Trabajo practico de introducción a la programación

Integrantes: Diaz Lucas Ariel y Avaca Matías
Comisión: Virtual

El proyecto consiste en implementar una aplicación web fullstack utilizando el framework Django que permita consultar las imágenes de la API pública proporcionada por la NASA. La información obtenida de esta API será renderizada por el framework en distintas cards, las cuales mostrarán la imagen, un título y una descripción.

Para iniciar el proyecto, descargamos e instalamos Visual Studio Code junto con sus extensiones necesarias. También creamos una cuenta en GitHub para almacenar el repositorio del grupo. Posteriormente, instalamos Django y las dependencias necesarias para ejecutar y visualizar la aplicación.

En esta primera etapa del trabajo, fue necesario resolver ciertas funciones en los archivos views.py y service_nasa_image_gallery.py.

Views.py

home (request): llama a la función auxiliar GetAllFavouriteList () y obtiene 2 listados: uno de las imágenes de la API y otro de favoritos por usuario.

```
def home(request):
    images= service_nasa_image_gallery. GetAllImages () #obtenemos las imágenes
    favouriteList= favouriteList. getAllFavouritesByUser(request) #obtenemos favoritos del usuario return render (request,'home.html',{'images' : images, 'favouriteList': favouriteList}))
```

```
def getAllImagesAndFavouriteList(request):
images,favouriteList=service_nasa_image_gallery.getAllImagesAndFavouriteList(request)
return images, favouriteList
#invoca a service_nasa_image_gallery. Y obtiene dos listados, uno de las imágenes de la API y otro de los favoritos del usuario.
```

service_nasa_image_gallery

obtiene un listado de imágenes desde transport.py y lo guarda en JSON collection.

recorre el listado de objetos JSON, lo transforma en una NASAcard y lo agrega en el listado de imágenes.

```
def getAllImages(input=None):
    json_collection = transport. getAllImages(input)
         images = [         mapper. NASACard(
title=item['data'][0]['title'],
description=item['data'][0]['description'],
image_url=item['links'][0]['href'],
date=item['data'][0]['date_created'][:10] # Asegurando el formato de
fecha YYYY-MM-DD

        )         for item in json_collection         if
'links' in item and item['links'] and 'href' in
item['links'][0]
    ]
return images
```

concluido el desarrollo de estas primeras funciones, la galería de imágenes se muestra adecuadamente (imagen, titulo y descripción).

Opcionales:

en esta parte del trabajo se presentó la dificultad en la función de inicio de sesión, ya que la información obtenida por medio de tutoriales resultaba confusa al implementarla en el trabajo del repositorio. Como también en el intento de realizar el opcional de paginación.

Inicio de sesión

```
@login_required def
index_page(request):
    return render (request, 'index.html')
```

se completa la función def index_page(request) en los archivos views.py
en urls.py (global)

```
from django. urls import include, path

urlpatterns = [
    path ('admin/', admin. site.urls),
path ('accounts/', include("django. contrib. auth.urls")),
path ('', include ('nasa_image_gallery. urls'))
]
```

Accounts: valida las urls de logout y login para su correcto funcionamiento.

Login_required: Este decorador indica a la función que, debe ejecutarse, una vez que el usuario se autenticó y además restringe el acceso hacia una vista especifica.

Para el deslogueo, se añadió la redirección hacia la pagina "index-page".

```
def exit(request):
    logout(request)
    return redirect('index-page')
```

Favoritos

En este apartado, se trabajó en los tres archivos indicados y hubo una pequeña modificación en el html, en una condicional de autenticación. A continuación se comentaran sus cambios respectivos.

En el archivo views.py, se modificó la funciones:

Def home: esta vista obtiene todas las imágenes y las favoritas del usuario, marca las imágenes que son favoritas, y luego renderiza la página principal (home.html) mostrando esta información.

```
@login_required
def home(request):
    images = services_nasa_image_gallery.getAllImages()  # Obtenemos
todas las imágenes
    favourite_list =
services_nasa_image_gallery.getAllFavouritesByUser(request)  # Obtenemos
favoritos del usuario

    # Crear un conjunto de títulos de imágenes favoritas para facilitar
la búsqueda
    favourite_titles = set(fav['title'] for fav in favourite_list)

    # se agrega la propiedad "estaEnTusFavoritos a cada imagen y se anexa
al home.html"
    for image in images:
        image.estaEnTusFavoritos = image.title in favourite_titles
```

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

getAllFavouritesByUser(request):

Esta función obtiene todas las imágenes favoritas del usuario que ha iniciado sesión. Llama a getAllFavouritesByUser del servicio services_nasa_image_gallery para obtener la lista de favoritos. Luego, renderiza la plantilla favourites.html y pasa la lista de favoritos al contexto para que se muestre en la página.

saveFavourite(request)

Esta función guarda una imagen como favorita para el usuario autenticado. Está protegida con @login_required para asegurar que solo los usuarios autenticados puedan acceder a ella. Verifica si el método de la solicitud es POST. Si es así, llama a saveFavourite del servicio services_nasa_image_gallery para guardar la imagen como favorita. Finalmente, redirige al usuario a la página principal (home) después de guardar la imagen.

deleteFavourite(request)

Esta función elimina una imagen de los favoritos del usuario que inició sesión. Cumple la misma funcionalidad que el "saveFavourite(request), pero para el propósito antes dicho.

```
@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('home')

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

service_image_gallery.py

Este archivo transporta los datos de cada función, el repositorio de datos y mapeo de objetos

getAllImages(input=None): se encarga de obtener una colección de imágenes en formato JSON desde la API de la NASA. Utiliza transport.getAllImages(input) para hacer la solicitud a la API. Luego, convierte cada imagen recibida en un objeto NASACard con propiedades como título, descripción, URL de la imagen y fecha. Finalmente, devuelve una lista de estos objetos NASACard.

getImagesBySearchInputLike(input): permite buscar imágenes basadas en un criterio específico. Simplemente llama a la función getAllImages(input) con el criterio de búsqueda proporcionado y devuelve los resultados obtenidos.

saveFavourite(request): guarda una imagen como favorita para el usuario que ha iniciado sesión. Toma los datos enviados en una solicitud POST (título, descripción, URL de la imagen y fecha) y crea un objeto NASACard con esos datos y el usuario autenticado. Luego, llama a repositories.saveFavourite(fav) para almacenar esta imagen favorita en la base de datos.

getAllImagesAndFavouriteList(request): obtiene tanto todas las imágenes de la API como la lista de imágenes favoritas del usuario autenticado. Primero, llama a getAllImages() para obtener todas las imágenes. Luego, utiliza repositories.getAllFavouritesByUser(get_user(request)) para obtener la lista de favoritos del usuario. Finalmente, devuelve ambas listas.

getAllFavouritesByUser(request): obtiene la lista de imágenes favoritas del usuario que ha iniciado sesión. Primero, obtiene el usuario autenticado usando get_user(request). Luego, llama a repositories.getAllFavouritesByUser(user) para obtener la lista de favoritos de este usuario y la devuelve.

deleteFavourite(request): elimina una imagen de los favoritos del usuario. Toma el ID del favorito a eliminar desde una solicitud POST. Luego, llama a repositories.deleteFavourite(fav_id) para eliminar la imagen favorita de la base de datos y devuelve el resultado de esta operación.

```
def getAllImages(input=None):

    json_collection = transport.getAllImages(input)

    images = [
        mapper.NASACard(
            title=item['data'][0]['title'],
            description=item['data'][0]['description'],
            image_url=item['links'][0]['href'],
```

```
            date=item['data'][0]['date_created'][:10]  # Asegurando el
formato de fecha YYYY-MM-DD
        )
        for item in json_collection
        if 'links' in item and item['links'] and 'href' in
item['links'][0]
    ]

    return images

def getImagesBySearchInputLike(input):
    return getAllImages(input)

def saveFavourite(request):
    fav = mapper.NASACard(
        title=request.POST.get('title'),
        description=request.POST.get('description'),
        image_url=request.POST.get('image_url'),
        date=request.POST.get('date')[:10],  # Asegurando el formato de
fecha YYYY-MM-DD
        user=get_user(request)
    )
    repositories.saveFavourite(fav)

def getAllImagesAndFavouriteList(request):
    images = getAllImages()
    favourite_list =
repositories.getAllFavouritesByUser(get_user(request))
    return images, favourite_list

def getAllFavouritesByUser(request):
    user = get_user(request)
    favourite_list = repositories.getAllFavouritesByUser(user)
    return favourite_list

def deleteFavourite(request):
    fav_id = request.POST.get('id')
    return repositories.deleteFavourite(fav_id)
```

repositories.py

El archivo repositories.py generalmente contiene funciones o clases que interactúan directamente con la sección “backend. Estas funciones o clases se encargan de realizar

operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos o en el sistema de almacenamiento.

Hubo muchos inconvenientes en investigar e implementar las características para el sistema de gestión de favoritos o bookmark, entre muchas variantes de escrituras. Se buscaron videos en idioma español, ingles e hindi y también se utilizó la ia para encaminar las pruebas realizadas. Se ha intentado implementar AJAX o JQUERY para la actualización de un pequeño sector de la página.

LOAD SPINNER:

Se han buscado tutoriales de su creación. Se implementó css para realizar el diseño del logo de carga asociada de manera temática a la pagina en cuestión, javascript por su parte, fue implementado para ejecutar el funcionamiento y la detención del mismo, paralelamente al tiempo de carga de la pagina home.html. Html, dio la estructura correspondiente, con su correspondiente clase.

```
<!-- Load spinner -->
<div id="load_spinner" class="spinner-container">
  <div class="star"></div>
</div>

<style>
  .spinner-container {
    display: none; /* Inicialmente oculto */
    position: fixed;
    z-index: 1000;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(255, 255, 255, 0.7);
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .star {
    width: 50px;
    height: 50px;
    background: linear-gradient(
      to bottom right,
      #f1c40f,
      #f39c12,
```

```
      #e74c3c,
      #9b59b6,
      #3498db,
      #1abc9c,
      #2ecc71,
      #27ae60,
      #16a085
    );
    clip-path: polygon(
      50% 0%,
      61% 35%,
      98% 35%,
      68% 57%,
      79% 91%,
      50% 70%,
      21% 91%,
      32% 57%,
      2% 35%,
      39% 35%
    );
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%) rotate(0deg);
    animation: rotateStar 1s infinite linear;
  }

  @keyframes rotateStar {
    from {
      transform: translate(-50%, -50%) rotate(0deg);
    }
    to {
      transform: translate(-50%, -50%) rotate(360deg);
    }
  }
</style>

<script>
  // Mostrar el spinner al iniciar la carga de la página
  document.addEventListener("DOMContentLoaded", function () {
    var spinnerContainer = document.getElementById("load_spinner");
    spinnerContainer.style.display = "flex"; // Mostrar el spinner
  });

  // Ocultar el spinner cuando la página haya cargado completamente
  window.addEventListener("load", function () {
    var spinnerContainer = document.getElementById("load_spinner");
    spinnerContainer.style.display = "none";
```

```
  });
</script>
```

Renovación de interfaz gráfica:

Se añadió un formato distinto a las cartas y al background, se incluyó una gradiente radial de dos colores para que haga un efecto temático tipo “espacial” al igual que los títulos, que poseen un text shadow o sombreado.

Se implementaron las media queries que sirven para hacer la pagina responsiva, es decir, que se adapte tanto para mobile como para pc.

Se cambió el formato de las cartas, para que haga un efecto “hover”. Esta sección fue la dinámica y maleable.

```
body {
    font-family: 'Roboto', sans-serif;
    background: #0f0f0f; /* Fondo oscuro */
    color: #fff; /* Texto principal en color claro */
    line-height: 1.6;
    margin: 0;
    padding: 0;
}

/* Encabezado */
.header {
    background: linear-gradient(to right, #00ff40, #3333ff); /* Gradiente
neón */
    color: #fff;
    padding: 20px 0;
    text-align: center;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.3);
}

.header h1 {
    font-size: 36px;
    margin: 0;
    text-transform: uppercase;
    letter-spacing: 3px;
    text-shadow: 2px 2px 6px rgba(0, 0, 0, 0.5); /* Sombra para destacar
el texto */
}

/* Navegación */
.navbar {
```

```
    background: rgba(0, 0, 0, 0.6); /* Fondo oscuro semi-transparente */
    color: #fff;
    padding: 10px 0;
    text-align: center;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
}

.navbar ul {
    list-style-type: none;
    padding: 0;
}

.navbar ul li {
    display: inline-block;
    margin-right: 20px;
}

.navbar ul li a {
    color: #fff;
    text-decoration: none;
    font-size: 18px;
    transition: color 0.3s;
}

.navbar ul li a:hover {
    color: #18a73b; /* Rosa neón al hacer hover */
}

/* Contenido principal */
.main-content {
    padding: 50px 20px;
}

.section-title {
    font-size: 28px;
    color: #fff;
    margin-bottom: 30px;
    text-align: center;
    text-shadow: 0 0 10px rgba(255, 255, 255, 0.8); /* Sombra para
destacar el título */
}

/* Cards */
.card {
    background: rgba(0, 0, 0, 0.8); /* Fondo oscuro semi-transparente */
    border-radius: 15px;
    box-shadow: 0 0 20px rgba(255, 255, 255, 0.2); /* Sombra más suave */
    overflow: hidden;
```

```
    position: relative;
    transition: transform 0.3s ease-in-out, box-shadow 0.3s ease-in-out;
    margin-bottom: 30px;
}

.card::before {
    content: '';
    position: absolute;
    top: -5px;
    left: -5px;
    right: -5px;
    bottom: -5px;
    background: linear-gradient(45deg, #00ff40, #3333ff); /* Gradiente
neón */
    z-index: -1;
    border-radius: 15px;
    opacity: 0;
    transition: opacity 0.3s;
}

.card:hover::before {
    opacity: 1;
}

.card img {
    width: 100%;
    height: 200px; /* Altura fija para todas las imágenes */
    object-fit: cover;
    border-top-left-radius: 15px;
    border-top-right-radius: 15px;
    transition: transform 0.3s ease-in-out;
}

.card:hover img {
    transform: scale(1.05);
}

.card .card-body {
    padding: 20px;
}

.card .card-title {
    font-size: 1.8rem; /* Tamaño más grande para el título */
    font-weight: bold;
    color: #fff; /* Texto blanco para contraste */
    text-shadow: 0 0 10px rgba(255, 255, 255, 0.8); /* Sombra para
destacar */
    margin-bottom: 10px;
```

```css
}

.card .card-text {
    color: #ccc; /* Texto gris claro */
    line-height: 1.6;
    margin-bottom: 20px;
}

.card .btn {
    background-color: #00ccff; /* Color azul neón */
    color: #fff;
    border: none;
    border-radius: 5px;
    padding: 10px 20px;
    transition: background-color 0.3s;
}

.card .btn:hover {
    background-color: #0099cc; /* Cambio de tono al hacer hover */
}

/* Tabla de favoritos */
.table-striped tbody tr:nth-of-type(odd) {
    background-color: rgba(0, 0, 0, 0.8); /* Fondo oscuro para filas
impares */
}

.table-striped tbody tr:nth-of-type(even) {
    background-color: rgba(0, 0, 0, 0.6); /* Fondo oscuro para filas
pares */
}

.table-hover tbody tr {
    position: relative;
    transition: background-color 0.3s, color 0.3s;
}

.table-hover tbody tr::before {
    content: '';
    position: absolute;
    top: -2px;
    left: -2px;
    right: -2px;
    bottom: -2px;
    background: radial-gradient(circle, #3333ff,#00ff40); /* Gradiente
radial en tonos neón */
    z-index: -1;
    border-radius: 5px;
```

```
    opacity: 0;
    transition: opacity 0.3s;
}

.table-hover tbody tr:hover::before {
    opacity: 1;
}

.table-hover tbody tr:hover {
    color: #fff; /* Texto blanco al pasar el ratón */
}

.table-hover tbody td {
    position: relative;
    z-index: 1;
    color: #fff; /* Texto blanco por defecto */
}

.table-hover tbody td img {
    max-width: 200px;
    max-height: 200px;
}

.table thead th {
    background-color: rgba(0, 0, 0, 0.8);
    color: #fff;
}

/* Footer */
.footer {
    background: linear-gradient(to top, #00ff40, #3333ff); /* Gradiente
neón */
    color: #fff;
    text-align: center;
    padding: 20px 0;
    position: fixed;
    bottom: 0;
    width: 100%;
}

/* Media Queries */
@media (max-width: 768px) {
    .header h1 {
        font-size: 28px;
    }
    .navbar ul li a {
        font-size: 16px;
    }
```

```
    .main-content {
        padding: 30px 10px;
    }
    .card {
        padding: 15px;
    }
}

@media (max-width: 576px) {
    .header h1 {
        font-size: 24px;
    }
    .navbar ul li a {
        font-size: 14px;
    }
    .card {
        padding: 10px;
    }
}
```