

### Licenciatura en Sistemas

# Trabajos Práctico

Nombre del TP

### Introducción a la Programación

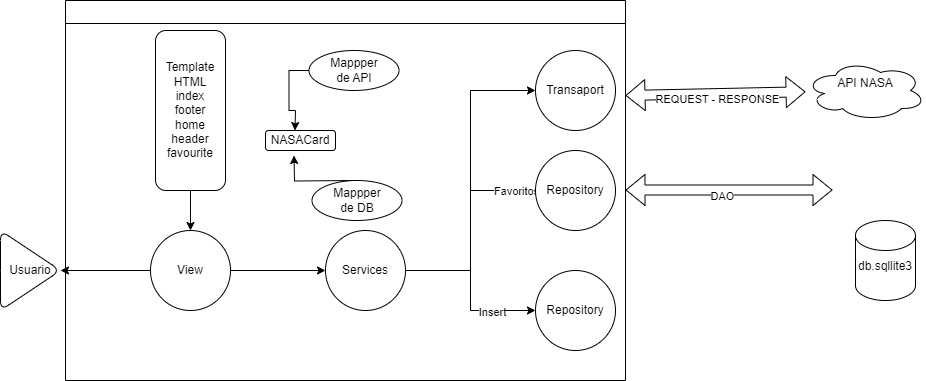
1st semestre del primer año 2024

Resumen: El trabajo consiste en hacer funcionar correctamente funciones que nos proporciona el programa para que la página web se ejecute correctamente

Integrantes: Facundo Rios Matías. Facundo.rios.colegio@gmail.com

Este trabajo consiste en una pagina con tematica Nasa. El programa ya viene casi armado y con funciones que deben ser arregladas para hacerlas funcionar correctamente.

La pagina tiene la siguiente arquitectura:



Para arreglar el programa primero se reviso las funciones que lo componen. Al revisar y analizar las funciones, se procedió a hacer que transport pida la información necesaria de la Api para las funciones de services puedan procesar esa información y convertirla en “NASA CARD”. Estas “NASA CARD” son procesadas para que las funciones de View las rendericen y sean mostrados al usuario correctamente.

Además, se arregló la función de search para que funcione el buscador de la pagina. Para que funcione se tuvo que modificar la función search del services para que devuelva las imágenes relacionadas con la palabra que busque el usuario.

Todas estas modificaciones y arreglos hace que la pagina renderice la información de la API y hacer funcionar el buscador. Este trabajo se logro gracias a los conocimientos obtenidos de en la cursada y a mi padre que es un desarrollador con muchos años de experiencia.

# Introducción a la Programación - primer semestre del 2024.

## Trabajo práctico: galería de imágenes de la NASA 🚀

[](https://camo.githubusercontent.com/ef2e766c9c1b9b176427e96abf8e0fd33586d2965af5e47fad63a5de4d54629d/68747470733a2f2f6170692e6e6173612e676f762f6173736574732f696d672f67656e6572616c2f61706f642e6a7067)

### Introducción

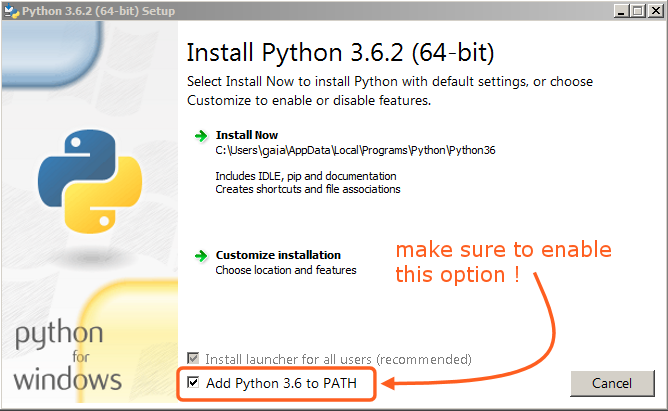
* El trabajo consiste en implementar una aplicación web fullstack usando [**Django Framework**](https://docs.djangoproject.com/en/4.2/) que permita consultar las imágenes de la API pública que proporciona la NASA. La información que provenga de esta API será renderizada por el framework en distintas cards que mostrarán -como mínimo- la imagen en cuestión, un título y una descripción. Adicionalmente -y para enriquecerla- se prevee que los estudiantes desarrollen la lógica necesaria para hacer funcionar el buscador central y un módulo de autenticación básica (usuario/contraseña) para almacenar uno o más resultados como **favoritos**, que luego podrán ser consultados por el usuario al loguearse. En este último, la app deberá tener la lógica suficiente para verificar cuándo una imagen fue marcada en favoritos.
* Gran parte de la aplicación ya está resuelta: solo falta implementar las funcionalidades más importantes 😉.

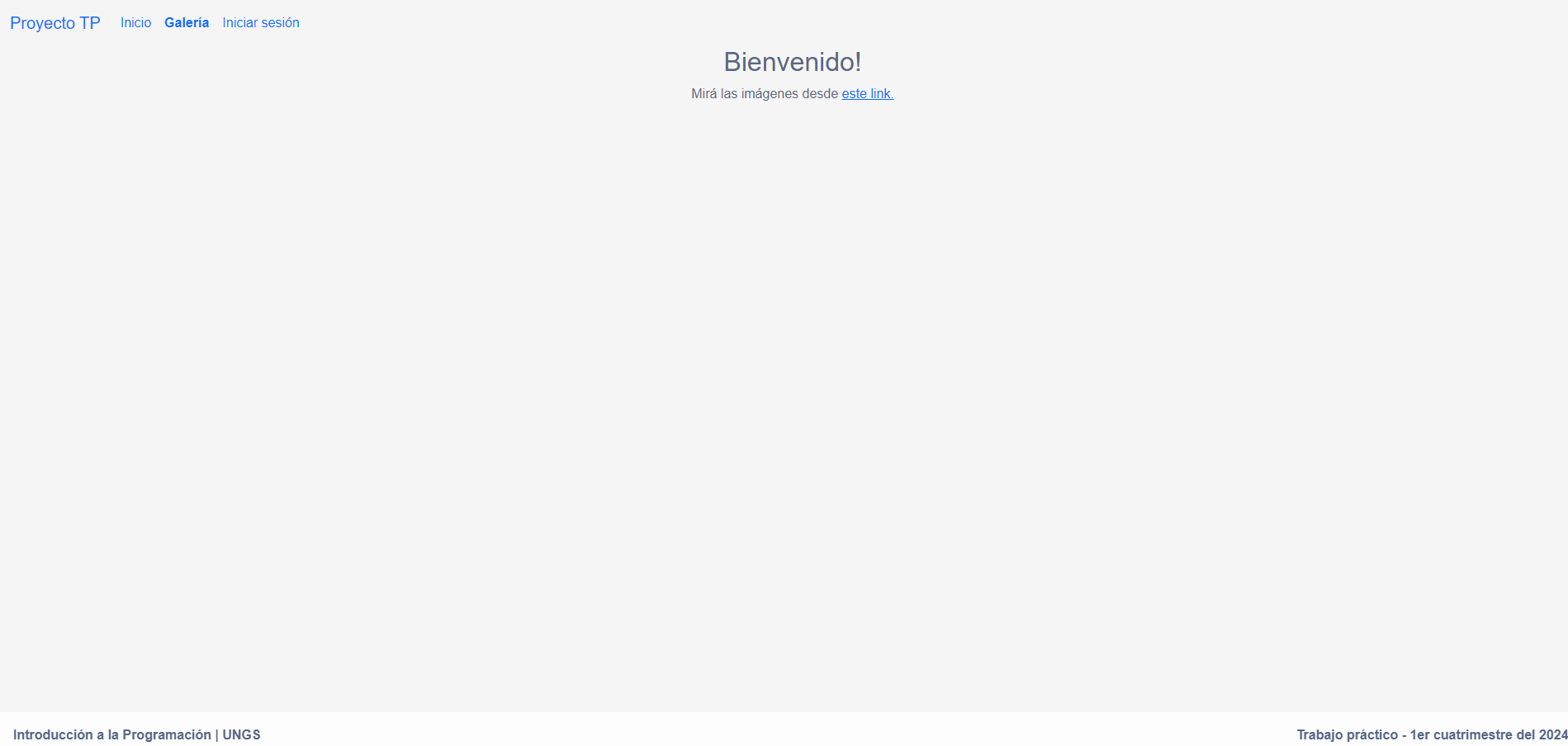
### ¿Cómo empiezo?

1. Descargá e instalá **Visual Studio Code** desde https://code.visualstudio.com/
   * Adicionalmente, se recomienda la instalación de las siguientes extensiones para facilitar el desarrollo:
     + After Dark.
     + Prettier - Code formatter.
     + Pylance.
     + Python.
     + Python Debugger.

Finalizada la instalación, ejecutá el programa. Deberías ver algo como lo siguiente (muestra dentro del mismo TP): [](https://camo.githubusercontent.com/6e29710b7e20b2cb807a9d52ef41561119c1f2f1d9ad7d51a25c34415982c194/68747470733a2f2f692e6962622e636f2f3042476b59727a2f657874656e732e706e67) Guía oficial de instalación de extensiones disponible [aquí](https://code.visualstudio.com/docs/editor/extension-marketplace).

1. Instalá la última versión de Python desde www.python.org. **Asegúrate de agregarlo al PATH durante la instalación:**

[](https://camo.githubusercontent.com/aef2adc9aa576f55d1b509fc70075c18a58aa55de8546f01d4ed3af721f5dd7f/68747470733a2f2f692e706f7374696d672e63632f4a6e5932635657712f707974686f6e2d696d6167652e706e67)

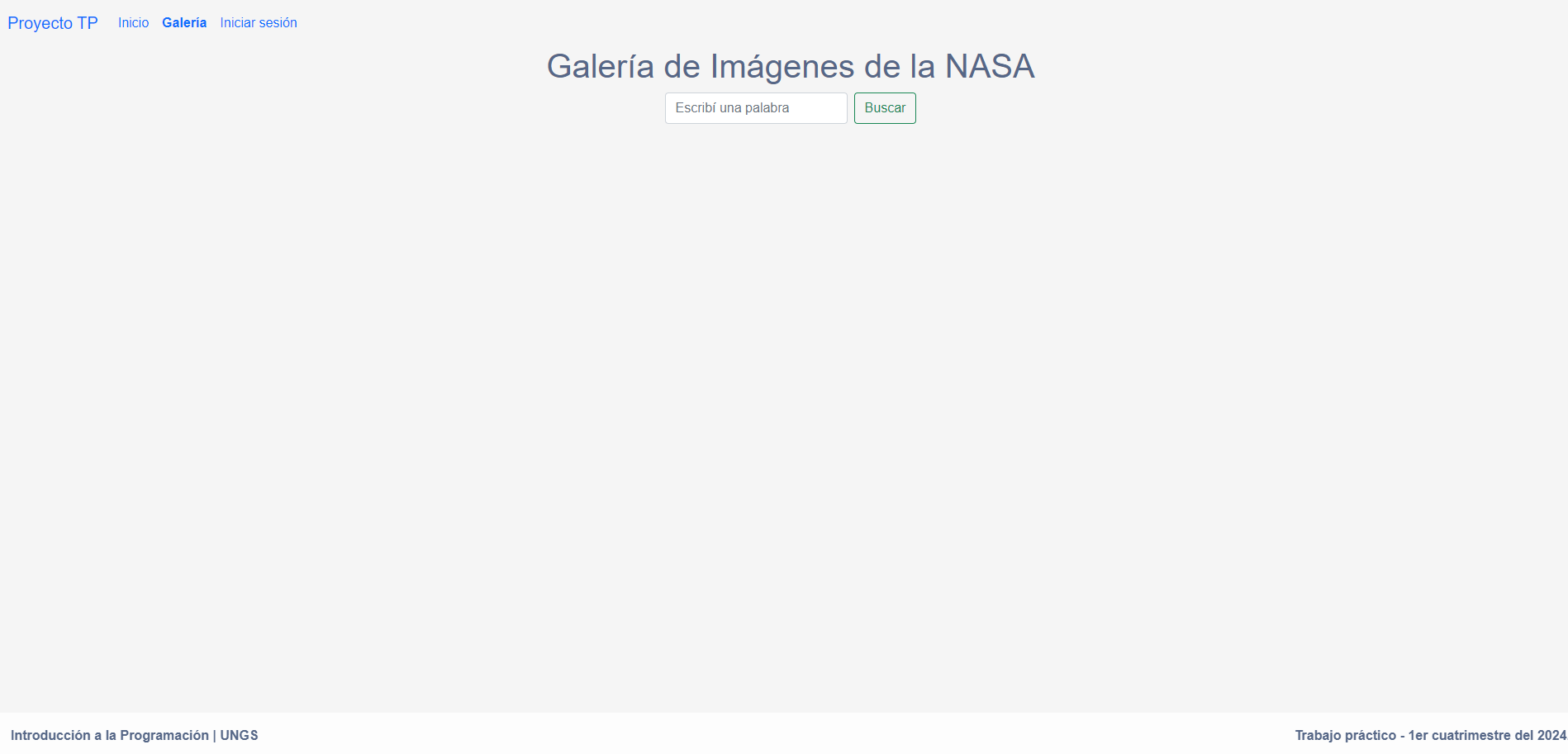
1. Creá una cuenta en GitHub [desde acá](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home). Luego, debés efectuar un fork (copia) del proyecto a tu repositorio (o al del grupo): [tutorial para hacer forks - thx. MitoCode](https://www.youtube.com/watch?v=9YUaf-uxuRM).
2. Cloná el repositorio copiado en tu máquina local (git clone). A continuación, dentro de la carpeta del repositorio local, abrí una terminal de VS Code e instalá Django ejecutando el siguiente comando: pip install django==4.2.10
3. Instalá las dependencias necesarias: pip install -r requirements.txt
4. Ejecutá el servidor Django (3000 representa el puerto donde se ejecutará la app): python manage.py runserver 3000
5. Abrí tu navegador y dirigíte a http://localhost:3000 para ver la aplicación. Deberá mostrar una pantalla como la siguiente: [](https://camo.githubusercontent.com/9de58e1f57d674cffb35868e104c4977492038650b6abea3e8fe1ba7a96fa1d8/68747470733a2f2f692e6962622e636f2f47464a646748722f67616c657269612d64656661756c742e706e67)
6. Por último, para ver el contenido de la base integrada (SQLite), recomendamos el uso de **DB Browser for SQLite**. Link de descarga: <https://sqlitebrowser.org/dl/>
   * El archivo que se debe abrir es **db.sqlite3**.

### Lo que ya está implementado

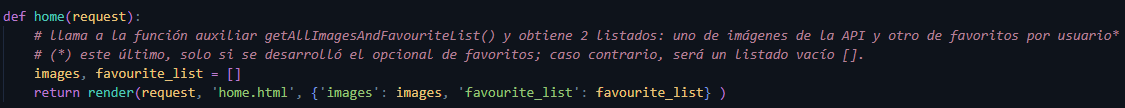
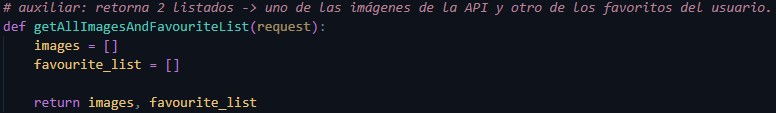
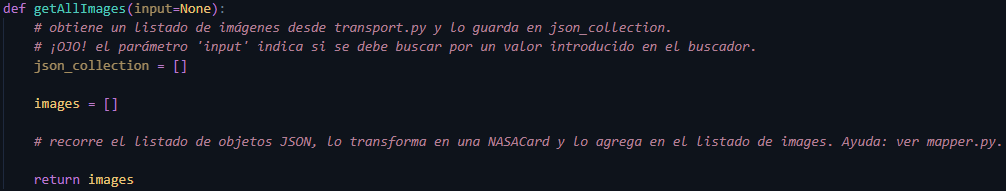
* A nivel **template**, se cuenta con 4 HTMLs: **header (cabecera de la página)**, **footer (pie de página)**, **home (sección donde se mostrarán las imágenes y el buscador)** e **index (contener principal que incluye a los 3 HTMLs anteriores).** Para el caso del header, se implementó cierta lógica para determinar si un usuario está logueado (o no) y obtener así su nombre; para el caso del home, éste tiene un algoritmo que permite recorrer cada objeto de la API y dibujar su información en pantalla. El footer no posee acciones a nivel código relevantes para el desarrollo.
* A nivel **views**, en el archivo **views.py** encontrarán algunas funciones semidesarrolladas: index\_page(request) que renderiza el contenido de 'index.html'; home(request) que obtiene todas las imágenes mapeadas de la API -a través de la capa de servicio- y los favoritos del usuario, y muestra el contenido de 'home.html' pasándole dicha información. Esta última hace uso de la función auxiliar getAllImagesAndFavouriteList(request) que devuelve 2 listas: una de las imágenes de la API y otra de las imágenes marcadas como favoritos del usuario.
* A nivel **lógica**, se incluye el archivo **transport.py** completo con todo el código necesario para consumir la API. Además, se anexa un **mapper.py** con la lógica necesaria para convertir/mapear los resultados en una **NASACard** (objeto que finalmente se utilizará en el template para dibujar los resultados).
* El proyecto está construido sobre una [**arquitectura multicapas**](https://medium.com/@e0324913/multilayered-software-architecture-1eaa97b8f49e), donde cada capa posee una única responsabilidad y se encuentra desacoplada del resto. Son las siguientes:
  + **DAO** (empleada para el alta/baja/modificación -CRUD/ABM- de objetos en una base de datos integrada, llamada [**SQLite**](https://github.com/Facuri13/Grupo-7/blob/main/X)).
  + **Services** (usada para la lógica de negocio de la aplicación).
  + **Transport** (utilizada para el consumo de la API en cuestión).

Si bien no es un parámetro de evaluación dónde colocan las funciones, es altamente recomendado que las funciones que se agreguen estén en las capas que correspondan (consultar con los docentes en caso de dudas).

### ¿Qué voy a ver al iniciar la app?

* Al iniciar la aplicación y hacer clic sobre **Galería**, verás lo siguiente: [](https://camo.githubusercontent.com/f8f6ee45b0e54949b817230f9b7c12dd24785ee8f50041b6b1fb53876ea76f09/68747470733a2f2f692e6962622e636f2f624e36626856472f67616c657269612d312e706e67)

### Lo que falta hacer

* Aún faltan implementar ciertas funciones de los archivos **views.py** y **services\_nasa\_image\_gallery.py**. Éstas son las encargadas de hacer que las imágenes de la galería se muestren/rendericen:
  + **views.py**:
    - home(request): invoca a getAllImagesAndFavouriteList(request) para obtener 2 listados que utilizará para renderizar el template. [](https://camo.githubusercontent.com/508597af76f769618866f65470ddea1a4473838df7ac22cdd840ee8a99652e4c/68747470733a2f2f692e6962622e636f2f306d4d4c5272762f67616c657269612d342e706e67)
    - getAllImagesAndFavouriteList(request): invoca al servicio correspondiente para obtener 2 listados, uno de las imágenes de la API y otro -si corresponde- de los favoritos del usuario. [](https://camo.githubusercontent.com/30dad0aa3097c7000505a96b899828b022866f4b3b487d8508a31be78cd00eda/68747470733a2f2f692e6962622e636f2f44705258586a352f67616c657269612d342e706e67)
  + **services\_nasa\_image\_gallery.py**:
    - getAllImages(input=None): obtiene un listado de imágenes de la API. El parámetro input, si está presente, indica sobre qué imágenes debe filtrar/traer. [](https://camo.githubusercontent.com/e04b62fcd9f689f1f7a7a773ac3d6ed52135fcda92cef99ab70dafed7f73a374/68747470733a2f2f692e6962622e636f2f6d7147524e66522f67616c657269612d332e706e67)

**Concluido su desarrollo, deberían ver algo como lo siguiente:** [](https://camo.githubusercontent.com/423354e33e934a4483da894057ddc23d0420d63d7854fae8c32aa0d0abb68b2c/68747470733a2f2f692e6962622e636f2f797074624733672f67616c657269612d322e706e67)

### Condiciones de entrega

* Requisitos de aprobación y criterio de corrección
  + El TP debe realizarse en grupos de 2 o 3 integrantes (no 1). Para aprobar el trabajo se deberán reunir los siguientes ítems:
    - La galería de imágenes se muestra adecuadamente (**imagen**, **título** y **descripción**).
    - El código debe ser **claro**. Las variables y funciones deben tener nombres que hagan fácil de entender el código a quien lo lea -de ser necesario, incluir comentarios que clarifiquen-. **Reutilizar el código mediante funciones todas las veces que se amerite.**
    - No deben haber variables que no se usan, funciones que tomen parámetros que no necesitan, ciclos innecesarios, etc.
  + **El 'correcto' funcionamiento del código NO es suficiente para la aprobación del TP, son necesarios todos los ítems mencionados arriba.**

### Opcionales

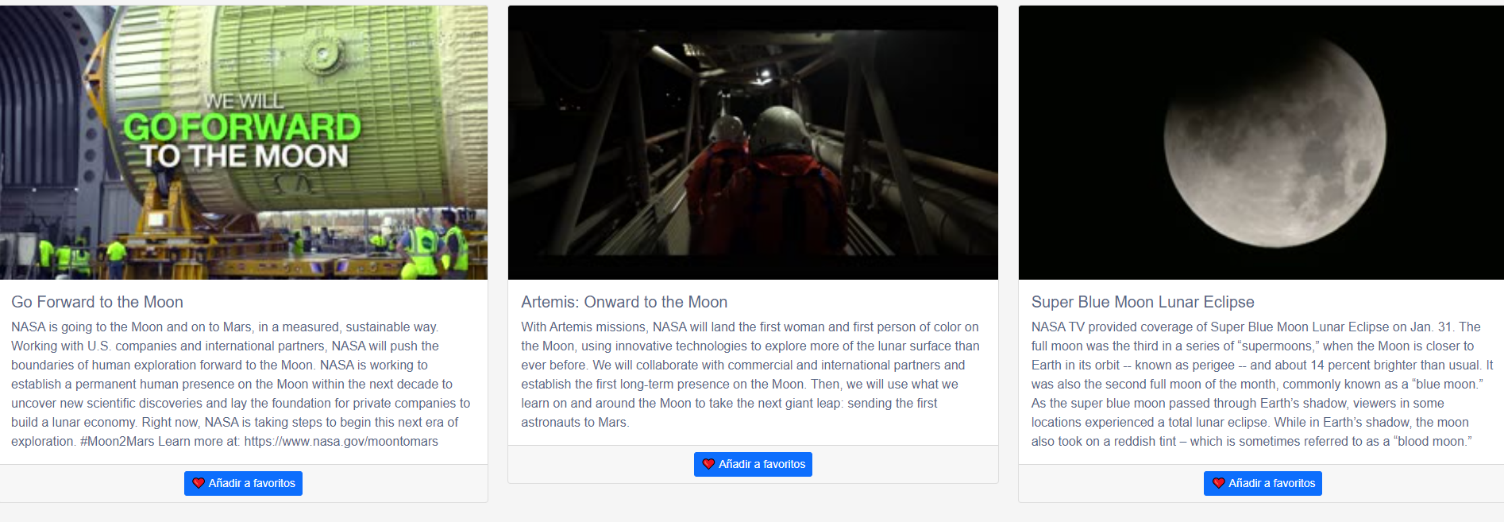
Las siguientes funcionalidades del juego NO son necesarias para la aprobación (con nota mínima), pero sirven para mejorar la nota del trabajo. De optar por hacerlas, se aplican las mismas reglas y criterios de corrección que para las funcionalidades básicas. Cualquier otra funcionalidad extra que se desee implementar debe ser antes consultada con los docentes.

**Los opcionales notados con ⭐ ya están parcialmente resueltos.** Se sugiere comenzar con ellos y luego seguir con los demás.

**⚠️NO ES NECESARIO REALIZAR TODOS LOS OPCIONALES.⚠️** Enfóquense en los más relevantes, teniendo en cuenta el tiempo de desarrollo y pruebas.

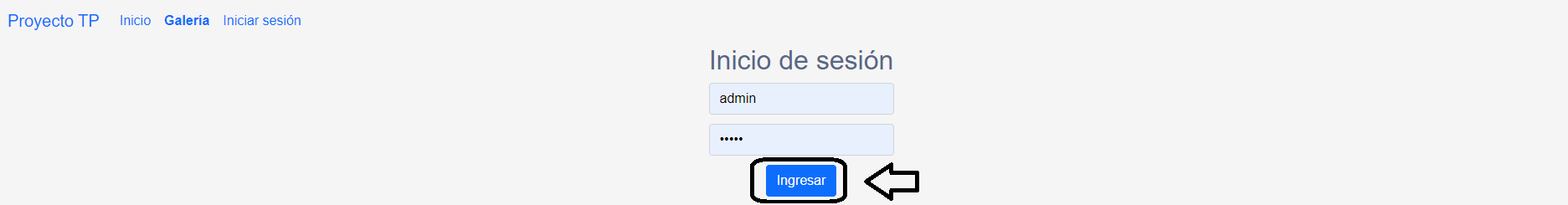
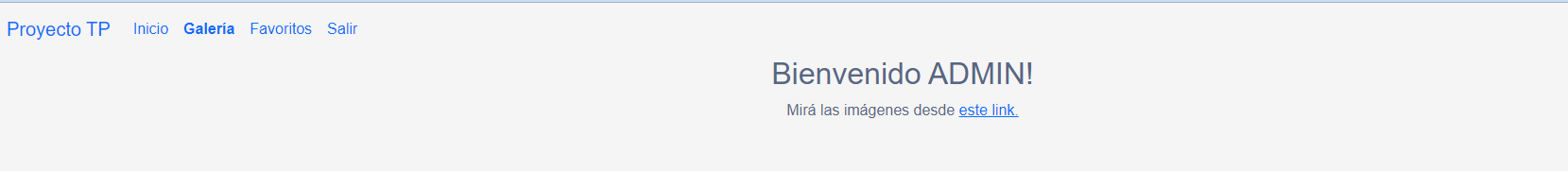
#### **Buscador** ⭐

* + Se debe **completar** la funcionalidad para que el buscador filtre adecuadamente las imágenes, según los siguientes criterios:
    - Si el usuario **NO** ingresa dato alguno y hace clic sobre el botón 'Buscar', debe filtrar por el valor predeterminado (space).
    - Si el usuario ingresa algún dato (ej. sun -sol, en inglés-), al hacer clic se deben desplegar las imágenes filtradas relacionadas a dicho valor.

Ejemplo para moon (luna). **ATENCIÓN, las imágenes pueden variar**: [](https://camo.githubusercontent.com/21b9c0735705e86c3311aced2526644d8af50db46cf543a6ea5c254f7ab11ebd/68747470733a2f2f692e6962622e636f2f707a356654716b2f67616c6572696131332e706e67)

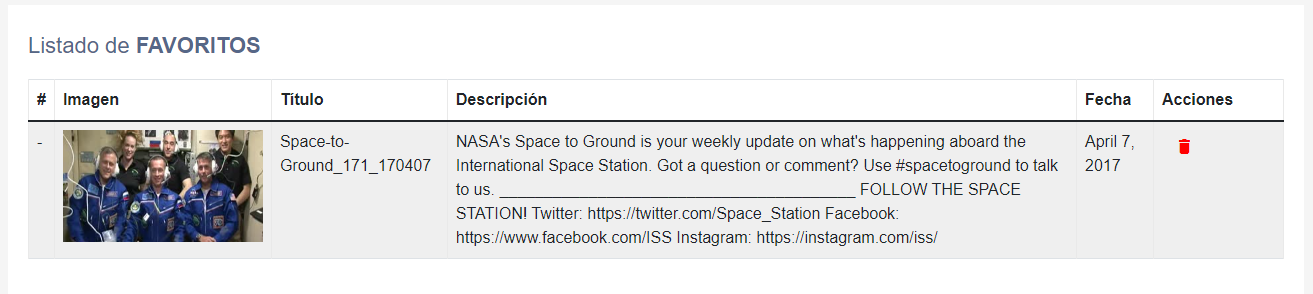
#### **Inicio de sesión** ⭐⭐

* + Se debe **completar** la feature de inicio de sesión de la app. El usuario y contraseña a utilizar, preliminarmente, es **admin**/**admin** (ya se encuentra guardado sobre la base SQLite, tabla auth\_user).
  + Consideraciones:
    - **NO** se permite utilizar Django Admin para emular la autenticación de los usuarios, la sección **Iniciar sesión** debe funcionar adecuadamente.
    - Solo los usuarios que hayan iniciado sesión podrán añadir las imágenes como favoritos y visualizarlas en su sección correspondiente.
  + Ayuda: [tutorial de autenticación login/logout básica](https://www.youtube.com/watch?v=oKuZQ238Ncc)

Una posible visualización del inicio de sesión es: [](https://camo.githubusercontent.com/9f3dee734561f4a948fea321898437e15adcfffd49574f817cf82c19e7876fae/68747470733a2f2f692e6962622e636f2f6e4d48474644392f73657373696f6e2d312e706e67) [](https://camo.githubusercontent.com/375597c9504a3df2d2cd17db2c7f084b6a360aae7ff9097dbf04d7bcac241643/68747470733a2f2f692e6962622e636f2f63777a63424e782f73657373696f6e2d322e706e67)

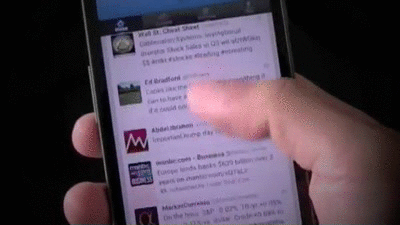
#### **Favoritos** ⭐⭐

* + Se debe **completar** la lógica presente para permitir que un usuario logueado pueda almacenar una o varias imágenes de la galería como **favoritos**, mediante el clic de un botón en la parte inferior.
  + **Observaciones**
    - Este punto puede realizarse SOLO si el ítem anterior (inicio de sesión) está desarrollado/funcionando bien.
    - Si el favorito ya fue añadido, debe mostrarse un botón que impida reañadirlo.
    - Debe existir una sección llamada 'Favoritos' que permita listar todos los agregados por el usuario, mediante una tabla. Además, debe existir un botón que permita removerlo del listado (**si fue removido, desde la galería de imágenes podrá ser agregado otra vez**).
  + **Parte del código ya está resuelto**. Revisar los archivos views.py, repositories.py y services\_nasa\_image\_gallery.py.

Una posible visualización de este ítem resuelto es: [](https://camo.githubusercontent.com/7558a7d9530c90a2280a6640f61b6c62376c3997613d0a6e85bf89c3ceb646c0/68747470733a2f2f692e6962622e636f2f30394c6b6e584e2f67616c657269612d31312e706e67) [](https://camo.githubusercontent.com/4e8d9e07c178944080dac1111fe1af5694be26c889a6b1f891f0b9e275070944/68747470733a2f2f692e6962622e636f2f6e445172584c632f67616c6572696131322e706e67)

#### Paginación de resultados (A) ó **infinite scroll** (B) (estilo Instagram ó Facebook).

* + (A) Se desea implementar la paginación de los resultados de búsqueda, de forma tal que:
    - Por cada página, se muestren 5\* imágenes. Es de interés que este número lo pueda escoger el usuario (definir/investigar la mejor forma de lograrlo).
    - Se deben listar TODAS las imágenes de la API. Si el número de imágenes no es **múltiplo** de la cantidad escogida, el **resto** debe figurar en una página adicional.
  + (B) Se desea implementar un algoritmo que permita mostrar cierta cantidad de imágenes y, a partir del swap/deslizamiento de la barra de scroll vertical, cargue las demás hasta completar su tope (similar al scroll infinito que poseen aplicaciones como Instagram, Facebook o X). **Es de interés que el usuario pueda configurar la cantidad de imágenes mostradas en un principio.**

[](https://camo.githubusercontent.com/6ccfa1723a6a1e2c1b5589332f37f8a7f5578297222c39367de626f6efd2e2c7/68747470733a2f2f692e6d616b65616769662e636f6d2f6d656469612f31312d31302d323031342f64584162755f2e676966)

#### Añadir comentarios en imágenes marcadas en favoritos

* + Se desea que, cada vez que se añada una nueva imagen a favoritos, se visualice un mensaje cargado por el usuario al hacer clic sobre el botón correspondiente. **Este mensaje debe visualizarse en la tabla de la sección en cuestión.**

#### ALTA de nuevos usuarios

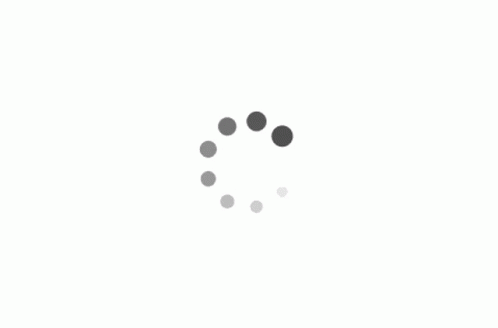
* + Actualmente la aplicación no permite el registro/alta de nuevos usuarios. Se desea implementar esta sección, para permitir que cualquier persona pueda registrarse en la aplicación.
  + Consideraciones:
    - Se debe solicitar nombre, apellido, usuario, contraseña y correo electrónico. **Si dos personas poseen el mismo nombre de usuario se anulará el alta, visualizando un mensaje descriptivo del error.**
    - El registro exitoso debe disparar un correo a la casilla indicada por el usuario, que indique en el cuerpo del mismo las credenciales de acceso.
  + Ayuda: [envío de emails usando cuenta @gmail a través de Django](https://github.com/akjasim/cb_django-sending-emails)

#### Internacionalizar (i18n) aplicación para soportar múltiples idiomas

* + Se debe desarrollar una lógica que permita switchear el idioma de la aplicación, de español a inglés o portugués y viceversa.
    - El switch debe ejecutarse desde la misma página, según el usuario lo requiera.

#### Loading Spinner para la carga de imágenes

* + Se desea implementar una pantalla de carga/loading spinner que indique el usuario que espere hasta que la carga de imágenes se complete.

[](https://camo.githubusercontent.com/7a93a9de21524e0c5f8bdd64887b5d9a6304a54aec1f79e933e97e8a4d68f692/68747470733a2f2f6d656469612e74656e6f722e636f6d2f7445426f5a753149534a3841414141432f7370696e6e696e672d6c6f6164696e672e676966)

#### Eliminar resultado de búsqueda no interesante (botón 'no mostrar esta img')

* + El usuario tendrá la posibilidad de marcar una imagen de la galería como **no interesante**. Si una imagen fue marcada de esta forma, además de eliminarse automáticamente de la galería, NO se mostrará en futuras búsquedas.
  + Ayuda: pensar a las imágenes **no interesantes** como una **lista** de NASACards, con un **atributo especial** que permita decidir, para determinado usuario, si resulta o no relevante en la búsqueda.

Debería verse algo similar a la siguiente idea: [](https://camo.githubusercontent.com/38766bc092e04031efbee545202bc2ad68c93107e2cd52e16b7247760dcf5a22/68747470733a2f2f692e6962622e636f2f47704457474d662f616e756e63696f2e706e67)

#### Renovar interfaz gráfica

* + Se debe proponer una nueva interfaz gráfica para los distintos templates de la aplicación.
  + Recomendaciones:
    - Pueden usar el framework CSS que deseen, sea [Bootstrap](https://getbootstrap.com/), [Tailwind](https://tailwindcss.com/), [Foundation](https://get.foundation/), etc., siempre y cuando **consideren que el código debe resultar LEGIBLE para su corrección**.
    - Verificar que la lógica implementada en los templates funcione bien a medida que se modifica la interfaz.

#### Mapeo/traducción automática de palabras ingresadas en español, a inglés, para el correcto funcionamiento del buscador

* + En la actualidad, el buscador solo admite palabras escritas en inglés, lo cual puede dificultar su uso para personas que no hablen ese idioma. Se debe implementar un algoritmo que permita mapear las palabras ingresadas en el buscador, de español a inglés, permitiendo que aún si el usuario ingresa algo en su idioma nativo lo traduzca antes de enviar la petición a la API, no alterando el normal funcionamiento de ésta y ampliando la utilidad del mismo.
  + Tips:
    - Opción 1: pensar un archivo tipo JSON (key, value), que contenga TODAS las posibles palabras que pueda ingresar la persona en el buscador (español e inglés).
    - Opción 2: usar la [API de Google Translate](https://codeloop.org/google-translate-api-with-python/) para traducir los ingresos de los usuarios internamente, antes de enviar la petición.

### Fecha de entrega

El trabajo debe ser entregado en la fecha estipulada en el cronograma. **Recordar que es requisito hacer pre-entregas.**

### Formato de entrega

* La entrega se dividirá de 2 partes: **código** e **informe**:
  + **Parte 1: código:** todo el desarrollo debe estar en un repositorio interno del grupo (fork del repo base del TP). Se deben añadir a los docentes de la comisión con motivo de verificar los avances del mismo (corregir funciones, brindar sugerencias o recomendaciones, etc). Dado el caudal de alumnos, **serán responsables los estudiantes de notificar a los docentes para evaluar una pre-entrega, corregir alguna duda o similar que bloquee/impida del avance del TP**.

Sugerimos:

* + - Que cada integrante tenga su propia cuenta de GitHub, NO usar una única en el proyecto.
    - Cada integrante debe commitear una o varias porciones de código, dependiendo cómo distribuyan el trabajo. **Se debe visualizar el aporte individual al TP.**
  + **Parte 2: informe:** deben redactar un documento donde exista una introducción que explique de qué se trata el trabajo (sin utilizar lenguaje técnico), que incluya el código de las funciones implementadas y una breve explicación de cada una de ellas junto con las **dificultades de implementación** y **decisiones tomadas** -con su correspondiente justificación-. **NO incluir explicaciones de funcionalidades de Python, Django o similares**. Este documento debe estar en formato PDF anexo dentro de la carpeta del TP.

🔥 **Se DEBE cumplir con ambas partes (código + informe) para aprobar el trabajo práctico.**

### Documentación adicional

* Documentación oficial de Django disponible aquí: <https://docs.djangoproject.com/en/4.2/>
* Sección **GIT**
  + Introducción a GIT: [clic acá](https://www.youtube.com/watch?v=mzHWafbVRyU).
  + Manejo de ramas/branches: [clic acá](https://www.youtube.com/watch?v=BRY9gamL9PE).
  + Merge & resolución de conflictos: [clic acá](https://www.youtube.com/watch?v=9YUaf-uxuRM).