



Universidad
Nacional de
General
Sarmiento

Informe

Trabajo práctico: Introducción a la programación

Profesores: Argañarás Omar - Benesch Alicia

Alumnos: Santillan Anabella Noemi - Díaz Sofía Belén

Fecha de entrega: 26/06/2024

Comisión 04 - Primer cuatrimestre 2024

Índice

1. Introducción
2. Desarrollo del código
3. Conclusión

1. Introducción

En este trabajo, tenemos como objetivo la creación de una aplicación web funcional mediante el uso de los conocimientos aprendidos en clase y los obtenidos mediante la investigación que debimos llevar a cabo para desarrollar el código. Recibimos funciones incompletas cuyos códigos debimos completar para que la página pudiera funcionar, y además de eso, los templates .html y un archivo .CSS que modificamos para implementar nuevas cosas. Mediante la utilización del framework Django, JavaScript, librerías de CSS como Bootstrap 5 y el correcto acceso a la API de la NASA, logramos crear una página funcional y estéticamente agradable. A continuación, mostraremos las modificaciones que aplicamos en los diferentes archivos del trabajo práctico, y explicaremos brevemente dichos cambios.

2. Desarrollo del código

Funcionamiento de la galería de imágenes y barra de búsqueda

Comunicación con la API: Para comenzar, la API de la NASA se comunica con la aplicación en formato de texto JSON, esto se realiza a través del archivo “transport.py”, mediante la función definida como “getAllImages”, la cual devuelve una lista con estos datos.

Capa de servicio/lógica de negocio: Dentro del archivo

“services_nasa_image_gallery.py”, la función “getAIIImages” recibe el listado de json y los guarda dentro de la variable ‘json_collection’. En siguiente paso, recorre esta lista y con la función ‘fromRequestIntoNASACard’ llamada desde el archivo “mapper.py” estos datos son transformados en NASACards y guardados dentro de la lista ‘images’.

```
def getAIIImages(input=None):

    json_collection = transport.getAIIImages(input)

    images = []

    for i in json_collection:

        foto = mapper.fromRequestIntoNASACard(i)

        images.append(foto)

    return images
```


Funcionamiento básico: Para que las “NASACards” sean renderizadas, se utiliza una función auxiliar desde el archivo “views.py”. Esta función es ‘getAIIImagesAndFavouriteList’ que devuelve dos listas, la primera una lista con todas las imágenes de la API y la segunda con las imágenes guardadas como favoritas por el usuario.

```
def getAIIImagesAndFavouriteList(request):

    images = services_nasa_image_gallery.getAIIImages()

    # favourite_list = []

    return images #, #favourite_list
```



Renderización de las “NASACards”: Estas imágenes son mostradas con la función ‘home’, que recibe los dos listados de la función anterior y los renderiza. Dentro de la página principal mostrará todas las imágenes tr aídas de la API en forma de NASACards.

Barra de búsqueda: Para el funcionamiento, se utiliza la función “search”, que comprueba si se ingresó o no, alguna palabra en la barra de búsqueda. La variable “search_msg” contiene el string ingresado por el usuario. En primer caso, si no se ingresó nada y se presionó el botón “buscar”, la función pasará por el primer if, y solamente recargará la página. En segundo caso, si hay alguna palabra ingresada en la barra de búsqueda, seguirá el else, y mostrará aquellas imágenes que únicamente contengan la palabra ingresada por el usuario.

def home(request):

images = getAllImagesAndFavouriteList(request)

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

def search(request):

images = getAllImagesAndFavouriteList(request) # images, favourite_list =
getAllImagesAndFavouriteList(request)

search_msg = request.POST.get('query', '')

if search_msg == "":

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

else:

```
filtro =
services_nasa_image_gallery.getImagesBySearchInputLike(search_msg)

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

Inicio de sesión admin:

Vistas de autenticación: dentro del archivo de “urls.py”, se importa desde “django.contrib.auth” las vistas de autenticación con el siguiente comando:

```
from django.urls import include

from django.contrib.auth import views as auth_views #acceso a las vistas de
autenticación
```

Path de inicio de sesión y salida de la sesión: siguiendo el archivo “urls.py” se agregan los path (camino) vinculan las URLs a las vistas de autenticación y cierre de sesión, respectivamente. Esto permite que, al redireccionar a esas URLs, se inicie y se cierre la sesión del usuario:

```
path('login/', auth_views.LoginView.as_view(), name='login'), #Cuando el usuario
es redireccionado a login, es autenticado
```

```
path('exit/', auth_views.LogoutView.as_view(), name='exit'), #Cuando el usuario es
redireccionado a exit, es desauntenticado
```

Vistas de “Iniciar sesión” y “Salir” para el usuario: en el archivo “header.html” en primer caso de que el usuario no esté logueado, se mostrará un botón de “iniciar sesión”, el cual tiene asignado el path de “login”. En segundo caso, cuando el usuario quiera salir de su sesión, se mostrará un botón de “salir”, que tiene asignado el path de “exit”:

```
<a class="nav-link" href="{% url 'exit' %}">Salir</a> {% else %}

<a class="nav-link" href="{% url 'login' %}">Iniciar sesión</a> {% endif %}
```

Login del usuario: en el archivo "login.html", una vez que el usuario ingrese su nombre de usuario y contraseña, el formulario se enviará a la URL asociada con 'login', la cual hemos vinculado a la vista de autenticación:

```
<form action="{% url 'login' %}" method="POST" style="display: inline-block;">
```

MODIFICACIONES DENTRO DE HEADER.HTML:

- Dentro de <head>:

```
<title>GALERÍA DE IMÁGENES DE LA NASA</title>
```

Modificamos el título que aparece en la pestaña de la página.

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
```

```
<script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcldslK1eN7N6jleHz
" crossorigin="anonymous"></script>
```

Se implementaron los enlaces a los archivos CSS y JavaScript de Bootstrap (la librería que utilizamos para cambiar la interfaz gráfica de la página), para poder aplicar estilos y clases CSS junto con los componentes interactivos de Bootstrap.

```
<style>
```

```
body {

    background-image: url('{% static "imagen/fondo.jpg" %}');

    background-size:contain;

    background-position: center;

    z-index: -1;

}

body.login-page {

    background-size:cover;
```

```
z-index: -1;

}

body.favourites-page {

    background-size:cover;

    z-index: -1;

}

</style>
```

Dentro de <head>, aplicamos el elemento <style> para asegurarnos de que la imagen y el formato del fondo se vea igual en todas las páginas en las que se extienda header.html, a excepción de la página de los favoritos y el inicio de sesión. Para lograr dicha excepción, agregamos las clases “login-page” y “favourites-page”, y les asignamos un estilo diferente al de la página principal.

- Dentro de <body>:

```
<body class="{% block fondo_login %}{% endblock %}">
```

```
<div

    class="background-image"

    style="background-image: url('static/imagen/fondo.jpg');"

></div>
```

```
<div class="color-overlay d-flex justify-content-center align-items-center">
```

Primero agregamos el bloque “fondo_login”, que luego utilizamos en las páginas de favoritos y del inicio de sesión para sobrescribir el estilo y poder personalizarlo a gusto en cada una.

Luego, dentro del <div>, agregamos la clase de la imagen de fondo y con el elemento “style” le otorgamos el url correspondiente para que pudiera encontrar la imagen que añadimos a los archivos del trabajo práctico. La imagen “fondo.jpg” se encuentra dentro de la carpeta “static”, en donde agregamos una nueva carpeta llamada “imagen”, que

contiene la foto .jpg que elegimos para el fondo.

Finalmente, agregamos la clase “color-overlay”, a la cual simplemente utilizamos para darle un overlay más oscuro a la imagen que utilizamos de fondo.

```
<nav class="navbar navbar-expand-lg navbar-dark">
```

Modificamos el color de la barra de navegación con “navbar-dark”.

```
<ul class="navbar-nav ms-auto">
```

Decidimos agregar la clase “ms-auto” para alinear hacia la derecha los botones de la barra de navegación.

```
<li class="nav-item">
```

```
    <a class="nav-link active" aria-current="page" href="{% url 'index-page' %}"><strong>Inicio</strong></a>
```

```
</li>
```

```
<li class="nav-item">
```

```
    <a class="nav-link active" aria-current="page" href="{% url 'home' %}"><strong>Galería</strong></a>
```

```
</li>
```

Agregamos a los hipervínculos Inicio y Galería para enfatizarlos y que se vean en negrita.

MODIFICACIONES DENTRO DE INDEX.HTML:

```
{% extends 'header.html' %} {% block content %}
```

```
<div class="container-fluid">
```

```
    <div class="content-container">
```

```
        {% if request.user.is_authenticated %}
```

```
            <h2 class="text-center titulo"><strong>¡Bienvenido/a {{ user.username | upper }}!</strong></h2>
```

```
        {% else %}
```



```

<h2 class="text-center titulo"><strong>¡Bienvenido/a!</strong></h2>

{% endif %}

<p class="text-center custom-text2">

    Mirá las imágenes desde

    <a href="{% url 'home' %}" class="btn btn-outline-light custom-btn btn-sm"> AQUÍ</a>

</p>

</div>

</div>

{% endblock %}

```

- Agregamos la clase “container-fluid”, que engloba todo el contenido de index.html, para que el contenido se expanda y ajuste fluidamente en toda la pantalla.
- Implementamos el div "content-container" para luego poder modificar el estilo desde el archivo CSS (se explicarán los cambios aplicados cuando mostremos las modificaciones realizadas en dicho archivo).
- Agregamos la clase “titulo” al h2 para luego aplicar estilos CSS al texto de bienvenida. También se utiliza para resaltar el texto en negrita. Además, cambiamos “Bienvenido” por “Bienvenido/a”.
- Agregamos la clase “custom-text2” para poder modificar el texto desde el archivo CSS.
- Lo que antes era un hipervínculo que mostraba “este link”, fue transformado en un botón pequeño (btn-sm) y transparente con bordes blancos (btn-outline-light) que muestra “AQUÍ” gracias a la implementación de “class=“btn btn-outline-light btn-sm””

MODIFICACIONES DENTRO DE HOME.HTML:

```

<style>

```

```
.loader{
  position:fixed;
  top: 0;
  left:0;
  width: 100vw;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  background: #413f3f;
  z-index: 1000;
  transition: opacity 0.75s, visibility 0.75s;
}

.loader--hidden {
  opacity: 0;
  visibility: hidden;
}

.loader::after {
  content: "";
  width: 75px;
  height: 75px;
  border: 15px solid #ddd;
  border-top-color: rgb(107, 171, 197);
}
```

```

border-radius: 50%;

animation: loading 0.75s ease infinite;
}

@keyframes loading {

  from { transform: rotate(0turn) }

  to {transform: rotate(1turn);}

}
</style>

```

Al principio de todo en el archivo home.html, luego de “{% extends 'header.html' %} {% block content %}”, utilizamos el elemento `<style>` para la implementación del spinner de carga, agregándole un estilo y una animación. A continuación, explicaremos qué hicimos con cada clase:

- **.loader{}:** creamos el estilo para la pantalla que se mostrará mientras esté el loader (modificamos alineación, color de fondo, la transición que hará antes de desaparecer cuando termine de cargar la página, etc).
- **.loader--hidden{}:** definimos la clase que quitará al spinner de la pantalla una vez que cargue el contenido de la página.
- **.loader::after{}:** creamos un pseudoelemento con “::after” que nos permite modificar el estilo del spinner en CSS sin necesidad de modificar el código en html.
- **@keyframes loading{}:** definimos la animación de rotación que hará el spinner con la regla “@keyframes”.

- Dentro de `<body>`:

```
<div class="loader"></div>
```

- Dentro de `<main>`:

Creamos la clase “loader”, que es la que implementamos dentro de <style>.

```
<main class="container">
```

Luego, capturamos todo el contenido de home.html dentro de un container. Tomamos esta decisión ya que lo necesitábamos para luego poder colocar el script del spinner al final, y que este desapareciera solo después de que cargara todo el contenido dentro del container (las imágenes).

```
<h1 class="tituloHome text-center">Galería de Imágenes de la NASA</h1>
```

Agregamos la clase “tituloHome” para luego poder modificar el título desde el archivo CSS.

```
<button class="boton btn-outline-dark btn-sm" type="submit">Buscar</button>
```

Modificamos el código del botón de la barra de búsqueda para cambiarlo de verde a un color oscuro (btn-outline-dark) y hacerlo más pequeño (btn-sm).

```
<div class="row row-cols-1 row-cols-md-3 g-4 mt-4">
```

Agregamos la clase de Bootstrap “mt-4” para darle un mayor margen desde el top de la página al contenedor de las imágenes, ya que considerábamos que estaban excesivamente cerca a la barra de búsqueda.

```
<script>
```

```
window.addEventListener("load", () => {  
    document.querySelector(".loader").classList.add("loader--hidden");  
});  
</script>
```

Este código fue escrito dentro del <body>, pero fuera del contenedor <main>. Se trata del script que le da el funcionamiento al spinner, y se encuentra fuera del contenedor <main> ya que, como mencionamos antes, debe desaparecer sólo después de que completen su carga los elementos que se encuentren dentro del contenedor (en este caso, nos enfocamos principalmente en la carga de las

imágenes). Básicamente, el programa espera a que cargue el contenido (`window.addEventListener("load", () =>)` y sólo entonces oculta el spinner (`document.querySelector(".loader").classList.add("loader--hidden");`)).

MODIFICACIONES DENTRO DE FOOTER.HTML:

```
<div class="text-center p-3" style="display: flex; justify-content: space-between;">
```

En el caso del footer, sólo quitamos del `<div>` la propiedad “background-color: rgb(255 255 255 / 78%)”, ya que consideramos que el color no se veía estéticamente correcto con las modificaciones que aplicamos a la interfaz gráfica de la página, y preferimos que quede transparente.

MODIFICACIONES DENTRO DE LOGIN.HTML:

```
{% extends 'header.html' %}
```

```
{% block fondo_login %}login-page{% endblock %}
```

```
{% block content %}
```

Antes del código, entre las declaraciones de plantilla, agregamos “`{% block fondo_login %}login-page{% endblock %}`” para que la página de inicio de sesión herede el estilo que definimos en `header.html`.

```
<div class="login-form mt-5" style="text-align: center;">
```

Agregamos “mt-5” para darle al formulario de inicio de sesión un mayor margen desde el tope de la página.

```
<h2 class="inicioSesion text-center">Inicio de sesión</h2>
```

Definimos la clase “inicioSesion” para luego poder modificar el texto desde el archivo CSS.

```
<button type="submit" class="btn btn-outline-light btn-block">Ingresar</button>
```

Cambiamos el color del botón “Ingresar” a uno transparente con bordes blancos (`btn-outline-light`).

MODIFICACIONES DENTRO DE FAVOURITES.HTML:



```
{% extends 'header.html' %}
```

```
{% block fondo_login %}favourites-page{% endblock %}
```

```
{% block content %}
```

Antes del código, entre las declaraciones de plantilla, agregamos “{% block fondo_login %}favourites-page{% endblock %}” para que la página de inicio de sesión herede el estilo que definimos en header.html.

MODIFICACIONES DENTRO DE STYLES.CSS:

```
.background-image {  
    background-image: url('{% static "static/imagen/fondo.jpg" %}');  
    height: 100vh;  
    width: 100vw;  
    position: absolute;  
    text-shadow: 2px 2px 2px rgba(0, 0, 0, .2);  
    top: 0;  
    left: 0;  
    z-index: -1;  
}
```

Aplicamos estilo a la imagen de fondo que elegimos en header.html: le otorgamos el url, elegimos su altura y el ancho, la posición, modificamos el texto que se verá encima, lo anclamos a la esquina superior izquierda y nos aseguramos de que esté detrás del texto.

```
.color-overlay {  
    position: absolute;  
    width: 100%;
```


```
height: 100%;  
  
background-color: rgba(0, 0, 0, .2);  
  
z-index: -1; }
```

Aquí simplemente le damos un overlay más oscuro a la imagen de fondo, con la idea de que ayude a resaltar el texto al cual le dimos un color blanco. También nos aseguramos que se encuentre detrás del contenido de la página con z-index.

```
.content-container {  
  
    position: relative;  
  
    z-index: 1;  
  
}
```

Modifica el container de index.html. Decidimos incluir los contenidos de index.html dentro de un container, para luego poder modificarlo en CSS y asegurarnos de que esté en la posición correcta. Nótese que z-index tiene un valor de “1” en lugar de “-1” para que esté sobre el resto del contenido: recurrimos a esto debido a que teníamos problemas con el contenido que se ocultaba detrás del fondo, y la implementación de esta propiedad nos ayudó a corregir el error.

```
.titulo {  
  
    font-family: "Roboto", sans-serif;  
  
    font-weight: 700;  
  
    font-style: normal;  
  
    position: relative;  
  
    top: 150px;  
  
    font-size: 5rem;  
  
    color: white;  
  
}
```



Modifica la fuente, el tamaño, la posición y el color del título del index (“BIENVENIDO/A”).

```
.custom-text2 {  
    font-family: "Raleway", sans-serif;  
    font-optical-sizing: auto;  
    font-weight: 300;  
    font-style: normal;  
    position: relative;  
    top: 155px;  
    font-size: 1.3rem;  
    color: white;  
}
```

Modifica la fuente, el tamaño, la posición y el color del texto del index.

```
.bg-body-tertiary {  
    font-size: 1rem;  
    color: rgb(158, 155, 155);  
}
```

Modifica la fuente y el color del texto presente en el footer.

```
.tituloHome {  
    color: white;  
    font-family: "Roboto", sans-serif;  
    font-weight: 700;  
    font-style: normal;
```



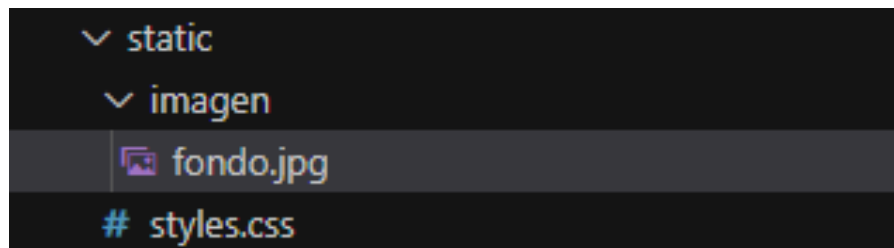
```
}
```

Modifica el color y la fuente del título de home.html (“Galería de Imágenes de la NASA”)

```
.inicioSesion {  
  
    color: white;  
  
    font-family: "Roboto", sans-serif;  
  
    font-weight: 700;  
  
    font-style: normal;  
  
}
```

Modifica el color y la fuente del texto de login.html (“Inicio de sesión”).


MODIFICACIONES DENTRO DE CARPETA “STATIC”



Dentro de la carpeta “static”, añadimos otra carpeta llamada “imagen”, la cual contiene la foto que elegimos para el fondo en formato jpg.

3. Conclusión

En conclusión, en este trabajo práctico logramos alcanzar los objetivos principales requeridos para el funcionamiento básico de la página. Además, conseguimos el correcto funcionamiento de la barra de búsqueda, el cual filtra el contenido mostrado basándose en las palabras que busquen los usuarios. Modificamos el código HTML y CSS, e implementamos un poco de JavaScript junto con elementos de la librería



Bootstrap 5, hasta lograr una interfaz gráfica estéticamente agradable a la vista que complementa la temática de la página. También logramos entender el funcionamiento de Django, lo que nos permitió completar el feature de inicio de sesión. Por último, implementamos el spinner de carga, superando los errores que se nos presentaron en un principio gracias a la investigación que realizamos para la resolución de problemas.