

# TALLER # 1

## Git, GitHub y Django

### Contenido

|   |    |
|---|----|
| INTRODUCCIÓN A GIT Y GITHUB.....                                | 2  |
| Acerca del control de versiones.....                            | 2  |
| ¿Qué es Git? .....  | 2  |
| Comandos básicos de Git .....                                   | 2  |
| ¿Qué es GitHub? .....   | 3  |
| INTRODUCCIÓN A PYTHON Y DJANGO .....                            | 4  |
| ¿Qué es Python? .....   | 4  |
| ¿Qué es Django? .....   | 5  |
| DESCRIPCIÓN DE LA APLICACIÓN WEB A DESARROLLAR.....             | 7  |
| INSTRUCCIONES PARA EL TALLER .....                              | 7  |
| CREACIÓN DE UN PROYECTO EN DJANGO .....                         | 8  |
| Creación de proyecto con comando <i>startproject</i> .....      | 8  |
| Lanzamiento del servidor con comando <i>runserver</i> .....     | 8  |
| Estructura de un proyecto Django .....                          | 10 |
| Creación de una aplicación con el comando <i>startapp</i> ..... | 11 |
| Definición de URLs .....  | 12 |
| USO DE PLANTILLAS .....   | 14 |
| Uso de plantillas sin parámetros.....                           | 14 |
| Uso de plantillas con parámetros .....                          | 16 |
| USO DE GIT .....  | 17 |
| USO DE MODELOS.....   | 27 |
| Modelos y migraciones .....                                     | 27 |
| Uso de la interfaz de administrador .....                       | 29 |
| Visualización y búsqueda de películas .....                     | 33 |
| USO DE BOOTSTRAP.....   | 36 |
| Inclusión de Bootstrap .....                                    | 36 |
| BIBLIOGRAFÍA.....   | 40 |

# INTRODUCCIÓN A GIT Y GITHUB

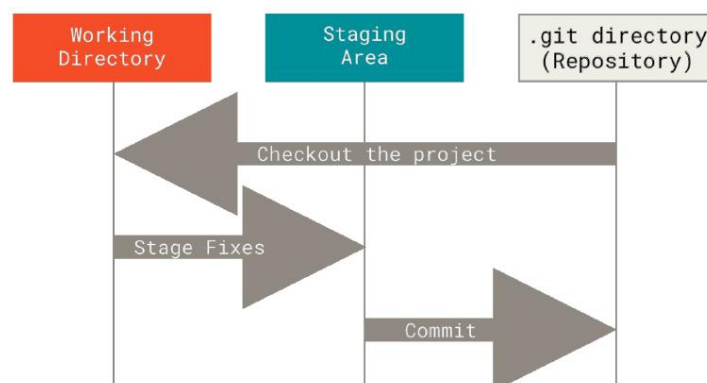
## Acerca del control de versiones

Un Sistema de Control de Versiones (Version Control System - VCS) registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo para poder recuperar versiones específicas más adelante. Un VCS permite revertir archivos seleccionados a un estado anterior, revertir todo el proyecto a un estado anterior, comparar los cambios a lo largo del tiempo y ver quién modificó por última vez algo que podría estar causando un problema. Existen VCS locales, centralizados y distribuidos.

## ¿Qué es Git?

Git es un VCS distribuido, diseñado principalmente para el manejo eficiente de proyectos de software. Permite a los usuarios llevar un registro de los cambios realizados en los archivos a lo largo del tiempo, facilitando la colaboración entre diferentes personas en un proyecto y la gestión de diferentes versiones del mismo. Con Git, cada vez que se hace un *commit* o se guarda el estado de un proyecto, Git básicamente toma una foto de cómo están todos los archivos en ese momento y almacena una referencia a esa foto. Para ser eficiente, si los archivos no han cambiado, Git no almacena el archivo de nuevo, sólo un enlace al archivo idéntico anterior que ya ha almacenado. La documentación oficial se encuentra disponible en: <https://git-scm.com/>.

Git tiene tres estados principales en los que pueden residir los archivos. **Modified** (modificado): significa que el archivo ha cambiado, pero aún no se ha confirmado para ser almacenado. **Staged** (preparado): significa que el archivo ha sido modificado en su versión actual para ir en la próxima foto de confirmación. **Committed** (confirmado): significa que los datos están almacenados de forma segura en local.

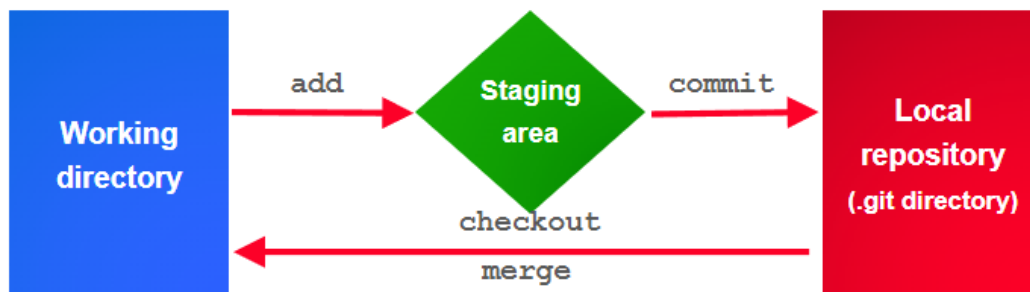


Fuente: Chacon, S., Straub, B. Pro Git. 2024.

## Comandos básicos de Git

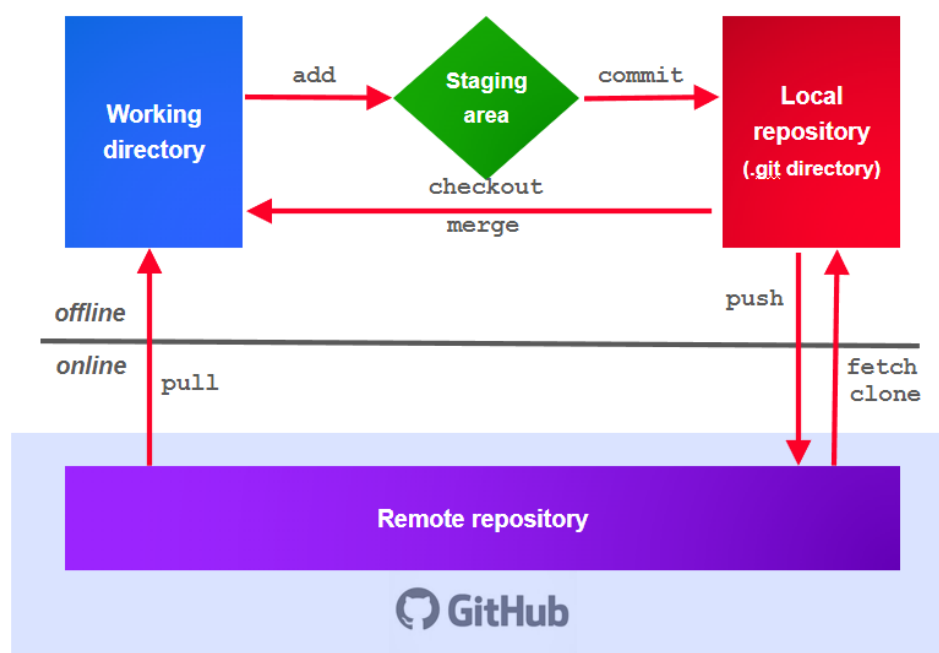
- **git init:** crea un subdirectorio llamado .git que contiene los archivos necesarios para hacer el seguimiento a los archivos.

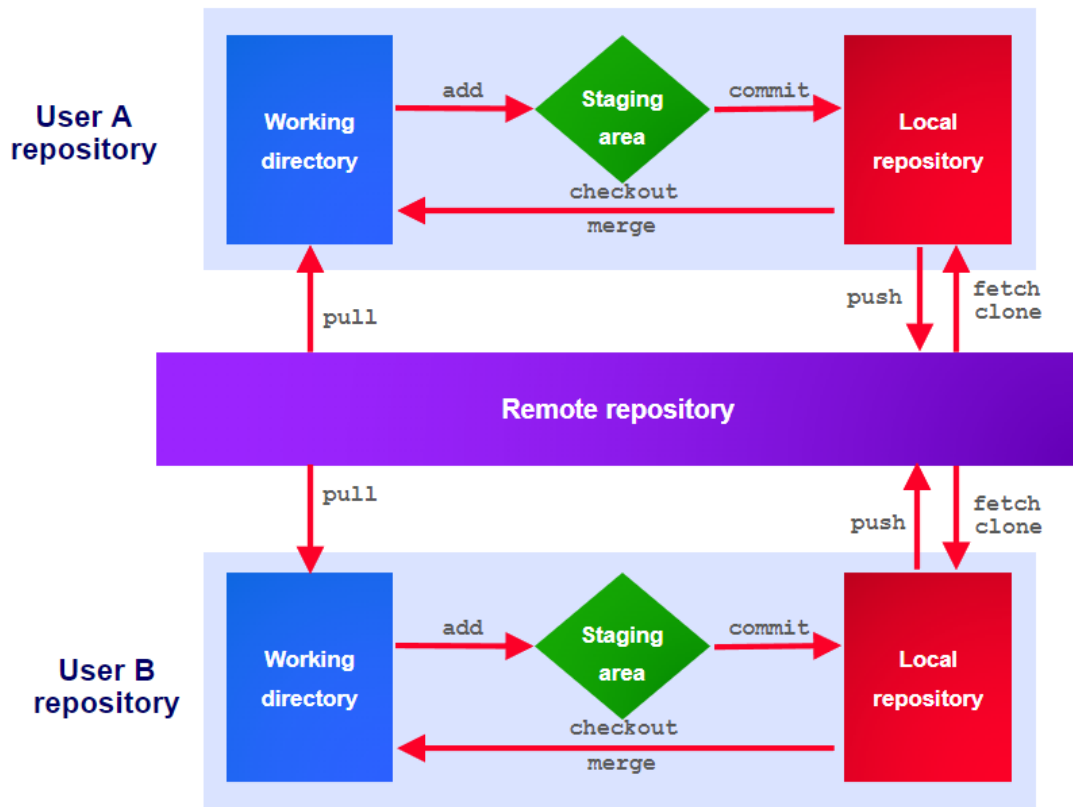
- **git clone <url>**: crea una copia de un repositorio existente, permite colaborar en el proyecto clonado.
- **git status**: muestra el estado de los archivos.
- **git push origin main**: sube los commits locales a un repositorio remoto.
- **git pull**: obtiene cambios desde un repositorio remoto y los fusiona con tu rama local.
- **git branch**: lista todas las ramas en el repositorio y muestra cuál se está utilizando actualmente.
- **git checkout <rama>**: cambia entre ramas.
- **git add <archivo>**: agrega un archivo al área de preparación (staging area) para que pueda ser incluido en el próximo commit.
- **git commit -m "<mensaje>"**: crea un commit con los cambios en el área de preparación, incluyendo un mensaje que describe los cambios realizados.
- **git merge <rama>**: fusiona cambios desde otra rama a la rama actual.



## ¿Qué es GitHub?

GitHub es una plataforma para alojar proyectos utilizando Git. Permite crear repositorios, los cuales son lugares virtuales alojados en la nube en donde los usuarios pueden almacenar cualquier tipo de archivos. Esta plataforma está disponible en: [github.com](https://github.com).



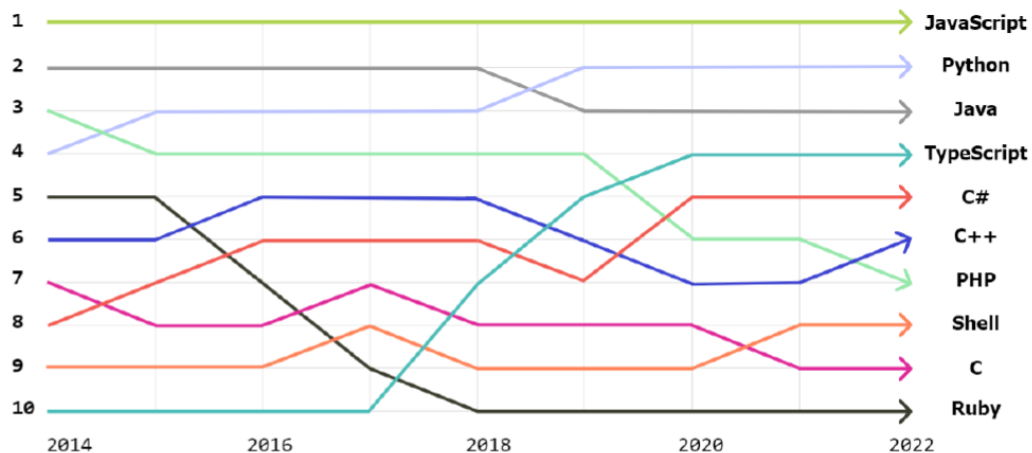


# INTRODUCCIÓN A PYTHON Y DJANGO

## ¿Qué es Python?

Python es un lenguaje de programación de alto nivel (<https://www.python.org/>). Python posee una licencia de código abierto, que, entre otras cosas, permite que los programadores puedan modificar su código, utilizarlo y/o redistribuirlo libremente sin tener que pagar al autor original. Python se caracteriza por ser un lenguaje de programación amigable y fácil de aprender.

Durante los últimos años Python ha estado entre los lenguajes más utilizados.

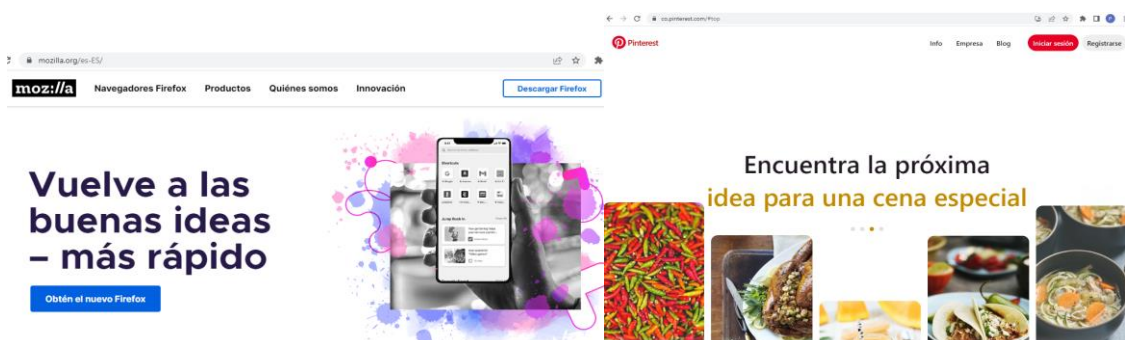
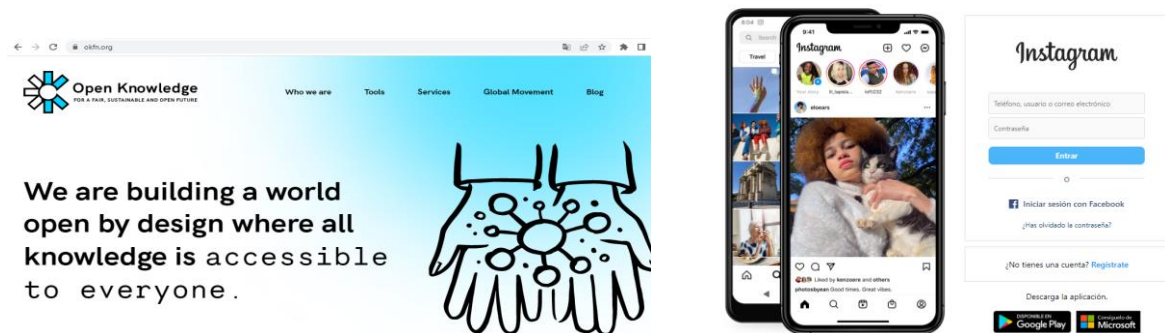
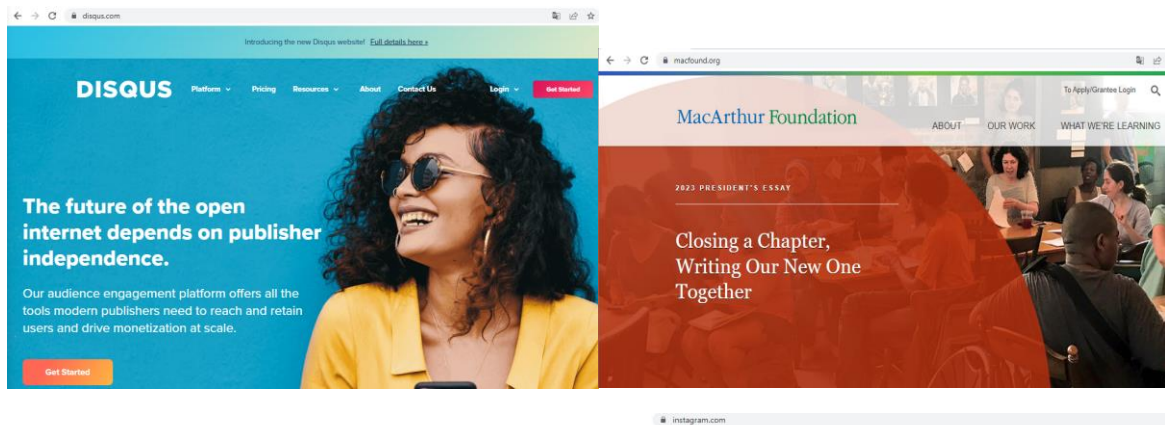


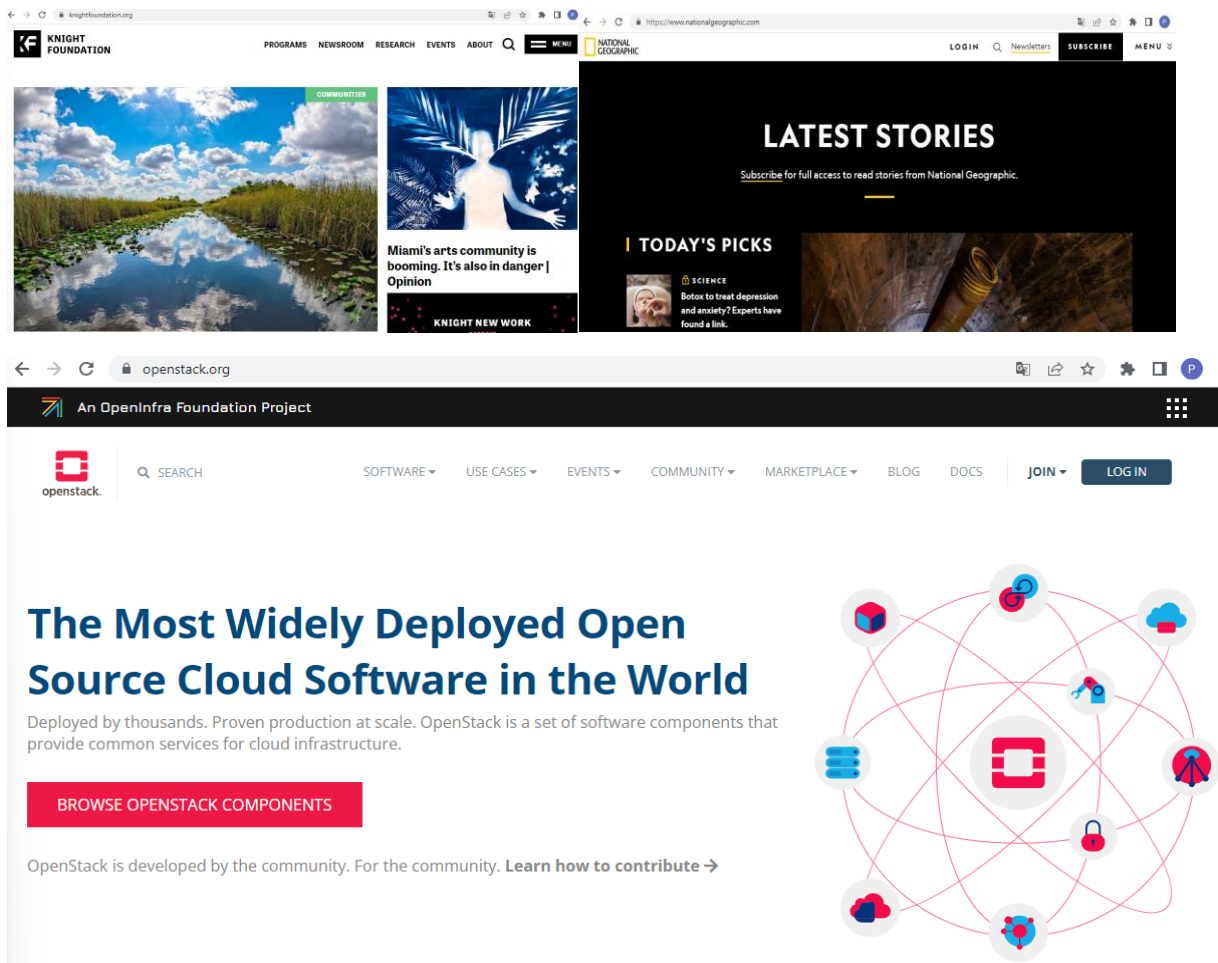
Fuente: Octoverse.

## ¿Qué es Django?

Django es un *framework* gratuito y de código abierto para desarrollar aplicaciones web en Python. Entre sus características están la rapidez para el desarrollo, la seguridad que ofrece y la escalabilidad de las aplicaciones. Django fomenta el desarrollo rápido y pragmático. Sus desarrolladores indican que Django es ridículamente rápido, tranquilizantemente seguro, extremadamente escalable e increíblemente versátil. Todos los detalles sobre el *framework* y su documentación se encuentran disponibles en línea (<https://www.djangoproject.com/>).

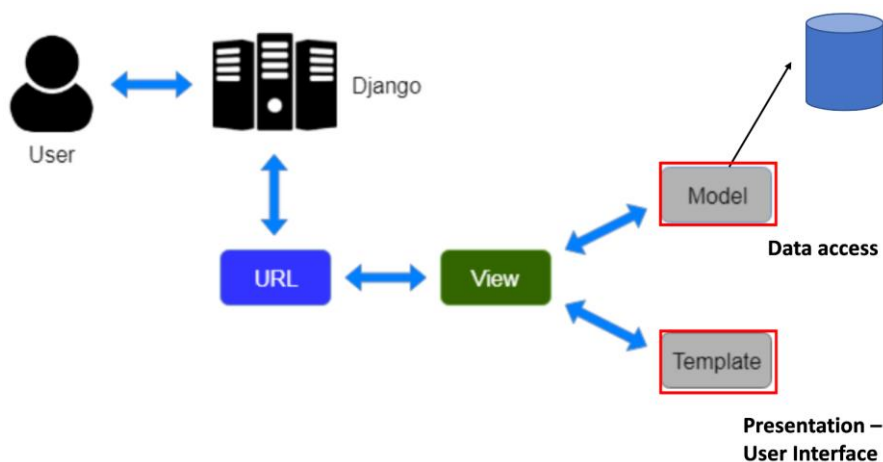
Algunos sitios web que usan Django son: Disqus (<https://disqus.com/>), MacArthur Foundation (<https://www.macfound.org/>), Open Knowledge Foundation (<https://okfn.org/>), Instagram (<https://www.instagram.com/>), Mozilla (<https://www.mozilla.org/es-ES/>), Pinterest (<https://co.pinterest.com/>), Knight Foundation (<https://knightfoundation.org/>), National Geographic (<https://www.nationalgeographic.com/>), Open Stack (<https://www.openstack.org/>).





Fuente : <https://djangostars.com/blog/10-popular-sites-made-on-django/>

Django utiliza el patrón MVT (Model-View-Template). La capa **Model** (modelos) permite estructurar y manipular los datos de la aplicación web, representa los datos y la lógica de la base de datos. La capa **View** (vistas) se usa para encapsular la lógica responsable de procesar la petición de un usuario y devolver la respuesta. La capa **Template** (plantillas) proporciona una sintaxis de fácil diseño para representar la información que se presentará al usuario. Este enfoque ayuda a separar las responsabilidades dentro de una aplicación web, lo que facilita el mantenimiento y la escalabilidad del proyecto.





# DESCRIPCIÓN DE LA APLICACIÓN WEB A DESARROLLAR

*Welcome to our Movie Reviews application, a user-friendly app designed for movie enthusiasts of all levels. Here's a simplified overview:*

*Imagine a space where you can discover and search for movies by name. Once you've found a movie you love, our application lets you log in and share your thoughts through reviews. Each movie comes with all the essential details: name, description, URL, and an eye-catching image to capture your interest.*

*Expressing your opinions is made easy with our review feature. Not only can you critique a movie, but you can also show your appreciation by using the "watch again" button in the review form. Your review includes the date it was submitted, the content, and your name. Once you're logged in, you can post, edit, and delete your own reviews. We've made sure to respect everyone's contribution by limiting the ability to modify or delete reviews from other users.*

*Adding more excitement, we've included a news section with headlines, stories, and dates. The latest news always takes the spotlight, creating a dynamic experience. All this content, including movies and news, is neatly organized in our database for efficient management.*

*Looking ahead, our focus is on making your experience even better. We're working on user authentication, where signing up is a breeze with a username, password, and confirmation. Successful authentication brings you to a personalized home page.*

*Currently, our project is running on our local machine, but we're gearing up to share it with the world.*

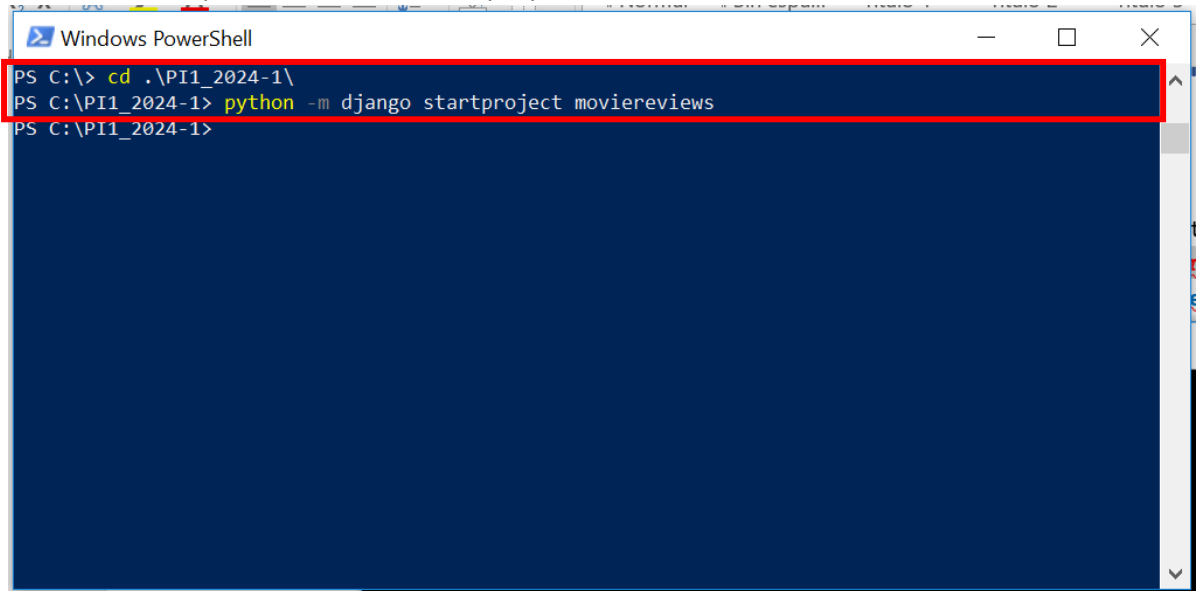
## INSTRUCCIONES PARA EL TALLER

- El taller está acompañado de una introducción temática realizada por el profesor, también habrá resolución de dudas durante la actividad.
- Este taller es de carácter individual, evaluativo y presencial.
- La duración del taller es de aproximadamente 2 horas.
- El taller consiste en seguir paso a paso este documento y entregar, antes de finalizar la clase, en el Buzón de Interactiva Virtual las evidencias (imágenes, enlaces, etc.) solicitadas.
- **En el buzón de Interactiva Virtual entregue un documento PDF que contenga lo siguiente:**
  1. El enlace de su repositorio en GitHub.
  2. Una captura de pantalla de la página principal con el nombre "a color".
  3. Una captura de pantalla de la página About.
  4. Una captura de pantalla de la rama development en GitHub, donde se vea su contenido y commits.
  5. Una captura de pantalla de la rama main en GitHub, donde se vea su contenido y commits.
  6. Una captura de pantalla de la lista de todas las películas.
  7. Una captura de pantalla de la lista de las películas que coinciden con un nombre buscado.

# CREACIÓN DE UN PROYECTO EN DJANGO

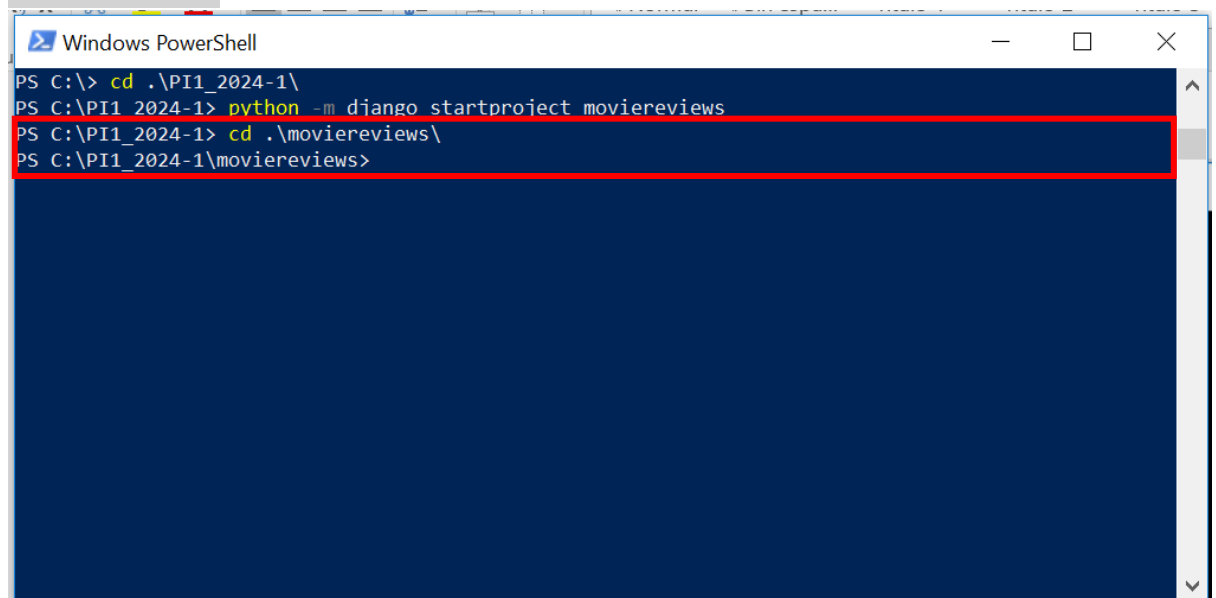
## Creación de proyecto con comando *startproject*

1. Desde la Terminal, ubíquese en la carpeta donde quiere crear el proyecto. Ejecute el siguiente comando: `python -m django startproject moviereviews`. *moviereviews* representa el nombre del proyecto.



```
Windows PowerShell
PS C:\> cd .\PI1_2024-1\
PS C:\PI1_2024-1> python -m django startproject moviereviews
PS C:\PI1_2024-1>
```

2. Acceda a la carpeta del proyecto que acaba de crear. En la Terminal, ejecute: `cd moviereviews`.



```
Windows PowerShell
PS C:\> cd .\PI1_2024-1\
PS C:\PI1_2024-1> python -m django startproject moviereviews
PS C:\PI1_2024-1> cd .\moviereviews\
PS C:\PI1_2024-1\moviereviews>
```

## Lanzamiento del servidor con comando *runserver*

1. Ejecute el servidor web local de Django por medio del comando: `python manage.py runserver`.

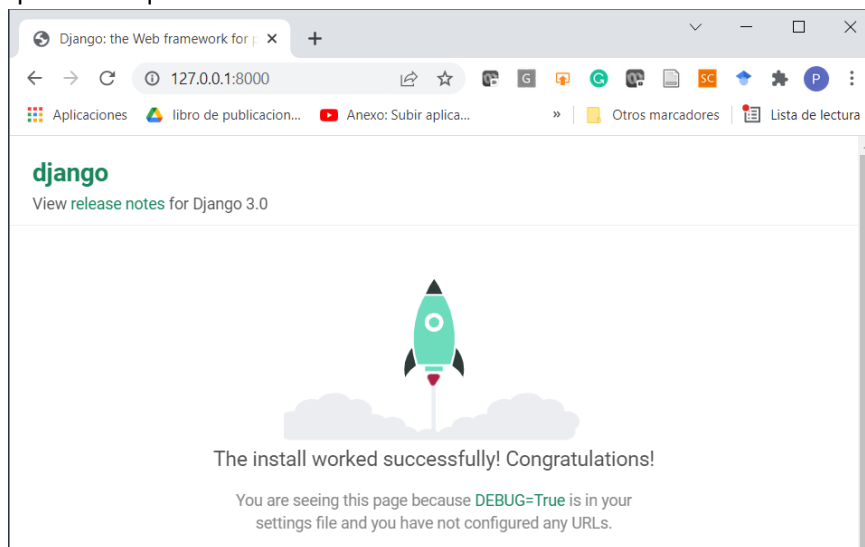


```
Windows PowerShell
PS C:\> cd .\PI1_2024-1\
PS C:\PI1_2024-1> python -m django startproject moviereviews
PS C:\PI1_2024-1> cd .\moviereviews\
PS C:\PI1_2024-1\moviereviews> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 06, 2024 - 10:45:04
Django version 5.0.1, using settings 'moviereviews.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

2. En un navegador acceda a la dirección <http://127.0.0.1:8000/>. Deberá observar un mensaje que indica que todo está funcionando correctamente.



3. Detenga el servidor desde la Terminal presionando las teclas **Ctrl** y **c** al mismo tiempo.

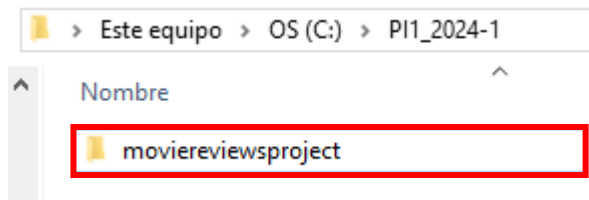
```
Windows PowerShell
PS C:\PI1_2024-1> python -m django startproject moviereviews
PS C:\PI1_2024-1> cd .\moviereviews\
PS C:\PI1_2024-1\moviereviews> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 06, 2024 - 10:45:04
Django version 5.0.1, using settings 'moviereviews.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

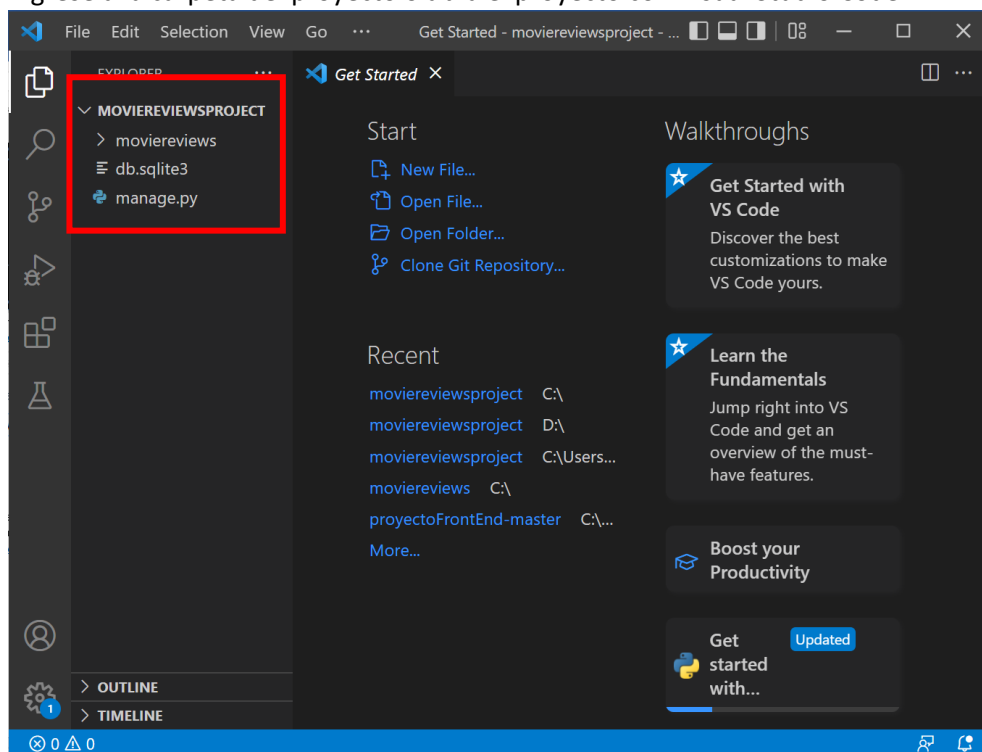
[06/Feb/2024 10:45:57] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[06/Feb/2024 10:45:58] "GET /favicon.ico HTTP/1.1" 404 2116
PS C:\PI1_2024-1\moviereviews>
```

4. Cambie el nombre de la carpeta del proyecto, agregando la palabra **project** al final, así: **moviereviewsproject**. Esto evitará confusiones entre el nombre del proyecto y el nombre de la aplicación que se desarrollará.



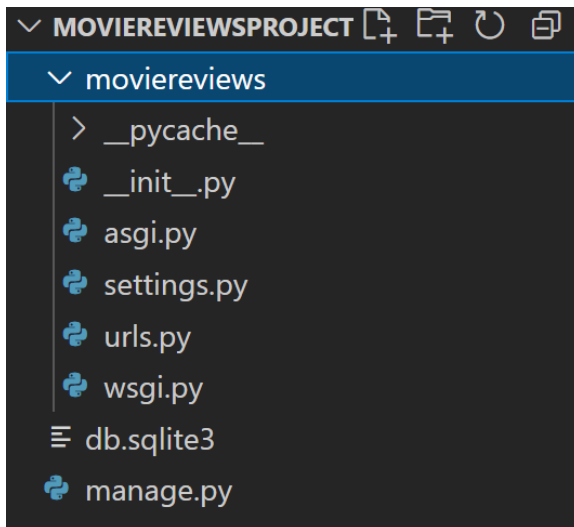
## Estructura de un proyecto Django

1. Ingrese a la carpeta del proyecto o abra el proyecto con Visual Studio Code.



2. Verifique que el proyecto tiene la siguiente estructura: una carpeta (llamada **moviereviews**, la cual contendrá la aplicación que desarrollará), el archivo **db.sqlite3** (nos ayudará a gestionar la persistencia) y el archivo **manage.py** (NO se puede modificar, es el que permite ejecutar el servidor local).

En la carpeta **moviereviews** encontrará los archivos: **asgi.py** (permite desplegar la aplicación desarrollada), **\_\_init\_\_.py** (especifica qué se ejecuta cuando Django se inicia por primera vez), **urls.py** (Indica qué páginas mostrar después de una petición) y **settings.py** (permite definir la configuración del proyecto).



## Creación de una aplicación con el comando *startapp*

1. Desde la Terminal, ubíquese en la carpeta del proyecto *moviereviewsproject* (NO en la carpeta interna) y ejecute el comando: `python manage.py startapp movie` para crear una app. *movie* representa el nombre de la aplicación.

```
Windows PowerShell
Quit the server with CTRL-BREAK.

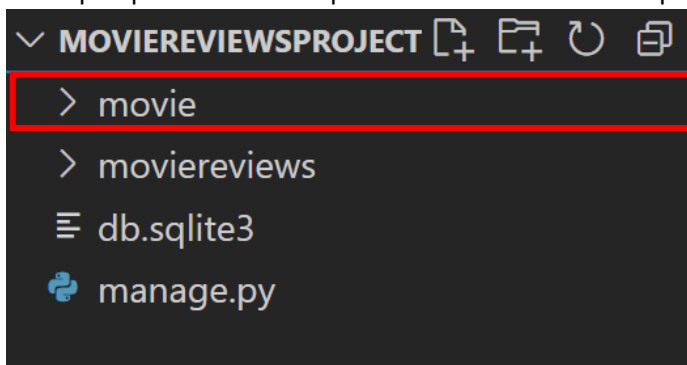
[06/Feb/2024 10:45:57] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[06/Feb/2024 10:45:58] "GET /favicon.ico HTTP/1.1" 404 2116
PS C:\PI1_2024-1\moviereviews> cd ..
PS C:\PI1_2024-1> dir

Directorio: C:\PI1_2024-1

Mode                LastWriteTime         Length Name
----                -
d-----          6/02/2024 10:45 a. m.          moviereviewsproject

PS C:\PI1_2024-1> cd .\moviereviewsproject\
PS C:\PI1_2024-1\moviereviewsproject> python manage.py startapp movie
PS C:\PI1_2024-1\moviereviewsproject>
```

2. Verifique que se creó la carpeta *movie* dentro de la carpeta del proyecto.



3. Agregue la app al archivo `settings.py` de *moviereviews*, en la sección de `INSTALLED_APPS`.

```
settings.py •
moviereviews > settings.py > ...
32
33     INSTALLED_APPS = [
34         'django.contrib.admin',
35         'django.contrib.auth',
36         'django.contrib.contenttypes',
37         'django.contrib.sessions',
38         'django.contrib.messages',
39         'django.contrib.staticfiles',
40         'movie',
41     ]
```

4. En la Terminal, ejecute: `python manage.py migrate`.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> python manage.py startapp movie
PS C:\PI1_2024-1\moviereviewsproject> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
PS C:\PI1_2024-1\moviereviewsproject>
```

5. Ejecute el servidor desde la Terminal, con el comando: `python manage.py runserver`. Verifique que todo sigue funcionando correctamente.

## Definición de URLs

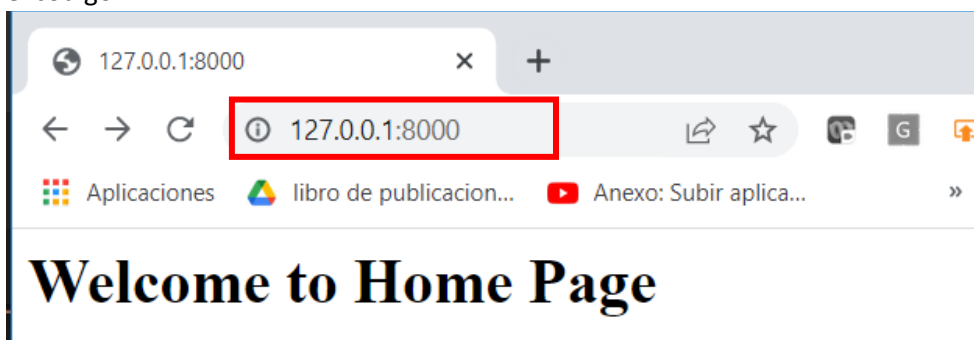
1. En el archivo `views.py` de `movie` importe la librería `HttpResponse` y agregue la función `home`. `HttpResponse` es una clase utilizada para construir respuestas HTTP en Django. La función `home` toma un objeto `request` como parámetro y devuelve la respuesta HTTP con el contenido HTML.

```
views.py
movie > views.py > home
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 # Create your views here.
5
6 def home(request):
7     return HttpResponse('<h1>Welcome to Home Page</h1>')
```

2. En el archivo **urls.py** de **moviereviews** agregue una ruta para la página principal de la app, la cual invoque la función **home** de **views.py**. Añadimos un nuevo objeto **path** con la ruta **'/'**. Es decir, coincide con la url **'localhost:8000/'** para una página de inicio. Si hay tal coincidencia, devolvemos **movieViews.home** que es una función que devuelve la vista de la página de inicio.

```
urls.py
moviereviews > urls.py > ...
13 1. Import the include() function: from django.u
14 2. Add a URL to urlpatterns: path('blog/', in
15 """
16 from django.contrib import admin
17 from django.urls import path
18 from movie import views as movieViews
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', movieViews.home),
23 ]
24
```

3. Lance el servidor y, en el navegador ingrese al enlace <http://localhost:8000> para visualizar el mensaje *Welcome to Home Page*. NOTA: Cuando se hacen cambios en un archivo y estos cambios se guardan, Django observa los cambios en el archivo y recarga el servidor con los cambios. No es necesario reiniciar manualmente el servidor cada vez que hay un cambio en el código.



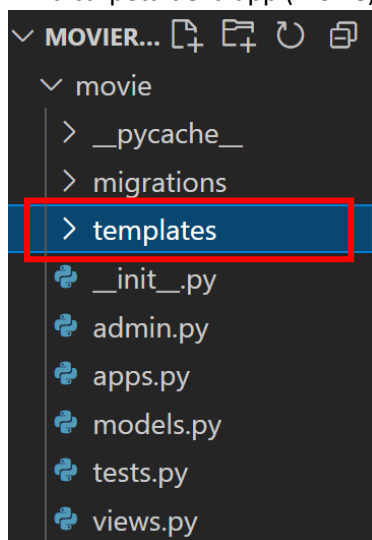
4. Ahora, cree la función about y una ruta para una página 'About', accesible cuando un usuario ingrese a localhost:8000/about.



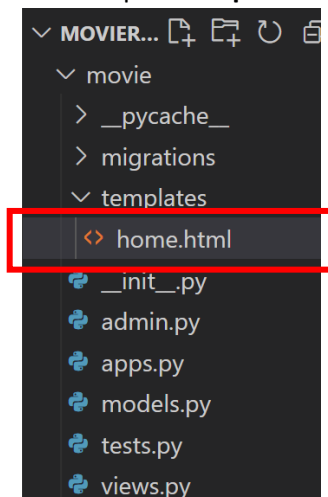
## USO DE PLANTILLAS

### Uso de plantillas sin parámetros

1. En la carpeta de la app (movie), cree una carpeta llamada **templates**.



2. En la carpeta **templates** cree un archivo llamado **home.html**.



3. En el archivo **home.html**, agregue el código HTML de la página principal.

```
<> home.html ●

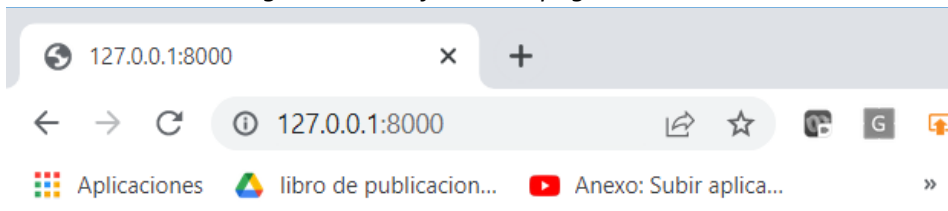
movie > templates > <> home.html
1 <h1>Welcome to Home Page</h1>
2 <h2>This is the full home page</h2>|
```

4. Modifique el archivo **views.py** para referenciar el archivo home.html. Tenga en cuenta que ahora está utilizando render en lugar de HttpResponse. Y en render, está especificando home.html.

```
views.py ●

movie > views.py > ...
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 # Create your views here.
5
6 def home(request):
7     #return HttpResponse('<h1>Welcome to Home Page</h1>')
8     return render(request, 'home.html')
```

5. En el navegador ingrese (o refresque) al enlace <http://localhost:8000> y visualizará el mensaje *Welcome to Home Page This is the full home page.*



# Welcome to Home Page

## This is the full home page

Si encuentra algún error relacionado con el reconocimiento de la ruta templates/home, abra el archivo **settings.py** y cambie la línea '**DIRS**: []', por '**DIRS**: [os.path.join(BASE\_DIR, 'movie/templates')]', y agregue **import os** al inicio del archivo.



## Uso de plantillas con parámetros

1. Modifique el archivo **views.py** de movie para pasar datos como parámetro. Pase un diccionario con un par clave-valor {'name': 'Greg Lim'} a home.html.

```
views.py X
movie > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  # Create your views here.
5
6  def home(request):
7      #return HttpResponse('<h1>Welcome to Home Page</h1>')
8      #return render(request, 'home.html')
9      return render(request, 'home.html', {'name': 'Greg Lim'})
```

2. Modifique el archivo **home.html** para recibir el parámetro. Recupere los valores del diccionario con {{ name }}. Django proporciona etiquetas de plantilla para ayudar a renderizar HTML. La lista completa de etiquetas se encuentra en la documentación oficial: <https://docs.djangoproject.com/en/3.2/ref/templates/language/>.

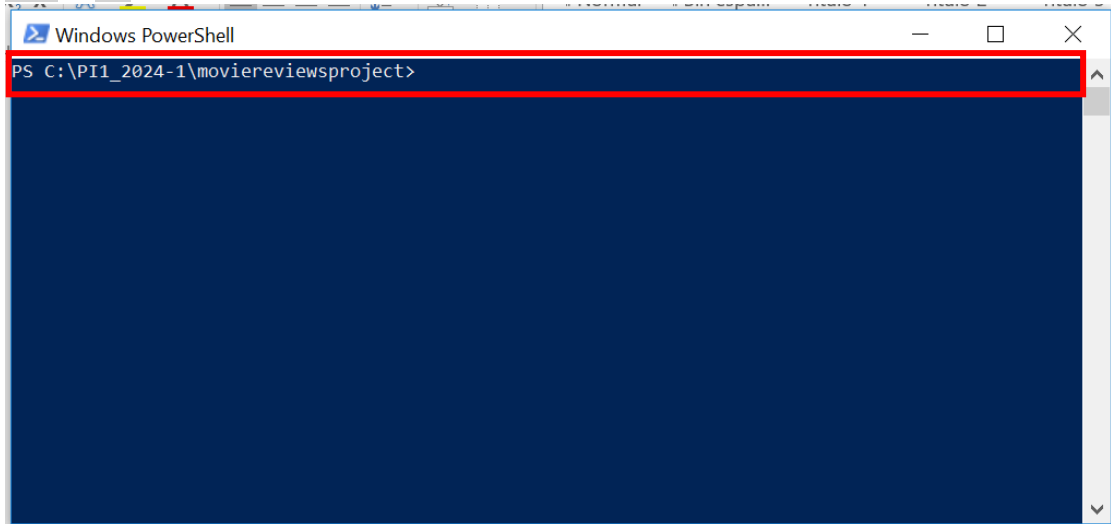
```
<> home.html X
movie > templates > <> home.html
1  <!-- <h1>Welcome to Home Page</h1> -->
2  <h1>Welcome to Home Page, {{name}}</h1>
3  <h2>This is the full home page</h2>
```

3. En el navegador ingrese (o refresque) al enlace <http://localhost:8000> y visualizará el mensaje *Welcome to Home Page, Greg Lim This is the full home page*.



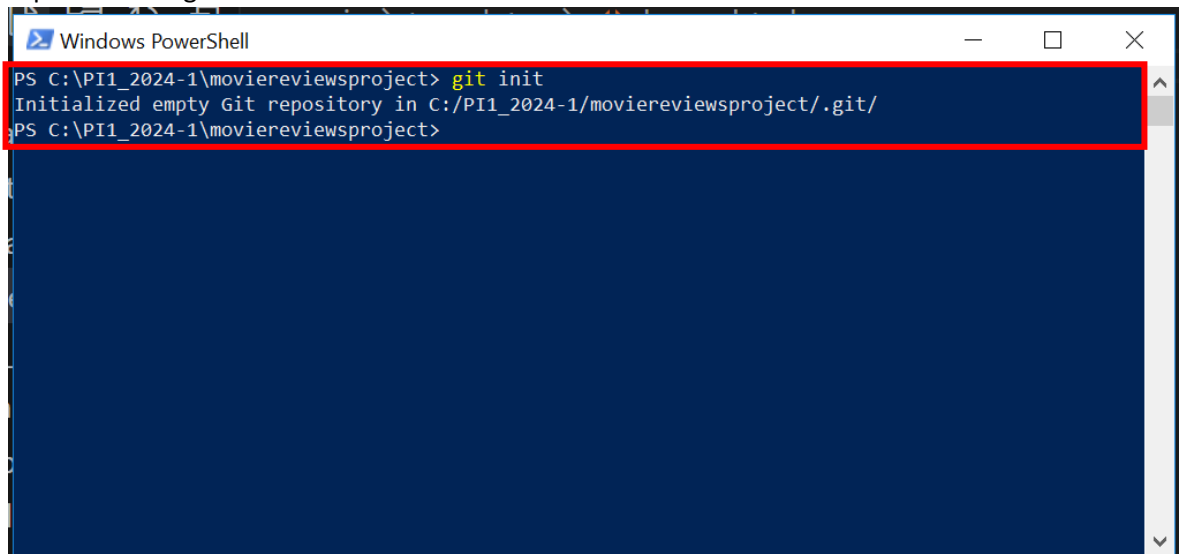
# USO DE GIT

1. Desde la Terminal, acceda a la carpeta del proyecto **moviereviewsproject** (use los comandos **dir** y **cd** para ubicarse en la carpeta).



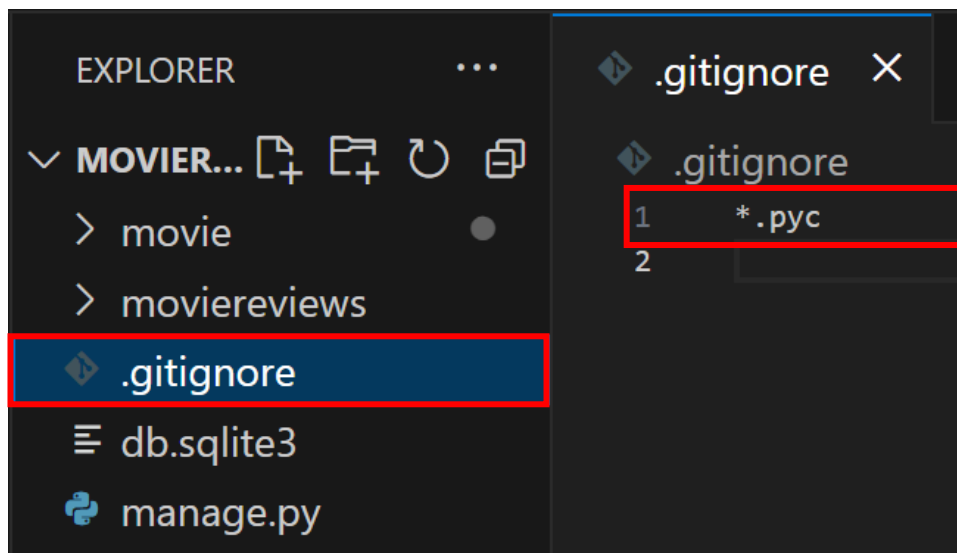
```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject>
```

2. En la Terminal, ejecute **git init** para inicializar la carpeta del proyecto como un repositorio de git.



```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git init
Initialized empty Git repository in C:/PI1_2024-1/moviereviewsproject/.git/
PS C:\PI1_2024-1\moviereviewsproject>
```

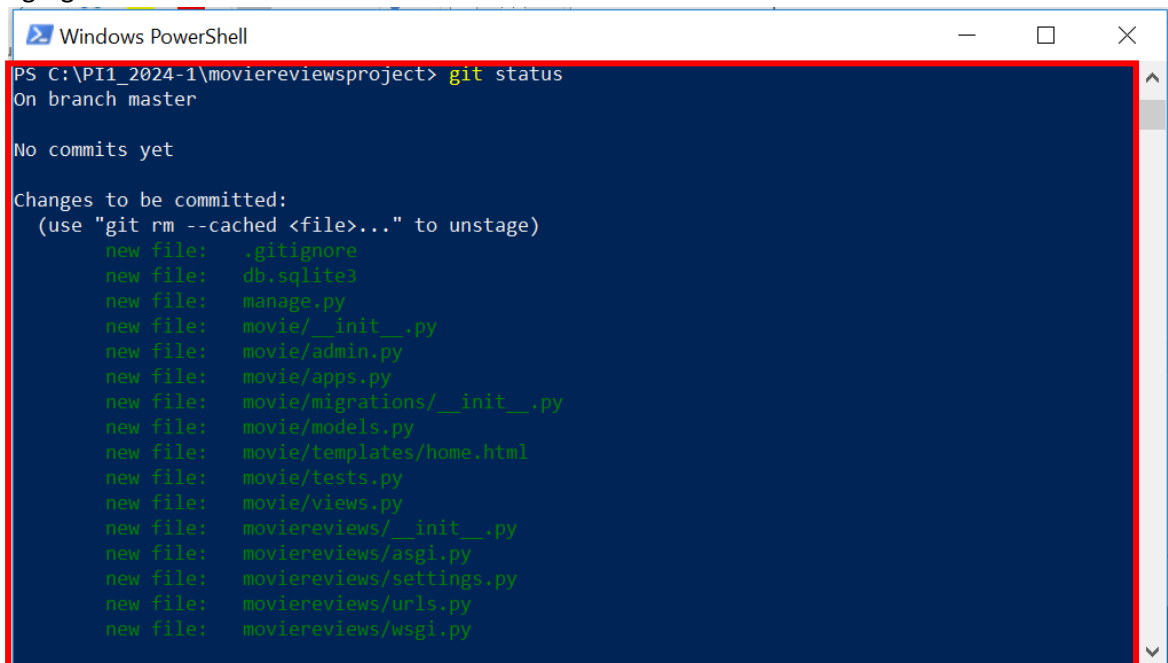
3. Cree el archivo **.gitignore** en la carpeta del proyecto y agregue la línea **\*.pyc**, esto ignorará los archivos .pyc (archivos bytecode de Python) al realizar operaciones como git add y git commit.



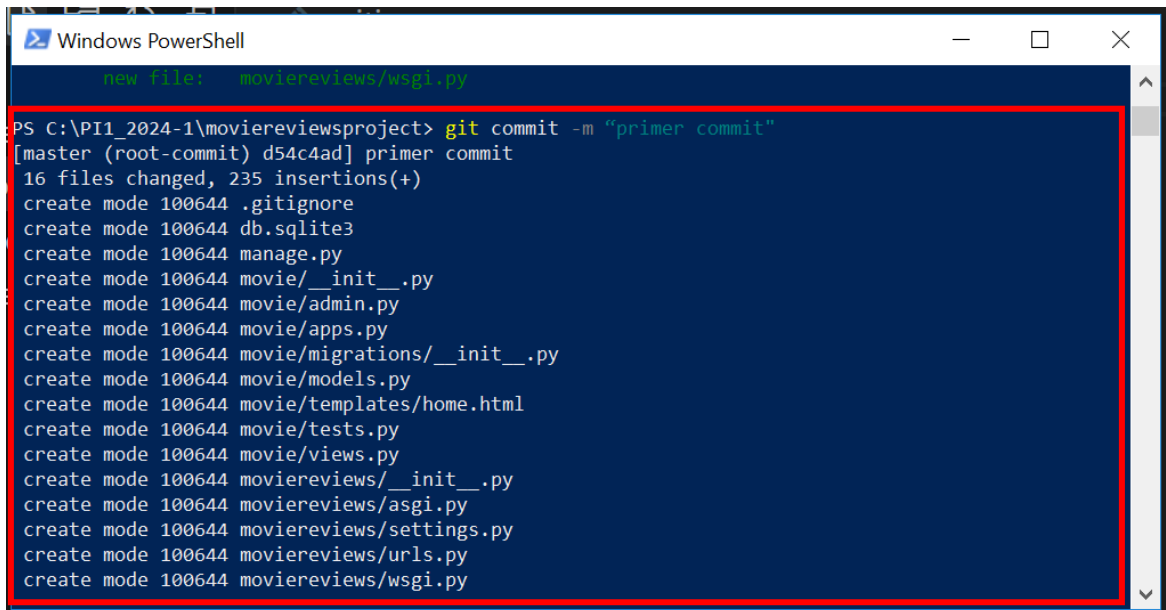
4. Ejecute **git add .** en la Terminal para hacer seguimiento a los cambios que se realizaron desde el último commit. Como es esta la primera vez que haces commit se añadirán todos los archivos.



5. Ejecute **git status** en la Terminal para verificar que los archivos fueron detectados y agregados.



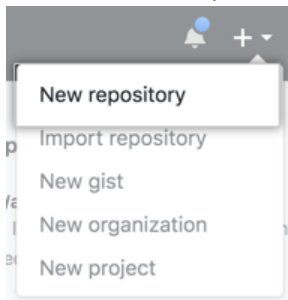
6. En la Terminal, ejecute **git commit -m "primer commit"**.



```
new file:   moviereviews/wsgi.py

PS C:\PI1_2024-1\moviereviewsproject> git commit -m "primer commit"
[master (root-commit) d54c4ad] primer commit
16 files changed, 235 insertions(+)
create mode 100644 .gitignore
create mode 100644 db.sqlite3
create mode 100644 manage.py
create mode 100644 movie/__init__.py
create mode 100644 movie/admin.py
create mode 100644 movie/apps.py
create mode 100644 movie/migrations/__init__.py
create mode 100644 movie/models.py
create mode 100644 movie/templates/home.html
create mode 100644 movie/tests.py
create mode 100644 movie/views.py
create mode 100644 moviereviews/__init__.py
create mode 100644 moviereviews/asgi.py
create mode 100644 moviereviews/settings.py
create mode 100644 moviereviews/urls.py
create mode 100644 moviereviews/wsgi.py
```

7. Cree un nuevo repositorio en GitHub. Solamente seleccione la opción **Public**.



8. Después de crear el proyecto, en la página del proyecto en GitHub busque las instrucciones para hacer **push desde un repositorio local**.

El comando **git remote add origin <URL>** se utiliza para agregar un nuevo repositorio remoto al repositorio local. **git remote** es el comando para administrar los repositorios remotos en Git. **add origin** indica que está agregando un nuevo repositorio remoto y le está dando el alias *origin*, el cual es un nombre comúnmente utilizado para referirse al repositorio remoto principal. **<URL>** es la URL del repositorio remoto que está agregando.

**git branch** se utiliza para administrar ramas en Git. **-M** es una opción que indica que se debe renombrar una rama existente. **main** es el nuevo nombre que se le dará a la rama.

**git push** es el comando usado para enviar los cambios locales a un repositorio remoto. **-u** establece la conexión entre la rama local y la rama remota main. Después de usar esta opción una vez, en futuros git push y git pull, Git entenderá automáticamente que desea trabajar en la rama main del repositorio remoto origin. **origin** es el nombre dado por defecto al repositorio remoto cuando clonó el repositorio. **main** es el nombre de la rama en la que está trabajando localmente y que desea enviar al repositorio remoto origin.

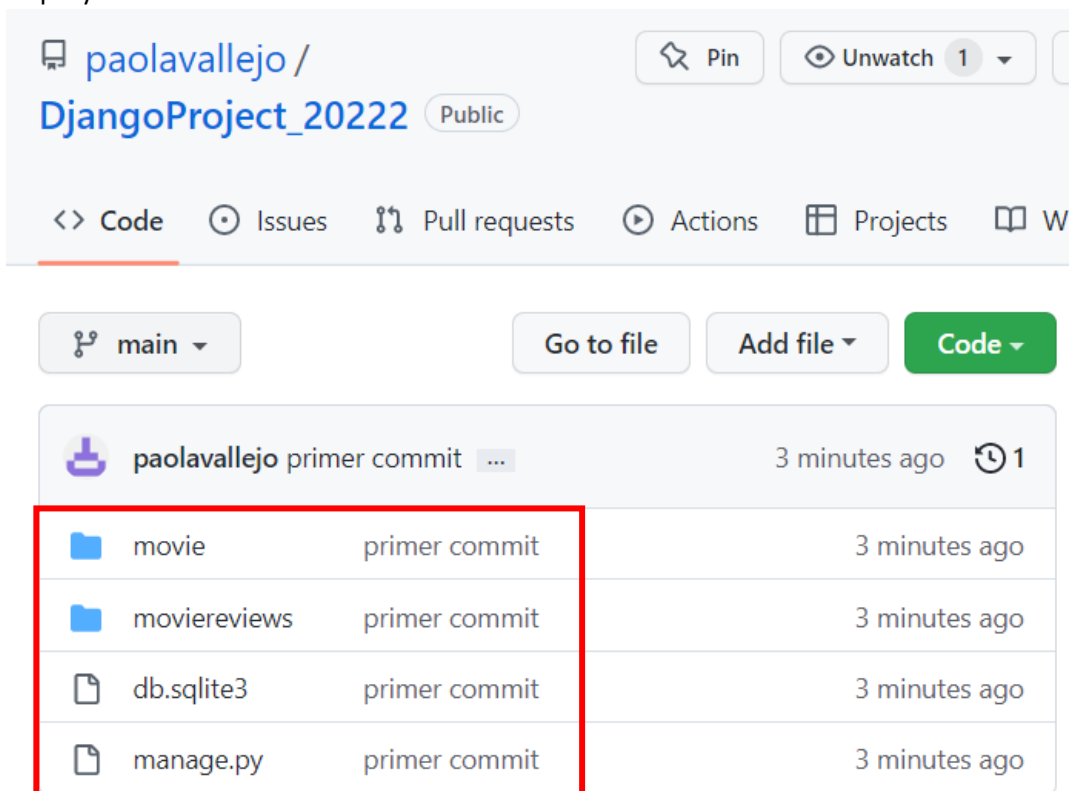
### ...or push an existing repository from the command line

```
git remote add origin https://github.com/paolavallejo/DjangoProject.git
git branch -M main
git push -u origin main
```

9. Copie las instrucciones y ejecútelas en la Terminal. Es posible que deba ingresar su usuario y contraseña de GitHub.

```
Selecionar Windows PowerShell
create mode 100644 moviereviews/asgi.py
create mode 100644 moviereviews/settings.py
create mode 100644 moviereviews/urls.py
create mode 100644 moviereviews/wsgi.py
PS C:\PI1_2024-1\moviereviewsproject> git remote add origin https://github.com/paolavallejo/Taller-PI1-20241.git
PS C:\PI1_2024-1\moviereviewsproject> git branch -M main
PS C:\PI1_2024-1\moviereviewsproject> git push -u origin main
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (20/20), 8.40 KiB | 860.00 KiB/s, done.
Total 20 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/paolavallejo/Taller-PI1-20241.git
* [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

10. En la página del proyecto en GitHub, acceda a la pestaña **Code**, allí encontrará el código de su proyecto.



paolavallejo / DjangoProject\_20222 Public

<> Code Issues Pull requests Actions Projects W

main Go to file Add file Code

paolavallejo primer commit 3 minutes ago 1

|              |               |               |
|--------------|---------------|---------------|
| movie        | primer commit | 3 minutes ago |
| moviereviews | primer commit | 3 minutes ago |
| db.sqlite3   | primer commit | 3 minutes ago |
| manage.py    | primer commit | 3 minutes ago |

11. En el archivo **views.py** de **movie**, en la función **home**, cambie el nombre Greg Lim por su nombre.

```
views.py M X
movie > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  # Create your views here.
5
6  def home(request):
7      #return HttpResponse('<h1>Welcome to Home Page</h1>')
8      #return render(request, 'home.html')
9      return render(request, 'home.html', {'name': 'Paola Vallejo'})
```

12. Ejecute **git status** en la Terminal para verificar que los archivos fueron detectados.

```
Selecionar Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   movie/views.py

no changes added to commit (use "git add" and/or "git commit -a")
```

13. Ejecute **git add movie/views.py** en la Terminal para hacer seguimiento a los cambios (del archivo views.py) que se realizaron desde el último commit.

```
Selecionar Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git add movie/views.py
```

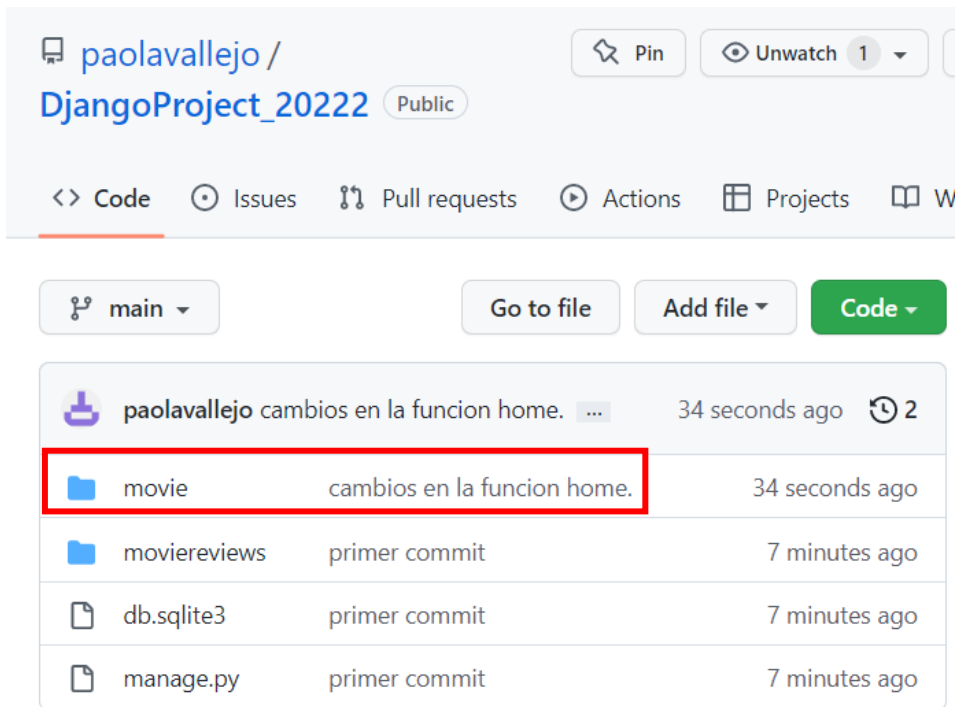
14. En la Terminal, ejecute **git commit -m "cambios en la funcion home"**.

```
Selecionar Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git commit -m "cambios en la funcion home"
[main 7bd1e29] cambios en la funcion home
1 file changed, 1 insertion(+), 1 deletion(-)
```

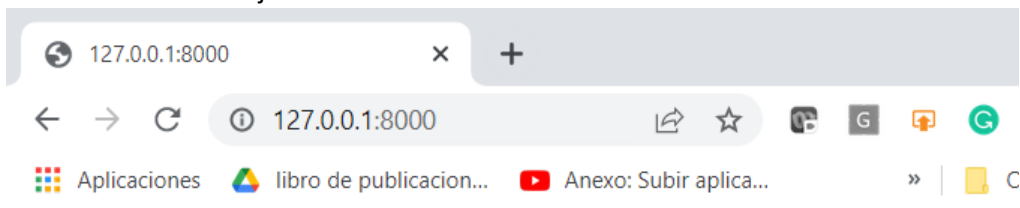
15. Ejecute **git push** para agregar el código actualizado al repositorio remoto.

```
Selecionar Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 369 bytes | 184.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/paolavallejo/Taller-PI1-20241.git
   d54c4ad..7bd1e29  main -> main
```

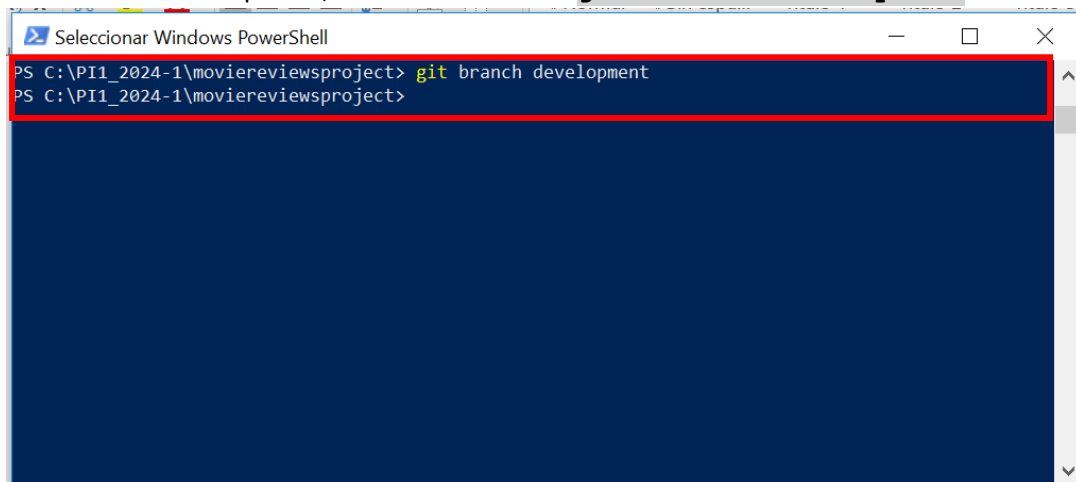
16. En la página del proyecto en GitHub, acceda a la pestaña **Code**, allí encontrará el código de su proyecto.



17. En el navegador ingrese (refresque) al enlace <http://localhost:8000>. Recuerde que el servidor debe estar ejecutándose.

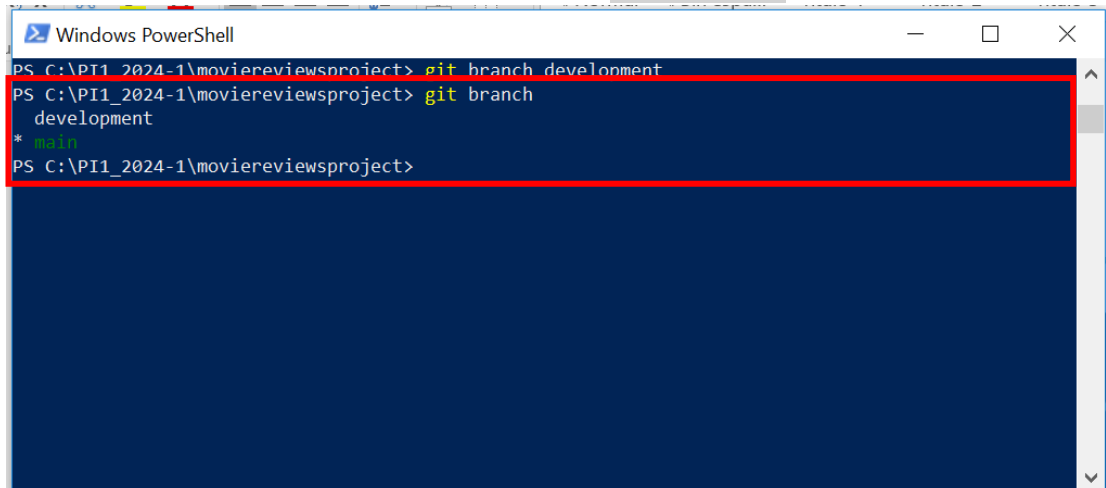


18. Cree la rama development, usando el comando `git branch development`.





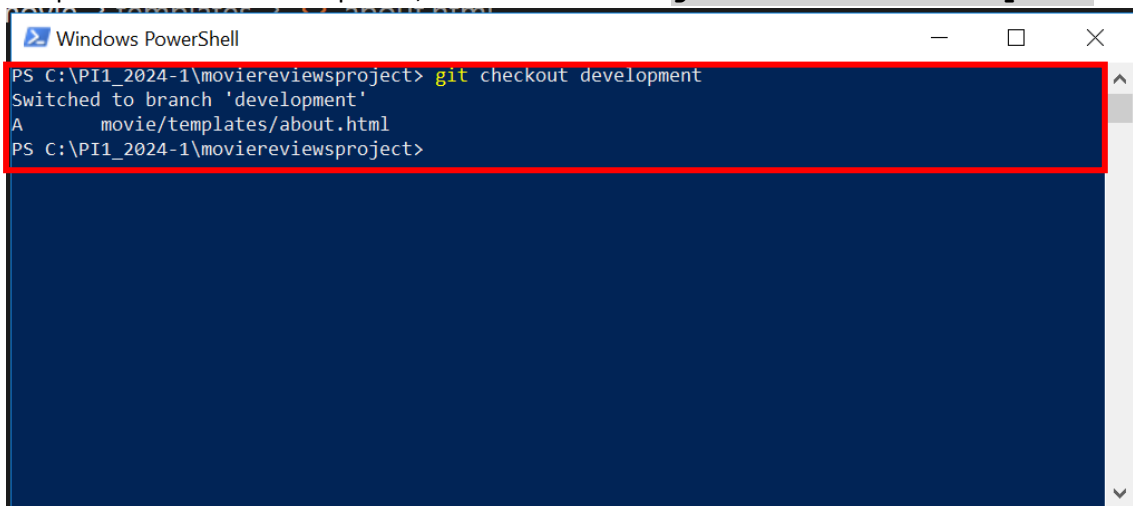
19. Verifique que la rama se creó, usando el comando `git branch`.



```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git branch development
PS C:\PI1_2024-1\moviereviewsproject> git branch
  development
* main
PS C:\PI1_2024-1\moviereviewsproject>
```

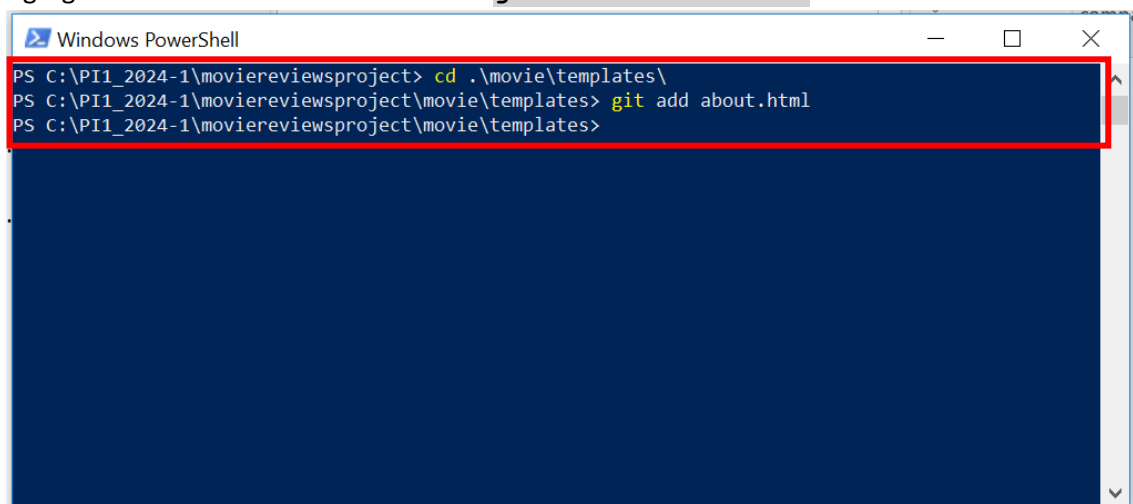
20. Cree el archivo **about.html** en la carpeta templates y ponga el contenido HTML que estaba inicialmente en la función **about** de **views.py**.

21. Ubíquese en la rama development, usando el comando `git checkout development`.



```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git checkout development
Switched to branch 'development'
A       movie/templates/about.html
PS C:\PI1_2024-1\moviereviewsproject>
```

22. Agregue el archivo html con el comando `git add about.html`.



```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> cd .\movie\templates\
PS C:\PI1_2024-1\moviereviewsproject\movie\templates> git add about.html
PS C:\PI1_2024-1\moviereviewsproject\movie\templates>
```

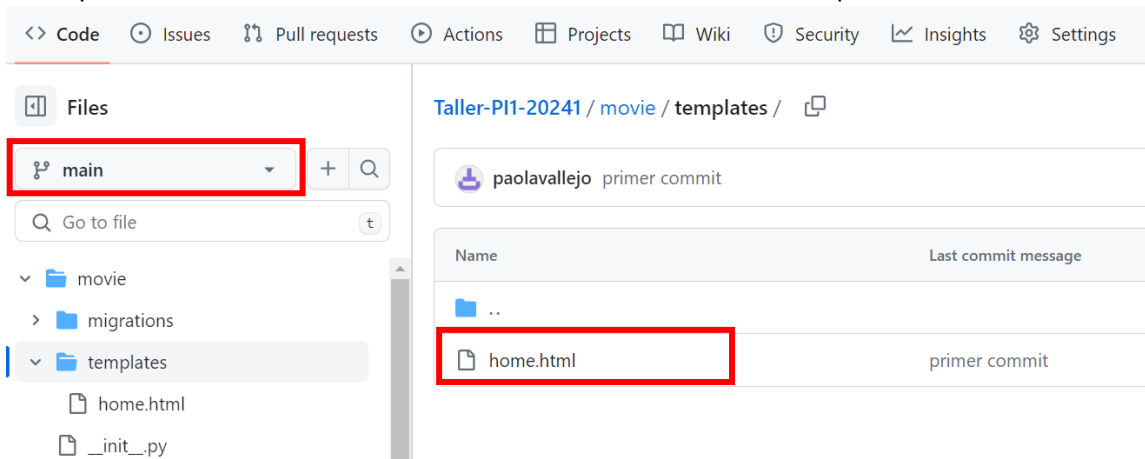
23. Haga un commit con `git commit -m "nuevo archivo about.html"`.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject\movie\templates> git commit -m "nuevo archivo about.html"
[development 1cca65a] nuevo archivo about.html
1 file changed, 1 insertion(+)
create mode 100644 movie/templates/about.html
PS C:\PI1_2024-1\moviereviewsproject\movie\templates>
```

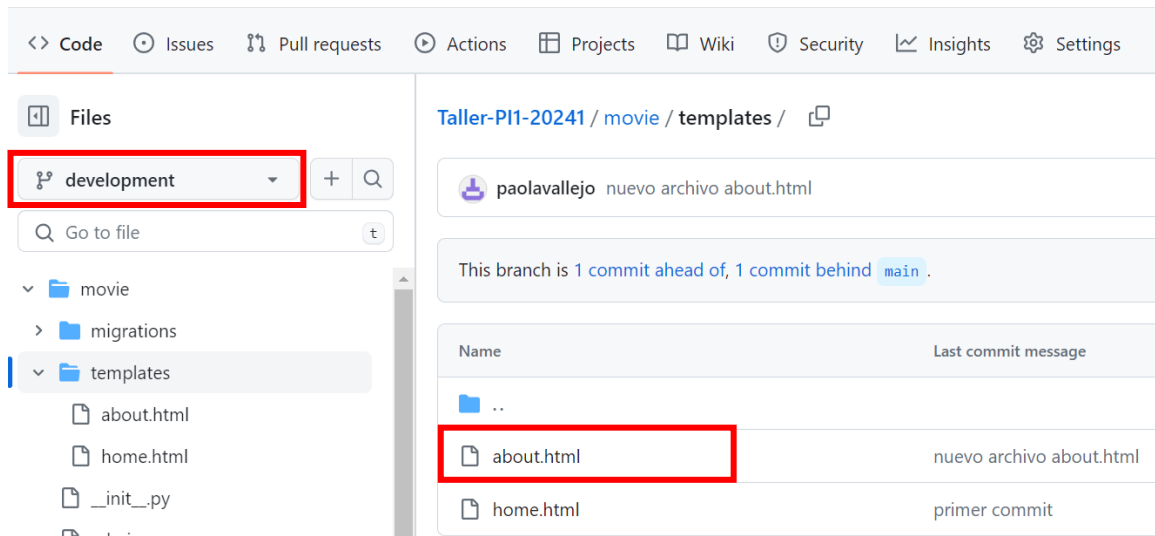
24. Haga push de la rama development `git push -u origin development`.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject\movie\templates> git push -u origin development
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 438 bytes | 438.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'development' on GitHub by visiting:
remote:   https://github.com/paolavallejo/Taller-PI1-20241/pull/new/development
remote:
To https://github.com/paolavallejo/Taller-PI1-20241.git
 * [new branch]      development -> development
branch 'development' set up to track 'origin/development'.
PS C:\PI1_2024-1\moviereviewsproject\movie\templates>
```

25. Verifique el contenido de la rama **main** en GitHub. Solo debe tener la plantilla de home.



26. Verifique el contenido de la rama **development** en GitHub. Debe tener la plantilla de about.



27. Ubíquese en la rama main, haciendo uso del comando *checkout*.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject\movie\templates> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\PI1_2024-1\moviereviewsproject\movie\templates>
```

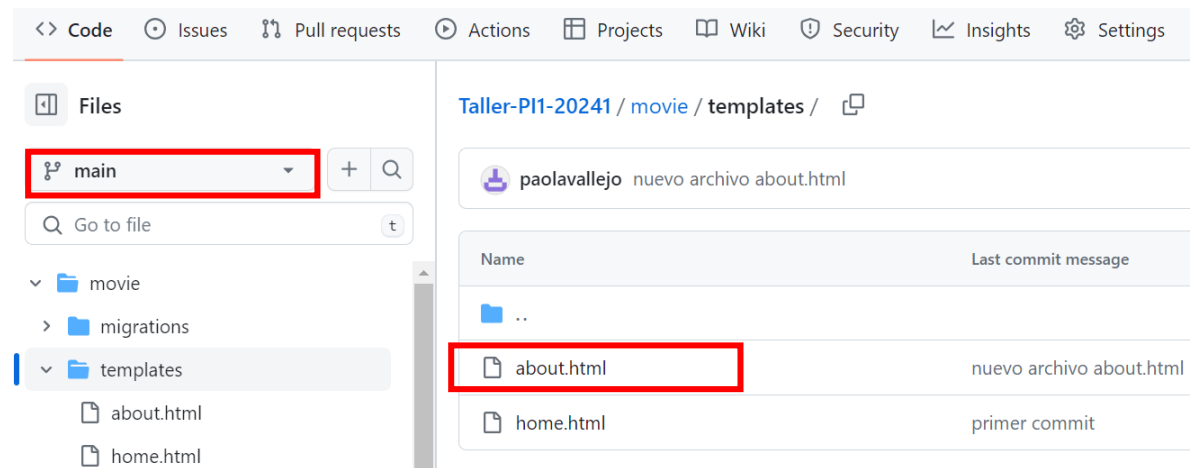
28. Fusione el contenido de la rama main con el de la rama development. Use el comando `git merge development`.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject\movie\templates> git merge development
Merge made by the 'ort' strategy.
 movie/templates/about.html | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 movie/templates/about.html
PS C:\PI1_2024-1\moviereviewsproject\movie\templates>
```

29. Es una buena práctica hacer push después de un merge para sincronizar tus cambios locales con el repositorio remoto. Use el comando `git push origin main`.

```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 350 bytes | 350.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/paolavallejo/Taller-PI1-20241.git
f30ef0b..ed74957 main -> main
PS C:\PI1_2024-1\moviereviewsproject>
```

30. Verifique el contenido de la rama main. Debe encontrar allí el archivo html que originalmente se creó en la rama development.

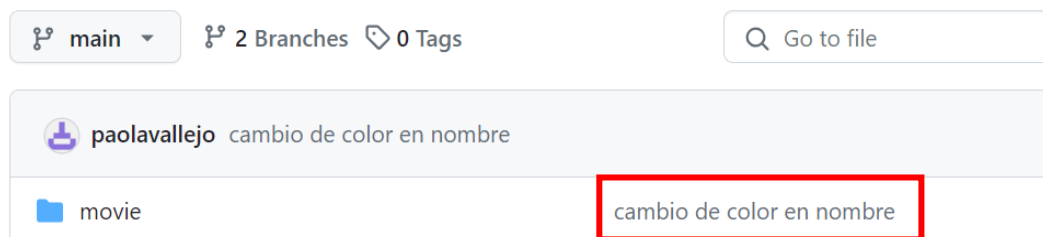


31. Ahora, modifique la función about para que use la plantilla about.html. Suba todos los cambios a la rama main y verifique que sí están en GitHub. Además, ejecute el servidor para validar que todo funciona correctamente.

32. Cambie el color del nombre que se muestra en la página principal. Únicamente se debe cambiar el color del nombre, NO de toda la línea.



33. Haga un commit al repositorio remoto con las modificaciones.



## USO DE MODELOS

Las bases de datos en Django se definen a través de modelos. Cada modelo define la estructura de una tabla de la base de datos, y Django convierte ese modelo en una tabla de base de datos.

### Modelos y migraciones

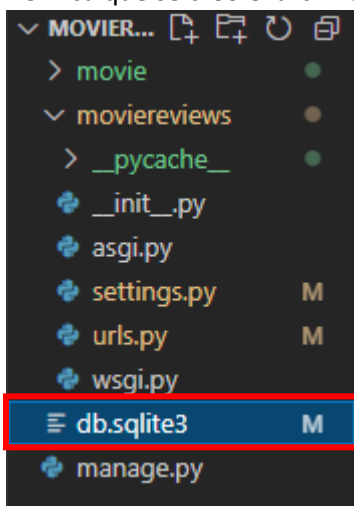
1. Agregue el modelo de Movie en el archivo **models.py**. Django importa el módulo **models** para ayudarnos a construir modelos de base de datos que definan las características de los datos en la base de datos. La clase **Movie** hereda de la clase **Model**, la cual permite interactuar con la base de datos, crear una tabla, recuperar información y realizar cambios en los datos de la base de datos.  
Este modelo contiene los atributos: *title*, *description*, *image* y *url*. Estos atributos tienen tipos como: *CharField* (representa strings), *ImageField* (representa imágenes) y *URLField* (representa direcciones URL). Para el atributo *image* se debe especificar la opción *upload\_to* para indicar un subdirectorío de **MEDIA\_ROOT** (que se encuentra en **settings.py**) en el que se almacenarán las imágenes subidas. *url* es de *URLField*, un *CharField* para una url. Para el atributo **url**, especificamos *blank=True* para indicar que este campo es opcional.  
Django proporciona muchos otros tipos como fechas, enteros y correos electrónicos. Para tener una documentación completa de los tipos y cómo usarlos, puede consultar la documentación: <https://docs.djangoproject.com/en/3.2/ref/models/fields/>.

```
models.py
movie > models.py > Movie
1 from django.db import models
2
3 # Create your models here.
4
5 class Movie(models.Model):
6     title = models.CharField(max_length=100)
7     description = models.CharField(max_length=250)
8     image = models.ImageField(upload_to='movie/images/')
9     url = models.URLField(blank=True)
```

2. Instale Pillow para el manejo de imágenes. En la Terminal ejecute el comando `pip install pillow`.

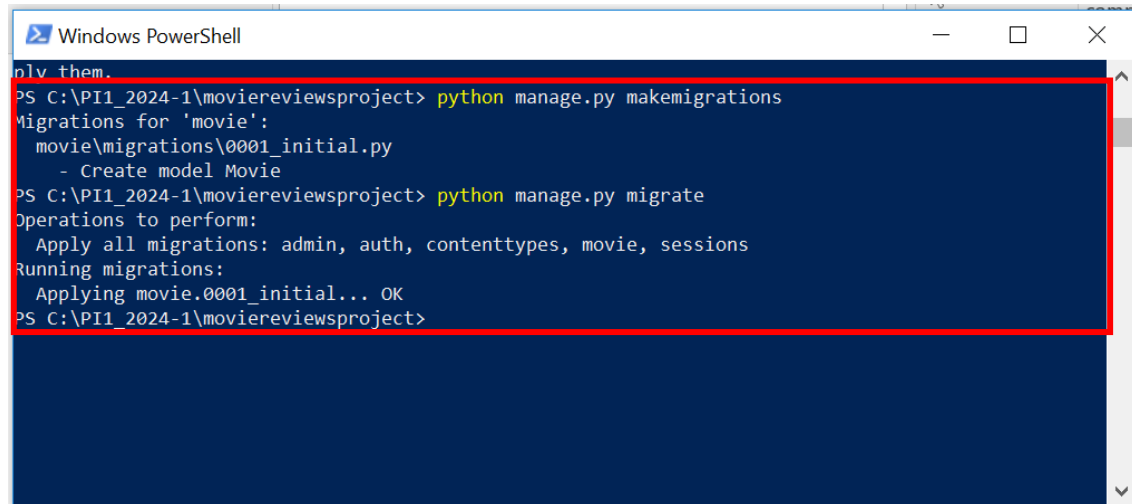
```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> pip install pillow
Collecting pillow
  Obtaining dependency information for pillow from https://files.pythonhosted.org/packages/ce/a7/11a539c1e12dfb9d67c35e5d3d99c7a6853face9083e6483360f4d9cd1d8/pillow-10.2.0-cp312-cp312-win32.whl.metadata
  Downloading pillow-10.2.0-cp312-cp312-win32.whl.metadata (9.9 kB)
  Downloading pillow-10.2.0-cp312-cp312-win32.whl (2.3 MB)
----- 2.3/2.3 MB 1.8 MB/s eta 0:00:00
Installing collected packages: pillow
Successfully installed pillow-10.2.0
```

3. En la Terminal, ejecute: `python manage.py migrate`. Este comando crea una base de datos SQLite inicial en la carpeta del proyecto. La primera vez que se ejecuta el comando *migrate*, se crea una base de datos inicial basada en la configuración por defecto de Django. Verifica que se creó el archivo **db.sqlite3** en la carpeta del proyecto.



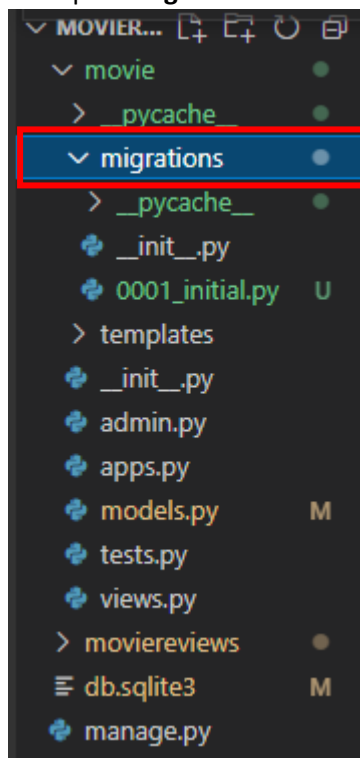
4. Cada vez que realice cambios en los modelos, ejecute los siguientes comandos en la Terminal: `python manage.py makemigrations` y `python manage.py migrate`. El comando *makemigrations* genera los comandos SQL para las aplicaciones preinstaladas en nuestra configuración *INSTALLED\_APPS*. Los comandos SQL aún no se ejecutan, sino que son solo un registro de todos los cambios realizados en nuestros modelos. Cada que se ejecuta el comando

*migrate*, la base de datos se sincroniza con el estado actual de los modelos.



```
Windows PowerShell
PS C:\PI1_2024-1\moviereviewsproject> python manage.py makemigrations
Migrations for 'movie':
  movie\migrations\0001_initial.py
    - Create model Movie
PS C:\PI1_2024-1\moviereviewsproject> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, movie, sessions
Running migrations:
  Applying movie.0001_initial... OK
PS C:\PI1_2024-1\moviereviewsproject>
```

5. Las migraciones se almacenan en la carpeta **migrations**. Verifique que su aplicación contiene la carpeta **migrations**.

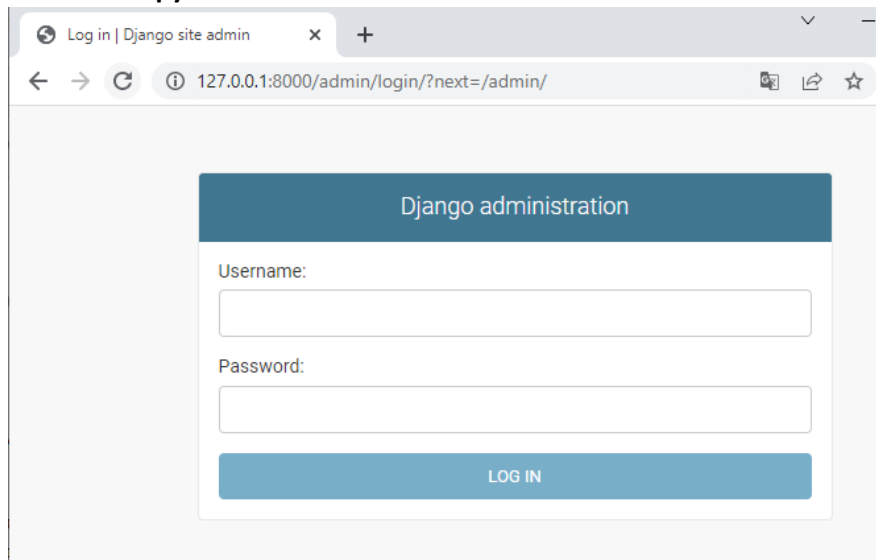


## Uso de la interfaz de administrador

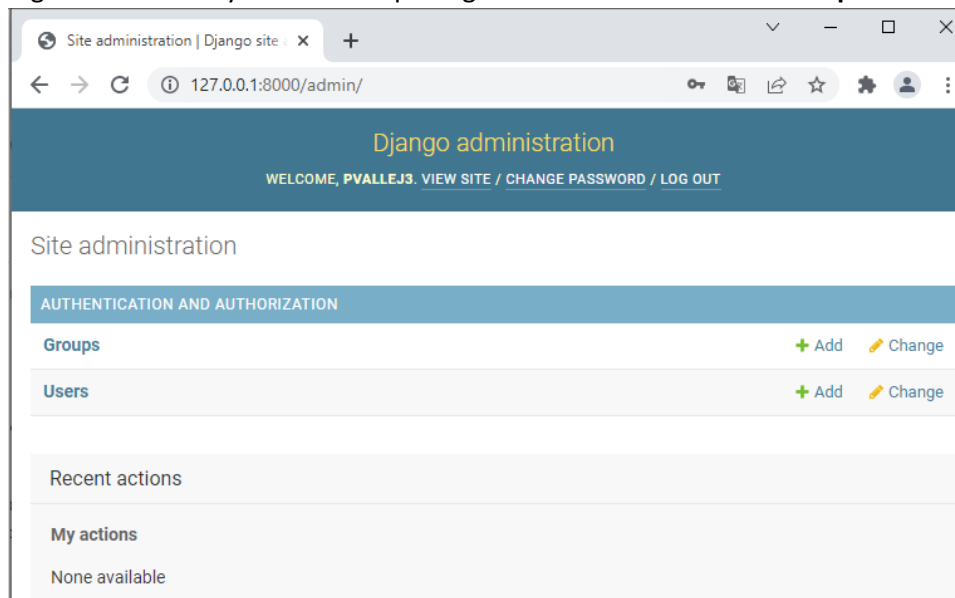
1. En la Terminal, ejecute: `python manage.py createsuperuser`. Ingrese el nombre de usuario, el correo y la contraseña con el cual se quiere registrar en la interfaz de administrador. Esta interfaz le permitirá manipular los datos de la base de datos. Si olvida el usuario y la contraseña ingresados no podrá acceder al administrador (consérvelos bien). Si en algún momento desea cambiar la contraseña puede hacerlo con el comando `python manage.py changepassword <username>`.
2. Ejecute el servidor desde la Terminal, con el comando: `python manage.py runserver`.



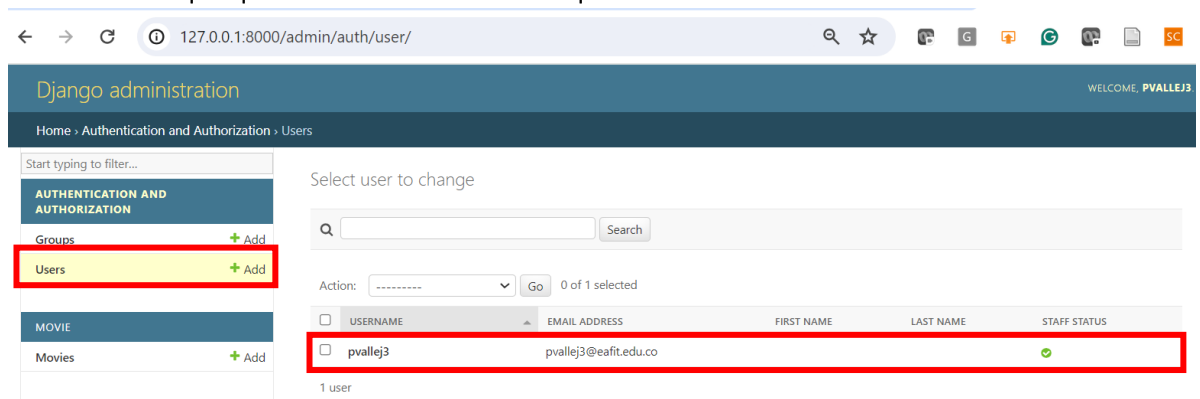
3. En el navegador ingrese al enlace <http://localhost:8000/admin>. Esta ruta fue definida en el archivo `urls.py`.



4. Ingrese el usuario y contraseña que registró con el comando `createsuperuser`.



5. En **Users** verifique que se encuentra el usuario que acaba de crear.



6. Agregue el modelo `Movie` a admin. Para esto, modifique el archivo `admin.py` de `movie`. Esta acción garantiza que el modelo se incluya en la interfaz de administración.

```
admin.py
movie > admin.py
1 from django.contrib import admin
2 from .models import Movie
3
4 # Register your models here.
5 admin.site.register(Movie)
```

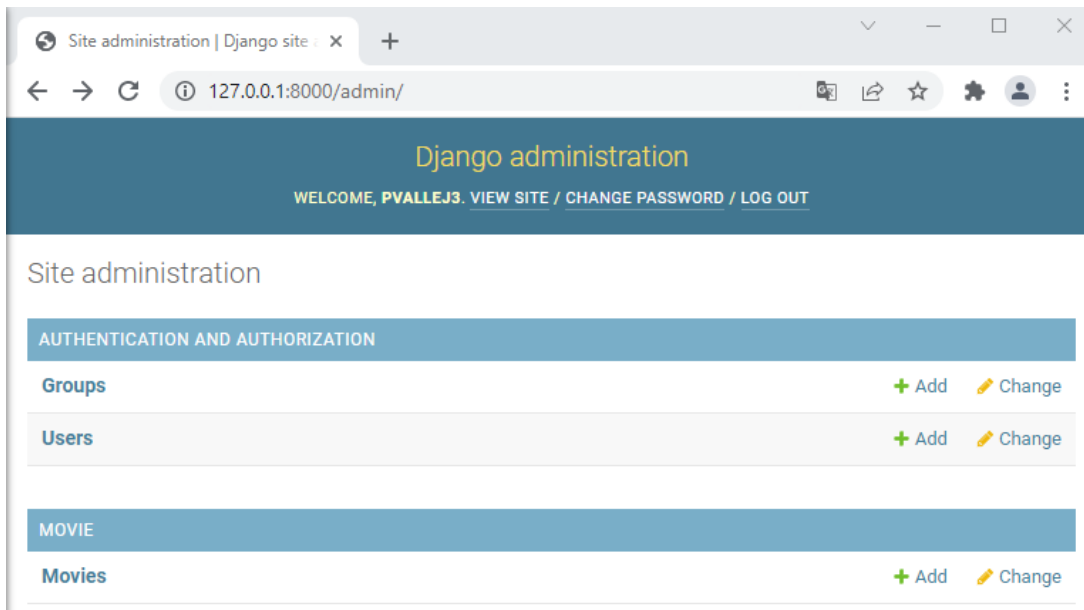
7. Configure dónde se almacenarán las imágenes. En **settings.py** de **moviereviews** agregue **MEDIA\_ROOT** y **MEDIA\_URL**, además importe **os**. **MEDIA\_ROOT** es la ruta absoluta de la carpeta que contendrá las imágenes de las películas. Unimos **BASE\_DIR** con **'media'** para indicar que habrá una subcarpeta llamada *media*. **MEDIA\_URL** es la URL que gestiona los archivos proporcionados desde **MEDIA\_ROOT**. Encontrará más información relacionada con la configuración en: <https://docs.djangoproject.com/en/3.2/ref/settings/>.

```
settings.py M
moviereviews > settings.py > MEDIA_URL
12
13 import os
14
15 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
16 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
17
18 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
19 MEDIA_URL = '/media/'
20
```

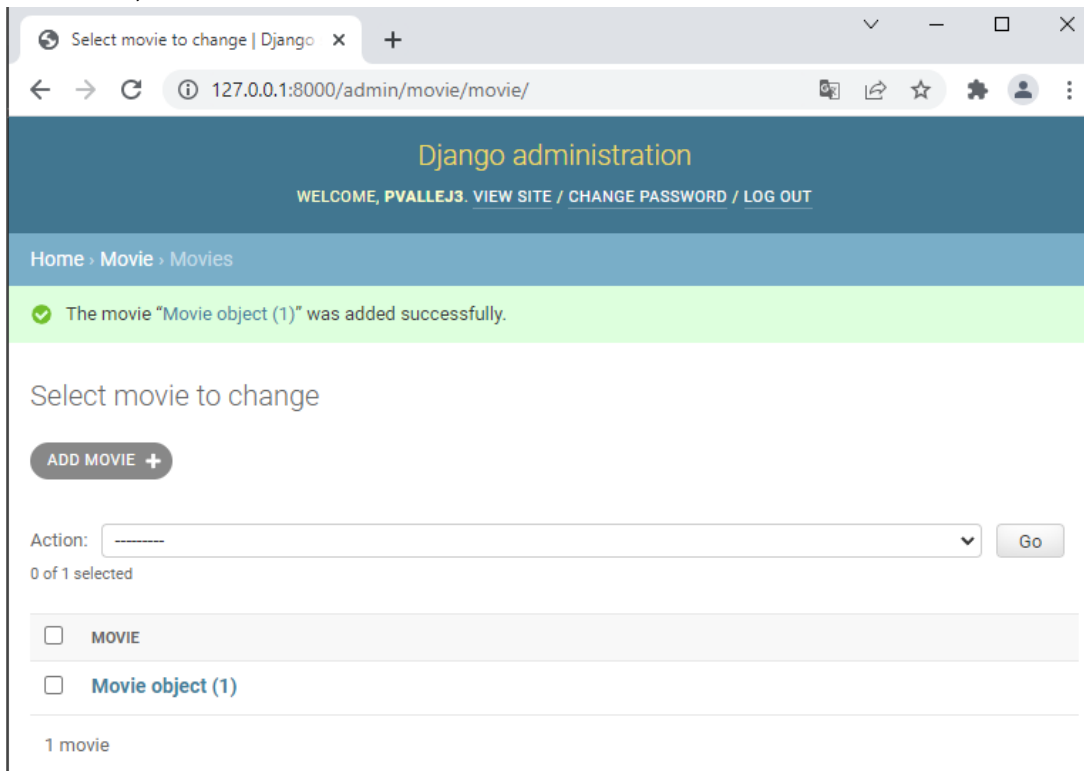
8. En **urls.py** de **moviereviews**, habilite al servidor para almacenar imágenes. **static()** es una función proporcionada por Django para servir archivos estáticos durante el desarrollo. **settings.MEDIA\_URL** es el punto final de URL donde se servirán los archivos (en el caso de este proyecto es  **'/media/'**). Django buscará los archivos en este directorio al servirlos.

```
urls.py M
moviereviews > urls.py > ...
16 from django.contrib import admin
17 from django.urls import path
18 from movie import views as movieViews
19
20 from django.conf.urls.static import static
21 from django.conf import settings
22
23 urlpatterns = [
24     path('admin/', admin.site.urls),
25     path('', movieViews.home),
26     path('about/', movieViews.about),
27 ]
28
29 urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
30
```

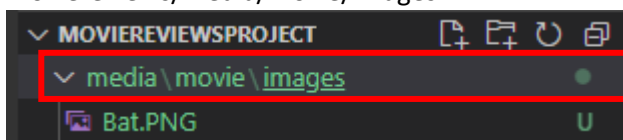
9. En el navegador ingrese (actualice) al enlace <http://localhost:8000/admin>.



10. En la tabla de Movies, agregue una película, usando la opción **Add**, luego **Add Movie** y finalmente, **Save**.

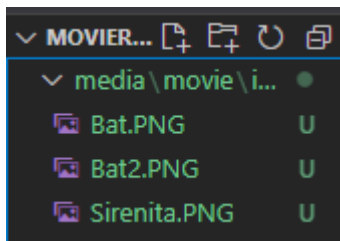
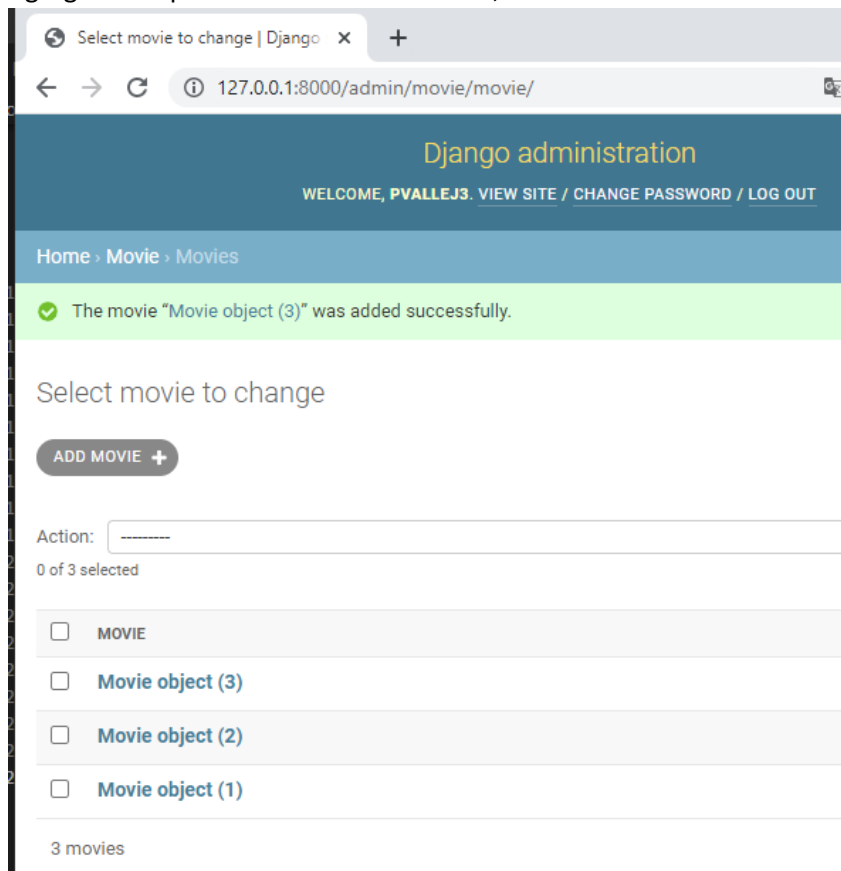


11. Verifique que la imagen cargada se almacenó automáticamente en la ruta `moviereviews/media/movie/images`.



## Visualización y búsqueda de películas

1. Agregue más películas a la base de datos, usando el Administrador.



2. En el archivo **views.py** de **movie** importe el modelo **Movie**, tome todos los objetos de **Movie** desde la base de datos y páselos a **home.html**. **GET** es uno de los métodos HTTP utilizados para solicitar datos del servidor. **request.GET** es un diccionario que contiene todos los parámetros pasados en la URL a través del método GET. **get('searchMovie')** se utiliza para obtener el valor asociado a la clave especificada. En este caso, **searchMovie** es el nombre de un parámetro en la URL que el cliente ha enviado. Por lo tanto, obtiene el valor asociado con la clave 'searchMovie' si existe, o devuelve **None** si no se proporciona ese parámetro en la URL.  
**movies = Movie.objects.all()** toma todos los objetos de película de la base de datos y los asigna a la variable **movies**.

```

views.py
movie > views.py > about
1 from django.shortcuts import render
2 from django.http import HttpResponseRedirect
3
4 from .models import Movie
5
6 # Create your views here.
7
8 def home(request):
9     #return HttpResponseRedirect('/h1>Welcome to Home Page</h1>')
10    #return render(request, 'home.html')
11    #return render(request, 'home.html', {'name': 'Paola Vallejo'})
12    searchTerm = request.GET.get('searchMovie')
13    movies = Movie.objects.all()
14    return render(request, 'home.html', {'searchTerm': searchTerm, 'movies': movies})
15
16 def about(request):
17     return HttpResponseRedirect('/h1>Welcome to About Page</h1>')

```

3. En el archivo **home.html** de **movie**, agregue elementos necesarios para mostrar las películas en la página principal. El código que está entre {} es funcional. El for recupera todas las películas de la base de datos. Los atributos title y description corresponden al título y descripción de cada película. El atributo url corresponde a la dirección url de la imagen de la película. Por defecto se listan todas las películas.

{% for movie in movies %} ... {% endfor %} se usa para hacer un ciclo a través de *movies* (uno de los parámetros que se recibió desde la función *home* de *views.py*) con *movie* actuando como una variable temporal para mantener el elemento de la iteración actual. Se usa {{ ... }} para mostrar variables como el título de la película, la descripción y la url de la imagen. Dado que *url* es opcional, puede ser nula, por lo tanto, se hace la validación {% if movie.url %} para mostrar la url únicamente cuando no sea nula.

```

5 <div class="container">
6     <form action="">
7         <div class="mb-3">
8             <label for="searchMovie" class="form-label">
9                 Search for Movie:
10            </label>
11            <input type="text" class="form-control" name="searchMovie"/>
12        </div>
13        <button type="submit" class="btn btn-primary">Search</button>
14    </form>
15    <p>Searching for {{searchTerm}}</p>
16    {% for movie in movies %}
17        <h2>{{ movie.title }}</h2>
18        <h3>{{ movie.description }}</h3>
19        
20        {% if movie.url %}
21            <a href="{{ movie.url }}"> Movie Link </a>
22        {% endif %}
23    {% endfor %}
24    <br/>
25 </div>

```

4. En **views.py** complete la función *home* validando si la palabra buscada está contenida en el título de alguna de las películas; si es así, se listarán todas las películas que coincidan con la

búsqueda. Si se introduce un término de búsqueda (searchTerm), se llama al método de filtrado (*filter*) del modelo para que devuelva los objetos de película que coincidan con el término de búsqueda sin distinguir mayúsculas de minúsculas. Si no se introduce ningún término de búsqueda, simplemente se devuelven todas las películas.

```
views.py M
movie > views.py > about
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  from .models import Movie
5
6  # Create your views here.
7
8  def home(request):
9      #return HttpResponse('<h1>Welcome to Home Page</h1>')
10     #return render(request, 'home.html')
11     #return render(request, 'home.html', {'name': 'Paola Vallejo'})
12     searchTerm = request.GET.get('searchMovie')
13     if searchTerm:
14         movies = Movie.objects.filter(title__icontains=searchTerm)
15     else:
16         movies = Movie.objects.all()
17     return render(request, 'home.html', {'searchTerm':searchTerm, 'movies': movies})
18
19 def about(request):
20     return HttpResponse('<h1>Welcome to About Page</h1>')
```

5. En el navegador acceda al enlace <http://localhost:8000> para visualizar el detalle de todas las películas.



6. Busque una película específica y valide que el resultado es correcto.

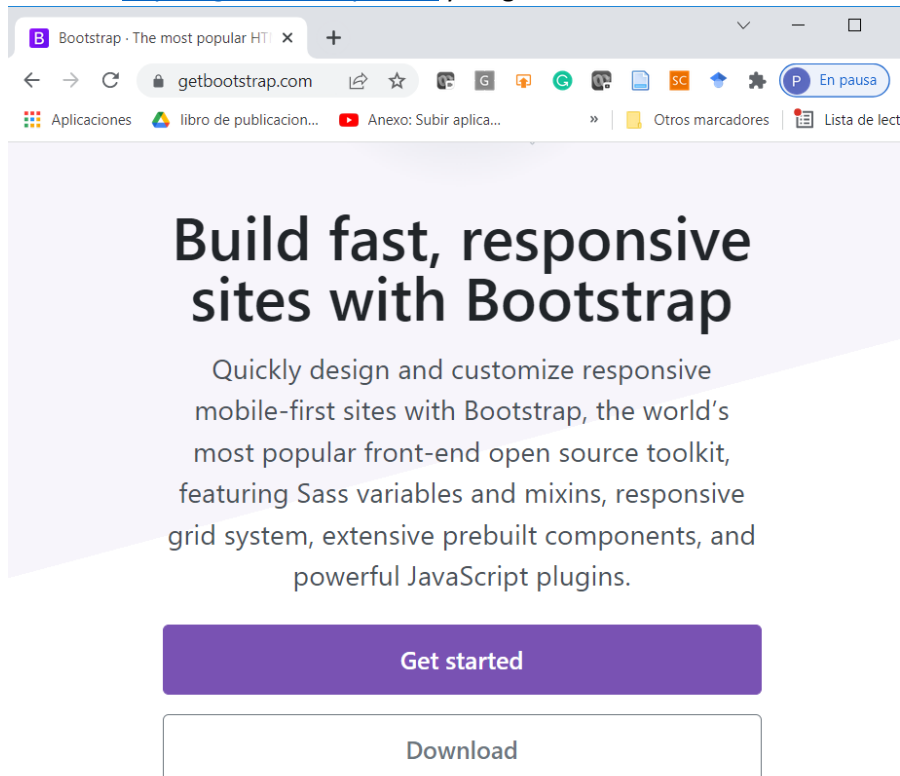
The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/?searchMovie=captura`. Below the address bar, there is a search form with the label "Search for Movie:" followed by a text input field containing the word "captura". A "Search" button is located to the right of the input field. Below the search form, the text "Searching for captura" is displayed. Underneath, the heading "La captura" is shown, followed by the subheading "Pelicula infantil". At the bottom of the visible area, there is a large, solid black silhouette of a person's head and shoulders.

7. Haga un commit (y un push al repositorio remoto en GitHub), cuyo mensaje sea “busqueda de peliculas”.

## USO DE BOOTSTRAP

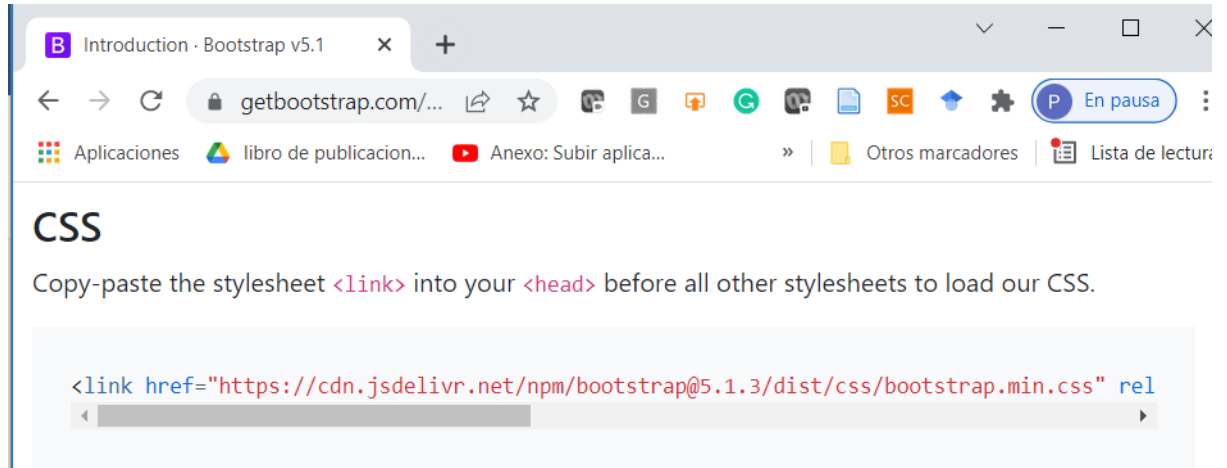
### Inclusión de Bootstrap

1. Acceda a <https://getbootstrap.com/> y luego a **Get Started**.

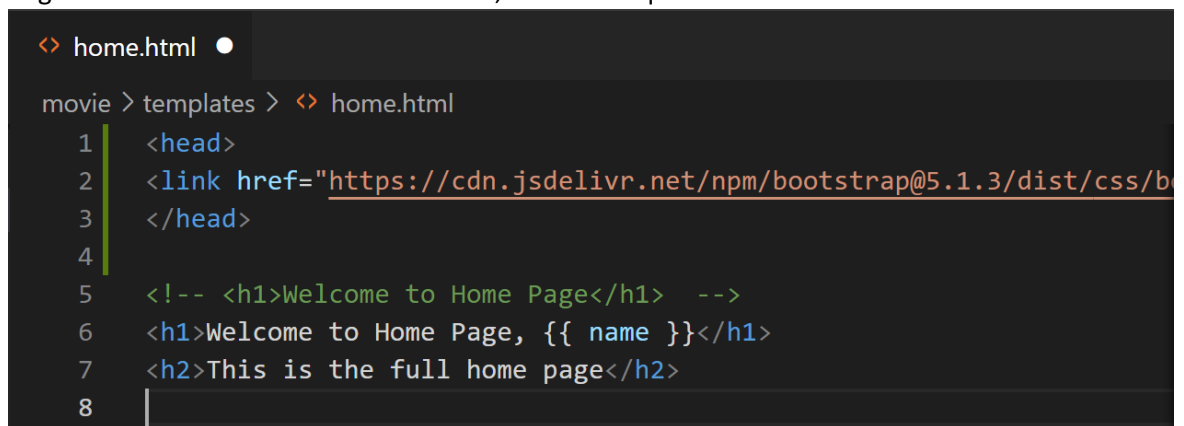




2. Copie el enlace que aparece en la sección CSS.



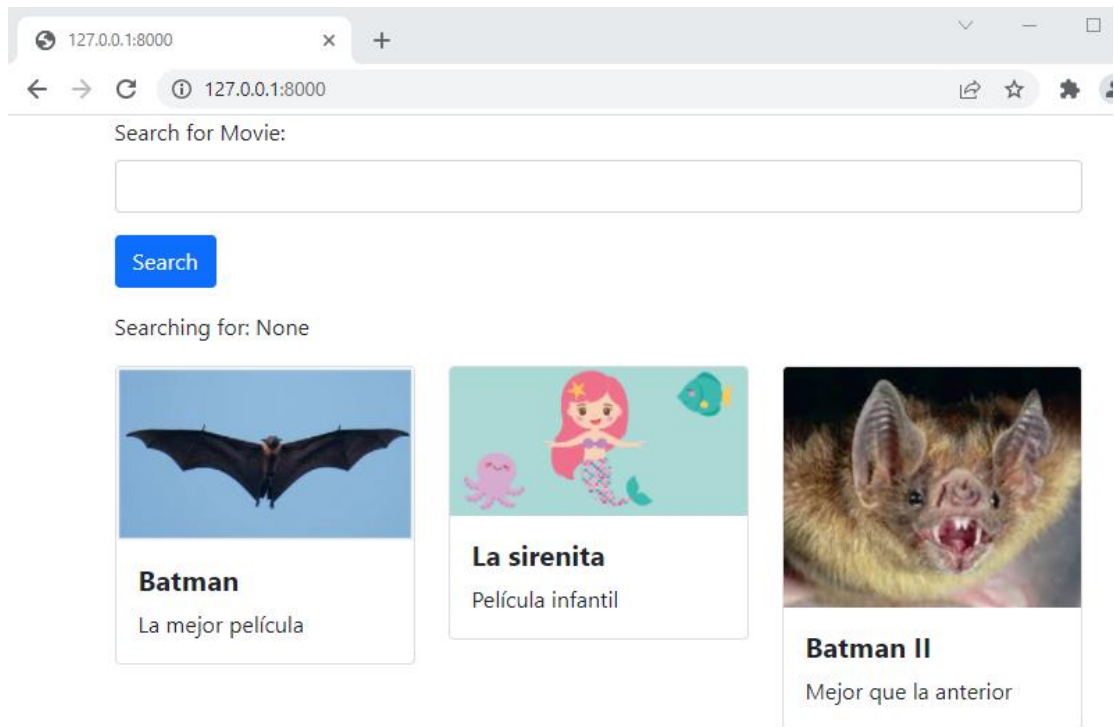
3. Pegue el enlace en el archivo **home.html**, entre la etiqueta **<head>**.



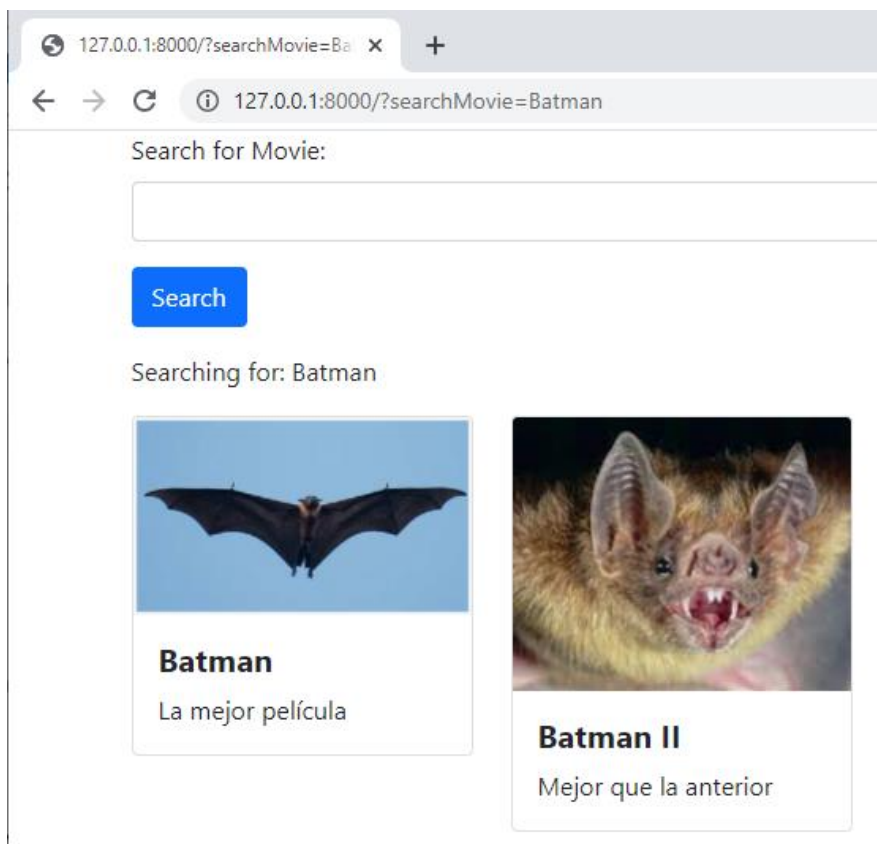
4. En el navegador ingrese (refresque) al enlace <http://localhost:8000> y visualizará la información de la página, pero esta vez, con un estilo diferente.
5. Mejore el aspecto visual de la página usando el componente Card de Bootstrap.

```
<> home.html M ●
movie > templates > <> home.html > div.container
1 <head>
2   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boots
3 </head>
4
5 <div class="container">
6   <!-- <h1>Welcome to Home Page</h1> -->
7   <!--<h1>Welcome to Home Page, {{ name }}</h1> -->
8   <!--<h2>This is the full home page</h2> -->
9   <form action="">
10    <div class="mb-3">
11      <label for="searchMovie" class="form-label">
12        Search for Movie:
13      </label>
14      <input type="text" class="form-control" name="searchMovie" />
15    </div>
16    <button type="submit" class="btn btn-primary">Search</button>
17  </form>
18  <p>Searching for: {{ searchTerm }}</p>
19  <div class="row row-cols-1 row-cols-md-3 g-4">
20    {% for movie in movies %}
21    <div v-for="movie in movies" class="col">
22      <div class="card">
23        
24        <div class="card-body">
25          <h5 class="card-title fw-bold">{{ movie.title }}</h5>
26          <p class="card-text">{{ movie.description }}</p>
27          {% if movie.url %}
28          <a href="{{ movie.url }}" class="btn btn-primary">
29            Movie Link
30          </a>
31          {% endif %}
32        </div>
33      </div>
34    </div>
35    {% endfor %}
36  </div>
37  <br />
38  <br />
39 </div>
```

6. En el navegador acceda al enlace <http://localhost:8000> para visualizar el detalle de las películas.



- En el navegador acceda al enlace <http://localhost:8000> y busque alguna película por su título.



- Haga un commit (y un push al repositorio remoto en GitHub), cuyo mensaje sea “visualizacion de películas con estilo”.

# BIBLIOGRAFÍA

Lim, G., Correa, D. Beginning Django 3 Development. Build Full Stack Python Web Applications. 2021.  
<https://github.com/danielgara/bookdjango4.0>

Chacon, S., Straub, B. Pro Git. 2024. <https://git-scm.com/book/en/v2>

Octoverse. 2024. <https://octoverse.github.com/2022/top-programming-languages>

Moure, B. Curso de GIT y GITHUB desde CERO para PRINCIPIANTES.  
<https://www.youtube.com/watch?v=3GymExBkKjE>

[https://twitter.com/Harsa\\_Dash/status/1750741820135596036?s=20](https://twitter.com/Harsa_Dash/status/1750741820135596036?s=20)