

# PROYECTO DE APLICACIÓN WEB

## PYTHON-FLASK-MONGODB

CICLO FORMATIVO GRADO SUPERIOR  
ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

**PEDRO LLULL ALLES**

# ÍNDICE

1-DESCRIPCIÓN DEL PROYECTO .....	3
2-DEFINICIONES.....	4
 ¿Qué es MongoDB? .....	4
¿Por qué Mongo db? Ventajas y desventajas.....	4
¿Qué es JSON? .....	6
Conceptos CRUD en MongoDB.....	7
 ¿Qué es Flask? .....	8
 ¿Qué es Python? .....	9
 ¿Qué es JQuery? .....	10
 ¿Qué es AJAX? .....	10
3-OBJETIVOS: justificación del proyecto .....	11
4-RECURSOS Y REQUISITOS NECESARIOS.....	11
5-PROYECTO .....	12
IMPLANTACIÓN DE REQUISITOS: .....	12
DESARROLLO DEL PROYECTO: .....	18
PRUEBAS: .....	33
6-CONCLUSIONES .....	39
FUTURAS MEJORAS: .....	39
OBJETIVOS ALCANZADOS: .....	39
7-BIBLIOGRAFÍA .....	40

# **1-DESCRIPCIÓN DEL PROYECTO**

Partimos de la base de que no tenemos ningún conocimiento sobre base de datos no relacionales, excepto su existencia, y elegimos el sistema gestor de base de datos no relacional más popular “MongoDB”, para llevar a cabo distintos objetivos. A partir de allí desarrollar en un Microframework local “Flask” con un Front-end un interfaz de entrada amigable para el cliente, para registrar y autenticar usuarios, que una vez registrados, podrán entrar en una zona de consulta de datos sobre los mismos usuarios e información sobre películas. Esta información será mostrada en pantalla. Las consultas en la parte Back-end se harán a la base de datos “MongoDB”.

Para el desarrollo de la apariencia frontal o parte cliente se usarán las tecnologías de HTML5 (Css, HTML, JavaScript, JQuery) y para la comunicación y obtención de datos desde la base de datos se usaran las tecnologías de Python y Ajax.

Como se comentó en el primer párrafo, tampoco se tienen conocimientos sobre los lenguajes JQuery, Python y comunicación Ajax los cuales se aprenderán las bases para poder llevar a cabo el proyecto.

## 2-DEFINICIONES



### *¿Qué es MongoDB?*

**MongoDB** (de la palabra en inglés “**humongous**” que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. Grandes compañías como Facebook utilizan esta base de datos por la gran cantidad de datos que necesitan consultar en un tiempo mínimo. Otras empresas que lo utilizan también son: Bosch, Forbes, Expedia, IBM, Codecademy, eBay...

### *¿Por qué Mongo db? Ventajas y desventajas*

A grosso modo, y de manera muy resumida, mongo ofrece mayor velocidad y mejor escalabilidad que un sistema tradicional de base de datos de tipo relacional, pero no cumple **ACID**: Atomicidad, Consistencia, Aislamiento y Durabilidad. Por lo que he podido investigar tampoco permiten Transacciones algo que muchos desarrolladores demandan, pero recientemente se ha anunciado que a partir de la versión 4.2 de “MongoDB” que saldrá en Julio de este año también se incluirán las Transacciones.

### *Modelado de datos*

Una de las partes más difíciles es crear una base de datos no relacional desde cero, para ello lo principal es tener claro los tipos de datos:

### *Tipos de datos*

A continuación detallamos, algunos de los tipos de datos que soporta mongo:

- **Integer** – Números enteros.

- **Double** – Números con decimales.
- **Boolean** – Booleanos verdaderos o falsos.
- **Date** – Fechas.
- **Timestamp** – Estampillas de tiempo.
- **Null** – Valor nulo.
- **Array** – Arreglos de otros tipos de dato.
- **Object** – Otros documentos embebidos.
- **ObjectID** – Identificadores únicos creados por MongoDB al crear documentos sin especificar valores para el campo `_id`.
- **Data Binaria** – Punteros a archivos binarios.
- **Javascript** – Código y funciones Javascript.
- **String** – Cadenas de caracteres.

Podemos guardar los diferentes datos, según un patrón, los dos más comunes son:

### **Embeber**

Este patrón se enfoca en incrustar documentos uno dentro de otro con la finalidad de hacerlo parte del mismo registro y que la relación sea directa.

### **Referenciar**

Este patrón busca imitar el comportamiento de las claves foráneas para relacionar datos que deben estar en colecciones diferentes.

### ***Estructura de Mongo***

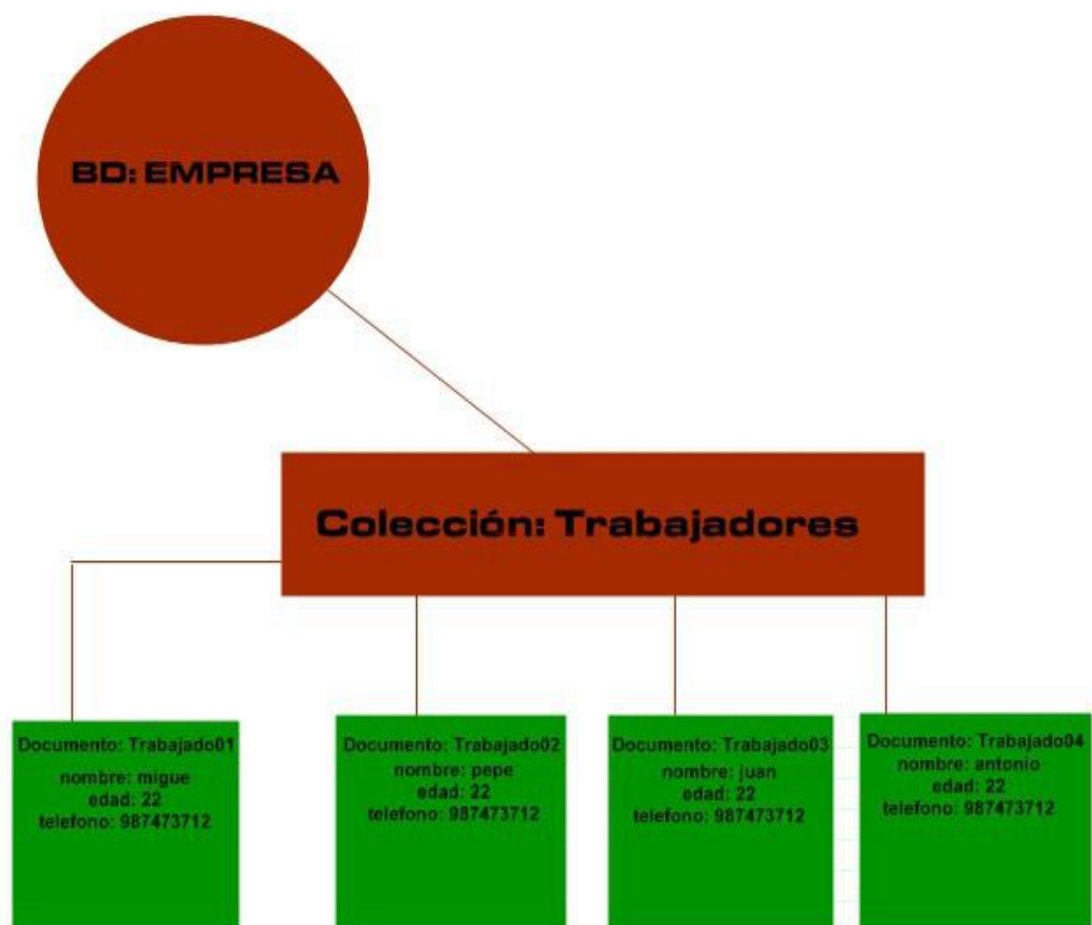
Lo principal sería comprender cómo funciona mongo, es decir la estructura que sigue a la hora de guardar o alojar los datos.

Mongo, puede tener varias base de datos, dentro de estas base de datos va a guardar colecciones y las colecciones no son más que un conjunto de documentos, los cuales guardan información de algo en concreto, con el formato clave-valor.

Un ejemplo básico, supongamos que tenemos una base de datos llamada: EMPRESA, dentro de esta tenemos una colección llamada TRABAJADORES, y dentro de esta colección tenemos un

documento llamado TRABAJADOR01, y dentro de estos documentos los valores para las claves deseadas:

A continuación se detalla mejor en la imagen:



## *¿Qué es JSON?*

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

### ¿Qué ventajas tiene la representación de nuestros datos utilizando un formato JSON?

- JSON presenta un framework flexible y conciso tanto para consultas, como para almacenar registros.
- La sintaxis JSON es similar a la de otras estructuras de datos utilizados en muchos lenguajes de programación y es por tanto, familiar para los desarrolladores.
- JSON es independiente del lenguaje de programación que se utilice.

Estructura de JSON y BSON de un documento:

```
Trabajador1 = {  
  "id": 1,  
  "name": "David",  
  "age": 34,  
  "address": {  
    "city": "Valladolid",  
    "postalCode": "47011"}  
}
```

## Conceptos CRUD en MongoDB

Algunos ejemplos de CRUD (Create, Read, Update, Delete) en “MongoDb”:

```
db.trabajadores.find({'Nombre': 'Inma'})
```

Esto sería al equivalente en una base de datos relacional a un select con where nombre = Inma, es decir sacar todos los datos de los ficheros con el atributo “Nombre” con valor “Inma”.

```
db.trabajadores.insert({'Nombre:  "Jose  Alejandro",  FechaNac:  "26/03/2014",  Direccion:  
"C/Aviacion nº30",  Uid: "3"})
```

Esto sería equivalente a un insert en una base datos relacional un nuevo registro.

```
db.trabajadores.update({"Uid": "1"}, {"$set":{"Apellidos":"Martin Serrano"}})
```

Este sería el equivalente a hacer un update en una base de datos relacional. Modificar el usuario con el Uid con valor 1, el apellido por el mostrado.

```
db.trabajadores.remove({"Uid": "2"})
```

Esto sería el equivalente a “delete” en una base de datos relacional, eliminar el usuario con Uid = 2

## *¿Qué es Flask?*



Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD.

El código siguiente muestra una aplicación simple que imprime "¡Hola Mundo!":

```
from flask import Flask  
app = Flask(__name__)  
  
@app.route("/")  
def hello():  
    return "¡Hola Mundo!"  
  
if __name__ == "__main__":  
    app.run()
```

Abriríamos el navegador web e introduciríamos la ruta en el que estaría activo el servidor Flask (127.0.0.1:5000) y nos aparecería el mensaje “¡Hola Mundo!”





## *¿Qué es Python?*

Python es un lenguaje de programación desarrollado como proyecto de código abierto y es administrado por la empresa Python software Foundation.

Fue creado por Guido van Rossum y su nombre se debe a la afición de su creador a los humoristas británicos Monty Python.

Se trata de un lenguaje de programación en scripts, competencia directa de Perl.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. También viene con una gran colección de módulos estándar que proporcionan E/S de ficheros, llamadas al sistema, sockets, interfaces GUI, etc.

Se trata de un lenguaje interpretado, lo que permite ahorrar el proceso de compilado.

Características generales de Python:

- Lenguaje de programación de alto nivel del tipo scripting.
- Diseñado para ser fácil de leer y simple de implementar.
- Es código abierto (de libre uso).
- Puede ejecutarse en Mac, Windows y sistemas Unix; también ha sido portado a máquinas virtual JAVA y .NET.
- Es a menudo usado para desarrollar aplicaciones web y contenido web dinámico.
- Se utiliza para crear extensiones tipo plug-ins para programas de 2d y 3d como Autodesk Maya, GIMP, Blender, Inkscape, etc.
- Los scripts de Python tienen la extensión de archivo .PY, que pueden ser parseados y ejecutados inmediatamente.
- Permite grabar programas compilados con extensión de archivo .PYC, los cuales suelen ser usados como módulo que pueden ser referenciados por otros programas Python.



## *¿Qué es JQuery?*

jQuery es una biblioteca multiplataforma de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. jQuery es la biblioteca de JavaScript más utilizada.



## *¿Qué es AJAX?*

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

JavaScript es un lenguaje de programación (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

### **3-OBJETIVOS: justificación del proyecto**

El objetivo es aprender nuevos lenguajes y tecnologías que conforman el panorama actual del desarrollo de aplicaciones web, tanto del lado cliente (HTML5) como del lado servidor (PYTHON; AJAX), y saber aplicarlo a un caso similar a una página web real. Tener una visión total de las partes que lo conforman, desde un interfaz amigable al cliente a una buena comunicación, mandar y recibir información desde la base de datos y devueltas al cliente de forma interactiva y agradable. También, en este caso, aprender los conceptos básicos sobre las bases de datos no SQL, su estructura, funcionamiento y finalmente poder hacer CRUD sobre ellas.

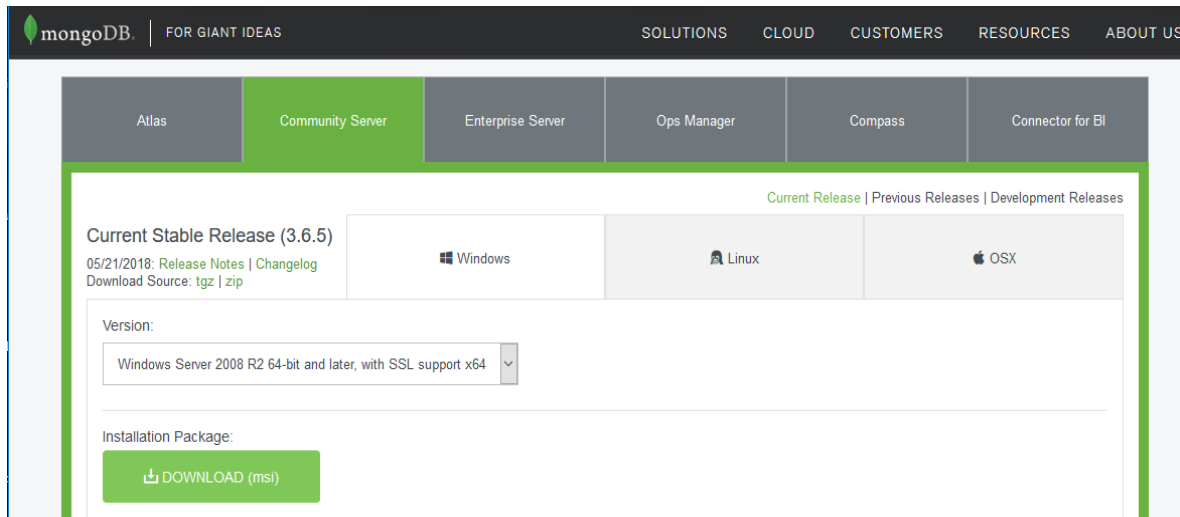
### **4-RECURSOS Y REQUISITOS NECESARIOS**

- **HARDWARE:** Para llevar a cabo el proyecto voy a utilizar mi ordenador portátil LENOVO con 12GB RAM, 1T de Disco Duro y WINDOWS 10.
- **SOFTWARE:** Mencionado anteriormente el proyecto se montará sobre el sistema operativo WINDOWS 10, utilizaré el editor de texto SUBLIMETEXT 3 para escribir las páginas y los scripts, el Framework FLASK, base de datos MONGODB, lenguajes de programación: PYTHON, JAVASCRIPT, JQUERY, HTML5, CRUD para Mongoddb y librería PYMONGO de python y AJAX para comunicación con Mongoddb. También utilizare la librería JQuery disponible en su página oficial y la librería de BOOTSTRAP bootstrap.min.css para dar formato y estilo a la página web que también se puede descargar en la página oficial de bootstrap.

## 5-PROYECTO

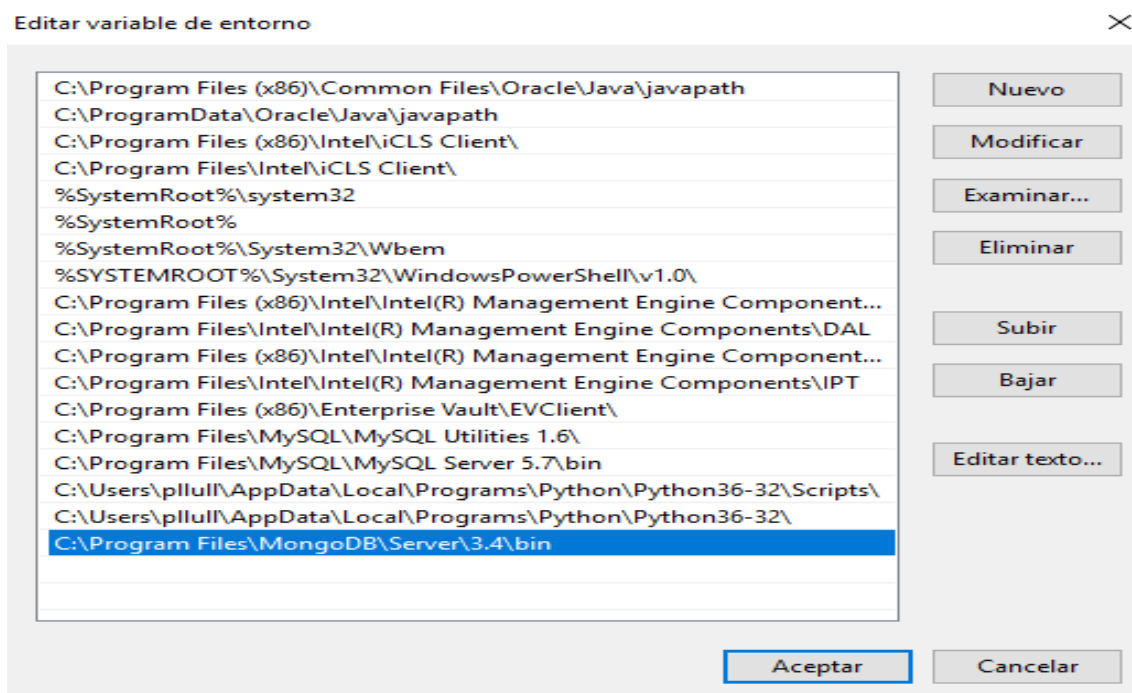
### IMPLANTACIÓN DE REQUISITOS:

**MongoDB:** Para la instalación de MongoDB entramos en la página oficial de mongo > downloads, descargamos el ejecutable para Windows de 64b de MongoDB Community Server

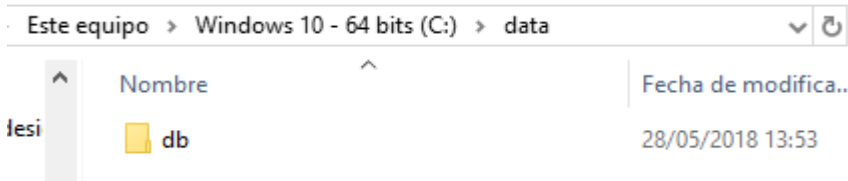


-Abrimos el ejecutable, damos aceptar a las políticas y términos e instalamos.

-Una vez instalado demos de agregar a la PATH o variables del sistema la ruta donde está el ejecutable de MongoDB



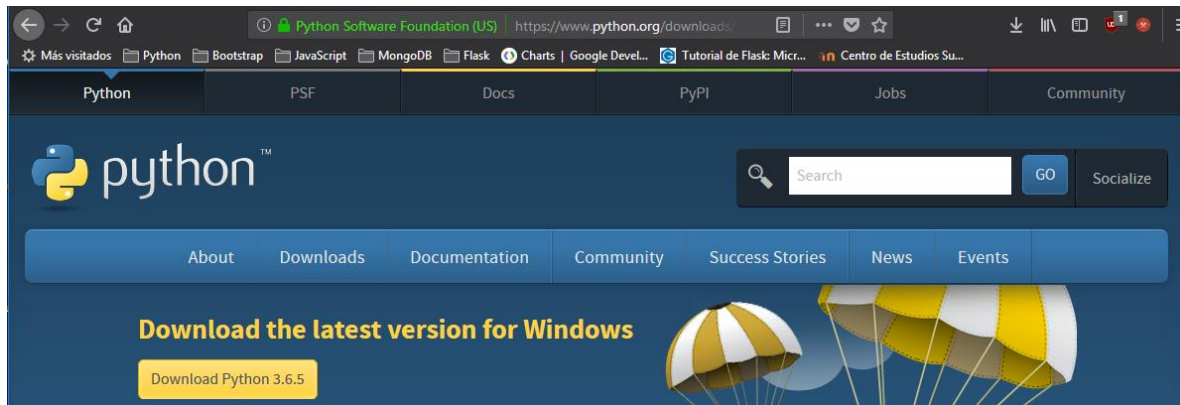
-También debemos crear las carpetas donde se almacenarán desde ahora la información y archivos .LOG de MongoDB. En C:\ creamos la carpeta DATA y dentro la carpeta DB.



-Ahora abrimos un terminal de CMD Windows y ejecutamos MONGOD que se encarga de tener mongo en escucha para poder trabajar. Dejamos este terminal abierto y ejecutándose y abrimos otro terminal y ejecutamos MONGO, este conectará con la Base de Datos que está a la escucha y hemos abierto anteriormente y nos mostrará el terminal de mongo. En este terminal ya podemos operar con la base de datos MongoDB y hacer todas las consultas CRUD.

A screenshot showing two overlapping Windows Command Prompt windows. The top window, titled 'Símbolo del sistema - mongod', shows the MongoDB server startup logs, including the path 'C:\data\db' and the version '3.4.15'. The bottom window, titled 'Símbolo del sistema - mongo', shows the MongoDB shell interface. It displays the connection to 'mongodb://127.0.0.1:27017' and the server version '3.4.15'. A warning message is visible: '\*\* WARNING: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted.' The user has entered the command 'show dbs' and the output shows the sizes of the 'admin', 'local', 'peliculas', and 'usuarios' databases, all at 0.000GB.

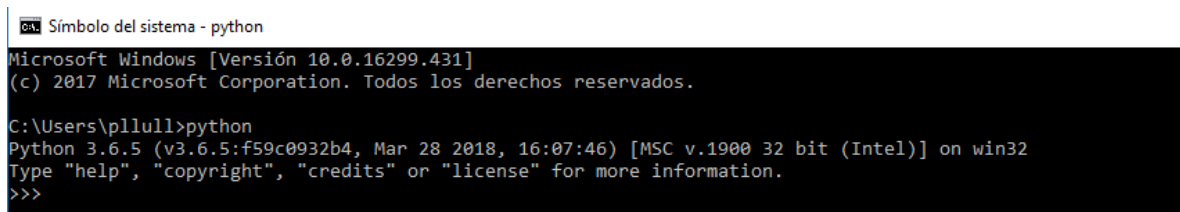
**Python:** Para instalar Python 3 en Windows, solo hay que ir a la página oficial de Python y descargarlo desde DOWNLOADS.



-Una vez descargado abrimos el instalador, aceptamos las políticas y términos, como en el caso de MongoDB también es necesario agregarlo al PATH o variables del sistema aunque el instalador de Python nos lo facilita con una pestaña que podemos confirmar al instalar.



-Una vez instalado ya podríamos utilizar Python en su propia consola o abriendo un CMD de Windows y escribiendo “python” también se abriría el terminal para ejecutar código

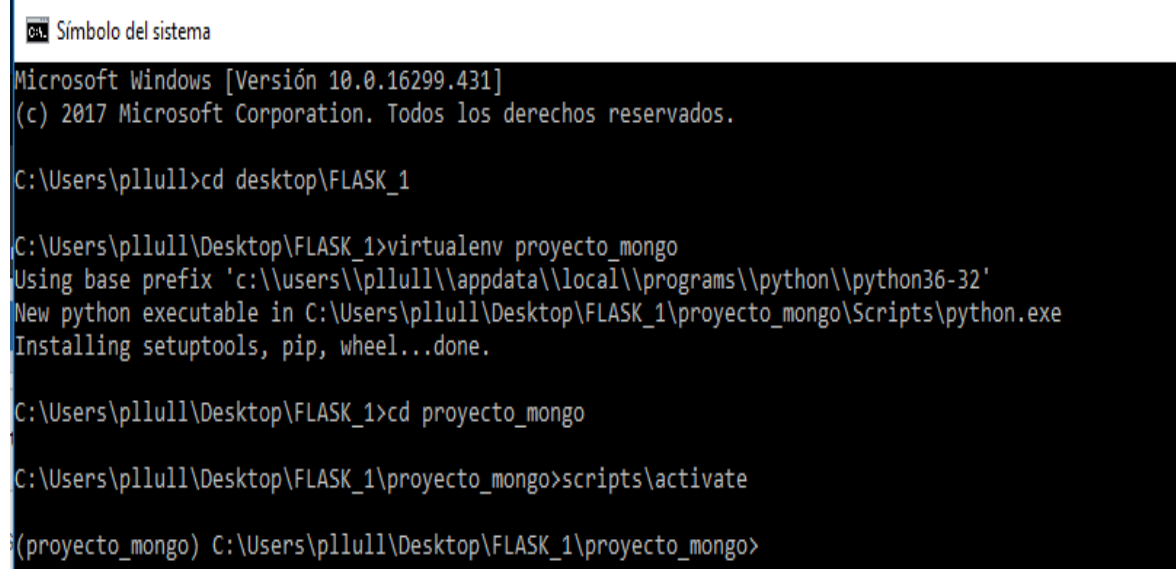


**Flask:** Para instalar y utilizar Flask en Windows, tenemos que utilizar otro programa que nos genera “Proyectos” en python en un entorno virtual y necesario ya que Flask esta escrito y se utiliza con el lenguaje de python. El programa que necesitamos es VIRTUALENV que se instala con PIP.

-PIP es un instalador de librerias de python que se instala automaticamente al instalar python en el equipo. Para instalar VIRTUALENV y teniendo PIP al haber instalado python anteriormente, solo tenemos que abrir un CMD de Windows e introducir:

***pip install virtualenv***

-Una vez instalado, creamos una carpeta principal donde queramos albergar nuestros proyectos con Flask, en mi caso tengo una carpeta en mi escritorio que se llama FLASK\_1, con un terminal CMS de Windows abierto nos posicionamos en la ruta de la carpeta. Una vez en la ruta ejecutamos:



```
Microsoft Windows [Versión 10.0.16299.431]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\Users\pllull>cd desktop\FLASK_1

C:\Users\pllull\Desktop\FLASK_1>virtualenv proyecto_mongo
Using base prefix 'c:\users\pllull\appdata\local\programs\python\python36-32'
New python executable in C:\Users\pllull\Desktop\FLASK_1\proyecto_mongo\Scripts\python.exe
Installing setuptools, pip, wheel...done.

C:\Users\pllull\Desktop\FLASK_1>cd proyecto_mongo

C:\Users\pllull\Desktop\FLASK_1\proyecto_mongo>scripts\activate

(proyecto_mongo) C:\Users\pllull\Desktop\FLASK_1\proyecto_mongo>
```

-Donde la definicion del nombre de nuestro proyecto será la palabra que pongamos despues de VIRTUALENV, en mi caso proyecto\_mongo, se creara una carpeta con nuestro proyecto y se instalara el entorno virtual en esa carpeta para poder ejecutar python.

-Como se puede apreciar en el pantallazo anterior tenemos que posicionarnos dentro de la carpeta que se a creado y ejecutar SCRIPTS\ACTIVATE para que se active el entorno del proyecto, se puede ver ahora a la izquierda de la ruta, entre paréntesis, el nombre del proyecto activado.

-Ahora que tenemos el proyecto y la virtualizacion de este a punto tenemos que instalar Flask en ella con:

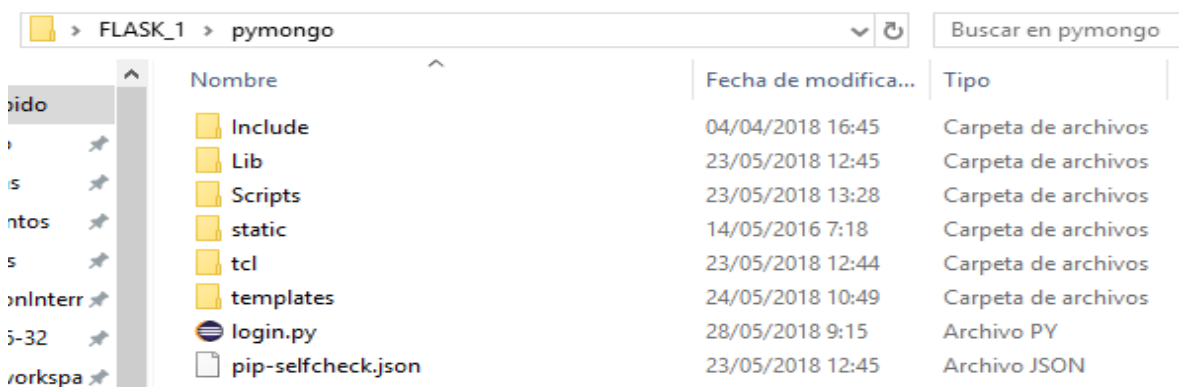
***pip install flask***

```
(proyecto_mongo) C:\Users\pllull\Desktop\FLASK_1\proyecto_mongo>pip install flask
Collecting flask
  Using cached https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edba/Flask-1.0.2-py2.py3-none-any.whl
Collecting Jinja2>=2.10 (from flask)
  Using cached https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a73/Jinja2-2.10-py2.py3-none-any.whl
Collecting Werkzeug>=0.14 (from flask)
  Using cached https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682bef8d0aae04fe335243/Werkzeug-0.14.1-py2.py3-none-any.whl
Collecting click>=5.1 (from flask)
  Using cached https://files.pythonhosted.org/packages/34/c1/8806f99713ddb993c5366c362b2f908f18269f8d792aff1abfd700775a7/click-6.7-py2.py3-none-any.whl
Collecting itsdangerous>=0.24 (from flask)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10->flask)
Installing collected packages: MarkupSafe, Jinja2, Werkzeug, click, itsdangerous, flask
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 flask-1.0.2 itsdangerous-0.24

(proyecto_mongo) C:\Users\pllull\Desktop\FLASK_1\proyecto_mongo>
```

-Ya estaría Flask preparado para su utilización y podemos observar que dentro del proyecto proyecto\_mongo se han creado varias carpetas que utiliza Flask, como las librerías de python.

-No entra por defecto, pero Flask suele utilizar una estructura de carpetas para los proyectos para poder acceder a los diferentes componentes que compondrán el proyecto web y en este caso tenemos que crear la carpeta TEMPLATES que es donde irán las páginas HTML que se irán consultando en el servidor local, y también debemos crear la carpeta STATIC y dentro de esta las carpetas: CSS, IMG y JS, que es donde irán los archivos CSS (para dar formato y estilo a las páginas), IMG para las imágenes que queramos enseñar en alguna de las páginas y los archivos JAVASCRIPT (para enlazar los scripts de JQUERY que darán vida a las páginas y se encargarán de la comunicación con AJAX). Así quedaría el proyecto en la carpeta: *(La imagen es del proyecto principal ya operativo llamado “pymongo”)*



Nombre	Fecha de modifica...	Tipo
Include	04/04/2018 16:45	Carpeta de archivos
Lib	23/05/2018 12:45	Carpeta de archivos
Scripts	23/05/2018 13:28	Carpeta de archivos
static	14/05/2016 7:18	Carpeta de archivos
tcl	23/05/2018 12:44	Carpeta de archivos
templates	24/05/2018 10:49	Carpeta de archivos
login.py	28/05/2018 9:15	Archivo PY
pip-selfcheck.json	23/05/2018 12:45	Archivo JSON

-El archivo LOGIN.PY es el archivo central que se encargará de llamar a las librerías, establecer las rutas donde se alojarán las páginas del servidor a través de el código python.

-Además para nuestro proyecto necesitaremos instalar librerías extra para poder coger los datos de formulario (request), la comunicación con la base de datos MongoDB (pymongo y mongoclient), para crear rutas estáticas (url\_for), para hacer redirección de páginas (redirect) y para transformar



datos STRING en datos JSON (jsonify). Tenemos que instalarlas con *pip install* y *el nombre de cada librería* (las que están entre paréntesis).

**Sublime text 3:** Desde la página web oficial del editor, se puede descargar e instalar Sublime text, es fácil, rápido y para mí un gran editor de texto que ayuda mucho con sus contrastes y colores a identificar palabras clave (funciones, variables...) y distinción entre diferentes lenguajes, haciendo más fácil y legible el código.

## DESARROLLO DEL PROYECTO:

Una vez instalados todos los requisitos necesarios vamos paso a paso a conformarlo para llegar a la puesta en marcha del proyecto.

Empezaremos por la parte de la base de datos MongoDB, con la introducción de datos desde el terminal para poder visualizar más adelante los datos de películas, ya que los datos de los usuarios se crearán e introducirán directamente al introducirlos en los campos del interfaz para el cliente. Abrimos pues el terminal de MongoDB en un CMD de Windows (recordar que hay que tener otro terminal ejecutando MONGODB para poder, en otro terminal, hacer las consultas en mongo) y empezamos por crear la base de datos peliculas, solo es necesario poner USE PELICULAS para que se cree la base de datos:

```
C:\Users\plull>mongo
MongoDB shell version v3.4.15
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.15
Server has startup warnings:
2018-05-28T05:19:42.291-0700 I CONTROL [initandlisten]
2018-05-28T05:19:42.291-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-05-28T05:19:42.292-0700 I CONTROL [initandlisten] **      Read and write access to data and configuration is u
nrestricted.
2018-05-28T05:19:42.292-0700 I CONTROL [initandlisten]
> show dbs
admin      0.000GB
local      0.000GB
usuarios   0.000GB
> use peliculas
switched to db peliculas
```

Después introducir datos en una colección para que se cree tanto la base de datos, como la colección que contendrá documentos con cada película:

```
> peli = {
...   "titulo":"Thor 3",
...   "año":"2015",
...   "actores":{
...     "principal":"Chris Hemsworth",
...     "secundario":"Natalie Portman"
...   }
... }
{
  "titulo" : "Thor 3",
  "año" : "2015",
  "actores" : {
    "principal" : "Chris Hemsworth",
    "secundario" : "Natalie Portman"
  }
}
> db.marvel.insert(peli)
WriteResult({ "nInserted" : 1 })
>
```

Se ha creado así, la base de datos PELICULAS que contiene una colección MARVEL con un documento THOR 3, he utilizado una variable para introducir este documento (me parece más

limpio) pero también se podría introducir directamente y entre llaves ( {} ) la información que quisiéramos introducir.

También se pueden introducir varios documentos a la vez agregando estos dentro de un ARRAY.

```
> peli = [
... {
...   "titulo":"Thor 3",
...   "año":"2015",
...   "actores":{"
...     "principal":"Chris Hemsworth",
...     "secundario":"Natalie Portman"
...   }
... },
... {
...   "titulo":"X-men",
...   "año":"2003",
...   "actores":{"
...     "principal":"Huge Jackman",
...     "secundario":"Patrick Stewart"
...   }
... },
... {
...   "titulo":"Deadpool",
...   "año":"2014",
...   "actores":{"
...     "principal":"Ryan Reynolds",
...     "secundario":"Stan Lee"
...   }
... },
... {
...   "titulo":"Guardianes de la galaxia vol.1",
...   "año":"2015",
...   "actores":{"
...     "principal":"Chris Pratt",
...     "secundario":"Vin Diesel"
...   }
... },
... ]
```

```
> db.marvel.insert(peli)
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 4,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

Hacemos una consulta a la colección MARVELy que nos enseñe el documento de la película X-MEN.

```
> db.marvel.findOne({ "titulo":"X-men"})
{
  "_id" : ObjectId("5b0c0c5a812dafddc9c71a70"),
  "titulo" : "X-men",
  "año" : "2003",
  "actores" : {
    "principal" : "Huge Jackman",
    "secundario" : "Patrick Stewart"
  }
}
```

El campo ID es un identificador ÚNICO que da MongoDB a cada documento. Podemos observar que este documento lo que sería en SQL una relación 1-1 o en mongo es un documento dentro de otro o embeber, esa película tiene esos actores y esos actores salen en esa película. En MongoDB existen las relaciones pero se contemplan de diferente manera que en las BBDD SQL.

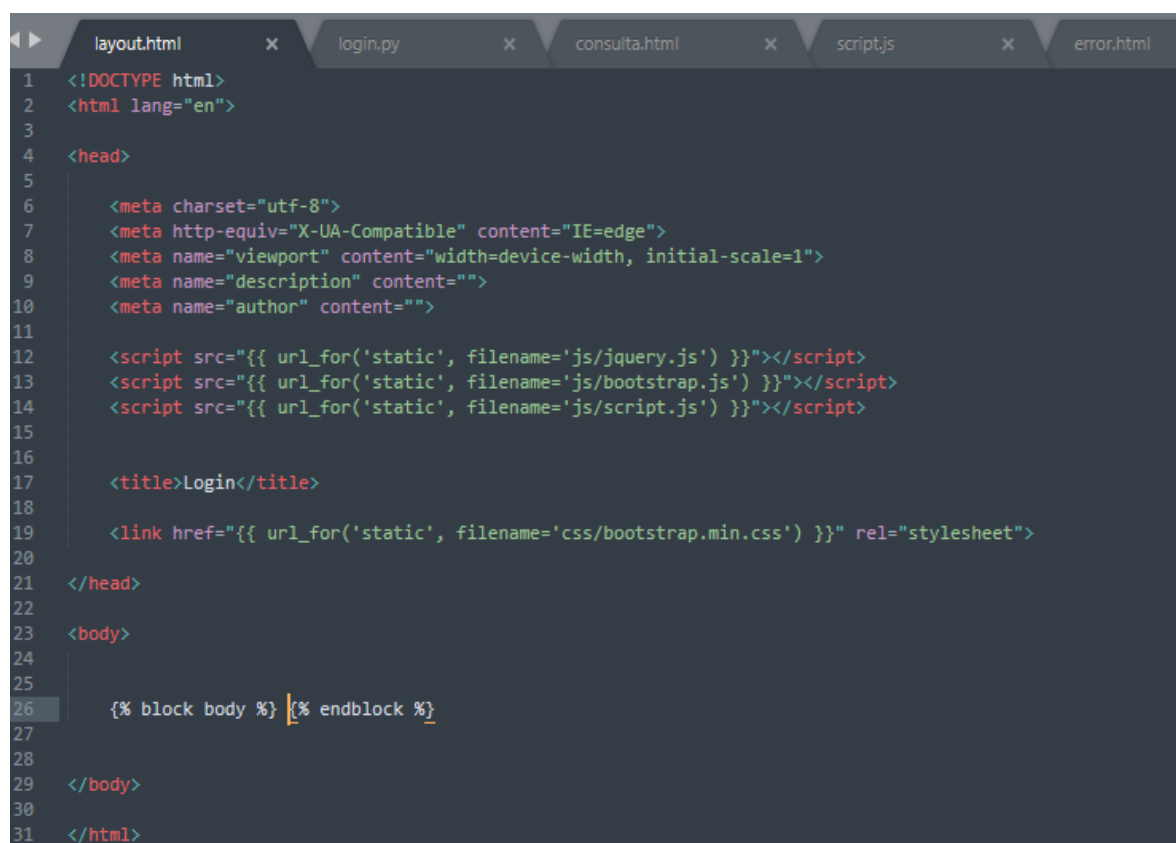
Hacemos una consulta de la colección MARVEL y que nos enseñe todos los documentos y de forma “bonita”.

```
> db.marvel.find().pretty()
{
  "_id" : ObjectId("5b0c09d0812dafddc9c71a6e"),
  "título" : "Thor 3",
  "año" : "2015",
  "actores" : {
    "principal" : "Chris Hemsworth",
    "secundario" : "Natalie Portman"
  }
}
{
  "_id" : ObjectId("5b0c0c5a812dafddc9c71a6f"),
  "título" : "Thor 3",
  "año" : "2015",
  "actores" : {
    "principal" : "Chris Hemsworth",
    "secundario" : "Natalie Portman"
  }
}
{
  "_id" : ObjectId("5b0c0c5a812dafddc9c71a70"),
  "título" : "X-men",
  "año" : "2003",
  "actores" : {
    "principal" : "Huge Jackman",
    "secundario" : "Patrick Stewart"
  }
}
{
  "_id" : ObjectId("5b0c0c5a812dafddc9c71a71"),
  "título" : "Deadpool",
  "año" : "2014",
  "actores" : {
    "principal" : "Ryan Reynolds",
    "secundario" : "Stan Lee"
  }
}
{
  "_id" : ObjectId("5b0c0c5a812dafddc9c71a72"),
  "título" : "Guardianes de la galaxia vol.1",
  "año" : "2015",
  "actores" : {
    "principal" : "Chris Pratt",
    "secundario" : "Vin Diesel"
  }
}
```

Solo pongo unos ejemplos y sobretodo para que se vea que información estará dentro de la base de datos PELÍCULAS, en la que más adelante haremos consultas directamente desde el interfaz web.

En la sección ¿Qué es MongoDB? ya explique brevemente como se podian hacer las consultas básicas CRUD de Insertar, Ver, Modificar y Borrar datos.

Vamos ahora a la parte de Flask para crear la interfaz web y puesta en marcha del proyecto, empezando por crear las páginas HTML dentro de la carpeta TEMPLATES que necesitaremos para ir navegando en las diferentes partes de la interfaz. Necesitamos pues, una página NÚCLEO para no tener que estar escribiendo ni tener tanto código HTML en todas las páginas que necesitamos. Para ello python y Flask nos ofrecen un método para reutilizar código desde una página que hará de núcleo a otras páginas que contendrán la información de la página núcleo más la propia de cada página, veamos la página núcleo llamada LAYOUT.html :

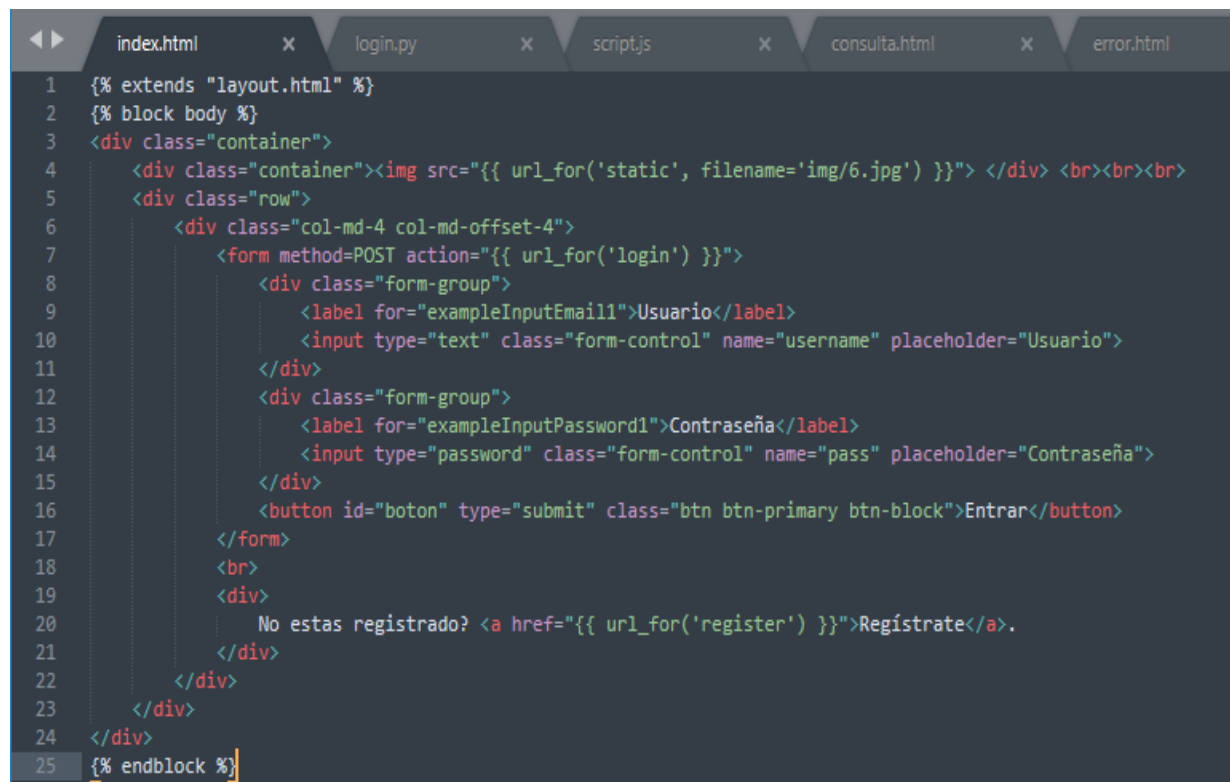


```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5
6     <meta charset="utf-8">
7     <meta http-equiv="X-UA-Compatible" content="IE=edge">
8     <meta name="viewport" content="width=device-width, initial-scale=1">
9     <meta name="description" content="">
10    <meta name="author" content="">
11
12    <script src="{{ url_for('static', filename='js/jquery.js') }}"></script>
13    <script src="{{ url_for('static', filename='js/bootstrap.js') }}"></script>
14    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
15
16
17    <title>Login</title>
18
19    <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}" rel="stylesheet">
20
21 </head>
22
23 <body>
24
25
26     {% block body %} {% endblock %}
27
28
29 </body>
30
31 </html>
```

Podemos observar que tiene la parte del entera de las etiquetas HTML o document que alberga todos los apartados del documento, el HEAD donde están los links para que se extraigan las librerías de CSS que están en su carpeta STATIC/CSS y de JavaScript que están en STATIC/JS y contiene las funciones de JQUERY y a la vez dentro de esta también está nuestro propio documento JavaScript con nuestros scripts de animación JQUERY y peticiones AJAX. Podemos ver también el TITLE o

título que sale en la pestaña al abrir la página web, el BODY que es la parte importante de la página núcleo ya que con la función BLOCK podemos ahorrarnos todo el código que hay en este documento y en las demás páginas HTML solo llamar el BLOCK para solo tener que escribir el código necesario del BODY, que explicaré ahora con las demás páginas. Al final como siempre vemos las etiquetas de cierre del documento.

Vamos a seguir con la página que será la de inicio o index del servidor y que gracias a la página núcleo solo tendremos que escribir la parte del body ya que lo demás lo cogerá de la página núcleo:



```
1 {% extends "layout.html" %}
2 {% block body %}
3 <div class="container">
4   <div class="container"> </div> <br><br><br>
5   <div class="row">
6     <div class="col-md-4 col-md-offset-4">
7       <form method=POST action="{{ url_for('login') }}">
8         <div class="form-group">
9           <label for="exampleInputEmail1">Usuario</label>
10          <input type="text" class="form-control" name="username" placeholder="Usuario">
11        </div>
12        <div class="form-group">
13          <label for="exampleInputPassword1">Contraseña</label>
14          <input type="password" class="form-control" name="pass" placeholder="Contraseña">
15        </div>
16        <button id="boton" type="submit" class="btn btn-primary btn-block">Entrar</button>
17      </form>
18      <br>
19      <div>
20        No estas registrado? <a href="{{ url_for('register') }}">Regístrate</a>.
21      </div>
22    </div>
23  </div>
24 </div>
25 {% endblock %}
```

Se puede observar la referencia a la página núcleo justo al principio de esta, se extiende el código de LAYOUT.html y se anida dentro del BODY el código que escribamos en esta página, como podemos ver dentro del bloque BODY que se abre arriba y se cierra abajo, sin duda una función de gran utilidad para reutilizar código y no tener que escribir ni repetir tanto. Se podría clonar otras partes de una página con este método pero para nuestro fin con poder enlazar el BODY ya nos es suficiente y reutilizaremos en todas las páginas HTML que tenemos para nuestro proyecto.

Podemos ver que esta página (INDEX.html) está formada por DIV que van conteniendo y clasificando las diferentes partes de la página. Los CLASS llaman a clases definidas dentro de la librería CSS que está en STATIC/CSS/bootstrap.min.css que es una librería de BOOTSTRAP para

dar formato de forma fácil a las páginas web (posición, formato y tamaño de fuente, aspecto del botón, campos de formulario...) y al estar anclado el link en la página núcleo, solo me hace falta dar el nombre apropiado a cada CLASS de los DIV para que tomen el aspecto deseado en todas las páginas que cree y llamen a la página núcleo.

Podemos ver que la página tendrá en la parte de arriba una imagen anclada que está en STATIC/IMG que es donde se alojan las imágenes. En el FORM se enviarán los datos de formulario por el método POST y se almacenarán en “login”, también se puede ver que es una página con dos LABEL/INPUT o campos de formulario, uno para el nombre de usuario y el segundo para la contraseña, justo después un botón de envío SUBMIT con el mensaje entrar y debajo un mensaje con un link que lleva a la página para registrar usuarios.

La página quedaría así:

Vamos a ver ahora la página REGISTER.html que es una página para que si un usuario no está inscrito en la base de datos y no puede acceder a la zona de clientes, poder registrarse:

```
register.html x index.html x login.py x script.js x consulta.html x
1 {% extends "layout.html" %}
2 {% block body %}
3 <div class="container">
4 <div class="container"> </div> <br><br><br>
5 <div class="row">
6 <div class="col-md-4 col-md-offset-4">
7 <form method=POST action="{{ url_for('register') }}">
8 <div class="form-group">
9 <label for="exampleInputEmail1">Usuario</label>
10 <input type="text" class="form-control" name="username" placeholder="Usuario">
11 </div>
12 <div class="form-group">
13 <label for="exampleInputPassword1">Contraseña</label>
14 <input type="password" class="form-control" name="pass" placeholder="Contraseña">
15 </div>
16 <button id="boton2" type="submit" class="btn btn-primary btn-block">Registrarse</button>
17 </form>
18 <br>
19 </div>
20 </div>
21 </div>
22 {% endblock %}
```

Se puede observar al igual que en la página INDEX.html el anclaje a la página núcleo para tanto tener anclados los estilos CSS como los scripts de JS y no tener que repetir tanto código, sino solo centrarnos en el código de la etiqueta BODY.

Esta página es muy similar a la anterior, tiene etiquetas DIV para organizar el formulario con sus CLASS que ponen estilo a cada parte o DIV, el método POST para que los datos introducidos se almacenen en “register”, 2 campos para usuario y contraseña y un botón que pone “registrar” que mandara la orden de introducir los datos a la base de datos.

La página quedaría así:



Usuario

Contraseña

Registrarse

Vamos a ver ahora la página (CONSULTA.html) donde accedera el cliente si ya esta registrado en la base de datos y donde podrá consultar los datos de los usuarios y los datos de las películas que están en MongoDB:

```
1  {% extends "layout.html" %}
2  {% block body %}
3  <div class="container">
4      <div class="container"></div> <br><br><br>
5      <div class="row">
6          <div class="col-md-4 col-md-offset-4">
7              <form method=POST action="{{ url_for('consulta') }}">
8                  <div class="form-group">
9                      <label for="exampleInputEmail1">Introduce dato sobre usuarios o sobre películas</label>
10                     <input type="text" class="form-control" name="username" placeholder="Usuario">
11                 </div>
12                 <button id="boton3" type="submit" class="btn btn-primary btn-block">Consulta gente</button>
13                 <button id="boton4" type="submit" class="btn btn-primary btn-block">Consulta pelis</button>
14             </form>
15             <br>
16             <h4>DATOS RESULTANTES:</h4>
17             <div id="datos"> </div><br>
18         </div>
19     </div>
20 </div>
21 {% endblock %}
```



También es muy similar a las anteriores páginas, con el anclaje a la página núcleo, la imagen de cabecera, el metodo de envio de datos POST que almacenará el dato del campo de formulario en “consulta” y la única diferencia es que esta solo tiene un campo para introducir texto y tiene 2 botones, uno que llamará a la información de los usuarios y otro que llamará a la información de las películas. Podemos ver un mensaje en la etiqueta H4 que gracias a JQuery estara oculto hasta que se accione alguno de los botones y un DIV con ID=”datos” vacio en donde irán los datos de la consulta al hacer CLICK en alguno de los botones, datos recogidos por AJAX y enseñados por JQUERY que explicare junto al código más adelante.

La página quedaría así:



Introduce dato sobre usuarios o sobre peliculas

Usuario

Consulta gente

Consulta pelis

También tengo 3 páginas (ERROR, ERROR1 y ERROR2.html) de error para los casos en que el usuario ya este registrado (no se puede registrar 2 usuarios con el mismo nombre), en el caso de que haya campos vacios en el formulario y en el caso de que al entrar usuario o contraseña estos sean incorrectos: (*expongo un caso ya que los demás son iguales pero con diferente imagen para cada caso*)

```
login.py x script.js x error.html x register.html x
{% extends "layout.html" %}
{% block body %}

<div class="container">
  <div id="foto">  </div>
  <h1 id="titulo">EL USUARIO YA EXISTE!</h1>
</div>

{% endblock %}
```

Se puede ver de nuevo el anclaje a la página núcleo y 2 DIV para almacenar una imagen y otro para almacenar un mensaje, estos se moverán con una animación hecha en un script JS y con funciones JQuery que estan en STATIC/JS/script.js y es el documento que contiene nuestros scripts

propios tanto de animación como de trato de información con AJAX. No puedo poner la animación pero esta página en particular al final de la animación quedaría así:



## EL USUARIO YA EXISTE!

Una vez hecha la parte de los HTML que están en la carpeta TEMPLATES vamos a crear el pilar de Flask, el archivo LOGIN.PY que pone en marcha el servidor local, aloja y conecta las diferentes páginas, conecta con la base de datos MongoDB, recibe y envía los datos que coge de los distintos formularios y los envía a la base de datos: *(pantallazos por partes para explicar cada parte del código)*

```
login.py x script.js x error.html x register.html x
1 # -*- coding: utf-8 -*-
2 from flask import Flask, render_template, url_for, request, redirect, jsonify
3 from flask_pymongo import PyMongo, MongoClient
4
```

En la cabecera del archivo se llaman las librerías, instaladas con pip anteriormente, para poder utilizar sus funciones dentro de cada “index” o ruta de nuestro proyecto en Flask. Podemos ver que desde Flask se llaman las librerías de:

- render\_template: Para llamar a la ruta una página HTML.
- url\_for: Para poner como estáticas rutas a otros documentos.
- request: Para que puedan recoger la información introducida en los campos de formulario.
- redirect: Para poder redireccionar una ruta a otra página HTML distinta para por ejemplo, si hay un error del servidor ir a una página de error 504.
- jsonify: Para devolver al POST o GET la información recogida con “request” pero en formato JSON ya que todos los lenguajes aceptan y utilizan este método para recibir y mandar la información.
- pymongo/mongoclient: Para poder establecer conexión con la base de datos MongoDB y poder hacer operaciones de envío y consulta de datos.

```

5
6  app = Flask(__name__)
7
8  client = MongoClient('mongodb://localhost:27017')
9  dbu = client.usuarios
10 dbp = client.peliculas
11

```

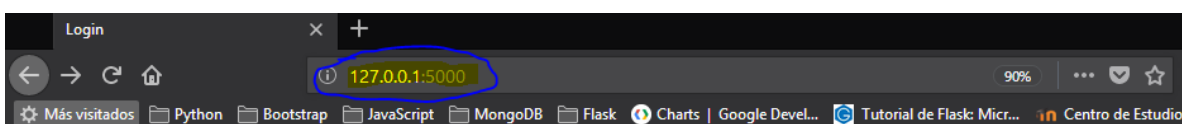
Avanzamos para ver el proximo bloque de código que en este caso “app” es común a todas las aplicaciones Flask, esta parte describe el documento entero como app que tomará los datos de aquí en adelante y al final del archivo podremos ver como llama de nuevo a “app” para poner en marcha el servidor. En la línea 8 se introduce en la variable “client” la conexión a la base de datos local de MongoDB (localhost y puerto 27017) y dentro de las variables “dbu” y “dbp” preparadas la conexión a las Bases de Datos de usuarios y de películas.

```

12
13 @app.route('/')
14 def index():
15
16     return render_template('index.html')
17

```

Seguimos con el primer “index” que creamos para la ruta raíz de nuestra “app”. En “app.route” le indicamos que en la raíz defina la función “index” que nos devuelva gracias a “render\_template” la página INDEX.html, así de sencillo es, si se pusiera en marcha Flask, se abriera un navegador y en la barra de búsqueda se introduce el servidor local sin añadir ninguna ruta extra, sino solamente la raíz, pues se muestra la página INDEX.html:



Usuario

Contraseña



No estas registrado? [Registrate.](#)

```

18
19 @app.route('/login', methods=['POST'])
20 def login():
21     users = dbu.users
22     login_user = users.find_one({'name' : request.form['username']})
23
24     if login_user:
25         if request.form['pass'] == login_user['password'] and request.form['username'] == login_user['name']:
26             return render_template('consulta.html')
27         else:
28             return render_template('error_2.html')
29
30     elif request.form['username'] == '' or request.form['pass'] == '':
31         return render_template('error_3.html')
32     else:
33         return render_template('error_2.html')
34

```

Seguimos ahora con el próximo “index” o ruta que se añadira a nuestra “app” Flask. En esta ocasión le decimos que en la ruta “/login” del servidor y con el metodo “POST” preparado, introducimos la colección “users” de la base de datos “usuarios” de MongoDB en la variable “users” y dentro de la variable “login\_user” introduciremos el resultado de la consulta “find\_one” a MongoDB donde en la colección “users” encuentra uno donde el nombre sea igual a el nombre introducido en el campo “usuario” del formulario. “request.form” de la librería “request” coge el texto introducido en el campo de formulario donde el nombre en el HTML sea el que esta entre comillas, en este caso “username”, pongo un pantallazo del HTML para que se vea de donde cogerá ese campo:

```

8
9
10
11

```

`<input type="text" class="form-control" name="username" placeholder="Usuario">`

El texto introducido en el campo de usuario será recogido por la función “request” y se utilizará para la xconsulta a MongoDB, el próximo paso es hacer comprobaciones con el resultado de esa consulta. Primer IF en que si “login\_user” es TRUE pues si este es igual a los datos de la consulta o, en otras palabras, el usuario entra los datos correctos y este está almacenado en la base de datos, pues nos devolvera la página CONSULTA.html que es la página de usuarios registrados que pueden consultar datos de usuarios y películas. Si los datos no coinciden con la base de datos saltará a una página de ERROR\_2.html, en la que se indica que el usuario o contraseña no son correctos y en el caso del ELIF, si se dejan campos de formulario vacios también se devolvera una página de ERROR\_3.html.

Vamos a ver la parte de si en caso de que un usuario no está registrado y quiere pues, al ir al link de “registro” de la página de inicio “login”, este podrá introducir sus datos para darse de alta en la base de datos:

```

37 @app.route('/register', methods=['POST','GET'])
38 def register():
39     if request.method == 'POST':
40         users = dbu.users
41         existing_user = users.find_one({'name' : request.form['username']})
42
43         if request.form['username'] == '' or request.form['pass'] == '':
44             return render_template('error_3.html')
45
46         elif existing_user is None:
47             users.insert({'name' : request.form['username'], 'password' : request.form['pass']})
48             return redirect(url_for('index'))
49
50         elif existing_user:
51             return render_template('error.html')
52
53     return render_template('register.html')
54

```

En esta ocasión, en la ruta del servidor “/register” con los metodos “GET” y “POST” a la escucha, definimos la función “register”. Como el caso anterior introducimos en variables la colección “users” de la base de datos “usuarios” y la consulta “find\_one” para poder operar con los resultados, así pues el primer IF despues de las variables es para descartar un error si se dejan campos vacios, el ELIF siguiente le decimos que si en la variable que consulta los datos de un usuario “existing\_user” y este nos devuelve “none” o ninguno pues procederemos a introducirlo a la base de datos con un “insert” con los datos recogidos por “request” y nos redireccionara automáticamente a la página de “login” para ya poder entrar los datos. Si el caso no fuera “none” y ya existiera el usuario, nos redirecciona a la página de ERROR.html con el mensaje de “usuario ya existe” y la animación JQuery.

En las dos siguientes partes de código del pilar Flask cogen desde la página en la que los usuarios ya estan logeados y pueden consultar datos CONSULTA.html la información introducida. Este código definirá las rutas que cogerán datos del formulario, se consultarán a la base de datos MongoDB y se devolveran en formato JSON para que el script de SCRIPT.js reciba los datos desde la rutas definidas en la función “app.route” y AJAX de forma asincrona devuelva los datos a la página HTML donde le digamos, cuando le digamos y en el formato que le digamos, vamos a explicar un poco el código:

```

57 @app.route('/consulta', methods=['POST'])
58 def consulta():
59     users = dbu.users
60     consulting_user = users.find_one({'name' : request.form['username']})
61
62     if request.form['username'] == '':
63         return render_template('error_3.html')
64
65     elif consulting_user:
66         usu = users.find_one({'name' : request.form['username']})
67         return jsonify({'Nombre':usu['name'], 'Contraseña':usu['password']})
68
69

```

Le decimos que en la ruta del servidor “/consulta” y con metodo “POST” a la escucha, coja los datos de la consulta que estará en la variable “consulting\_user”, en el primer IF es una comprobación de error, el ELIF le diremos que si en la variable es TRUE que haya datos, los metemos en la variable “usu” y devolvemos via “POST” y en formato JSON con “jsonify” a la ruta “/consulta” los datos de “nombre” y de “contraseña” almacenados en la base de datos.

Vamos a ver y para seguir el flujo de información, la parte del script que tomará los datos y AJAX devolverá a la página en SCRIPT.js:

```
30 $(document).ready(function() { //PONEMOS EL DOCUMENTO HTML LISTO PARA RECIBIR INSTRUCCIONES
31
32     $("h4").hide(); //LE DECIMOS QUE ESconda LA ETIQUETA H4 Y EL TEXTO QUE CONTIENE
33     $("#boton3").click(function(e) { //CUANDO SE DA CLICK AL BOTON, EN ESTE CASO DE USUARIOS, QUE TIENE DEFINIDO EL id="boton3"
34         e.preventDefault(); //SE ACTIVA LA FUNCION AJAX, preventDefault evita que el navegador pueda saltar a otra página para
35         //enseñar la informacion, queremos la informacion al momento sin recargar ni saltar a otra página
36
37         $.ajax({
38             url: "/consulta", //LA FUNCION AJAX DE JQUERY COGE LOS DATOS QUE SE HAN ALMACENADO VIA POST EN LA RUTA DEL SERVIDOR "/consulta"
39             data: $("form").serialize(), //SE RECOGEN TODOS LOS DATOS Y SE SERIALIZAN O ALMACENAN EN UN ARRAY
40             type: 'POST',
41             dataType: "json", //LE INDICAMOS QUE LOS DATOS SE DEVOLVERAN CON METODO POST Y FORMATO JASON
42             success: function(response) { //SI TIENE EXITO success LOS DATOS ESTARAN ALMACENADOS EN response
43
44                 var info = "<p> Nombre: " + response.Nombre + "</p>" ; //COMO LOS DATOS ESTAN EN FORMATO JSON TENEMOS QUE ACCEDER A CADA VALOR DE CADA CLAVE
45                 info += "<p> Password: " + response.Contraseña + "</p> <br>"; // CON LOS DOCUMENTOS JSON SE ACCEDE A ELLOS CON LA nomenclatura del punto
46                 // DONDE response ES JSON Y nombre ES LA CLAVE Y EN ELLA ESTA EL VALOR
47                 // QUE INTRODUCIMOS EN UNA ETIQUETA <p>
48                 $("#datos").css({ //QUIERO QUE SE ENSEÑE AHORA EL CONTENIDO ESCONDIDO DE H4 Y EL DIV CON id="datos" QUE ESTA VACIO
49                     'font-weight':'bold', //RECOJA LA INFORMACION DE LA VARIABLE info LA PONGAMOS GRUESA
50                     'font-size':'30px', //AUMENTE EL TAMAÑO DEL TEXTO
51                     'color':'#5882FA', //CAMBIE EL COLOR A AZULADO
52                     'text-align':'center', //CENTRE DENTRO DEL DIV EL TEXTO
53                     'border-style':'solid' //Y LE DEE AL DIV UN BORDE SOLIDO
54                 }).fadeIn().html(info); //LO ENSEÑE Y AÑADA EL CONTENIDO
55             },
56             error: function(error) { //SI EL RESULTADO DE AJAX NO FUERA success Y FUERA error DENTRO DEL DIV con id="datos" NOS ENSEÑA UN MENSAJE DE ERROR
57                 $("#datos").html("EL CAMPO DE CONSULTA ES REQUERIDO Y EL DATO DEBE EXISTIR");
58             }
59         }); //se cierra AJAX
60     }); //se cierra CLICK
61 }); //se cierra READY
62
63
64
```

He explicado con comentarios que función tiene cada línea del script. También se podrá ver con más detalle en la entrega en PDF o en el propio proyecto que adjuntaré.

Vamos a la explicación del código del archivo pilar del proyecto Flask, que mandara al script de JQuery y AJAX información que luego será mostrada en la página de consultas de la misma manera que en el apartado anterior, pero esta tratará los datos sobre películas:

```
70
71 @app.route('/consulta2', methods=['POST'])
72 def consulta2():
73     peli = dbp.marvel
74     pe = peli.find_one({'titulo' : request.form['username']})
75
76     if request.form['username'] == '':
77         return 'El campo de consulta es requerido'
78
79     elif pe:
80         return jsonify({'Titulo':pe['titulo'], 'Año':pe['año'], 'Actores':pe['actores']})
81
```

Le decimos que en la ruta del servidor “/consulta2” y con metodo “POST” a la escucha, que haga una consulta a la colección “marvel” con los datos de formulario, que mirará registro con la clave “titulo”, coja los datos de la consulta que estará en la variable “pe”, en el primer IF es una comprobación de error, el ELIF le diremos que si en la variable es TRUE que haya datos, devolvemos via “POST” y en formato JSON con “jsonify” a la ruta “/consulta2” los datos de “titulo”, “año” y “actores” almacenados en la base de datos peliculas y colección “marvel”.

Vamos a ver y para seguir el flujo de información, la parte del script que tomará los datos y AJAX devolverá a la página en SCRIPT.js:

```
66 $(document).ready(function() { //PONEMOS EL DOCUMENTO HTML LISTO PARA RECIBIR INSTRUCCIONES
67
68     $("h4").hide(); //LE DECIMOS QUE ESconda LA ETIQUETA H4 Y EL TEXTO QUE CONTIENE
69     $("#boton4").click(function(e) { //CUANDO SE DA CLICK AL BOTON , EN ESTE CASO DE PELICULAS, QUE TIENE DEFINIDO EL id="boton4"
70         e.preventDefault(); //SE ACTIVA LA FUNCION AJAX, preventDefault evita que el navegador pueda saltar a otra página para
71         //enseñar la informacion, queremos la informacion al momento sin recargar ni saltar a otra página
72
73         $.ajax({
74             url: "/consulta2", //LA FUNCION AJAX DE JQUERY COGE LOS DATOS QUE SE HAN ALMACENADO VIA POST EN LA RUTA DEL SERVIDOR "/consulta2"
75             data: $("form").serialize(), //SE RECOGEN TODOS LOS DATOS Y SE SERIALIZAN O ALMACENAN EN UN ARRAY
76             type: 'POST',
77             dataType: "json", //LE INDICAMOS QUE LOS DATOS SE DEVOLVERAN CON METODO POST Y FORMATO JSON
78             success: function(response2) { //SI TIENE EXITO success LOS DATOS ESTARAN ALMACENADOS EN response2
79
80                 var info = "<p> Titulo: " + response2.Titulo + "</p>" ; //COMO LOS DATOS ESTAN EN FORMATO JSON TENEMOS QUE ACCEDER A CADA VALOR DE CADA CLAVE
81                 info += "<p> Año: " + response2.Año + "</p>"; //CON LOS DOCUMENTOS JSON SE ACCEDE A ELLOS CON LA NOMENCLATURA DEL PUNTO
82                 info += "<p> Actor principal: <br>" + response2.Actores.principal + "</p>"; //DONDE response2 ES JSON Y actores ES EL NOMBRE DEL
83                 info += "<p> Actor secundario: <br>" + response2.Actores.secundario + "</p> <br>"; //DOCUMENTO INCRUSTADO Y EN ELLA ESTA EL VALOR
84                 // QUE INTRODUCIMOS EN UNA ETIQUETA <p>
85
86                 $("h4").show();
87                 $("#datos").css({ //QUIERO QUE SE ENSEÑE AHORA EL CONTENIDO ESCONDIDO DE H4 Y EL DIV CON id="datos" QUE ESTA VACIO
88                     'font-weight': 'bold', //RECOJA LA INFORMACION DE LA VARIABLE info LA PONGAMOS GRUESA
89                     'font-size': '30px', //AUMENTE EL TAMAÑO DEL TEXTO
90                     'color': '#5882FA', //CAMBIE EL COLOR A AZULADO
91                     'text-align': 'center', //CENTRE DENTRO DEL DIV EL TEXTO
92                     'border-style': 'solid' //Y LE DEE AL DIV UN BORDE SOLIDO
93                 }).fadeIn().html(info); //LO ENSEÑE Y AÑADA EL CONTENIDO
94             },
95             error: function(error) { //SI EL RESULTADO DE AJAX NO FUERA success Y FUERA error DENTRO DEL DIV con id="datos" NOS ENSEÑA UN MENSAJE DE ERROR
96                 $("#datos").html("EL CAMPO DE CONSULTA ES REQUERIDO Y EL DATO DEBE EXISTIR");
97             }
98         }); //se cierra AJAX
99     }); //se cierra CLICK
100
101 }); //se cierra READY
```

Vamos a comentar la ultima parte del pilar de Flask del archivo LOGIN.py:

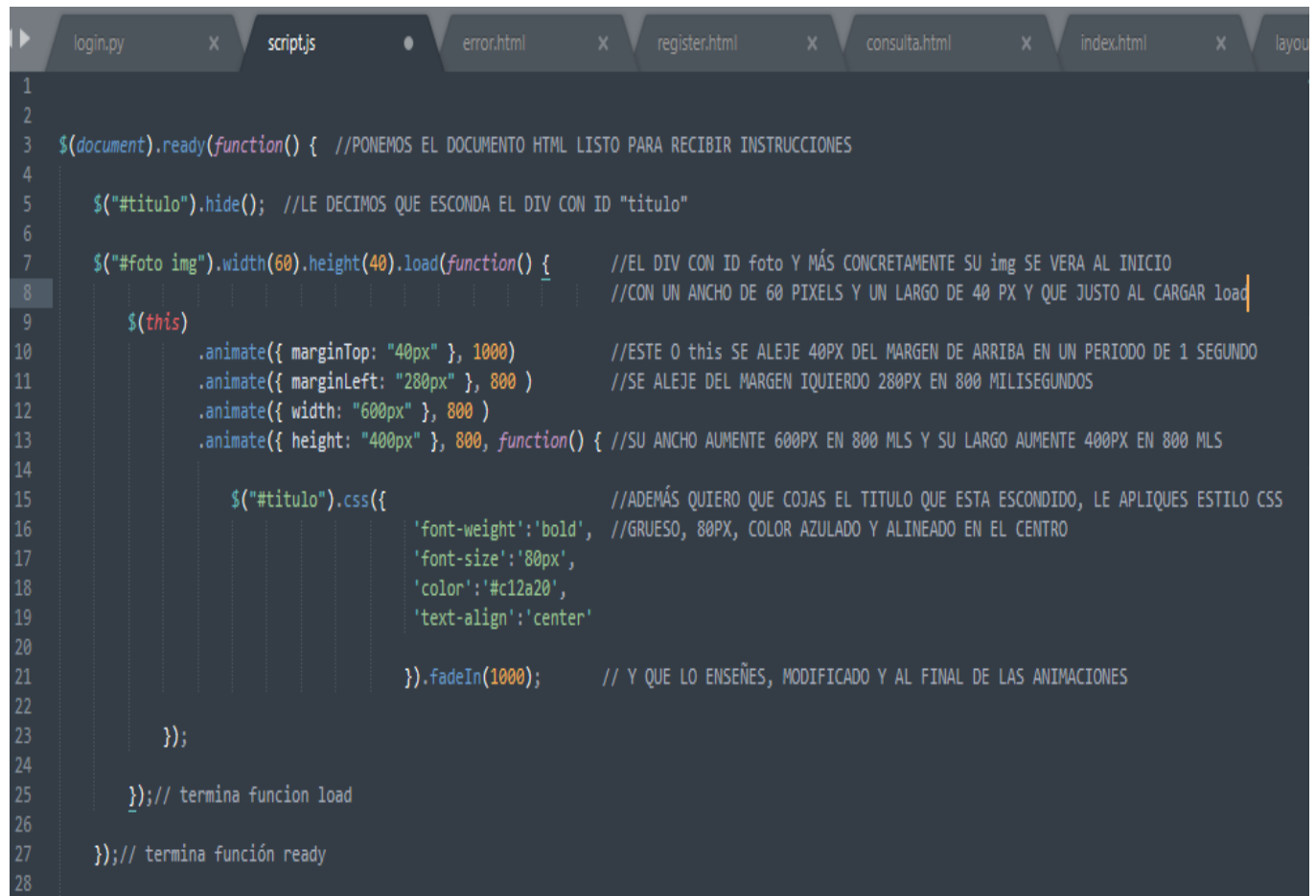
```
82
83
84 if __name__ == '__main__':
85     app.run(debug=True)
```

Simplemente al final del archivo de coge el nombre de la aplicación y se pone en marcha con “app.run”, “debug=True” es para que una vez puesto en marcha el servidor, si hiciéramos algún cambio en el archivo pilar o principal y lo guardáramos, el servidor detectaría esos cambios y



reiniciaría el servidor para hacer esos cambios efectivos, así no se tiene que andar apagando y reiniciando manualmente el servidor al hacer algún cambio en el archivo principal, muy útil.

Para terminar vamos a definir del archivo SCRIPTS.js el script de animación con funciones JQUERY y se aloja en STATIC/JS/script.js: *(un pantallazo de esa parte del archivo)*



```
1
2
3 $(document).ready(function() { //PONEMOS EL DOCUMENTO HTML LISTO PARA RECIBIR INSTRUCCIONES
4
5     $("#titulo").hide(); //LE DECIMOS QUE ESCONDA EL DIV CON ID "titulo"
6
7     $("#foto img").width(60).height(40).load(function() { //EL DIV CON ID foto Y MÁS CONCRETAMENTE SU img SE VERA AL INICIO
8         //CON UN ANCHO DE 60 PIXELS Y UN LARGO DE 40 PX Y QUE JUSTO AL CARGAR load
9         $(this)
10            .animate({ marginTop: "40px" }, 1000) //ESTE O this SE ALEJE 40PX DEL MARGEN DE ARRIBA EN UN PERIODO DE 1 SEGUNDO
11            .animate({ marginLeft: "280px" }, 800 ) //SE ALEJE DEL MARGEN IQUIERDO 280PX EN 800 MILISEGUNDOS
12            .animate({ width: "600px" }, 800 )
13            .animate({ height: "400px" }, 800, function() { //SU ANCHO AUMENTE 600PX EN 800 MLS Y SU LARGO AUMENTE 400PX EN 800 MLS
14
15                $("#titulo").css({ //ADEMÁS QUIERO QUE COJAS EL TITULO QUE ESTA ESCONDIDO, LE APLIQUES ESTILO CSS
16                    'font-weight': 'bold', //GRUESO, 80PX, COLOR AZULADO Y ALINEADO EN EL CENTRO
17                    'font-size': '80px',
18                    'color': '#c12a20',
19                    'text-align': 'center'
20                });
21                .fadeIn(1000); // Y QUE LO ENSEÑES, MODIFICADO Y AL FINAL DE LAS ANIMACIONES
22            });
23        });
24    }); // termina funcion load
25
26
27 }); // termina función ready
28
```

Ya estaría así el proyecto y el servidor listo para arrancar la aplicación para poder ver e interactuar con el interfaz.



## PRUEBAS:

Vamos a hacer un recorrido por la aplicación, para ver su estado final y sus opciones. Empezamos arrancando Flask, abrimos un terminal de CMD de Windows con la ruta puesta en nuestro proyecto y este a la vez ya activado para poder utilizar el entorno virtual (VIRTUALENV). Escribimos:

*python login.py*

```
(pymongo) C:\Users\pllull\Desktop\FLASK_1\pymongo>python login.py
* Serving Flask app "login" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 928-921-351
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Podemos ver, como habíamos especificado al final del archivo pilar del proyecto que el “debug” esta activo y que el servidor ya está arrancado en “127.0.0.1:5000”, además comentar que si en el momento de arrancar, en el archivo principal hubiese algún error de sintaxis de python, este no arrancaría y nos avisaría con un mensaje (útil para resolver problemas al programar).

Accedemos al navegador web, que en mi caso utilizo FIREFOX, ya que su consola de desarrollo web me es muy útil para ver y detectar problemas en las peticiones “GET” y “POST” y ver como se envían y reciben los datos en la aplicación:

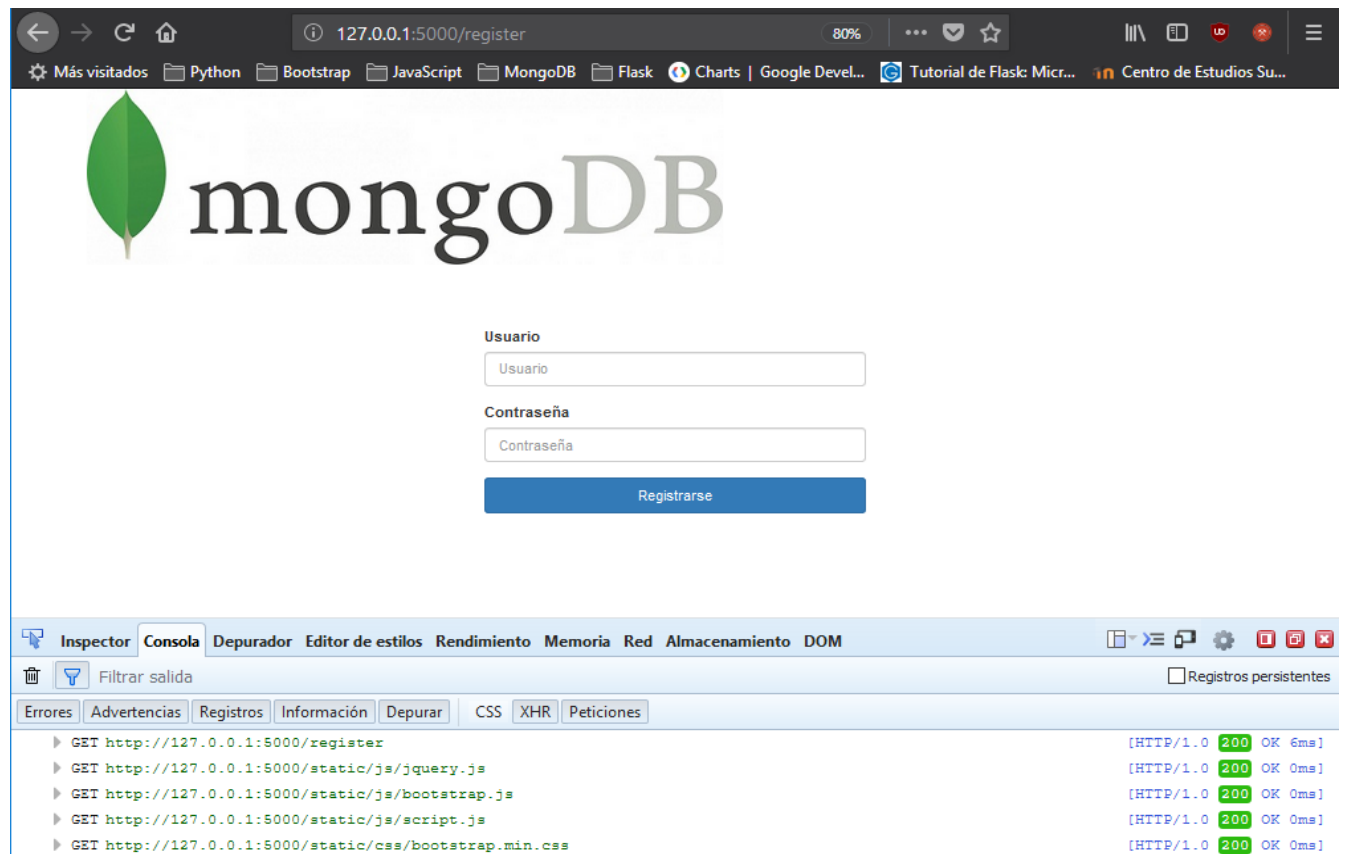
The screenshot shows a web browser window with the address bar set to `127.0.0.1:5000`. The page displays the MongoDB logo and a login form with fields for 'Usuario' and 'Contraseña', and an 'Entrar' button. Below the form is a link that says 'No estas registrado? Registrate.'.

The browser's developer console is open, showing the 'Consola' tab. It lists several GET requests to the server, all of which returned a 304 status code, indicating that the resources were not modified. The requests are:

- `GET http://127.0.0.1:5000/` [HTTP/1.0 200 OK 3ms]
- `GET http://127.0.0.1:5000/static/js/jquery.js` [HTTP/1.0 304 NOT MODIFIED 9ms]
- `GET http://127.0.0.1:5000/static/js/bootstrap.js` [HTTP/1.0 304 NOT MODIFIED 13ms]
- `GET http://127.0.0.1:5000/static/js/script.js` [HTTP/1.0 304 NOT MODIFIED 10ms]
- `GET http://127.0.0.1:5000/static/css/bootstrap.min.css` [HTTP/1.0 304 NOT MODIFIED 4ms]
- `GET http://127.0.0.1:5000/static/img/6.jpg` [HTTP/1.0 304 NOT MODIFIED 3ms]

Se puede ver que al acceder a la ruta que nos daba la consola de Flask podemos ver la página principal LOGIN.html y la consola de desarrollo con las diferentes peticiones “GET” de links a librerías e imágenes.

Si no estamos registrados y pulsamos en el link de “regístrate” accedemos a la página que nos dará de alta a la base de datos:



Es muy similar a la página de login pero tiene pequeños matices que se diferencian. Si introducimos usuario y contraseña y le damos a registrarse, los datos se almacenan en la base de datos y la página automáticamente nos redirecciona a la página de login para poder entrar a la sección de consultas.

En el siguiente pantallazo ya hemos introducido los datos de usuario y contraseña y nos ha redireccionado a la página de login otra vez. Se puede observar en la parte de la consola una petición “POST” con la información que se ha introducido y mandado a la base de datos:

The screenshot shows the MongoDB web interface in a browser. The page has a green leaf logo and the text "mongoDB". Below the logo, there are input fields for "Usuario" and "Contraseña", and a blue "Entrar" button. A link "No estas registrado? [Regístrate.](#)" is also present. The Chrome DevTools console is open, showing a POST request to "http://127.0.0.1:5000/register" with a status of 302 FOUND. The request body is visible under "Parámetros" and "Datos de formulario", showing "pass: moreno" and "username: Pepito". Below the console, a GET request to "http://127.0.0.1:5000/" is also visible.

Ahora que estamos registrados en la base de datos podemos acceder a la zona de consulta de datos de usuarios y películas, introducimos los datos correctamente:

The screenshot shows the MongoDB web interface after login. The page has the same green leaf logo and "mongoDB" text. Below the logo, there is a heading "Introduce dato sobre usuarios o sobre películas" and an input field labeled "Usuario". There are two blue buttons: "Consulta gente" and "Consulta pelis". The Chrome DevTools console is open, showing a POST request to "http://127.0.0.1:5000/login" with a status of 200 OK. The request body is visible under "Parámetros" and "Datos de formulario", showing "pass: moreno" and "username: Pepito". Below the console, a GET request to "http://127.0.0.1:5000/static/js/jquery.js" is also visible.

Ya estamos en la zona de consulta, muy similar a las zonas anteriores pero con solo un campo para introducir texto y dos botones, uno para consultar datos de usuarios y otro para consultar datos de películas. Si introducimos el nombre con el que nos hemos registrado y pulsamos el botón “consulta gente” vemos los datos que nos aparecen:

Introduce dato sobre usuarios o sobre películas

Pepito

Consulta gente

Consulta pelis

DATOS RESULTANTES:

**Nombre: Pepito**

**Password: moreno**

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento DOM

Filtrar salida

Errores Advertencias Registros Información Depurar CSS XHR Peticiones

XHR POST http://127.0.0.1:5000/consulta [HTTP/1.0 200 OK 6ms]

Cabeceras Cookies Parámetros Respuesta Tiempos Traza de la pila

Filtrar propiedades

JSON

Contraseña: **moreno**

Nombre: **Pepito**

Contenido de respuesta

```
1 {
2   "Contrase\u00f1a": "moreno",
3   "Nombre": "Pepito"
4 }
```

Se puede observar que el resultado aparece como se especifico en el script, donde, como y al pulsar el botón, también en la parte de la consola y al haber accionado el botón, podemos ver una petición “POST XHR” que es un paquete con la información solicitada, que viene desde la ruta “/consulta” del servidor Flask y son los datos devueltos desde AJAX y podemos ver en el JSON que nos devuelve.

Si introducimos una de las películas que teníamos en la base de datos de MongoDB “películas” y su colección “marvel” y dando al botón “consulta pelis”, vemos sus datos que substituyen a los que teníamos visualizando de usuarios:

Introduce dato sobre usuarios o sobre películas

X-men

Consulta gente

Consulta pelis

DATOS RESULTANTES:

**Titulo: X-men**

**Año: 2003**

**Actor principal:**  
**Huge Jackman**

**Actor secundario:**  
**Patrick Stewart**

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento DOM

Filtrar salida


Errores Advertencias Registros Información Depurar CSS XHR Peticiones

XHR POST http://127.0.0.1:5000/consulta [HTTP/1.0 200 OK 6ms]

XHR POST http://127.0.0.1:5000/consulta2 [HTTP/1.0 200 OK 3ms]

Se puede ver la información como se especificaba y otra petición en la consola, “POST XHR” que viene desde la ruta “/consulta2” del servidor Flask y son los datos devueltos desde AJAX y podemos ver en el JSON que nos devuelve.

No hemos contemplado ningún error sobre campos vacíos, datos erróneos o usuarios ya registrados, voy a poner un ejemplo (aunque la animación no se verá) de un error al introducir mal la contraseña de un usuario ya registrado que quiere netrar en la zona de consulta:



# USUARIO O CONTRASEÑA INVALIDOS!

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento DOM

Filtrar salida ☐ Registros persistentes

Errores	Advertencias	Registros	Información	Depurar	CSS	XHR	Peticiones
▶ POST http://127.0.0.1:5000/login							[HTTP/1.0 200 OK 6ms]
▶ GET http://127.0.0.1:5000/static/js/jquery.js							[HTTP/1.0 200 OK 0ms]
▶ GET http://127.0.0.1:5000/static/js/bootstrap.js							[HTTP/1.0 200 OK 0ms]
▶ GET http://127.0.0.1:5000/static/js/script.js							[HTTP/1.0 200 OK 0ms]
▶ GET http://127.0.0.1:5000/static/css/bootstrap.min.css							[HTTP/1.0 200 OK 0ms]
▶ GET http://127.0.0.1:5000/static/img/8.png							[HTTP/1.0 200 OK 0ms]

## 6-CONCLUSIONES

### ***FUTURAS MEJORAS:***

Como futura mejora creo que sería mejor y más atractivo que en la zona de consulta, el usuario pudiese directamente acceder a otras páginas, tanto de consulta como de introducción de cualquier tipo de dato en la base de datos MongoDB, para ser también más interactivo el uso desde la interfaz.

### ***OBJETIVOS ALCANZADOS:***

Con este proyecto y en mi opinión en poco tiempo he aprendido a ver las aplicaciones web de un modo más profundo, con todas las partes que lo componen. La utilización y aprendizaje de lenguajes como: Python, su sintaxis y su gran versatilidad que le dan sus librerías, y la utilización de librerías como: Bootstrap y JQuery, que en el caso de Bootstrap permiten dar formato a una página de forma muy sencilla y en el caso de JQuery la gran facilidad de código para manipular el DOM de una página web, dar vida con animaciones y la obtención y muestra de datos asíncronos con AJAX. También aprender una nueva forma y formato de Base de Datos como MongoDB, su sistema NO SQL de colecciones y documentos en formato BSON y hacer operaciones CRUD en ella.

La verdad que me parece muy interesante y muy útil el saber manejar el Front y el Back de una aplicación web, desde la apariencia como la comunicación y obtención de datos desde la base de datos.

## 7-BIBLIOGRAFÍA

### MongoDB:

- Página del manual oficial  
<https://docs.mongodb.com/manual/>
- Para saber instalar MongoDB en Windows 10  
<https://gist.github.com/AlejoJamC/b8635af765ac7495c4931403b97a0d78>
- Tutorial con buenas referencias para preparar una aplicación  
[http://www.bogotobogo.com/python/MongoDB\\_PyMongo/python\\_MongoDB\\_RESTAPI\\_with\\_Flask.php](http://www.bogotobogo.com/python/MongoDB_PyMongo/python_MongoDB_RESTAPI_with_Flask.php)

### Python:

- Magnifico profesor que explica en este tutorial de 53 capítulos las bases de Python  
<https://www.youtube.com/watch?v=G2FCfQj-9ig&index=1&list=PLU8oAIHdN5BlvPxziopYZRd55pdqFwkeS>
- Muy buen material para aprendizaje y consulta  
<http://librosweb.es/libro/python/>
- Página oficial de la documentación Python  
<https://docs.python.org/3/>

### Flask:

- Tutorial para entender y manejar Flask  
[https://www.youtube.com/playlist?list=PLf46te\\_1S1\\_KLT8\\_71ioCwmEBQlavOIC](https://www.youtube.com/playlist?list=PLf46te_1S1_KLT8_71ioCwmEBQlavOIC)
- Para saber a instalar y configurar Flask  
<https://www.solvetic.com/tutoriales/article/1417-instalar-y-configurar-flask/>
- Tutorial por fases para un proyecto con Flask  
<https://lomelisan.wordpress.com/2015/05/29/mega-tutorial-de-flask-python-1era-parte-hola-mundo/>

### jQuery y Ajax:

- Página oficial con la documentación jQuery y descargar librerías  
<http://api.jquery.com/>
- Buenas referencias sobre el uso de jQuery y Ajax  
[https://librosweb.es/libro/fundamentos\\_jquery/](https://librosweb.es/libro/fundamentos_jquery/)
- Magnifico profesor que explica en este tutorial de 78 capítulos JS, jQuery y Ajax  
<https://www.youtube.com/watch?v=m2nscBtQEIs&list=PLU8oAIHdN5BmpobVmjlIlnKIVLJ84TID>

### Bootstrap:

- Página oficial de bootstrap para ver ejemplos y descargar librerías  
<https://getbootstrap.com/>



