

Tienda en linea

Django Framework



17 de junio de 2024

Jarot Osiel Fierro Concha

Contenido

[Descargas e instalaciones Previas Backend 3](#_Toc173352402)

[Python 3](#_Toc173352403)

[Miniconda 3](#_Toc173352404)

[Instalación de Django 3](#_Toc173352405)

[Creación del entorno virtual 4](#_Toc173352406)

[Creación de un entorno virtual con Miniconda 4](#_Toc173352407)

[Creación de un entorno virtual Aislado 4](#_Toc173352408)

[Diferencias con entorno virtual aislado y con miniconda 4](#_Toc173352409)

[Creación de Proyecto Django 4](#_Toc173352410)

[Proyecto Django 4](#_Toc173352411)

[Creación de aplicaciones Django 4](#_Toc173352412)

[Añadir nuestra aplicación creada a las configuraciones del proyecto 4](#_Toc173352413)

[Realizar primera migración 5](#_Toc173352414)

[Creación de la primera página web 5](#_Toc173352415)

[Templates 5](#_Toc173352416)

[Views 6](#_Toc173352417)

[Mandar mensajes desde views 6](#_Toc173352418)

[URL 6](#_Toc173352419)

[Static 7](#_Toc173352420)

[Carpetas 7](#_Toc173352421)

[Styles 8](#_Toc173352422)

[JavaScript 8](#_Toc173352423)

[Img 8](#_Toc173352424)

[Super Usuario 8](#_Toc173352425)

[Crear login 9](#_Toc173352426)

[Archivo Base 9](#_Toc173352427)

[Creación del archivo login 9](#_Toc173352428)

[Obtener credenciales del login 11](#_Toc173352429)

[Autentificacion del login 12](#_Toc173352430)

[Authenticate 12](#_Toc173352431)

[Login 12](#_Toc173352432)

[Explicacion 13](#_Toc173352433)

[Redireccionar al Usuario 13](#_Toc173352434)

[Mensajes de autentificación 14](#_Toc173352435)

[Views código autentificación 14](#_Toc173352436)

[Login identifica que mensaje mostrar 15](#_Toc173352437)

[Mensajes por Include 15](#_Toc173352438)

[Cerrar Mensajes de Alerta 16](#_Toc173352439)

[Logout 17](#_Toc173352440)

[base 17](#_Toc173352441)

[Index 17](#_Toc173352442)

[Views 17](#_Toc173352443)

[funcion 17](#_Toc173352444)

[Mensajes de deslogeo index 18](#_Toc173352445)

[Registro de Usuarios 18](#_Toc173352446)

[URL 18](#_Toc173352447)

[Forms 19](#_Toc173352448)

[Views 20](#_Toc173352449)

[Llamar al formulario dentro del html 21](#_Toc173352450)

[Leer Credenciales de Registro 22](#_Toc173352451)

[Views 22](#_Toc173352452)

[Creación de Usuarios 23](#_Toc173352453)

[Validar Campos de Formulario 24](#_Toc173352454)

[Validar contraseña 25](#_Toc173352455)

[Método Save 26](#_Toc173352456)

[Font Awesome 27](#_Toc173352457)

[Como integrar Links 27](#_Toc173352458)

[Validar Opciones de Navbar 28](#_Toc173352459)

[Paginas de login y registro para usuarios logeados 29](#_Toc173352460)

[Aplicación de Productos 29](#_Toc173352461)

[Creación de modelo 30](#_Toc173352462)

[Como registrar un Modelo el Admin 31](#_Toc173352463)

[Listar Productos 32](#_Toc173352464)

[Productos representados por clase 34](#_Toc173352465)

[Detalles por producto representado por clase 35](#_Toc173352466)

[URLS por ID de producto 37](#_Toc173352467)

[Slug como identificador en la URL 38](#_Toc173352468)

[Nuevo identificador Slug 39](#_Toc173352469)

[Slugs Automáticos 40](#_Toc173352470)

[Primer método 40](#_Toc173352471)

[Ocultar slug automático 41](#_Toc173352472)

[Visualización de slug 41](#_Toc173352473)

[Segundo Método 42](#_Toc173352474)

[Slug Duplicados 43](#_Toc173352475)

# Descargas e instalaciones Previas Backend

## Python

* Descargar Python y añadirlo al path

## Miniconda

* Descargar Miniconda y añadirlo al path

# Instalación de Django

instalación de Django a través de la consola en la que se tiene el ambiente virtual activo

* Pip install django

Si queremos alguna versión en especifico

* Pip install django == 2.3

# Creación del entorno virtual

## Creación de un entorno virtual con Miniconda

* Conda créate -n myenv\_tienda\_online Python

Para activar el entorno virtual:

* Conda actívate myenv\_tienda\_online

## Creación de un entorno virtual Aislado

* Pip install virtualenv
* Python -m venv venv

Para activar el entorno virtual:

* Cd /”ubicación\_carpeta\_env”/Scripts
* Actívate

## Diferencias con entorno virtual aislado y con miniconda

El entorno aislado es mucho mas simple y ligero, para este tienes que instalar los paquetes que necesites para tu proyecto, además estos paquetes solo se instalarán en el entorno deseado.

El entorno con Conda es mucho más completo en términos de gestión de entornos virtuales y dependencias.

Para el entorno aislado para activarlo uno tiene que dirigirse al entorno y entrar a la carpeta Scripts, para luego escribes el comando “actívate”, y el entorno se activará

Para el entorno con Conda desde cualquier ubicación escribes el comando “ conda actívate ‘nombre\_entorno’ ”

# Creación de Proyecto Django

## Proyecto Django

Al dirigirse desde consola a la carpeta en la que se trabajará deberá escribir en consola el siguiente comando para la creación de la carpeta del proyecto Django en la que se trabajará:

* Django-admin startproject “nombre\_del\_proyecto” .

## Creación de aplicaciones Django

Esta aplicación no contendrá plantillas ya que esa parte será trabajo de react. El comando para la creación de la aplicación:

* Django\_admin startapp “nombre\_de\_la\_app”

## Añadir nuestra aplicación creada a las configuraciones del proyecto

Una vez la aplicación está creada el proyecto django no lo reconoce, es por eso que tendremos que entrar a los settings del proyecto y en la sección de “INSTALLER\_APPS =” añadimos en el mismo formato nuestra aplicación

* ‘nombre\_de\_la\_app’,

Texto

Descripción generada automáticamente

## Realizar primera migración

Una vez nuestra aplicación ya esté creada necesitamos realizar una migración en el proyecto para que se actualicen las tablas de la base de datos y estén listas para que se ocupen, y esto se hace con el siguiente comando en consola:

* Python manage.py migrate

# Creación de la primera página web

## Templates

Creamos una carpeta que se llame templates, en la que estarán todas las plantillas, es decir los archivos html, esta carpeta irá dentro de la aplicación que creamos como en nuestro ejemplo, se creó la aplicación web, y dentro encontramos la carpeta templates, y dentro de templates se crea la carpeta base, en la que irá el archivo html base

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Views

Ahora nos vamos al archivo views que es donde estarán agregadas todas las plantillas que se podrán ver, tenemos que retornar, renderizar y mandar una respuesta request, indicando la ruta en la que se encuentra nuestro archivo html, dentro de la carpeta templates

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

### Mandar mensajes desde views

También podemos mandar mensajes desde el archivo views e indicar que lo que se mostrará en pantalla será el mensaje que nosotros colocaremos

Texto

Descripción generada automáticamente

Indicando en el html que corresponde a message

Texto

Descripción generada automáticamente

*-Introduccion,’8 Cliclos y condicionales’*

## URL

Para que django reconozca donde están situadas todas las plantillas tenemos que importar el archivo views en el archivo URL y añadir la ruta de index que es la que queremos mostrar en pantalla

Texto

Descripción generada automáticamente

# Static

La carpeta static hace referencia a los archivos estáticos que se mostrarán dentro de nuestro sitio web, por ejemplo, los estilos, las imágenes, videos o archivos de ejecución de script como javascript.

Esta carpeta irá al mismo nivel que templates, y se tiene que llamar static, dentro de la carpeta creamos otra carpeta con el nombre de nuestra aplicación para llevar un mejor orden de las cosas, finalmente dentro de esta carpeta añadiremos las carpetas que necesitemos, styles,js o img.

Hay que recordar que para cargar los archivos estáticos en el archivo html es de la siguiente manera dentro del head:

Texto

Descripción generada automáticamente

## Carpetas

Imagen que contiene Texto

Descripción generada automáticamente

Para llamar a estos archivos desde el html seria de la siguiente manera:

### Styles



### JavaScript



### Img



# Super Usuario

Primero realizamos las migraciones para que las tablas se crean en la base de datos

* Python manage.py migrate

Y ahora creamos el super usuario que actuará como el administrador de nuestro sitio web, para crear super usuarios desde consola debemos ofrecer un nombre, un correo, y una contraseña que nos sepamos:

* Python manage.py createsuperuser
* User: Administrador
* E-mail: administrador@administrador.cl
* Password: ‘administrador-1’

# Crear login

## Archivo Base

Una vez se haya creado el archivo base tenemos que colocar los bloques en donde queremos que el contenido sea dinámico

Texto

Descripción generada automáticamente

## Creación del archivo login

Dentro de la aplicación web y dentro de la carpeta templates creamos la carpeta para los usuarios, esto es para llevar un orden, en esta carpeta se colocará el archivo login en el que accederán los usuarios para iniciar sesión

Interfaz de usuario gráfica

Descripción generada automáticamente

Nuestro archivo base ya no es el principal archivo que tiene que mostrarse (raíz), si no nuestro archivo index, el cual está dentro de una carpeta llamada web dentro de templates, este va a ser nuestro archivo principal, el cual tiene que heredar del archivo base, además de modificar en las URL el archivo principal.

Texto

Descripción generada automáticamente

*-Archivo url-*

Texto

Descripción generada automáticamente*-Archivo index-*

El archivo login va a heredar del archivo base, por ende, tenemos que decirle en lenguaje django que herede del archivo base.html, luego cargamos los archivo estáticos, y posteriormente abrimos los bloques indicados en el archivo base en donde irá el contenido dinámico, para así colocar el contenido que queremos mostrar.

No olvidar que para enviar estos archivos por el método post, django nos dice que tenemos que agregar csrf\_token, que es un token de seguridad contra falsificaciones de solicitudes entre sitios

Texto

Descripción generada automáticamente

## Obtener credenciales del login

Para obtener las credenciales dentro de views de la función login, creamos un condicional en el que indicamos si la petición se ha hecho por el método POST, si es así, entonces crearemos 2 variables una para el usuario y otra para la contraseña, en la que indicaremos la petición a través del método post, queremos obtener, e indicamos el name que tienen los inputs del html del login

Texto

Descripción generada automáticamente

# Autentificacion del login

Para la autentificación dentro del archivo views vamos a importar 2 funciones dentro de django.

Texto

Descripción generada automáticamente

## Authenticate

La primera sirve para buscar dentro de la base de datos si se encuentra el usuario y la contraseña, validando que estos se encuentren dentro del sistema.

## Login

La segunda es una función que permite generar una sesión, pero para esto necesitamos la petición y el usuario.

El nombre de nuestra función login fue cambiado ya que hay conflictos con respecto a la función de django llamada de la misma manera

Se Texto

Descripción generada automáticamente

### Explicacion

indicamos a través de un condicional **if** si la petición que se hizo fue a través del método POST, si eso es correcto creamos dos variables para obtener las credenciales que se pusieron en el formulario, en este caso username y password, “desde la petición. Por el método POST, obtendremos el texto dentro del input, indicando el **name** de nuestra etiqueta donde estamos escribiendo”.

Para atenticar al usuario llamamos a la función “authenticate” y le pasamos los argumentos de username y password, ahora porque se coloca el = para indicar username = username, bueno le estamos diciendo a la base de datos que busque algun “username” en tu fuente de datos que contenga la credencial obtenida anteriormente, que se guardó en username.

Posteriormente guardamos el objeto que nos devuelve y se almacena en una variable “user”

Luego se crea un condicional **if** para saber si la autentificación fue exitosa, con esto me refiero, si el objeto que devuelve “user” es de tipo None entonces nuestro no hará nada, de lo contrario entrará y se ocupará la función de django importada llamada “login”, la cual pide 2 argumentos, la petición y el usuario al que le queremos realizar la sesión

Para posteriormente con la función de django para redireccionar de página, retornamos la redirencion con el nombre que se le dio en views a la ruta a la que queremos redireccionar.

# Redireccionar al Usuario

Importamos una función importante que nos permitirá redireccionar al usuario a otra dirección.



Y dentro de la función login\_view, retornamos la redireccion mandando como argumento el name de la URL a la que queremos redireccionnar, en este caso el index.

# Mensajes de autentificación

Para poder mandar mensajes desde el servidor al cliente tenemos que importar una función de django que nos permite realizar esta comunicación



## Views código autentificación

Texto

Descripción generada automáticamente

Utilizamos message dentro de nuestro condicional para mandar un mensaje de éxito al usuario, por ende llamamos a la función message la cual contiene una función llamada “success” utilizada para los mensajes de éxito.

Para poder mandar los mensajes del servidor al cliente le tenemos que pasar 2 argumentos a nuestra función “success” que serían la petición y el mensaje.

Dentro del mensaje tenemos los corchetes que indica la posición en donde mostraremos el nombre del usuario que se llama a través del método .format el cual nos sirve para formatear una cadena de texto e insertar algún valor, en este caso de nuestra variable “user” el nombre del usuario que se está logeando, “.username”

De lo contrario se mandará un mensaje de error a través de la función error al cual le mandamos los 2 argumentos anteriores, la petición y el mensaje

## Login identifica que mensaje mostrar

Texto

Descripción generada automáticamente

En algún lugar de nuestro código donde queramos mostrar el mensaje, tenemos que condicionar si existen mensajes a través de nuestro condicional if pero en lenguaje django, luego iniciamos el bucle for para recorrer todos los mensajes que estén dentro de messages

Creamos la estructura HTML del mensaje junto con su respectivo estilo CSS.

Para no tener tanto código se condiciona dentro de las clases para definir cual es la clase que se le va a definir a la etiqueta div, ya que cada mensaje tiene un estilo diferente en este caso si un mensaje es de éxito o de error, esto lo hacemos con el condicional if dentro de la clase.

Condicionamos if con message. tags, que tags es un atributo que se refiere a que esta etiqueta va a tomar un respectivo estilo si el mensaje es igual a un mensaje de éxito, tag se utiliza para definir que tipo de mensaje es al que nos estamos refiriendo, puede ser warnning, error, success, info.

Si el mensaje es “success” ósea exitoso, la etiqueta tendrá la clase “container-alert-green”, de lo contrario tendría “container-alert-red”, siempre hay que cerrar el condicional dentro de la clase con lenguaje django, se cierra el bloque condicional if

Finalmente, dentro de las etiquetas se imprime el mensaje dentro de dos llaves, indicando que lo que queremos mostrar no es un bloque de código, si no una variable que contiene el mensaje

## Mensajes por Include

Texto

Descripción generada automáticamente

Otra forma de mostrar mensajes es incluyéndolos en el código correspondiente, en este caso a nuestro archivo login, se crea un nuevo html que va almacenar el mensaje de error o de éxito para el inicio de sesión abriendo las respectivas llaves de los condicionales.

Para posteriormente indicar en nuestro archivo html de login un include, en donde se indica la ruta del archivo que queremos mostrar en ese sitio

Texto

Descripción generada automáticamente

El maquetado del HTML puede verse distinto al presentado anteriormente, eso es debido a cambios de estilos.

## Cerrar Mensajes de Alerta

Para cerrar los mensajes de alerta utilizaremos un código en javascript local, primero que todo debemos ponerle un id a las etiquetas de tanto el botón como la etiqueta contenedor del mensaje de alerta

Texto

Descripción generada automáticamente

Es un código simple en el cual creamos una función de cierre de mensajes, en el cual indicamos el id del contenedor de las alertas, a través de un classlist indicamos un toggle que lo que hará es añadir una clase con un estilo definido por nosotros a nuestra etiqueta div, este nuevo estilo contiene un display: “none”, el cual ocultará todo el contenedor de la alerta.

Posteriormente indicamos el id del botón de cierre el cual va a tener un evento de onclick, ósea cuando sea clickeado se inicie una función no determinada la cual iniciará la primera función de close\_messages

# Logout

## base

Texto

Descripción generada automáticamente

Dentro del archivo base creamos los bloques donde queremos que se vea el icono para deslogearse, este aparecerá solo si se entra el index.html

## Index

Texto

Descripción generada automáticamente

Creamos las etiquetas correspondiente para poder darle lugar y estilo a nuestro icono, encerrado en una etiqueta de link la cual nos va a mandar a la url de logout

## Views

Para deslogearse tenemos que importar la función de logout que tiene por defecto django



### funcion

Texto

Descripción generada automáticamente

Creamos la función con del nombre de “logout\_views” le daremos este nombre ya que importamos la función de logout, la llamamos y le mandamos el parámetro de la petición “request”.

Mandamos un mensaje de sesión cerrada, este es un mensaje de éxito, para luego redirigir a login\_views

Texto

Descripción generada automáticamente

### Mensajes de deslogeo index

Realizamos lo mismo de los mensajes como en el login pero dentro de index para un mensaje de éxito, ósea que el usuario se a deslogeado con exito

Texto

Descripción generada automáticamente

# Registro de Usuarios

## URL

Registramos una nueva URL para la sección de creación de usuarios en donde la plantilla mostrará el formulario correspondiente creado por clases en la web

Texto

Descripción generada automáticamente

## Forms

Creamos el archivo que contendrá las clases para la creación de usuarios a la altura de models.py

Nuestro archivo será forms.py

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Importamos el modulo forms de django, el cual nos permite usar clases para la creacion de formularios html y validar los datos

La clase es creada llamando a los parámetros de forms.Form que es el modulo llamando a una función la cual nos permite crear formularios personalizador y darle especificaciones a este

Se crea la variable username que va a ser de tipo CharField, ósea de tipo texto, y dentro de este están las especificaciones de longitud

Se especifica que se ocupará un widget que define como se visualizará nuestro campo en el formulario html, en este caso de tipo TextInput, dentro de este se ocupa la librería “attrs” la cual es la abreviación de atributos, para indicar que este campo tendrá unos atributos adicionales en el campo, básicamente especificar los atributos que tienen las estiquetas html en el html, pero en nuestro caso de tipo clase.

Escala de tiempo

Descripción generada automáticamente

## Views

Importamos dentro de views nuestro formulario en forma de clase la cual queremos presentar dentro del html



Creamos la función y el archivo correspondiente para visualizar en el navegador nuestro registro, con su respectiva ruta.

Creamos la variable form que va a contener el objeto que devuelve la clase creada en forms, ahora en el contexto agregamos nuestra variable que apunta al objeto obtenido y le decimos que se asocie a la clave ‘form’

Esto nos permite poder llamar a la variable que contiene el objeto a través de la clave ‘form’ dentro del html, mientras se utilice la función la plantilla register.html

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Llamar al formulario dentro del html

Nuestro archivo extiende de base.html, cargamos los archivos estáticos si esque se necesitan, añadimos el bloque de código en el que cargaremos el contenido dinámico y creamos una etiqueta forms ya que al enviar solo nuestra clave forms nos renderizará el formulario sin la etiqueta forms, por ende, no podemos pasar las credenciales por el método post ni añadir el token csrf\_token para mayor seguridad, para finalmente llamar a la clave de form para renderizar nuestro formulario creado por clases.

Texto

Descripción generada automáticamente

# Leer Credenciales de Registro

## Views

Form es la creación de una instancia (rescata un objeto) del formulario “RegisterForm”, se indica si la petición fue a través del método POST las credenciales se rellenarán con los datos que el usuario ingresó, de lo contrario se creará el formulario vacío.

Se crea un condicional definiendo si la petición fue a través del método POST y además si el formulario es válido, Si este es correcto devuelve un True.

El método “is\_valid”, lo que hace es:

* Verificar si todos los campos requeridos tienen valor.
* Verificar si los valores ingresados en los campos cumplen con las reglas de validación predefinidas, como la longitud o el formato.
* Realiza validaciones personalizadas que hayas definido en el formulario.

Posteriormente creamos las variables que guardarán las credenciales, a través del atributo “cleaned\_data” que lo que hace es crear un DICCIONARIO con los datos limpios, añadiendo una clave que sería el nombre de los campos del formulario y valor que serían los datos limpios que ingresó el usuario, este permite obtener las credenciales, podemos obtener a través del metodo get los atributos que se usaron para crear el formulario de tipo clase (los nombres de los atributos de la clase “RegisterForm”).

Texto

Descripción generada automáticamente

# Creación de Usuarios

Para la creación de usuarios nos vamos a apoyar de la clase User, la cual nos va a permitir representar a los usuarios en la base de datos de nuestra aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Definimos la instancia “user” que va a guardar el objeto obtenido, llamamos a la clase User que nos permite hacer alusión a los usuarios de la base de datos, llamamos a “objects” el cual es un administrador de consultas que permite realizar consultas y operaciones dentro de la base de datos (crear, recuperar, filtrar y eliminar) en vez de hacer consultas SQL, luego llamamos al método “créate\_user” que lo que hace es recuperar las credenciales y crear un nuevo usuario dentro de la base de datos.

Condicionamos si la creación del usuario fue exitosa, ósea si se pudo concretar, si esto es así mandamos un mensaje de éxito y redirigimos al usuario a la pagina de login para que pueda iniciar sesión.

Texto

Descripción generada automáticamente

## Validar Campos de Formulario

En forms.py se crea la función “clean\_username” la cual indica a Django que se va a implementar una validación al campo username, por eso se utiliza este nombre, luego creamos una variable que va a almacenar el username, la función toma como primer argumento self, el cual nos indica que se va hacer referencia al objeto que se está creando y se usa para acceder a sus atributos y métodos.

Self.cleaned\_data.get nos recupera el valor del campo username del formulario, cleaned\_data no es más que un diccionario que contiene los datos limpios y validados del formulario.

Condicionamos en base a los usuarios de la base de datos, diciendo que les vamos a hacer una operación de filtrado, se filtra por el campo username de los usuarios con el username ingresado, para posteriormente, se llama a exists el cual nos va a indicar si la operación devuelve un resultado, si la operación devuelve al menos un resultado, significa que el usuario existe.

Raise es una palabra clave que se utiliza para generar excepciones, para posteriormente indicar donde se realizará la excepción, en este caso en el formulario indicando el tipo de excepción con el texto, el mensaje se realiza siempre que el condicional if ya indica si el usuario existe dentro de la base de datos.

Si el usuario no existe dentro de la base de datos, se retorna el valor, se realiza lo mismo con el email.

Texto

Descripción generada automáticamente

Lo anterior se pinta en el html de la siguiente manera, recordar que este archivo es el de registro,

La explicación es la siguiente: se crea un bucle for field in form el cual itera sobre los campos del formulario form, dándole estilo de la clase, field.label\_tag renderiza la etiqueta html del campo del formulario, field renderiza el capo del formulario en si (<input>), y finalmente se realiza un condicional en el cual indicamos si el campo contiene un error entonces se muestra el siguiente mensaje, con si respectivo mensaje.

Texto

Descripción generada automáticamente

## Validar contraseña

Para crear el usuario es necesario validar la contraseña, con esto se refiere a que se escriba 2 veces la misma contraseña por si el usuario se equivocó al escribir su contraseña definitiva.

Para esta sección en el archivo de forms, creamos una nueva variable password2 que almacenará la segunda contraseña, le añadimos sus atributos y su clase

Texto

Descripción generada automáticamente

Se sobre escribe el método clean, esto solo se hace si un campo depende del contenido de otro, se obtiene todos los campos del formularui con el super.clean, posteriormente se valida con un condicional, en el que obtenemos el password2, si es distinto a lo obtenido en password, se genera un error con self se utiliza el método add\_error, este método recibe 2 elementos, el primero el campo al cual le queremos añadir el error y el segundo el mensaje que queremos mostrar

Texto

Descripción generada automáticamente

## Método Save

En esta sección de delega la función de crear usuarios en views a forms.

Se imita la forma de obtener los datos en view, pero ahora los queremos obtener desde forms, desde nuestra clase, para eso creamos un nuevo método de guardar, el el cual retornamos a User, y le delegamos la función de crear un usuario, dentro de este obtenemos las credenciales que nos permitirán crear al usuario

Texto

Descripción generada automáticamente

Se reemplaza la forma de obtener los datos de view, porque ya los obtenemos desde form, e indicamos que user será igual a form con su método de seba que ya obtiene las credenciales

Texto

Descripción generada automáticamente

# Font Awesome

Para nuestro sitio web necesitamos colocar iconos, como los del login, logout, usuario, entre otros, para ello vamos a visitar una pagina que nos ofrece iconos de forma gratuita, <https://fontawesome.com/>

# Como integrar Links

Ya conocíamos una forma de hacerlo, ahora conoceremos otra forma, primero nos vamos a el html del nav, e indicamos después del load static, que la url del nombre que le dimos en urls.py (name=) se va a guardar en la variable siguiente e indicamos el nombre de la variable

Texto

Descripción generada automáticamente

Como ahora ya indicamos que llamando a la variable hacemos relación al nombre de la ruta que necesitamos, vamos a indicar en una etiqueta a de link solo la variable, y ¿cómo se llamaban a las variables en el html?, pues con doble juego de llaves {{ variable }}

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

# Validar Opciones de Navbar

En esta sección esconderemos algunas opciones de navbar que no queremos mostrar, por ejemplo, si tenemos 3 opciones inicio sesión, registro y cerrar sesión, entonces cuando el usuario esté logeado no queremos mostrar el botón de iniciarl sesión ni el de registrarse, solo el de logout, lo mismo al contrario, cuando no estemos logeados, no queremos mostrar el botón de logout, eso lo lograremos con un condicional en el que mostraremos si el usuario está validado mostraremos una cosa y si no otra, de la siguiente manera

Texto

Descripción generada automáticamente

# Paginas de login y registro para usuarios logeados

En esta sección veremos como limitar las páginas, un simple ejemplo para esto es, si el usuario está logeado, se le redirige a index, si accedemos a login a través de la url nos debería refirigir a index porque ya estamos logeado, lo mismo con registro, como estamos logeado y tenemos sesión, no debemos permitir al usuario poder entrar a estos sitios, para ello iniciamos un condicional if en views.py y decirmos, si el usuario está autenticado (tiene su sesión activa, se ha logeado) entonces lo redirigimos a index, sala principal, en caso de que el usuario no esté logeado entonces podrá acceder a las paginas de login y registro.

Texto

Descripción generada automáticamente

# Aplicación de Productos

Primero crearemos a través de la consola la aplicación en donde se manejarán los productos

Texto

Descripción generada automáticamente

Dentro de Settings vamos a registrar nuestra aplicación

Texto

Descripción generada automáticamente

## Creación de modelo

Nos vamos a models.py en la aplicación de products para crear nuestro primero modelo para los Productos del sitio web, creamos una clase llamada Product que va a heredar de models.Model, lo que quiere decir models.Model es que la clase a la que hace referencia Product se va a representar como una tabla en nuestra base de datos, y dentro de estas se definen atributos que serán la representación de columnas, en nuestro caso pondremos, un título, descripción, precio y la fecha de creación del producto.

Añadir que el créate\_at es la variable que va a guardar la fecha exacta cuando se creó, dentro de esta tiene la función de auto\_now\_add = True que nos indica que se agregará automaticamente

Captura de pantalla de un celular

Descripción generada automáticamente

Posterior a esto realizamos un makemigrations para que se generen las tablas en la base de datos, nos fijaremos que en la carpeta de migrations de la aplicación se generó un archivo que contiene los datos que creamos en el modelo

Texto

Descripción generada automáticamente

Posteriormente realizamos un migrate para que se apliquen los cambios en todo el proyecto

Texto

Descripción generada automáticamente

# Como registrar un Modelo el Admin

Nos dirigimos a la aplicación y dentro de esta nos vamos a admin para poder registrar el modelo creado en el panel de administradores

Captura de pantalla de computadora

Descripción generada automáticamente

Una vez hecho esto podemos revistar en la pagina de /admin el modelo de Product

Captura de pantalla de un celular

Descripción generada automáticamente

Al agregar un producto podemos observar que el producto no se refleja con el nombre que se le dio, para ello sobrescribiremos el método str colocando el titulo como nombre de este

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

# Listar Productos

Para mostrar los productos que tenemos en la base de datos, debemos irnos a views.py e importar nuestro modelo de Product de la aplicación producto, luego de esto indicaremos una variable que guardará estos datos, definimos que guardará todos los objetos dentro de el modelo, y luego indicar que se imprimirá en pantalla una biblioteca la cual hace referencia a products.

Texto

Descripción generada automáticamente

Para el template de index, mostraremos el titulo del producto y el precio del producto

Texto

Descripción generada automáticamente

Posteriormente en el archivo views.py indicaremos que al mostrar los elementos se muestren desde el último elemento agregado por el id

Texto

Descripción generada automáticamente

# Productos representados por clase

En esta sección lo que haremos es delegar la función de mostrar los productos desde la función de views.py a una clase creada en la aplicación de products.

Nos vamos a products e importamos el modulo ListView que contiene funciones para operar en las listas, luego importamos del models.py la clase Product

Creamos una clase que va a ser la encargada de listar todos los productos, definimos una variable que indicará la ruta del archivo que queremos mostrar, luego creamos una variable para realizar consultas en donde indicaremos que se mostrarán todos los productos registrados con un orden asendente, ósea el ultimo agregado es el primero en verse.

Para mostrar los productos tenemos que sobreescribir el metodo de get\_context\_data, el cual se encarga de mostrar el contexto de la clase al template, por eso obtenemos el contexto de la clase padre, (\*\*kwargs son elementos), definimos una llave de productos en el cual le vamos a dar los valores representantes de los productos.

Texto

Descripción generada automáticamente

Desde urls.py lo que haremos es indicar que es lo que queremos mostrar en index, por eso primero importamos del archivo views de producto, la clase ProductListView, para posteriormente utilizar el método as\_view() que nos indica que se utilizará la clase como vista dentro del template

Texto

Descripción generada automáticamente

# Detalles por producto representado por clase

Para mostrar los detalles de cada producto primero tendremos que importar la clase DetailView, el cual se encarga de tener un registro de los productos desde la base de datos, la búsqueda por producto se hará en base a el id.

Le indicaremos con que modelos vamos a trabajar, con el modelo de Product, posteriormente especificamos con que template vamos a trabajar en indicamos la ruta del archivo

Texto

Descripción generada automáticamente

Creamos la carpeta dentro de template y dentro de esta productos dentro de la aplicación product y dentro de este el archivo html de productos, indicando que extiende de base, es importante indicar que la ruta del include no está al mismo nivel que en la aplicación principal de los templates por eso cambia, indicamos a través de variables que queremos mostrar el titulo, el precio y la descripción.

Django siempre buscará los archivos html en la carpeta templates, por eso no importa en que aplicación se encuentren, siempre buscará en templates.

Captura de pantalla de computadora

Descripción generada automáticamente

## URLS por ID de producto

Creamos un archivo urls.py en la aplicación producto, para todas las rutas de los productos, luego importamos la clase del archivo view.py que es la encargada de obtener los productos como objetos. En la lista de las rutas indicamos a través de <pk> que referencia el id, llave primaria de cada producto, después de esta indicamos desde views la clase de detalles del producto y para este indicar que queremos mostrarlo como vista dentro del template, posteriormente indicamos un nombre de referencia para los templates.

Texto

Descripción generada automáticamente

Nos vamos al archivo principal de urls del proyecto, ósea la aplicación “proyecto\_tienda\_online”, en la cual importaremos la función include, para poder utilizar la ruta de los productos, añadimos una nueva ruta e incluimos la referencia del id del producto.

Texto

Descripción generada automáticamente

# Slug como identificador en la URL

En esta sección lo que queremos lograr es que en la url no se vea el ID del producto, si no que queremos un representante del ID, que es lo que se denomina como slug.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para ello tendremos que modificar nuestra Clase de Producto situada en models, para añadir esta nueva variable a los atributos de la clase, indicamos el nombre de la variable, el tipo de valor, y dentro de este indicamos los siguientes parámetros, el valor no puede ser nulo, no puede estar en blanco y tiene que ser un valor único.

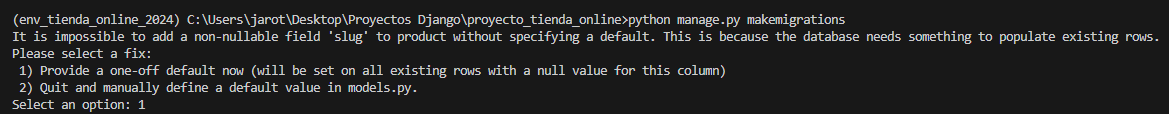
Captura de pantalla de un celular

Descripción generada automáticamente

Posterior a esto debemos realizar un makemigrations, pero sin antes saber que al realizarlo, si tenemos productos en la base de datos, debemos darle valores a través de la consola, como no llevamos tantos productos lo que haremos que es más fácil es entrar como administrador y borrar los productos creados.

Cualquier problema o error que salga debemos comentar la variable creada recientemente (SLUG) antes de entrar como administrador a la pagina, para luego de borrar los productos, guardar la modificación de la variable que estamos creando.

Realizamos el makemigrations y seleccionamos la primera opción (1)



Nos pide colocar un valor, por lo que pondremos una string vacia, luego realizamos una migración para la creación de la columna en la base de datos.

Texto

Descripción generada automáticamente

## Nuevo identificador Slug

Ahora nos dirigimos al archivo urls.py y modificamos la dirección de la url por la cual va a identificar al producto el cual seria primero definiendo el nombre de la variable (slug) : y el tipo de dato (slug), con esto logramos que la búsqueda sea por el slug

Texto

Descripción generada automáticamente

Lo que resulta como búsqueda lo siguiente (la url)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Slugs Automáticos

## Primer método

Para ello nos dirigimos a la clase de Product en nuestro archivo models.py y procedemos a importar la función de slugify para posterior, sobrescribir la función save, indicamos el self, junto con los argumentos, y el diccionario de argumentos, luego generamos el slug automático gracias a la ayuda de una función de django llamada slugify a partir del titulo, luego ejecutamos el método sabe de la clase padre, pasando como argumento los argumentos.

Captura de pantalla de computadora

Descripción generada automáticamente

### Ocultar slug automático

Nos dirigimos a el archivo de admin.py de la aplicación product ya que en el panel de administrador nos muestra todos los campos, cuando el administrador crea un producto el slug será un valor automático, por lo cual no queremos que este sea modificado a la hora de creación de producto.

Crearemos una clase llamada ProductAdmin en la que modificaremos el comportamiento del producto en el administrador, la clase heredará de admin.ModelAdmin, indicamos a través de fiel cuales son los atributos que queremos mostrar en el panel a traves de una tupla, para finalmente registrarla

Texto, Aplicación

Descripción generada automáticamente

### Visualización de slug

Una vez creado el slug, queremos visualizar el valor resultante automatico de la creación del producto en el administrador, para ello vamos a indicar en el archivo admin a través de un atributo list\_display que son los que queremos que se muestren en formato de tupla, en nuestro caso el método \_\_str\_\_ , el slug y la fecha de creación.

Captura de pantalla de computadora

Descripción generada automáticamente

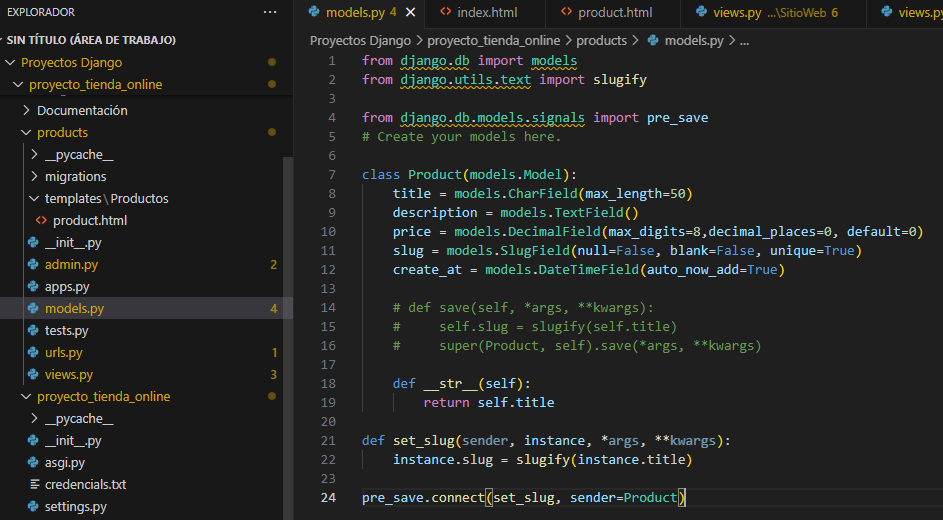
## Segundo Método

En este segundo metodo es el que se utilizará para el proyecto utilizaremos CALLBACKS que nos permiten realizar acciones (CRUD) antes o después de crear los productos, para Django se llama SIGNALS.

Generamos una nueva función llamada set\_slug el cual será el callback, encargado de generar el slug, se utilizará el signal pre\_save para ellos hay que importar pre\_save, dentro de este indicaremos que el callback recibirá 2 argumentos, sender e instance, y luego los argumentos y el diccionario de argumentos, la instancia es el producto que vamos a almacenar.

Por ellos utilizamos instance.slug que seria lo mismo que en el metodo anterior decir “ self.slug = slugify(self.title) “

Por ultimo llamamos a la función pre\_save en la que conectamos la función set\_slug con que modelo está relacionado en nuestro caso product



# Slug Duplicados

El atributo slug es un campo único, por lo que si se agrega algún producto que tenga el mismo identificados de slug que otro producto, el servidor nos arrojará un error, para ello haremos agregaremos validaciones en el CALLBACK

Se preguntará si el objeto posee un título y además no posee un slug, si esto es verdadero se generará un slug, pero mientras el slug filtrado existe por lo menos uno, se debe generar un nuevo slug, solo que tendrá un formato diferente, en el que el primer valor seguirá siendo el título, y el segundo será una cadena de caracteres pseudoaleatorio función que nos brinda la librería uuid, se ejecutará el metodo uuid el cual retornará como resultado un objeto, este hay que pasarlo a string, y por ultimo definimos de cuantos dígitos queremos que sea el uuid

Texto

Descripción generada automáticamente

# Imágenes de Productos

Lo primero que debemos hacer es instalar la librería Pillow, el cual se instala con el comando pip install Pillow.

Texto

Descripción generada automáticamente

Posterior a esto, verificamos si tenemos productos en la base de datos a través del panel de administrador, en caso de que si tengamos procedemos a eliminarlos para no tener conflictos, ya que vamos a agregar un nuevo atributo a la clase.

Nos dirigimos a nuestra clase de Product en el archivo models.py para agregar un nuevo atributo

En el cual indicamos que las imágenes se guarddarán en la carpeta Media dentro de esta en products.

Una captura de pantalla de una computadora

Descripción generada automáticamente

Realizamos un makemigrations y luego una migración para que se aplique.

Texto

Descripción generada automáticamente

Creamos la carpeta en donde se almacenarán las imágenes de los productos a la altura de las aplicaciones del proyecto.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Configuración de URL de media

Nos dirigimos a settings y agregamos 2 constantes

Texto

Descripción generada automáticamente

Ahora nos dirigimos a urls.py del proyecto, e importaremos 2 modulos, static y settings.

Condicionamos sobre settings, si estamos en proceso de desarrollo, entonces vamos a extender el listado de las direcciones, agregando las direcciones de las imagenes

Texto

Descripción generada automáticamente

## Visualizar el campo Image

Nos dirigimos al archivo admin de products y agregamos a la tupla el campo de image

Texto

Descripción generada automáticamente

Para mostrar el template nos ayudaremos del atributo url {{ producto.image.url }} Modificamos un poco el template para que la visualización de los datos sea mejor

Texto

Descripción generada automáticamente

El resultado es el siguiente:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente