

# Tienda Online

**Programacion Web 2**

Alumno

**Kevin Andree Llacma Quispe**

Profesor

**Richart Smith Escobedo Quispe**



# Sistema

El sistema desarrollado es una aplicación web para la gestión y visualización de productos, integrada mediante un backend en **Django** y un frontend en **React**.





```

14 class User(Time):
15
16     username = models.CharField(max_length=50, unique=True)
17     email = models.EmailField(unique=True)
18     password = models.CharField(max_length=100)
19
20     def __str__(self):
21         return self.username
22
23 You, 3 weeks ago | 1 author (You)
24 class Category(Time):
25     name = models.CharField(max_length=100)
26     description = models.TextField(blank=True)
27
28     def __str__(self):
29         return self.name
30
31 You, 3 weeks ago | 1 author (You)
32 class Product(Time):
33     name = models.CharField(max_length=100)
34     description = models.TextField()
35     price = models.DecimalField(max_digits=10, decimal_places=2)
36     stock_quantity = models.PositiveIntegerField()
37
38     def __str__(self):
39         return self.name

```

# Modelo de Datos

El modelo de datos se implementa utilizando el **ORM** de Django. Los principales modelos de datos incluyen:

- Producto:** Representa los productos en la base de datos con atributos como nombre, descripción, precio, stock .

# Backend

```
You, 3 weeks ago | 1 author (You)  
1 web: gunicorn tienda_online.wsgi --log-file -
```

El backend está desarrollado en **Django** y utiliza Django Rest Framework para crear una API REST. Algunas configuraciones importantes incluyen:

- Configuración de la base de datos: SQLite para simplificar el despliegue y desarrollo.
- Configuración de archivos estáticos: Se usa whitenoise para servir archivos estáticos en producción.
- Configuración de seguridad: Uso de HTTPS en el despliegue de producción para asegurar la comunicación de datos.



# API Rest

La API RESTful implementada con Django Rest Framework proporciona endpoints para la gestión de productos. Las principales operaciones que soporta la API son:

- GET: Recuperar la lista de productos o un producto específico.
- POST: Crear un nuevo producto.
- PUT/PATCH: Actualizar un producto existente.
- DELETE: Eliminar un producto





```

import React, { useState } from 'react';

const CreateProduct = () => {
  const [name, setName] = useState('');
  const [price, setPrice] = useState('');
  const [success, setSuccess] = useState(false);
  const [error, setError] = useState(null);

  const handleSubmit = (event) => {
    event.preventDefault();

    fetch('/api/products/', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ name, price }),
    })
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then(data => {
        setSuccess(true);
        setName('');
        setPrice('');
      })
      .catch(error => {
        console.error('Hubo un problema con la operacion fetch :', error);
        setError(error);
      });
  };


  return (

```


# FrontEnd


El frontend desarrollado en React incluye componentes clave como: ProductList: Muestra una lista de productos. ProductDetail: Muestra los detalles de un producto específico.


## Mis acciones

 ShoppingCart object (1)  
Shopping cart


 juan  
User


 0d1285972426684992205cdc996e...  
Token

 ivan  
Usuario

 b1b17675b10e6bb2da0f64fcc7825...  
Token

 miguel  
Usuario

 5a625e057c1827ea30158582859cf...  
Token

 ShoppingCart object (1)  
Shopping cart

 OrderDetail object (1)

# Hosting-Dominio HTTPS

El proyecto está desplegado en **Heroku**, una plataforma PaaS que facilita el despliegue de aplicaciones web. Heroku proporciona automáticamente un dominio con soporte **HTTPS**, lo que garantiza que toda la comunicación con el servidor esté cifrada.

VPN



tiendaonline-9c83954f4ca8.herokuapp.com/admin/

[Open app](#)[More](#)

Latest activity

[All Activity](#)

killacma@unsa.edu.pe: Deployed `299583b3`  
Today at 11:45 AM · v36



killacma@unsa.edu.pe: Build succeeded  
Today at 11:44 AM · [View build log](#)



killacma@unsa.edu.pe: Remove `DB_USER` config var  
Today at 11:22 AM · v35



killacma@unsa.edu.pe: Remove `DB_PORT` config var  
Today at 11:22 AM · v34



killacma@unsa.edu.pe: Remove `DB_PASSWORD` config var  
Today at 11:22 AM · v33



killacma@unsa.edu.pe: Remove `DB_NAME` config var  
Today at 11:22 AM · v32



killacma@unsa.edu.pe: Remove `DB_HOST` config var  
Today at 11:21 AM · v31

# NGINX-GUNICORN-DJANGO

Para el despliegue de la aplicación en **Heroku**, se utiliza **Gunicorn** como servidor **WSGI** para Django. Aunque no se usa NGINX en este despliegue específico, NGINX se suele emplear como proxy inverso en implementaciones más complejas, actuando como intermediario entre el cliente y Gunicorn para manejar solicitudes **HTTP** y servir archivos estáticos.



# Recomendaciones



Este trabajo tiene mucho rango de mejora, ya que no se exploró de manera muy profunda el desarrollo de estos. Pero satisfecho con el resultado, propongo las siguientes mejoras: para mejorar los tiempos de carga

- A. Migrar a una base de datos más robusta como PostgreSQL o MySQL para manejar mayores volúmenes de datos.
- B. Implementar autenticación y autorización para proteger los datos y las operaciones del sistema.



# Conclusión

El proyecto permitió explorar la integración de tecnologías frontend y backend, demostrando la eficacia de Django y React en la construcción de una aplicación. El despliegue en Heroku proporcionó una plataforma accesible, aunque se identificaron áreas para mejorar, especialmente en términos de seguridad y rendimiento.

