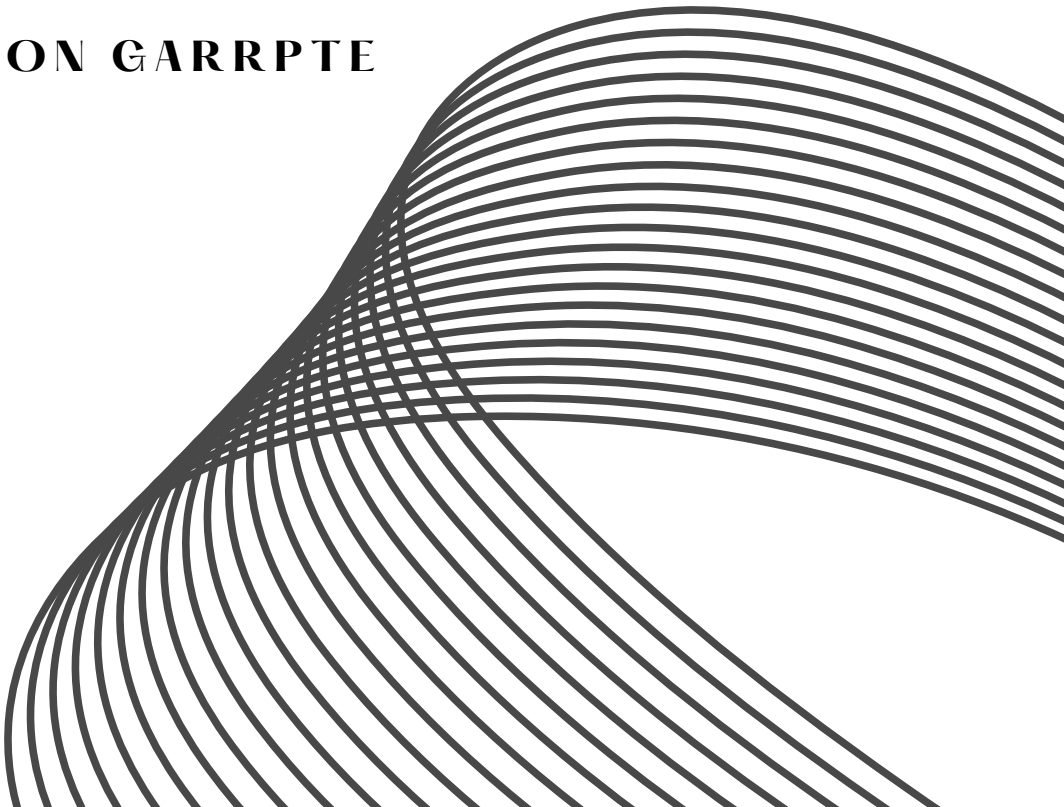
18/12/2024

Documentación Terminal Puto de Venta

REALIZADO POR
JAVIER CALDERON GARRPTE



Abstract

Este proyecto consiste en el desarrollo de un terminal de punto de venta (TPV) diseñado para pequeñas y medianas empresas (PyMEs), enfocado en optimizar los procesos de venta y la gestión comercial mediante una solución tecnológica moderna y accesible. La implementación utiliza Django con Python para el backend, ofreciendo una infraestructura robusta y segura para las operaciones del sistema. En el frontend, se emplearon HTML, CSS y Bootstrap para desarrollar una interfaz intuitiva, responsiva y adaptable a dispositivos móviles y de escritorio. SQLite fue seleccionado como base de datos por su ligereza y eficiencia, ideal para proyectos de este alcance.

El sistema está estructurado en dos roles principales: administrador y vendedor, cada uno con funciones específicas que garantizan una adecuada gestión de usuarios y operaciones. Los administradores tienen acceso a una amplia gama de funcionalidades, incluyendo la capacidad de realizar ventas, gestionar servicios mediante operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para productos, categorías y usuarios, además de visualizar los detalles de ventas y editar su perfil personal. Por su parte, los vendedores cuentan con acceso a funcionalidades esenciales como la realización de ventas, la gestión de servicios mediante operaciones CRUD y la edición de su perfil, asegurando un enfoque simplificado y eficiente para sus tareas diarias.

Este TPV también prioriza la usabilidad y el rendimiento, ofreciendo una experiencia de usuario optimizada que reduce los tiempos operativos y facilita la navegación. Al integrar estas funcionalidades, el proyecto no solo simplifica los procesos comerciales, sino que también promueve la productividad y la eficiencia en entornos de pequeñas y medianas empresas, posicionándose como una herramienta clave para la mejora competitiva en el mercado actual.

Indice

1. Justificación del Proyecto

El desarrollo de este terminal de punto de venta (TPV) responde a la necesidad de la empresa familiar de optimizar el control y registro de sus ventas diarias. Actualmente, la empresa enfrenta retos comunes en las pequeñas y medianas empresas (PyMEs), como la dificultad para gestionar las operaciones comerciales de manera eficiente, especialmente en lo que respecta al seguimiento de las ventas y el cálculo de las cifras diarias de la caja. Estas tareas, cuando se realizan de forma manual o con sistemas no integrados, tienden a generar errores, pérdida de tiempo y falta de información precisa para la toma de decisiones estratégicas.

El TPV se plantea como una solución tecnológica diseñada específicamente para la empresa, permitiendo automatizar los procesos relacionados con las ventas y la gestión de servicios. Esta herramienta ayudará a los administradores y vendedores a realizar un seguimiento más detallado de las transacciones, con un sistema adaptado a las necesidades específicas del negocio, mejorando así su eficiencia operativa y productividad.

Motivaciones Específicas

- Control de Ventas Diarias:
  Una de las motivaciones principales de este proyecto es facilitar el seguimiento de las ventas realizadas en un período de tiempo determinado, como el cálculo de la caja diaria. Este sistema permitirá registrar cada transacción de manera automática, consolidar la información de todas las ventas realizadas y generar reportes detallados que muestren las cifras finales del día.
- Gestión Eficiente de Servicios:
  El proyecto incluye funcionalidades para gestionar los servicios ofrecidos por la empresa, permitiendo a los usuarios realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre estos servicios. Esto garantiza que los administradores y vendedores puedan mantener un control actualizado y centralizado de las operaciones realizadas.
- Organización de Roles y Responsabilidades:
  La empresa cuenta con una estructura organizativa que requiere una gestión clara de las responsabilidades de los usuarios. En este sentido, el TPV diferencia entre dos roles principales:

- Administradores: Con acceso a funcionalidades avanzadas, los administradores podrán realizar ventas, gestionar servicios mediante operaciones CRUD, consultar y analizar los detalles de las ventas diarias y editar su perfil personal.
- Vendedores: Este rol está diseñado para simplificar las tareas de los empleados responsables de la atención al cliente. Los vendedores podrán realizar ventas, gestionar servicios básicos y editar su perfil, enfocándose exclusivamente en las actividades comerciales del día.

Impacto en la Productividad y Competitividad

La implementación de este TPV permitirá a la empresa familiar optimizar sus procesos comerciales, reduciendo los tiempos necesarios para registrar las ventas y calcular la caja diaria. Al proporcionar una herramienta centralizada y automatizada, se elimina la necesidad de registros manuales, minimizando errores y garantizando la precisión de la información financiera diaria.

Además, la mejora en la gestión de servicios y ventas permite a la empresa ofrecer un mejor servicio a sus clientes, asegurando que las operaciones se realicen de manera fluida y profesional. Esto contribuye directamente a mejorar la percepción del cliente sobre la empresa, lo que resulta en un aumento de la satisfacción y lealtad del cliente.

En términos de competitividad, el uso de este TPV coloca a la empresa en una posición más favorable frente a sus competidores al incorporar tecnología moderna que respalda la eficiencia y profesionalismo en sus operaciones comerciales.

Sostenibilidad y Crecimiento a Largo Plazo

La digitalización de procesos clave, como el seguimiento de ventas y la gestión de servicios, prepara a la empresa para enfrentar los desafíos del futuro con una base tecnológica sólida. Este TPV ha sido diseñado para ser escalable, permitiendo la integración de nuevas funcionalidades según las necesidades futuras del negocio. Además, fomenta una cultura de innovación dentro de la empresa, promoviendo el uso de herramientas modernas para resolver problemas operativos.

El impacto de este sistema no solo se limita a la operatividad diaria, sino que también contribuye a la sostenibilidad del negocio a largo plazo. Al automatizar los procesos y

proporcionar información precisa en tiempo real, el TPV facilita la planificación estratégica y asegura la continuidad del negocio en un mercado competitivo.

---

2. Introducción

En la actualidad, el uso de tecnologías avanzadas en el desarrollo de software es crucial para mejorar la eficiencia operativa en diferentes sectores. El diseño de sistemas como los terminales de punto de venta (TPV) se ha visto beneficiado por el uso de herramientas robustas que permiten a las empresas optimizar procesos clave, como el control de ventas y la gestión de servicios. Este proyecto se basa en una serie de tecnologías modernas que garantizan la funcionalidad, escalabilidad y seguridad del sistema desarrollado.

Tecnologías Utilizadas

El terminal de punto de venta desarrollado en este proyecto hace uso de diversas tecnologías ampliamente utilizadas en el desarrollo de aplicaciones web. A continuación, se detallan las herramientas clave que componen la arquitectura del sistema.

Django (Python)

Django es un framework de desarrollo web basado en Python, que se destaca por su enfoque en la simplicidad, la rapidez y la seguridad. Utilizado en el backend de este sistema, Django permite desarrollar aplicaciones robustas y escalables con una estructura modular y fácil de mantener. Con su potente sistema de administración, es posible gestionar de manera eficiente los registros de ventas y servicios, al tiempo que facilita la implementación de funcionalidades avanzadas como la autenticación de usuarios y la protección contra vulnerabilidades comunes.

La arquitectura de Django facilita la separación de la lógica de negocio del frontend, lo que asegura una mayor flexibilidad para el desarrollo de nuevas características. Además, su integración con bases de datos, como SQLite, permite gestionar grandes volúmenes de información de manera efectiva y eficiente.

HTML, CSS y Bootstrap (Frontend)

Para el desarrollo del frontend, se ha utilizado HTML, CSS y Bootstrap, tecnologías clave en la creación de interfaces web interactivas y atractivas. HTML proporciona la estructura básica de la página, mientras que CSS se encarga del diseño y la presentación visual. La combinación de ambos permite crear una interfaz clara y fácil de usar.

Bootstrap, un framework de diseño front-end, ofrece un conjunto de componentes prediseñados y una estructura flexible que hace que la aplicación sea completamente responsiva y accesible desde cualquier dispositivo, ya sea un ordenador o un móvil. Esto es especialmente importante en el contexto de un TPV, ya que los vendedores y administradores necesitan una interfaz rápida y cómoda para operar en tiempo real.

SQLite (Base de Datos)

SQLite es una base de datos ligera y eficiente que se utiliza para almacenar la información de las transacciones, usuarios y servicios del sistema. Al ser una base de datos embebida, no requiere un servidor de base de datos externo, lo que simplifica su instalación y mantenimiento, especialmente en entornos de pequeña escala o para empresas que no disponen de una infraestructura de servidores compleja.

SQLite es ideal para este proyecto debido a su capacidad de manejar grandes volúmenes de datos sin comprometer el rendimiento. Permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) de manera eficiente, lo cual es esencial para gestionar las ventas diarias y los servicios que ofrece la empresa.

Beneficios de la Tecnología Implementada

La elección de estas tecnologías no solo garantiza la eficiencia y escalabilidad del sistema, sino que también permite una fácil integración y mantenimiento a largo plazo. Django proporciona una base sólida para manejar grandes volúmenes de datos y procesos de negocio complejos, mientras que HTML, CSS y Bootstrap aseguran una experiencia de usuario amigable y accesible. SQLite, por su parte, ofrece una solución eficiente y fácil de usar para el almacenamiento de datos, lo que contribuye a la agilidad del sistema.

3. Objetivos

El objetivo principal de este proyecto es diseñar y desarrollar un sistema de terminal de punto de venta (TPV) optimizado, que permita a las pequeñas empresas gestionar de manera eficiente sus ventas y servicios, mejorando la productividad, reduciendo errores y facilitando el seguimiento de las operaciones comerciales. Este sistema, basado en tecnologías modernas y robustas como Django, HTML, CSS, Bootstrap y SQLite, se orienta a optimizar los procesos internos de ventas y gestión de servicios, adaptándose a las necesidades específicas de las pequeñas empresas.

Objetivo General

- Desarrollar un sistema de terminal de punto de venta (TPV) eficiente y accesible para pequeñas empresas, que facilite la gestión automatizada de ventas y servicios, garantizando una experiencia de usuario intuitiva y mejorando la eficiencia operativa mediante el uso de tecnologías como Django, HTML, CSS, Bootstrap y SQLite.

Objetivos Específicos

1. **Automatizar el registro y cálculo de las ventas diarias**
   Desarrollar una funcionalidad que permita automatizar el proceso de registro de ventas, calculando de forma precisa la caja diaria. Esto reducirá el tiempo invertido en tareas manuales y minimizará los errores humanos en la recopilación de datos, garantizando un control preciso y eficiente de las transacciones comerciales.
2. **Implementar un sistema de gestión de servicios**
   Crear un sistema que permita a los administradores y vendedores gestionar los servicios ofrecidos por la empresa, utilizando operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Esto asegurará que los servicios estén siempre actualizados y bajo control, optimizando su seguimiento y mejorando la toma de decisiones comerciales.
3. **Diseñar una interfaz de usuario sencilla y accesible**
   Desarrollar una interfaz intuitiva y visualmente atractiva para los usuarios del sistema, utilizando tecnologías como HTML, CSS y Bootstrap. Esta interfaz debe ser

fácil de navegar tanto para administradores como para vendedores, permitiendo que interactúen con el sistema de manera eficiente sin requerir conocimientos técnicos avanzados.

4. Crear un sistema de autenticación y roles de usuario diferenciados
   Implementar un sistema de autenticación robusto que permita gestionar el acceso al sistema según el rol del usuario. Los administradores tendrán acceso a funciones avanzadas, como la gestión de ventas y servicios, mientras que los vendedores tendrán acceso limitado a las funcionalidades necesarias para realizar ventas y gestionar servicios básicos.
5. Garantizar la escalabilidad y eficiencia del sistema de bases de datos
   Utilizar SQLite como base de datos ligera y eficiente para gestionar los datos relacionados con las ventas, servicios y usuarios. La base de datos debe ser escalable y capaz de manejar el crecimiento de las pequeñas empresas, permitiendo agregar nuevos registros sin afectar el rendimiento del sistema.
6. Facilitar la toma de decisiones mediante la generación de reportes detallados
   Incorporar una funcionalidad que permita generar reportes detallados sobre las ventas diarias, proporcionando a los administradores información precisa sobre el desempeño de la empresa. Estos reportes facilitarán la toma de decisiones estratégicas y la evaluación del rendimiento comercial.
7. Mejorar la eficiencia operativa y competitividad de las pequeñas empresas
   Implementar un sistema que ayude a las pequeñas empresas a optimizar sus procesos internos, reduciendo el tiempo dedicado a tareas administrativas y mejorando la precisión en la gestión de ventas y servicios. Este sistema también permitirá a las pequeñas empresas mantenerse competitivas al adoptar una solución tecnológica moderna y eficiente.

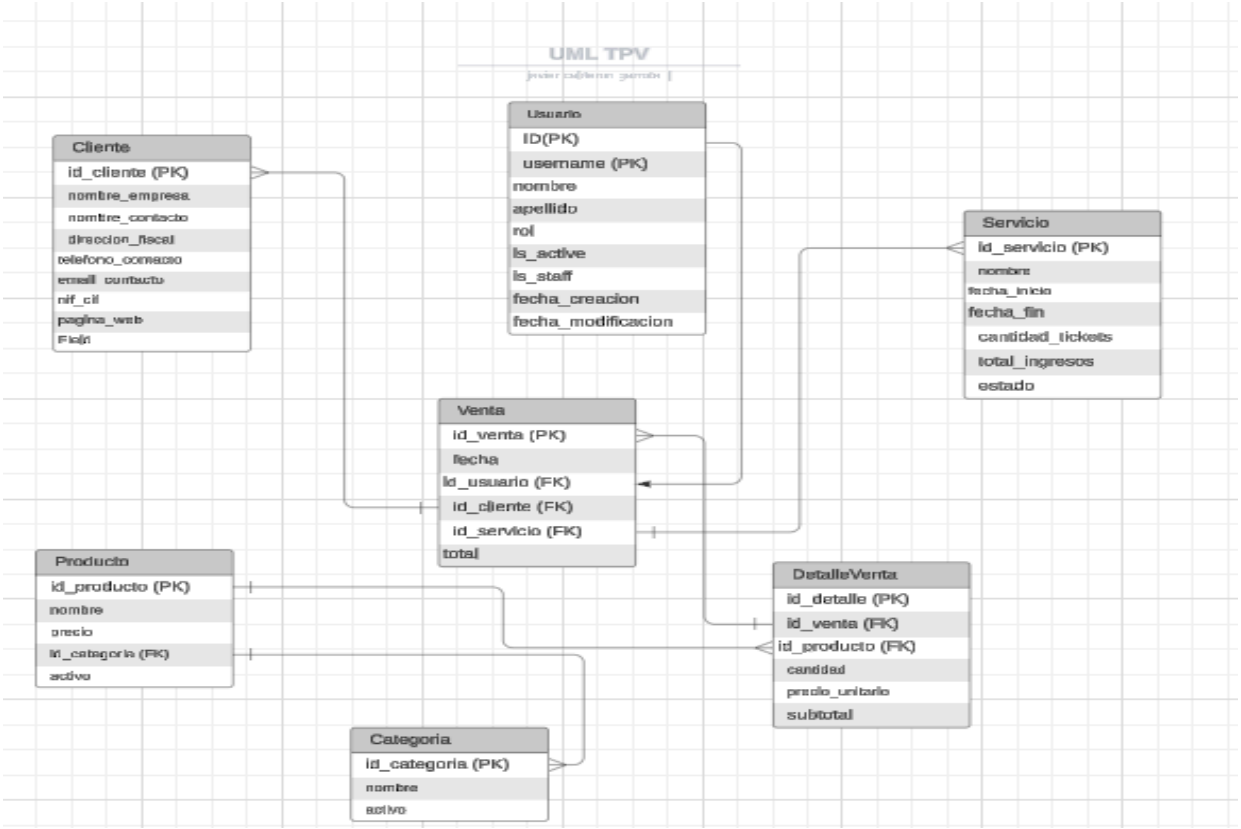
Estos objetivos están diseñados para asegurar que el sistema de TPV desarrollado sea una herramienta efectiva para mejorar la gestión y las operaciones de las pequeñas empresas, proporcionando una solución tecnológica que no solo automatiza procesos clave, sino que también facilita la toma de decisiones estratégicas y optimiza la experiencia del usuario.

---

4. Desarrollo

Funcionamiento de la Base de Datos (BBDD)

La base de datos del proyecto está estructurada para gestionar de manera eficiente la información relacionada con usuarios, productos, servicios, ventas y clientes. A continuación, se destacan los aspectos más relevantes de cada modelo de datos utilizado:



1. Modelo de Usuario

El Usuario es fundamental para gestionar los accesos al sistema, diferenciando entre administradores y vendedores. La autenticación y permisos son gestionados mediante AbstractBaseUser y PermissionsMixin. Los usuarios se crean a través de un UsuarioManager, con funciones específicas para la creación de superusuarios y usuarios normales. El modelo incluye:

- Campos: username, nombre, apellido, rol, is_active, is_staff.
- Relaciones: No tiene relaciones directas, pero se usa para gestionar el acceso al sistema.

2. Modelo de Producto

El Producto es central en la gestión de inventarios, con campos como nombre, precio y categoría. Se vincula con Categoría, lo que permite clasificar los productos de manera lógica. Este modelo tiene:

- Campos: id_producto, nombre, precio, id_categoria, activo.
- Relaciones: Muchos a uno con el modelo Categoría.

3. Modelo de Venta y Detalles de Venta

Las Ventas son el núcleo del proceso de transacción. Cada venta se asocia a un usuario, cliente y servicio. El total de la venta se calcula automáticamente sumando los subtotales de los productos en los Detalles de Venta.

- Campos de Venta: id_venta, fecha, id_usuario, id_cliente, id_servicio, total.
- Campos de DetalleVenta: id_detalle, id_venta, id_producto, cantidad, precio_unitario, subtotal.
- Relaciones:
    - Venta tiene una relación muchos a uno con Usuario, Cliente y Servicio.
    - DetalleVenta tiene una relación muchos a uno con Venta y Producto.

4. Modelo de Servicio

El Servicio representa las sesiones de ventas o actividades comerciales. Cada servicio tiene un estado (abierto o cerrado) y se vincula con las ventas realizadas. Además, el modelo gestiona automáticamente la actualización de ingresos y la cantidad de tickets vendidos cuando se guarda o elimina una venta.

- Campos: id_servicio, nombre, fecha_inicio, fecha_fin, cantidad_tickets, total_ingresos, estado.
- Relaciones: Uno a muchos con Venta.

5. Relaciones entre Modelos

- Usuario ↔ Venta: Un usuario (administrador o vendedor) puede realizar múltiples ventas.
- Cliente ↔ Venta: Un cliente puede realizar múltiples compras, asociadas a una venta específica.
- Producto ↔ DetalleVenta: Un producto puede aparecer en varias ventas, con un detalle específico (cantidad y subtotal).
- Servicio ↔ Venta: Un servicio puede tener múltiples ventas asociadas, y los ingresos del servicio se actualizan con cada venta.

6. Señales para Actualización Automática

El uso de señales (post_save y post_delete) asegura que la base de datos se mantenga actualizada automáticamente:

- Venta → Servicio: Cuando se guarda o elimina una venta, los ingresos y la cantidad de tickets del servicio asociado se actualizan automáticamente.
- Categoría → Producto: Cuando se elimina una categoría, los productos asociados se marcan como inactivos.

Funcionamiento de la Vista de Login

1. **Formulario de Login:**
    - El formulario de login en tu aplicación permite a los usuarios ingresar sus credenciales: **nombre de usuario** y **contraseña**.
    - Cuando el usuario hace un envío (POST), el servidor recibe los datos proporcionados y los valida.
2. **Proceso de Autenticación:**
    - En el backend, el sistema utiliza el método authenticate de Django para verificar si las credenciales coinciden con algún usuario registrado en la base de datos.
    - Si el **nombre de usuario** y la **contraseña** son correctos, el sistema autentica al usuario y lo registra en la sesión con el método login.
3. **Manejo de Errores:**
    - Si las credenciales no son correctas, se muestra un mensaje de error al usuario, indicando que el **usuario o contraseña son incorrectos**.
    - Los mensajes se gestionan utilizando el sistema de mensajes de Django, lo que permite mostrar retroalimentación al usuario en la interfaz.
4. **Redirección Post-login:**
    - Si la autenticación es exitosa, el usuario es redirigido a la página principal de la aplicación, generalmente el **home**.
    - La sesión del usuario se mantiene activa, lo que permite mantener al usuario autenticado durante su interacción con la aplicación.
5. **Mensajes de Confirmación:**
    - Si la autenticación tiene éxito, se envía un mensaje de **"Inicio de sesión exitoso"**.
    - En caso de error, se muestra un mensaje de **"Usuario o contraseña incorrectos"**.

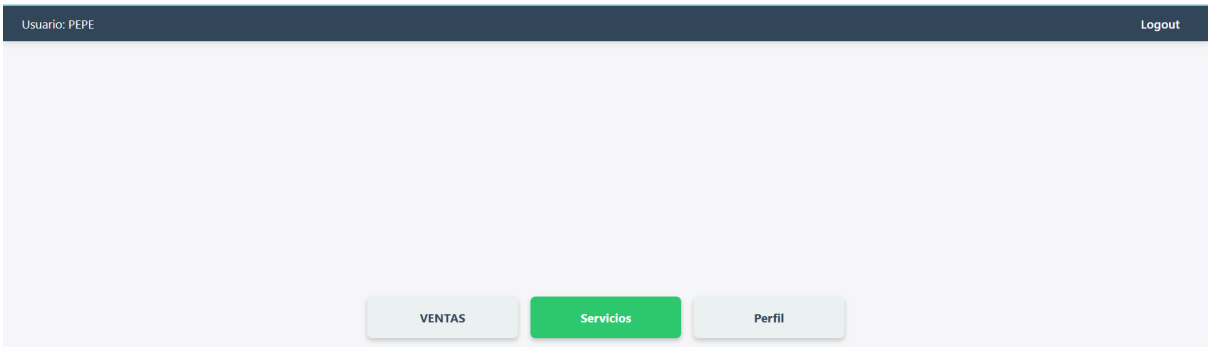
1. **Autenticación y Gestión de Sesión:**
    - Si la autenticación es exitosa, Django establece la sesión del usuario, lo que significa que el usuario podrá acceder a las vistas protegidas que requieren autenticación.
    - Si el usuario no está autenticado, se puede redirigir a una vista pública como la de login.

Funcionamiento de la Vista de Home

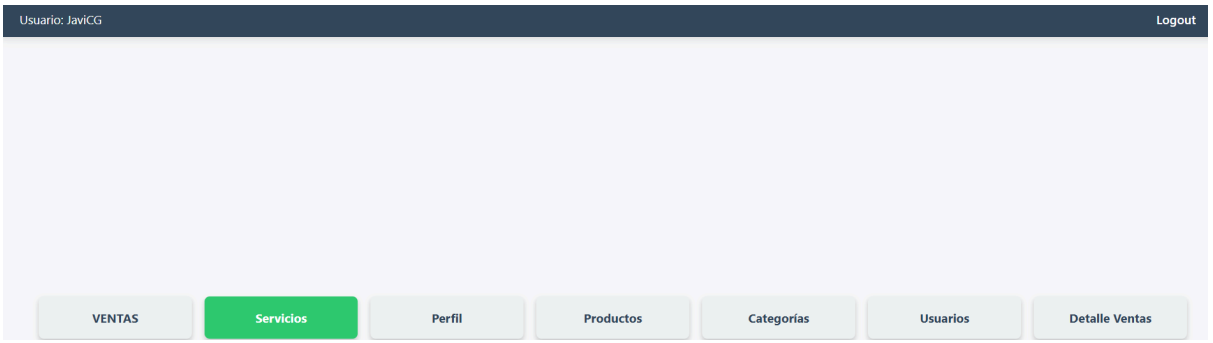
1. **Requerimiento de Autenticación (@login_required)**:
    - La función home está decorada con @login_required, lo que significa que solo los usuarios autenticados pueden acceder a esta vista. Si un usuario no está autenticado, será redirigido a la página de inicio de sesión automáticamente.
2. **Recuperación del ID del Usuario**:
    - Dentro de la vista, se utiliza request.session.get('usuario_id') para obtener el ID del usuario que está actualmente en la sesión. Este ID es almacenado en la sesión cuando el usuario se loguea.
    - Si no se encuentra un usuario_id en la sesión (lo que indica que el usuario no está autenticado o la sesión ha expirado), se redirige a la página de inicio de sesión utilizando redirect('login').
3. **Obteniendo Información del Usuario**:
    - Utilizando el usuario_id, se recupera el objeto Usuario de la base de datos mediante get_object_or_404(Usuario, id=usuario_id). Este método buscará al usuario correspondiente, y si no lo encuentra, lanzará una excepción 404 (página no encontrada).
4. **Verificación de Estado del Servicio**:
    - Se realiza una consulta a la base de datos para verificar si existe algún servicio cuyo estado sea abierto. Esto se hace mediante el filtro Servicio.objects.filter(estado='abierto').exists(). Si existe al menos un servicio abierto, se asigna True a la variable servicio_abierto; de lo contrario, se asigna False.
5. **Renderización de la Página**:
    - Finalmente, la función home renderiza la plantilla home.html, pasando dos variables al contexto:
        - usuario: el objeto Usuario correspondiente al usuario autenticado.
        - servicio_abierto: un valor booleano que indica si hay algún servicio abierto o no.

La plantilla home.html es la que se muestra al usuario después de que la vista home ha sido procesada. A continuación te explico cómo funciona la plantilla:

1. **Barra de Navegación**:
    - Se muestra el nombre de usuario actual en la barra de navegación mediante {{ usuario.username }}.
    - También hay un enlace para cerrar sesión, que redirige a la vista de logout.
2. **Contenido Principal**:
    - La plantilla incluye un contenedor principal con varios botones de acción que se muestran dependiendo del rol del usuario (por ejemplo, "Vendedor", "Administrador").
3. **Condiciones en los Botones**:
    - **Usuarios con rol "Vendedor" o "Administrador"**:
        - **Ventas**: Redirige a la vista para crear ventas.
        - **Servicios**: Si hay un servicio abierto (servicio_abierto es True), el botón tendrá el estilo green (verde), de lo contrario, tendrá el estilo red (rojo). Esto refleja si el servicio está activo o no.
        - **Perfil**: Redirige a la vista de edición del perfil del usuario.



    - **Usuarios con rol "Administrador"**:
        - **Productos**: Enlace a la vista de productos.
        - **Categorías**: Enlace a la vista de categorías de productos.
        - **Usuarios**: Enlace para gestionar usuarios.
        - **Detalle de Ventas**: Enlace a la vista de detalles de ventas.



4. **Responsividad**:
    - Los estilos de la página están diseñados para ser responsivos, lo que significa que se adaptan a diferentes tamaños de pantalla. En pantallas más pequeñas (por ejemplo, móviles), la barra de navegación se reorganiza verticalmente y el tamaño de los títulos se ajusta.

Funcionamiento de la Vista de Ventas

1. **Estructura de la Vista**:
    - **Columna izquierda**:
        - **Detalles de la venta**: Muestra una tabla con los productos añadidos a la venta, su cantidad, precio unitario, y total. Además, tiene un campo para ingresar la cantidad a través de un teclado numérico (con botones del 0 al 9 y un botón de "Borrar").
        - **Teclado numérico**: Permite ingresar la cantidad de productos a añadir mediante botones que modifican el campo screen.
    - **Columna derecha**:
        - **Selector de producto**: Un menú desplegable permite seleccionar la categoría de productos que se desea visualizar. Si no se selecciona ninguna categoría específica, se muestran todos los productos.
        - **Tarjetas de productos**: Se muestran los productos disponibles con su nombre y precio. Al hacer clic en una tarjeta de producto, este se añade al carrito de ventas con la cantidad seleccionada.
        - **Botones de control**: Incluye botones para finalizar la venta ("Caja"), asignar un cliente a la venta.
2. **Modal de Cliente**:
    - Al hacer clic en "Asignar Cliente", se muestra un modal para seleccionar un cliente de los disponibles. Este cliente se asigna a la venta y se muestra en el proceso de pago.

JavaScript (Lógica de Interacción)

1. **Teclado numérico**:
    - Los botones del teclado numérico permiten introducir la cantidad de productos.
    - Si el usuario pulsa el botón "Borrar", el valor de la cantidad ingresada se elimina.
2. **Selección de Producto**:
    - Al seleccionar un producto, si no se ha ingresado una cantidad, se usa la cantidad predeterminada de 1.
    - Se añade el producto al carrito de venta (tabla) con su nombre, cantidad, precio unitario y total. Si el producto ya está en la tabla, se actualiza la cantidad y el total.

- Los productos en la tabla tienen un botón de eliminación, que permite eliminar el producto del carrito.

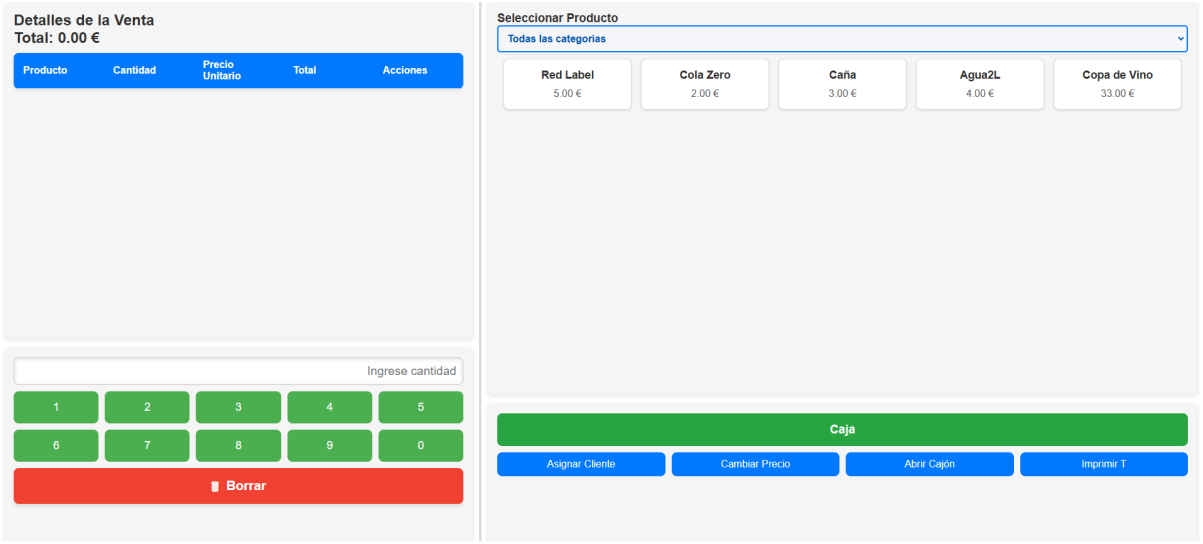
3. **Actualización del Total**:
   - Cada vez que se añade un producto o se elimina uno, se recalcula el total de la venta en base a los productos y cantidades seleccionadas.
4. **Filtrado por Categoría**:
   - Cuando se selecciona una categoría en el desplegable de categorías, los productos se filtran para mostrar solo los que pertenecen a la categoría seleccionada. Si se selecciona "Todas las categorías", se muestran todos los productos.
5. **Asignación de Cliente**:
   - Al hacer clic en el botón "Asignar Cliente", se muestra un modal para elegir un cliente. El cliente seleccionado se asigna a la venta.
   - Una vez asignado, el cliente se asocia con la venta cuando se finaliza la transacción.
6. **Finalizar Venta (Botón Caja)**:
   - Al hacer clic en el botón "Caja", se envían los productos seleccionados y el cliente a la vista de Django (crear_venta) mediante una petición POST. Los productos se envían junto con su cantidad, y el cliente seleccionado se incluye si es aplicable.
   - Si la venta es exitosa, se limpia el carrito, se muestra un mensaje de éxito, y se reinician las variables.

Vista Django (crear_venta)

1. **Recepción de Datos**:
   - Cuando se recibe la solicitud POST, Django procesa los datos enviados desde el frontend:
     - id_cliente: El cliente seleccionado para la venta.
     - producto_ids: Los IDs de los productos seleccionados.
     - cantidades: Las cantidades de cada producto.
2. **Validaciones**:
   - Se validan que se hayan enviado productos y cantidades.
   - Si la cantidad es inválida, se lanza un error.
3. **Creación de Venta**:

- Si la validación es exitosa, se crea una nueva venta asociada con el usuario actual, el servicio abierto (se supone que solo hay un servicio activo), y el cliente seleccionado (o None si no se selecciona ninguno).

4. **Creación de Detalles de Venta**:
   - Para cada producto, se crea un DetalleVenta con el precio, la cantidad y el subtotal correspondiente.
   - Se calcula el total de la venta.
5. **Respuesta**:
   - Si la venta se guarda correctamente, la respuesta JSON incluye el ID de la venta.
   - Si ocurre un error, la respuesta incluye el mensaje de error correspondiente.



Funcionamiento de la Vista de Gestión de Servicios

1. Barra Superior

- Muestra el nombre del usuario logueado y un enlace para volver al **Home**.

2. Botón para Crear Servicio

- **Crear Servicio**: Abre un modal donde se ingresa el nombre y estado del servicio (abierto o cerrado).

3. Tabla de Servicios

- Muestra los servicios en una tabla con:
    - **ID del servicio**.
    - **Nombre del servicio**.
    - **Fecha de inicio** (y fin, si existe).
    - **Cantidad de tickets**.
    - **Estado**: Resaltado con color (verde para abierto, rojo para cerrado).
    - **Acciones**: Botones para **editar** o **eliminar**.
- **Acciones**:
    - **Editar**: Abre el modal con los datos del servicio para editar.
    - **Eliminar**: Muestra una confirmación antes de eliminar el servicio.

4. Paginación

- Si la lista de servicios es grande, se implementa paginación con botones de **Anterior** y **Siguiente**, mostrando la página actual.

5. Modal de Crear/Editar Servicio

- **Crear Servicio**: Muestra un formulario con los campos **Nombre** y **Estado**.
- **Editar Servicio**: Carga los datos del servicio en el modal para editarlos.

6. Funcionalidad del Lado del Servidor (Django)

- **Listar Servicios**: Obtiene y muestra los servicios ordenados por fecha de inicio, con paginación.
- **Crear Servicio**: Al enviar el formulario, se crea un nuevo servicio y se redirige al usuario con un mensaje de éxito.

- **Editar Servicio**: Carga los datos existentes para editar y luego actualiza el servicio en la base de datos.
- **Eliminar Servicio**: Confirma la eliminación y luego borra el servicio de la base de datos.

7. JavaScript (Interactividad del Modal)

- **Abrir Modal**: Los modales se abren para crear o editar servicios, según la acción seleccionada.
- **Limpiar Formulario**: El formulario se limpia si el modal se cierra sin enviar los datos

Funcionamiento de la Vista de Editar Perfil

1. Formulario de Edición

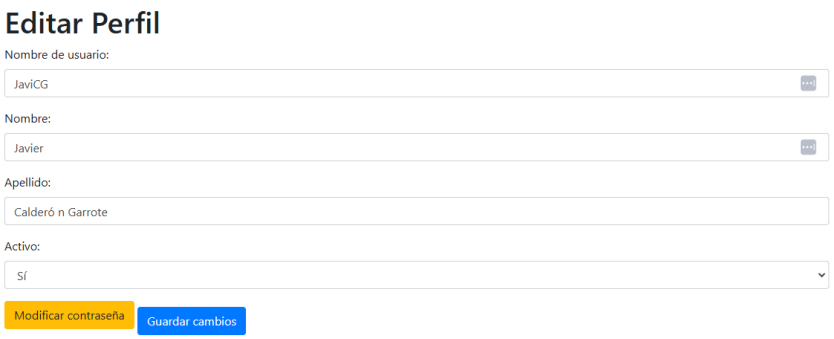
- Permite modificar:
    - **Nombre de usuario**, **nombre**, **apellido** y **estado de la cuenta** (activo/inactivo).
    - Contraseña: Opcional, activable mediante un botón que muestra campos para nueva contraseña y confirmación.

2. Validaciones en el Frontend

- Verifica que la nueva contraseña:
    - Coincida con su confirmación.
    - Sea numérica y tenga al menos 4 caracteres.
- Solo permite enviar el formulario si:
    - Las contraseñas son válidas, o
    - No se realizaron cambios en la contraseña.

3. Proceso en el Backend

- **Actualización de Perfil**:
    - Actualiza los campos básicos (nombre de usuario, nombre y apellido) con los datos enviados.
- **Gestión de Contraseña**:
    - Comprueba que la nueva contraseña (si se proporcionó):
        - Coincida con la confirmación.
        - Cumpla los requisitos (numérica y mínimo 4 caracteres).
    - Si es válida, la actualiza en la base de datos.
    - Si no es válida, devuelve un mensaje de error.
- **Guardar Cambios**:
    - Valida los datos y guarda la nueva información.
    - Muestra un mensaje de éxito al usuario tras actualizar el perfil.


Editar Perfil
Nombre de usuario:
JaviCG
Nombre:
Javier
Apellido:
Calderó n Garrote
Activo:
Sí
Modificar contraseña
Guardar cambios

Funcionamiento de la Vista de Gestión de Productos

La **Vista de Gestión de Productos** permite listar, crear, editar y eliminar productos de forma eficiente. A continuación, su funcionamiento:

1. Barra Superior

- Muestra el nombre del usuario autenticado o "Invitado".
- Incluye un enlace para volver al Home.

2. Botón para Crear Producto

- Abre un modal con campos para ingresar datos del producto: Nombre, Precio y Categoría (desplegable).
- El modal sirve tanto para crear como para editar productos.

3. Tabla de Productos

- **Columnas**: ID, Nombre, Precio, Categoría y Acciones.
- **Acciones**:
    - **Editar**: Abre un modal con los datos del producto cargados para su modificación.
    - **Eliminar**: Muestra un cuadro de confirmación y realiza un borrado lógico del producto.

4. Paginación

- Divide los productos en páginas, mostrando botones para avanzar y retroceder, con la página actual visible ("Página X de Y").

5. Modal de Crear/Editar Producto

- **Crear Producto**: Formulario vacío para ingresar datos del nuevo producto.
- **Editar Producto**: Carga los datos del producto seleccionado, ajustando el formulario para actualizar la información.

6. Lado del Servidor (Django)

- **Listar Productos**: Obtiene productos activos, aplica paginación y los envía a la plantilla para su renderización.

- **Crear Producto**: Guarda un nuevo producto en la base de datos y redirige con un mensaje de éxito.
- **Editar Producto**: Actualiza un producto existente en la base de datos y redirige con un mensaje de éxito.
- **Eliminar Producto**: Marca el producto como no activo en la base de datos (borrado lógico) y redirige con un mensaje de éxito.

Funcionamiento de la Vista de Gestión de Categorías

La **Vista de Gestión de Categorías** permite listar, crear, editar y eliminar categorías de productos o servicios. Su funcionamiento se detalla a continuación:

1. NavBar

- Incluye un enlace para volver al Home y muestra el nombre del usuario autenticado o "Invitado" si no hay sesión iniciada.

2. Tabla de Categorías

- **Columnas**: ID de la categoría, Nombre de la categoría, y Acciones (editar y eliminar).
- **Acciones**:
    - **Crear:** Abre un modal con los campos de categoria
    - **Editar**: Abre un modal con los datos de la categoría cargados para su edición.
    - **Eliminar**: Usa una confirmación emergente y realiza un borrado lógico, marcando la categoría como inactiva en la base de datos.

3. Paginación

- Divide las categorías en páginas cuando hay muchas, con botones para avanzar y retroceder. Muestra la página actual (ejemplo: "Página X de Y").

4. Modal de Crear/Editar Categoría

- **Crear Categoría**: Muestra un formulario vacío para ingresar el nombre de la nueva categoría.
- **Editar Categoría**: Muestra los datos actuales de la categoría seleccionada en el formulario, permitiendo actualizarlos.
- Ambas acciones usan solicitudes POST para guardar o actualizar los datos en la base de datos.

5. Lado del Servidor (Django)

- **Listar Categorías**: Obtiene las categorías activas desde la base de datos, aplica paginación y las envía a la plantilla.

- **Crear Categoría**: Guarda una nueva categoría en la base de datos y redirige a la lista con un mensaje de éxito.
- **Editar Categoría**: Actualiza el nombre de una categoría existente y redirige con un mensaje de éxito.
- **Eliminar Categoría**: Realiza un borrado lógico, marcando la categoría como no activa, y redirige a la lista con una notificación de éxito.

Funcionamiento de la Vista de Gestión de Usuarios

La **Vista de Gestión de Usuarios** permite listar, crear, editar y eliminar usuarios en una aplicación web con Django. Su funcionamiento se detalla a continuación:

1. NavBar

    - Incluye un enlace al Home y muestra el nombre del usuario autenticado o "Invitado".

2. Tabla de Usuarios

    - Presenta las columnas: ID, Nombre de Usuario, Nombre y Apellido, Rol, y Acciones.
        - **Acciones**:
            - **Editar**: Abre un modal con los datos cargados, con la opción de cambiar la contraseña si es necesario.
            - **Eliminar**: Usa SweetAlert para confirmar antes de eliminar al usuario y recargar la lista.

3. Paginación

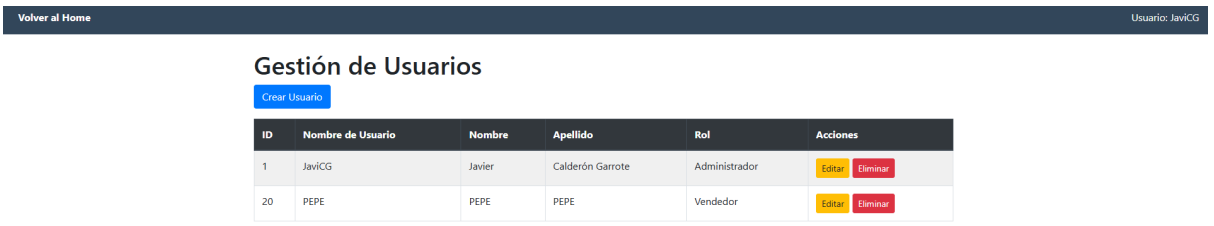
    - Si hay muchos usuarios, muestra botones para navegar entre páginas e indica la página actual.

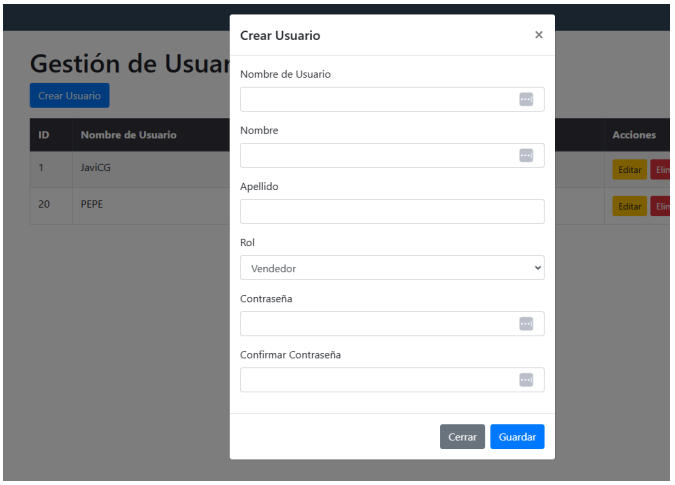
4. Modal de Crear/Editar Usuario

    - **Crear Usuario**: Abre un formulario vacío para ingresar datos básicos y la contraseña.
    - **Editar Usuario**: Carga los datos del usuario en el formulario, con la opción de modificar la contraseña si se habilita.

5. Lado del Servidor (Django)

    - **Listar Usuarios**: Obtiene usuarios con paginación y los muestra en la tabla.
    - **Crear Usuario**: Valida los datos y crea un usuario nuevo con Usuario.objects.create_user.
    - **Editar Usuario**: Actualiza los datos del usuario identificado por ID.
    - **Eliminar Usuario**: Confirma la eliminación y elimina físicamente al usuario con usuario.delete.


Volver al Home
Usuario: JaviCG
Gestión de Usuarios
Crear Usuario
ID
Nombre de Usuario
Nombre
Apellido
Rol
Acciones
1
JaviCG
Javier
Calderón Garrote
Administrador
Editar
Eliminar
20
PEPE
PEPE
PEPE
Vendedor
Editar
Eliminar


Crear Usuario
Nombre de Usuario
Nombre
Apellido
Rol
Vendedor
Contraseña
Confirmar Contraseña
Cerrar
Guardar

Funcionamiento de la Vista de Gestión de Clientes

1. Barra de Navegación

- **Enlace al Home**: Redirige a la página principal.
- **Usuario Logueado**: Muestra el nombre del usuario autenticado o "Invitado" si no hay sesión.

2. Listado de Clientes

- **Tabla con Clientes**:
    - Muestra: Nombre de la empresa, nombre de contacto, teléfono, y email.
    - **Acciones**:
        - **Editar**: Abre un modal con la información del cliente para editar.
        - **Eliminar**: Elimina al cliente tras confirmación.
    - **Paginación**: Muestra 6 clientes por página, con navegación entre páginas.

3. Modal de Crear/Editar Cliente

- **Campos del Modal**:
    - Nombre de la empresa, nombre del contacto, dirección fiscal, teléfono, email, NIF/CIF y página web.
- **Acciones**:
    - **Crear Cliente**: Utiliza el formulario vacío para crear un nuevo cliente.
    - **Editar Cliente**: Carga los datos existentes del cliente para su edición.

4. Paginación

- Si hay muchos clientes, se divide en páginas.
- Los usuarios pueden navegar con botones de **Anterior** y **Siguiente**, y se muestra la página actual.

5. Funcionalidad en el Lado del Servidor (Django)

- **Listar Clientes**:
    - Obtiene todos los clientes de la base de datos y aplica paginación (6 por página).
    - Renderiza la vista con la lista de clientes.
- **Crear/Editar Cliente**:
    - Si se proporciona un **id_cliente**, edita un cliente existente.

    - Si no hay **id_cliente**, crea un nuevo cliente.
    - Muestra un mensaje de éxito y redirige a la lista de clientes.
- **Eliminar Cliente**:
    - Elimina el cliente de la base de datos y redirige a la lista con un mensaje de éxito.

Este proceso permite gestionar los clientes de manera eficiente, facilitando tanto la creación, edición y eliminación de clientes, como la visualización paginada para mejorar la navegación.

Funcionamiento de la Vista Detalle Venta

1. **Requerimiento de Autenticación (@login_required)**: La vista detalle_venta está decorada con @login_required, lo que significa que solo los usuarios autenticados pueden acceder a esta vista. Si el usuario no está autenticado, será redirigido a la página de inicio de sesión automáticamente.
2. **Obtención de Datos para los Gráficos**:
    - **Productos Más Vendidos**: Se obtienen los 6 productos más vendidos. Esto se logra consultando la tabla DetalleVenta y sumando la cantidad vendida para cada producto, ordenándolos de mayor a menor cantidad vendida.
    - **Clientes con Más Ventas**: Se obtienen los 5 clientes con más ventas. Esto se hace a través de la tabla Venta, contando el número de ventas realizadas por cada cliente.
    - **Servicios con Más Ventas**: De forma similar, se obtienen los 5 servicios con más ventas, contando las ventas por servicio.
3. **Obtención de Detalles de Venta**: Se obtiene la lista completa de DetalleVenta y se aplica paginación. Cada página muestra 6 detalles de venta.
4. **Paginación**: La paginación se realiza usando Paginator de Django, lo que permite dividir los detalles de venta en varias páginas con un máximo de 6 elementos por página. Los enlaces de paginación permiten al usuario navegar entre las páginas.
5. **Preparación de Datos para los Gráficos**: Los datos obtenidos de las consultas de productos, clientes y servicios se preparan en listas con las etiquetas (nombres) y los valores (totales vendidos o totales de ventas). Estos datos se pasan a la plantilla para ser utilizados por los gráficos generados con Chart.js.
6. **Renderización de la Plantilla**: Finalmente, la vista renderiza la plantilla detalle_venta.html y pasa los datos obtenidos (productos, clientes, servicios y detalles de ventas) a la plantilla para su visualización.

Estructura de la Plantilla detalle_venta.html

1. **Barra de Navegación**:
    - En la parte superior se encuentra un nav con un enlace para volver al "Home" y el nombre de usuario actual, o "Invitado" si el usuario no está autenticado.
2. **Contenedor de Acciones**:

- Hay varios botones que permiten al usuario ver diferentes datos: gráficos de los productos más vendidos, clientes con más ventas, servicios con más ventas y detalles de ventas.
- Cada uno de estos botones tiene un id específico, y al hacer clic en ellos se muestran u ocultan los gráficos correspondientes o la tabla de detalles de venta.

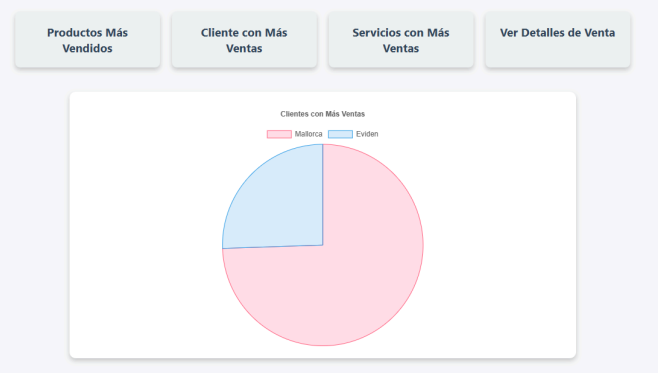
3. **Contenedores de los Gráficos**:
   - Se incluyen tres contenedores para los gráficos:
     - **Productos Más Vendidos**: Un gráfico de barras que muestra los productos más vendidos.
     - **Clientes con Más Ventas**: Un gráfico circular que muestra la distribución de ventas entre los clientes.
     - **Servicios con Más Ventas**: Un gráfico de barras similar al de productos, pero con los servicios.
4. **Tabla de Detalles de Venta**:
   - La tabla muestra una lista de detalles de venta, incluyendo el ID, producto, cantidad, precio unitario, subtotal y la ID de la venta correspondiente.
   - Los detalles se muestran con paginación, lo que permite navegar entre las diferentes páginas de ventas.

5. **Paginación**:
   - En la parte inferior de la tabla se encuentran los enlaces de paginación. Si hay más de 6 detalles de venta, se podrán ver otras páginas usando los enlaces "Anterior" y "Siguiente".
6. **Chart.js para la Visualización Gráfica**:
   - Los gráficos son creados con la biblioteca Chart.js. El código JavaScript se encarga de inicializar los gráficos de acuerdo con los datos pasados desde el

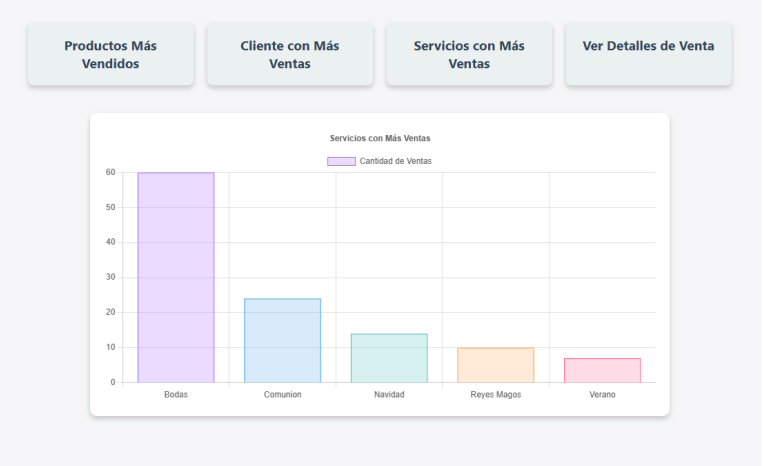
backend. Dependiendo del gráfico seleccionado por el usuario, el contenedor adecuado se muestra y se genera el gráfico.

- **Gráfico de Productos**: Se utiliza un gráfico de barras para mostrar los productos más vendidos.

- **Gráfico de Clientes**: Se utiliza un gráfico de pastel para mostrar las ventas por cliente.



- **Gráfico de Servicios**: También es un gráfico de barras para mostrar los servicios con más ventas.



5. Prueba Unitarias con Pytest

1. Test de la BBDD
    a. **test_usuario_creado_correctamente**: Verifica que un usuario se crea correctamente con los datos esperados.
    b. **test_superusuario_creado_correctamente**: Comprueba que un superusuario se crea con permisos de administrador (is_superuser y is_staff).
    c. **test_categoria_string_representation**: Verifica que la representación en texto de una categoría sea su nombre.
    d. **test_producto_string_representation**: Comprueba que la representación en texto de un producto sea su nombre.
    e. **test_producto_categoria**: Verifica que un producto está correctamente asociado a su categoría.
    f. **test_cliente_string_representation**: Comprueba que la representación en texto de un cliente sea su nombre de empresa.
    g. **test_servicio_crea_correctamente**: Verifica que un servicio se crea con el estado inicial correcto y se representa bien.
    h. **test_venta_string_representation**: Comprueba que la representación de una venta incluye "Venta".
    i. **test_venta_clean_sin_servicio_abierto**: Verifica que no se permita crear una venta si no hay servicios abiertos.
    j. **test_detalle_venta_calculo**: Verifica que el subtotal y el precio unitario en el detalle de venta se calculan correctamente.
    k. **test_detalle_venta_producto_inactivo**:Comprueba que no se puede agregar un producto inactivo al detalle de venta.

2. Test de Usuarios
    a. **test_editar_perfil_get**: Prueba la vista GET para editar el perfil de un usuario. Verifica que el usuario logueado pueda acceder a la vista de edición de perfil, que la respuesta sea correcta y que el contexto contenga la información del usuario.
    b. **test_editar_perfil_post**: Prueba la vista POST para editar el perfil. Envia un formulario con nuevos datos de usuario y verifica que estos cambios se guardan correctamente en la base de datos.
    c. **test_editar_perfil_post_contra_no_coincide**: Verifica que si las contraseñas no coinciden al editar el perfil, se muestre el mensaje adecuado de error.
    d. **test_login_view_post_success**: Prueba la vista de login con datos correctos, asegurando que el usuario sea redirigido correctamente y que su ID se almacene en la sesión.
    e. **test_login_view_post_failure**: Verifica que si se proporcionan credenciales incorrectas, la vista de login retorne un error de autenticación.

f. **test_logout_view**: Prueba la vista de logout. Asegura que al hacer logout, el usuario sea redirigido a la página de login y que su sesión sea eliminada.
g. **test_listar_usuarios**: Verifica que la vista de listado de usuarios funcione correctamente y que la paginación esté funcionando con múltiples usuarios.
h. **test_crear_usuario_post**: Prueba la creación de un nuevo usuario, asegurándose de que se guarde correctamente en la base de datos y que el usuario sea redirigido a la lista de usuarios.
i. **test_borrar_usuario**: Verifica que un usuario pueda ser eliminado correctamente de la base de datos y que la eliminación sea efectiva.
j. **test_seleccionar_usuario**: Asegura que al seleccionar un usuario, este sea almacenado en la sesión.

3. Test de Categorías

a. **test_listar_categorias**: Verifica que la vista de listar categorías muestre las categorías correctas y que la paginación funcione bien.
b. **test_crear_categoria**: Prueba la creación de una nueva categoría, asegurando que se guarde correctamente en la base de datos y que el usuario sea redirigido a la lista de categorías.
c. **test_editar_categoria**: Verifica que una categoría pueda ser editada correctamente. Después de la edición, se comprueba que el cambio se haya guardado.
d. **test_editar_categoria_invalid**: Prueba la edición de una categoría con datos inválidos, asegurando que la categoría no sea modificada y que se muestre el comportamiento esperado (redirección).
e. **test_borrar_categoria**: Prueba el borrado de una categoría de manera lógica (marcándola como inactiva) y verifica que el estado de la categoría cambie a inactivo.
f. **test_no_permitir_crear_categoria_sin_nombre**: Verifica que al intentar crear una categoría sin un nombre, se muestre un mensaje de error.
g. **test_crear_categoria_sin_nombre**: Asegura que no se permita la creación de una categoría sin nombre, mostrando un mensaje de error adecuado.
h.

4. Test de Productos

a. **test_listar_productos**: Verifica que la vista lista correctamente los productos y muestra su información, como el nombre y la categoría
b. .**test_crear_producto**: Comprueba que se pueda crear un producto con datos válidos y se guarde correctamente en la base de datos.

c. **test_borrar_producto**: Verifica que un producto pueda ser eliminado de forma lógica, marcándolo como inactivo.
d. **test_editar_producto**: Comprueba que un producto pueda ser editado y que los cambios se guarden correctamente

5. Test de Servicios

a. **test_listar_servicios**: Verifica que la vista muestra correctamente la lista de servicios disponibles.
b. **test_crear_servicio**: Comprueba que se pueda crear un nuevo servicio y se redirija correctamente a la lista de servicios.
c. **test_editar_servicio**: Verifica que se puedan editar los datos de un servicio existente y que los cambios se guarden correctamente.
d. **test_borrar_servicio**: Comprueba que se pueda eliminar un servicio existente y que se redirija correctamente a la lista de servicios.
e. **test_borrar_servicio_no_existente**: Verifica el comportamiento al intentar eliminar un servicio que no existe, asegurando que no se afecten los datos existentes.

6. Test de Ventas

a. **test_crear_venta_con_usuario_autenticado**:
    i. Verifica que un usuario autenticado pueda crear una venta correctamente.
    ii. Valida la creación de la venta, el cálculo del total y los detalles relacionados.
b. **test_crear_venta_sin_productos**:
    i. Comprueba que intentar crear una venta sin productos devuelve un error con el código 400.
    ii. Se valida el mensaje de error retornado.
c. **test_crear_venta_con_cantidades_invalidas**:
d.
    i. Prueba la validación al usar cantidades inválidas (por ejemplo, negativas).
    ii. Asegura que el servidor responda con 400 y un mensaje adecuado.
e. **test_detalle_venta_con_ventas**:
    i. Verifica que la vista de detalles de venta muestra correctamente la información de los productos.
    ii. Valida la presencia del producto y su subtotal.
f. **test_crear_venta_sin_servicio_abierto**:
    i. Prueba el caso donde no hay un servicio abierto, lo que impide la creación de la venta.
    ii. Se asegura de que el servidor devuelva un error con 400 y un mensaje informativo.

---

7. Mejoras Futuras

Sistema de Ticket por Cada Venta Realizada
Implementar un sistema automático de generación e impresión de tickets después de cada transacción realizada. Esto permitirá:

- Mejorar el control de ventas y facilitar la entrega de comprobantes al cliente.
- Ofrecer un registro físico y digital de todas las operaciones, lo cual es útil para auditorías y conciliaciones.

Sistema de Comandas a Través de Dispositivos Móviles
Desarrollar una solución que permita enviar comandas directamente desde dispositivos móviles (tabletas o smartphones) conectados a una red local. Esta mejora contribuirá a:

- Agilizar el proceso de toma de pedidos en restaurantes o negocios similares.
- Reducir errores humanos al evitar transcripciones manuales de los pedidos.
- Mejorar la comunicación entre el área de atención al cliente y la cocina o almacén.

Mejoras en el Diseño del TPV
Realizar una actualización del diseño de la interfaz del TPV, enfocada en:

- Optimizar la experiencia del usuario (UX) mediante un diseño más intuitivo y fácil de usar.
- Modernizar la estética visual para que sea más atractiva y profesional.
- Incorporar elementos gráficos que faciliten la navegación y el uso eficiente del sistema.

---

8. Problemas Encontrados

Uno de los principales problemas que hemos encontrado es que, al no haber utilizado antes tecnologías como **Django** y **SQLite**, ha costado un poco adaptarse y coger el ritmo. Algunas de las dificultades incluyen:

- **Curva de aprendizaje**: La necesidad de familiarizarse con el marco de trabajo Django y la configuración de la base de datos SQLite ha generado tiempos de adaptación más largos de lo esperado.
- **Falta de experiencia previa**: Al no contar con experiencia previa en estas tecnologías, algunos procesos o funcionalidades no se implementan con la misma rapidez que si se trabajara con herramientas más conocidas.
- **Optimización**: A medida que el proyecto crece, hemos identificado que es necesario optimizar tanto el backend (Django) como las consultas de la base de datos (SQLite) para garantizar un rendimiento eficiente a largo plazo.
