

PROYECTO FINAL

PYTHON

FRANCO VALENTIN ESPINEL



FRANCO VALENTIN ESPINEL

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

**Fecha:** 12, 10, 2024

ÍNDICE DE CONTENIDOS

1. **DESCRIPCIÓN GENERAL DEL PROYECTO**
2. **OBJETIVOS Y ALCANCE DEL PROYECTO**
3. **STACK TECNOLÓGICO Y ALTERNATIVAS**

**EVALUADAS**

1. **MODELO DE DATOS**
2. **EXPLICACIÓN DE LOS REQUISITOS DE LA APLICACIÓN**

**5.1) REQUISITOS FUNCIONALES**

**5.2) REQUISITOS NO FUNCIONALES**

1. **MANUAL DE INSTALACIÓN**
2. **CONCLUSIONES**
3. **EVOLUTIVOS DEL PROYECTO**
4. **DESCRIPCIÓN DE LOS COMPONENTES**
5. DESCRIPCIÓN GENERAL DEL PROYECTO

El Sistema de Gestión de Inventario (TechVentory) es una aplicación web diseñada para ayudar a las empresas a gestionar eficientemente su inventario, ventas y compras. Esta aplicación permite a los usuarios realizar un seguimiento de los productos, proveedores, ventas y compras, así como generar informes y análisis para tomar decisiones informadas. El sistema está diseñado para ser fácil de usar, escalable y adaptable a diferentes tipos de negocios.

1. OBJETIVOS Y ALCANCE DEL PROYECTO

OBJETIVOS

º Desarrollar una aplicación web intuitiva para la gestión de inventario.

º Implementar un sistema de seguimiento en tiempo real de productos y stock.

º Automatizar procesos de compra y venta.

º Proporcionar análisis y reportes detallados para la toma de decisiones.

º Mejorar la eficiencia operativa y reducir errores en la gestión de inventario.

ALCANCE

º Gestión de productos: añadir, editar, eliminar y visualizar productos.

º Gestión de proveedores: mantener una base de datos de proveedores.

º Control de stock: seguimiento de niveles de inventario y alertas de bajo stock.

º Procesamiento de ventas y compras.

º Generación de informes y gráficos de rendimiento.

º Interfaz de usuario responsive para acceso desde diferentes dispositivos.

º No incluye: integración con sistemas de contabilidad externos o gestión de recursos humanos.

1. STACK TECNOLÓGICO Y ALTERNATIVAS EVALUADAS

STACK TECNOLÓGICO

º Backend: Python con Flask

º Frontend: HTML, CSS (Tailwind CSS), JavaScript

º Base de datos: SQLite

º ORM: SQLAlchemy

º Autenticación: Flask-Login

º Gráficos: Chart.js

ALTERNATIVAS EVALUADAS

º Backend: Django vs Flask:

Se eligió Flask por su ligereza y flexibilidad, ideal para un proyecto de tamaño medio.

º Frontend: React vs Vue vs Vanilla JS:

Se optó por Vanilla JS con Tailwind CSS para mantener la simplicidad y reducir la curva de aprendizaje.

º Base de datos: PostgreSQL vs SQLite:

SQLite fue seleccionado por su facilidad de configuración y portabilidad, adecuado para la escala inicial del proyecto.

1. MODELO DE DATOS

EXPLICACIÓN DE LA BASE DE DATOS

Se utiliza SQLite como sistema de gestión de base de datos relacional. SQLite es ligero, no requiere un servidor separado y es ideal para aplicaciones de tamaño medio con una carga de trabajo moderada.

ESQUEMA DE LA BASE DE DATOS

<https://dbdiagram.io/d/Proyecto_final-670d2f7797a66db9a3ed3551>

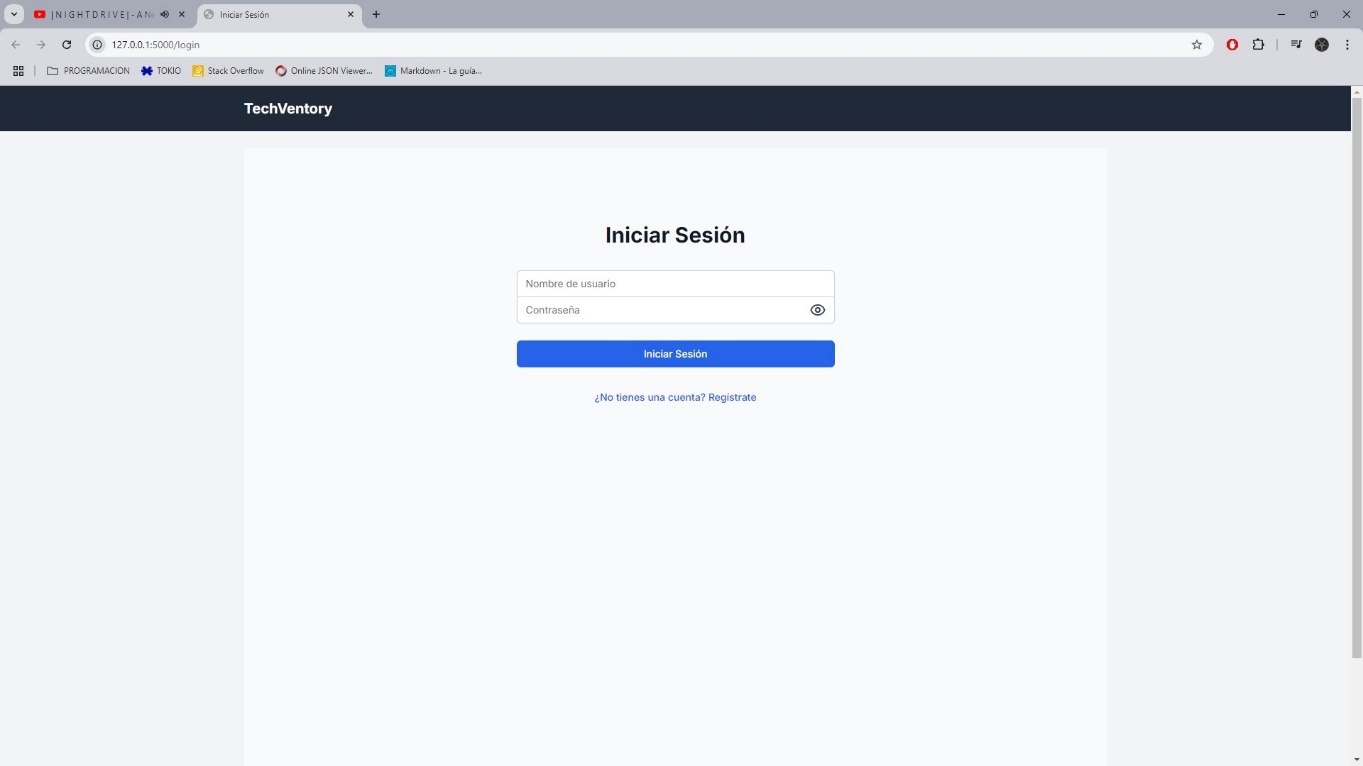
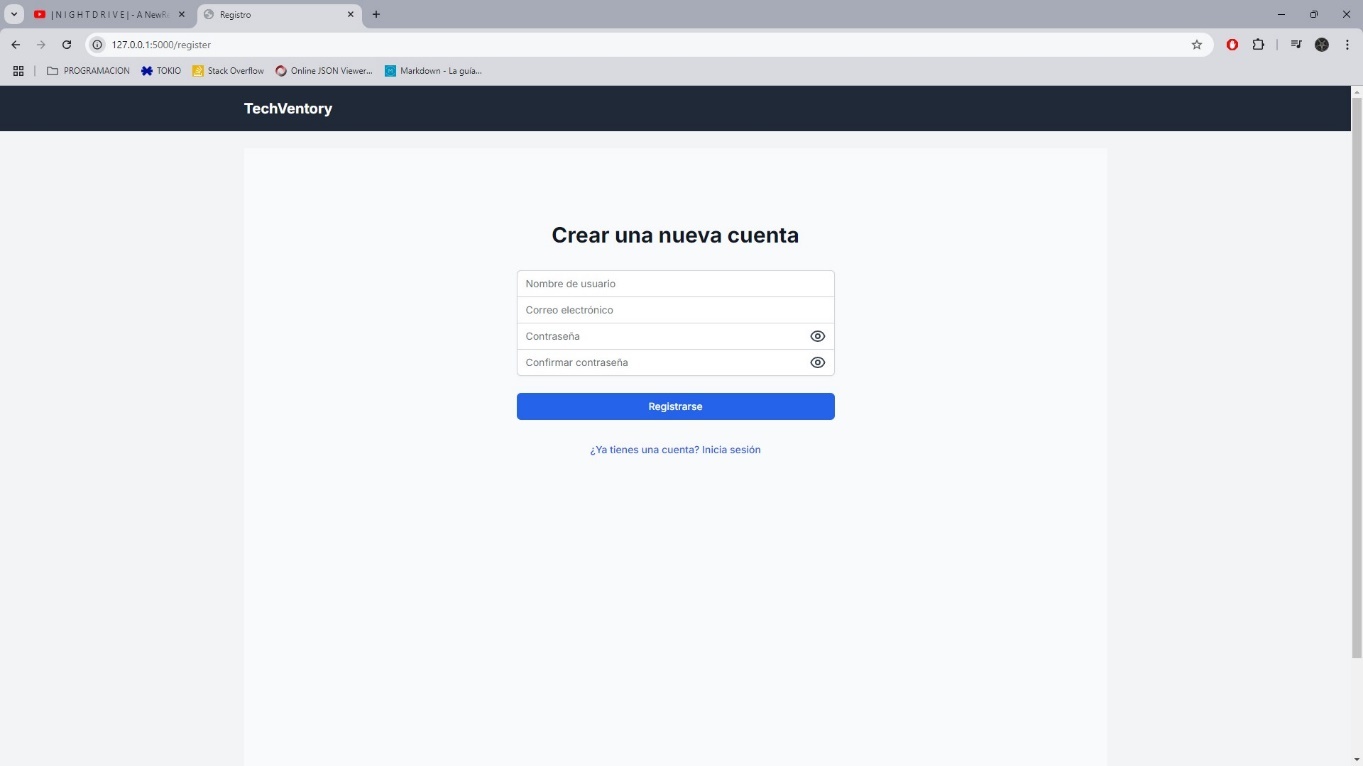
1. EXPLICACIÓN DE LOS REQUISITOS DE LA APLICACIÓN

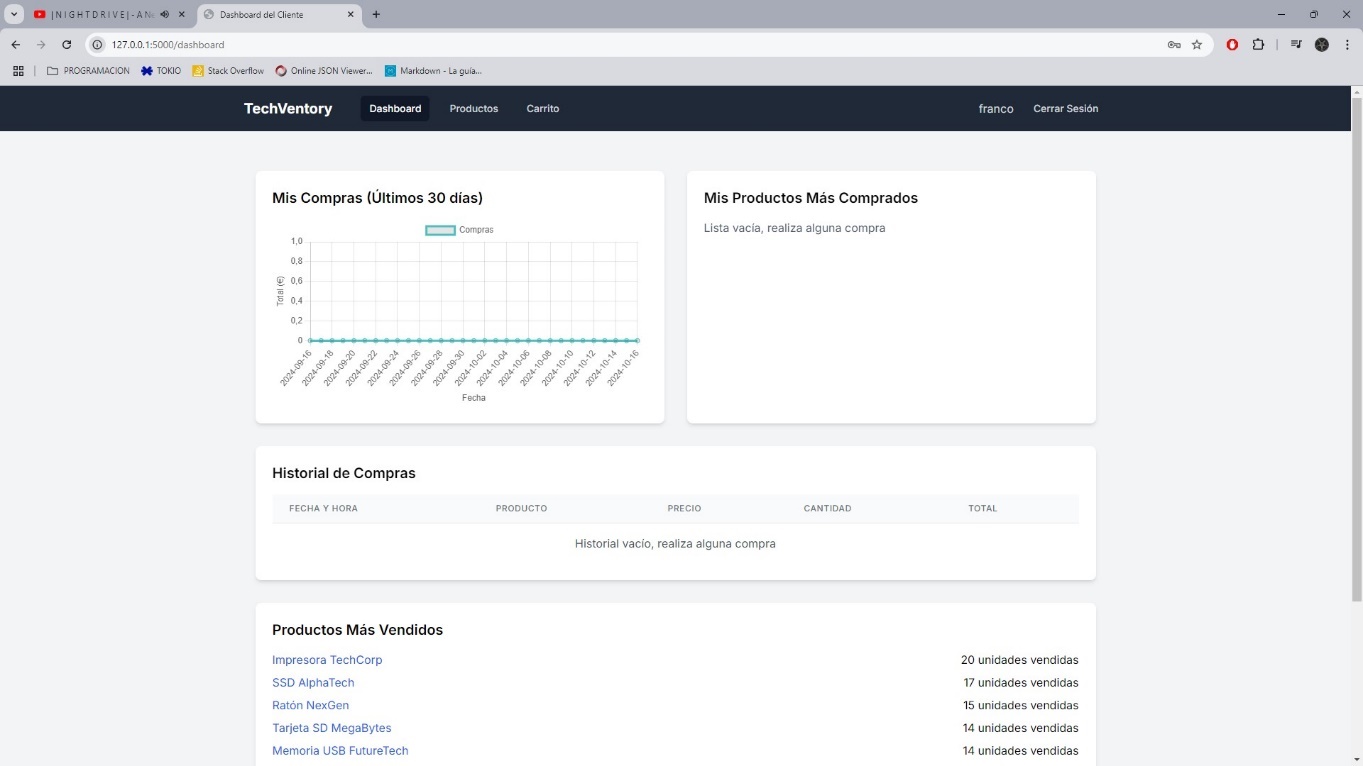
REQUISITOS FUNCIONALES:

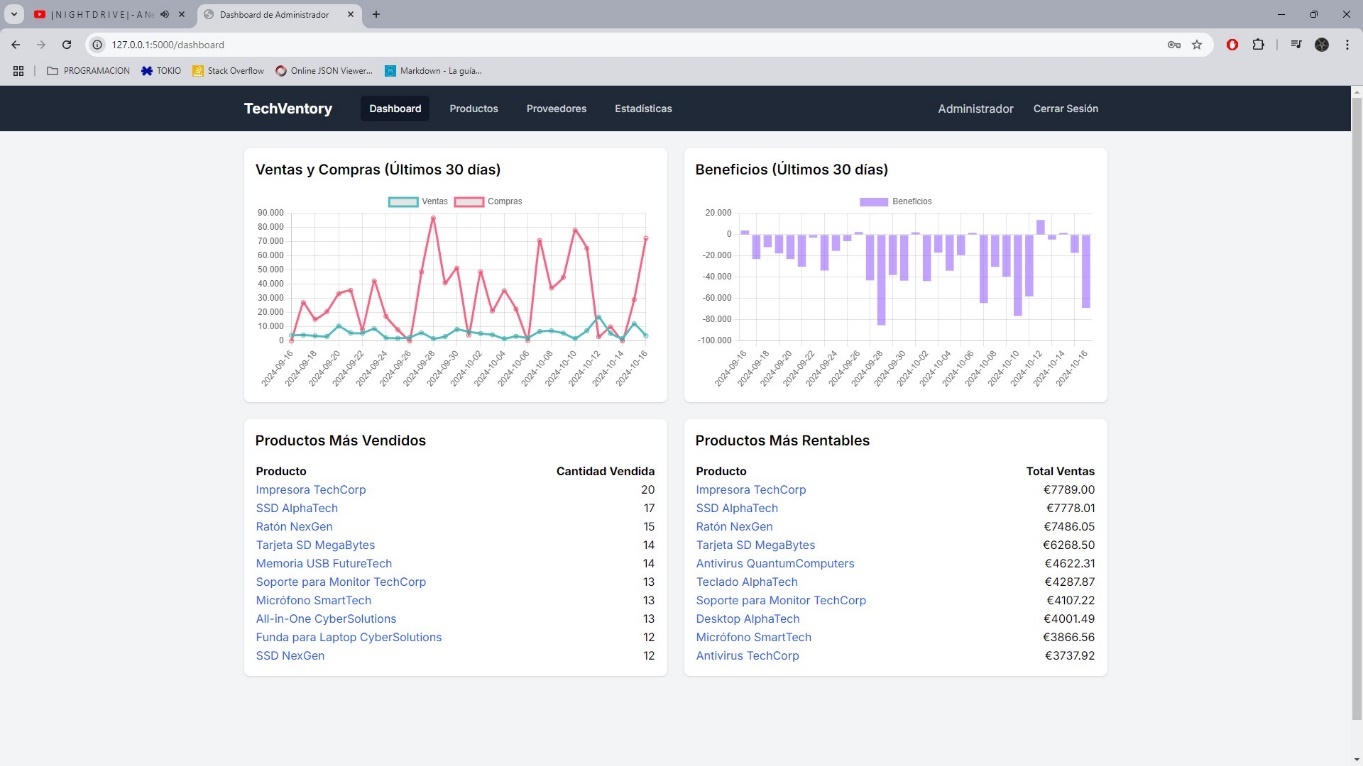
A) GESTIÓN DE USUARIOS

- Registro e inicio de sesión de usuarios.

- Diferenciación entre roles de administrador y cliente.

* **LOGIN**
* **REGISTRO**
* **DASHBOARD DE CLIENTES (se ha creado un nuevo usuario para realizar las capturas)**



* **DASHBOARD DEL ADMINISTRADOR**

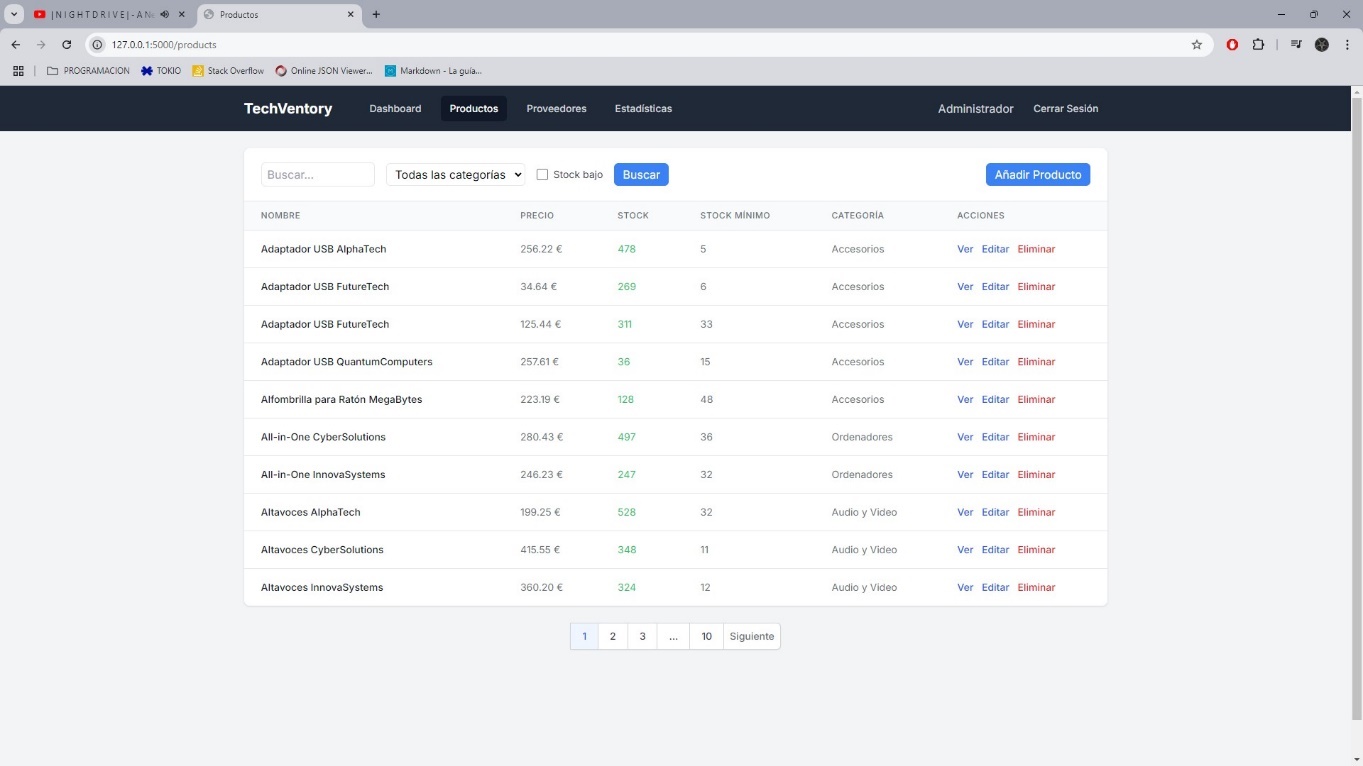
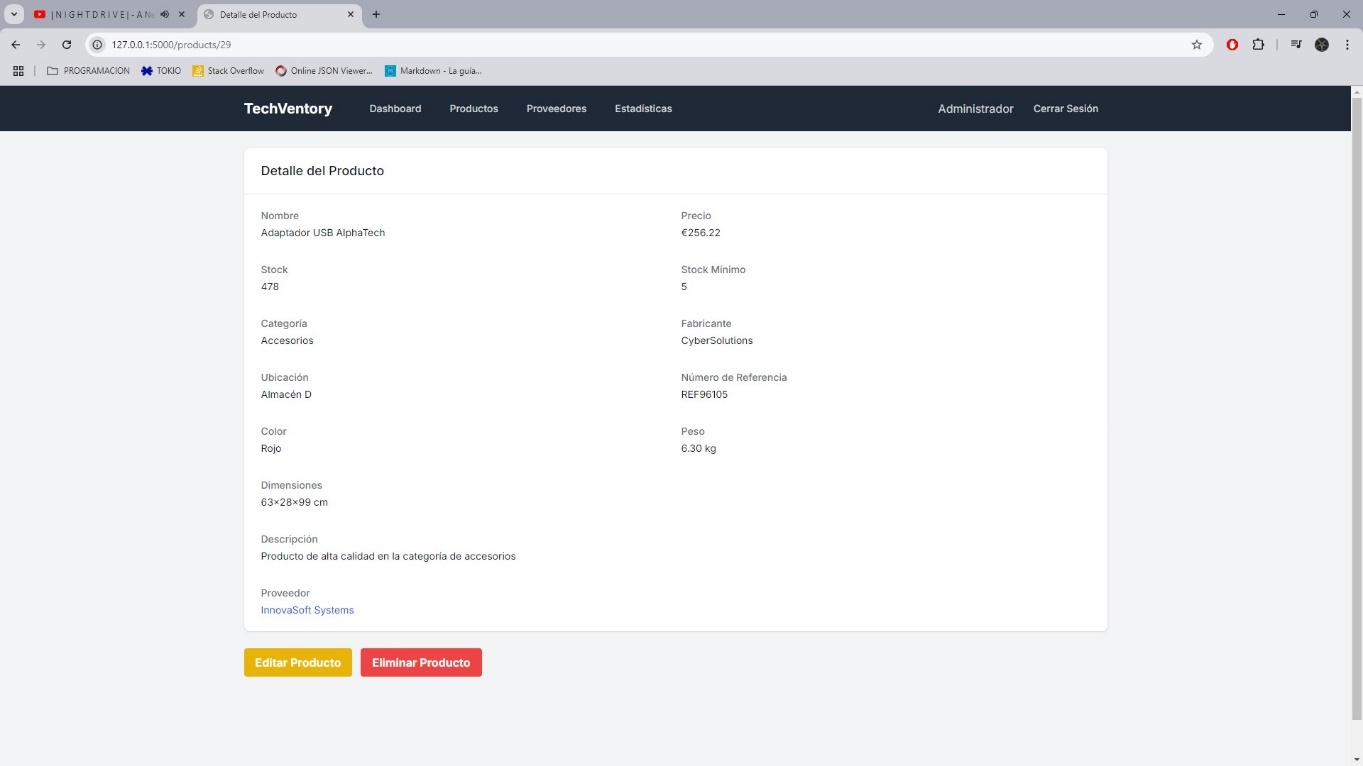
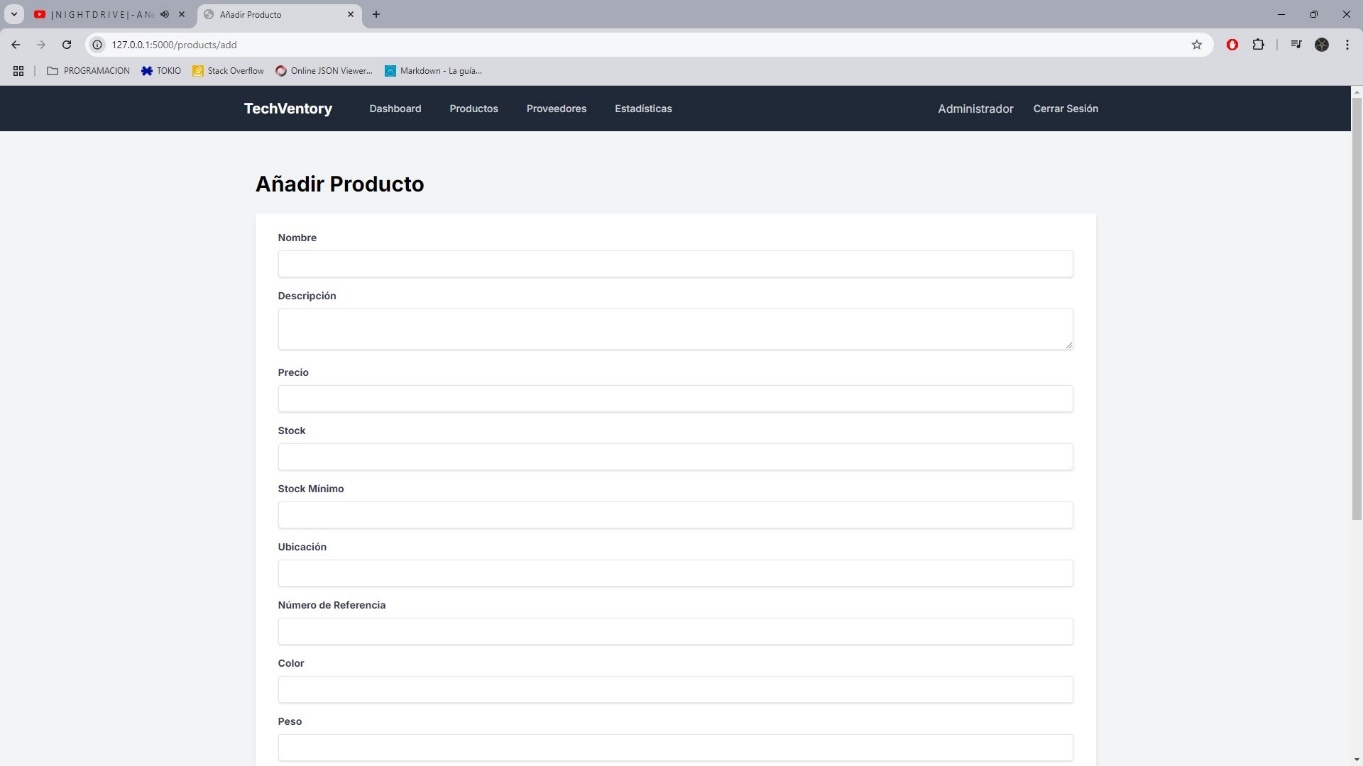
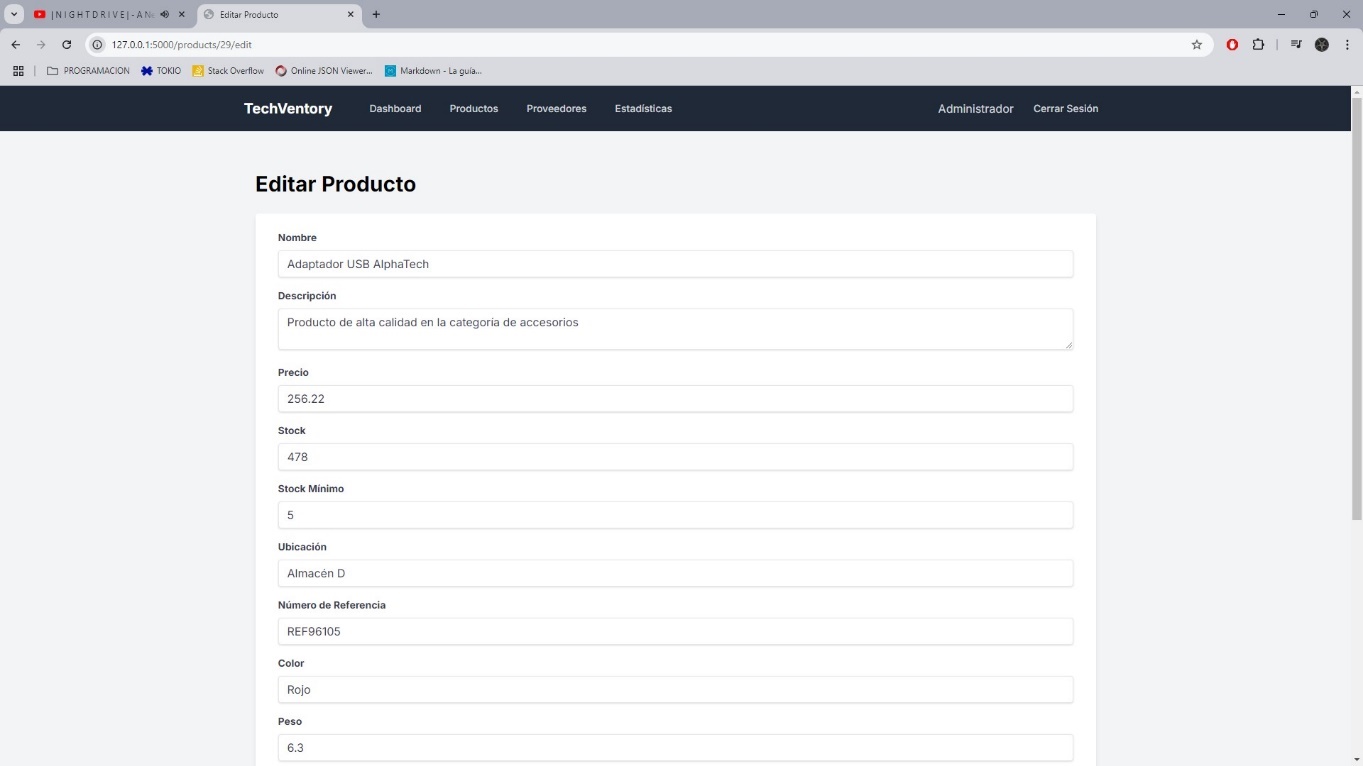
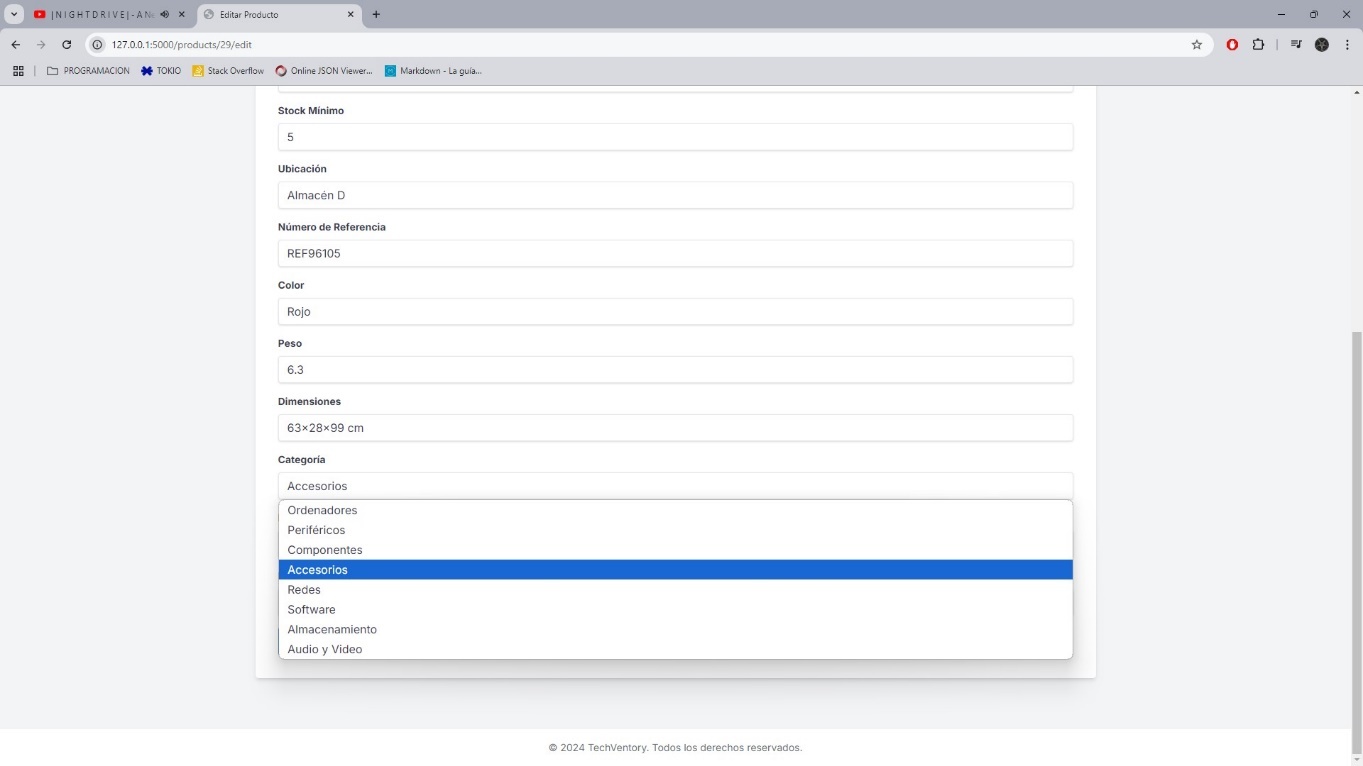
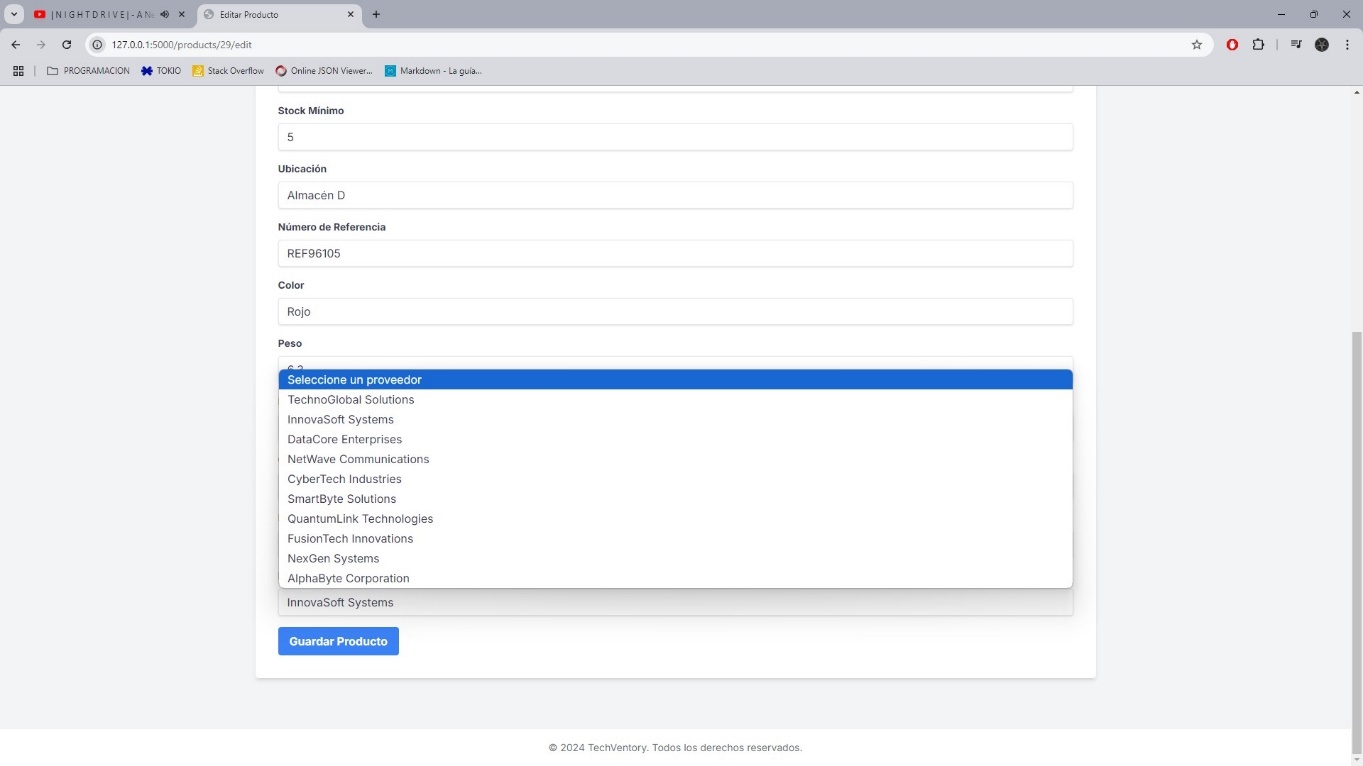
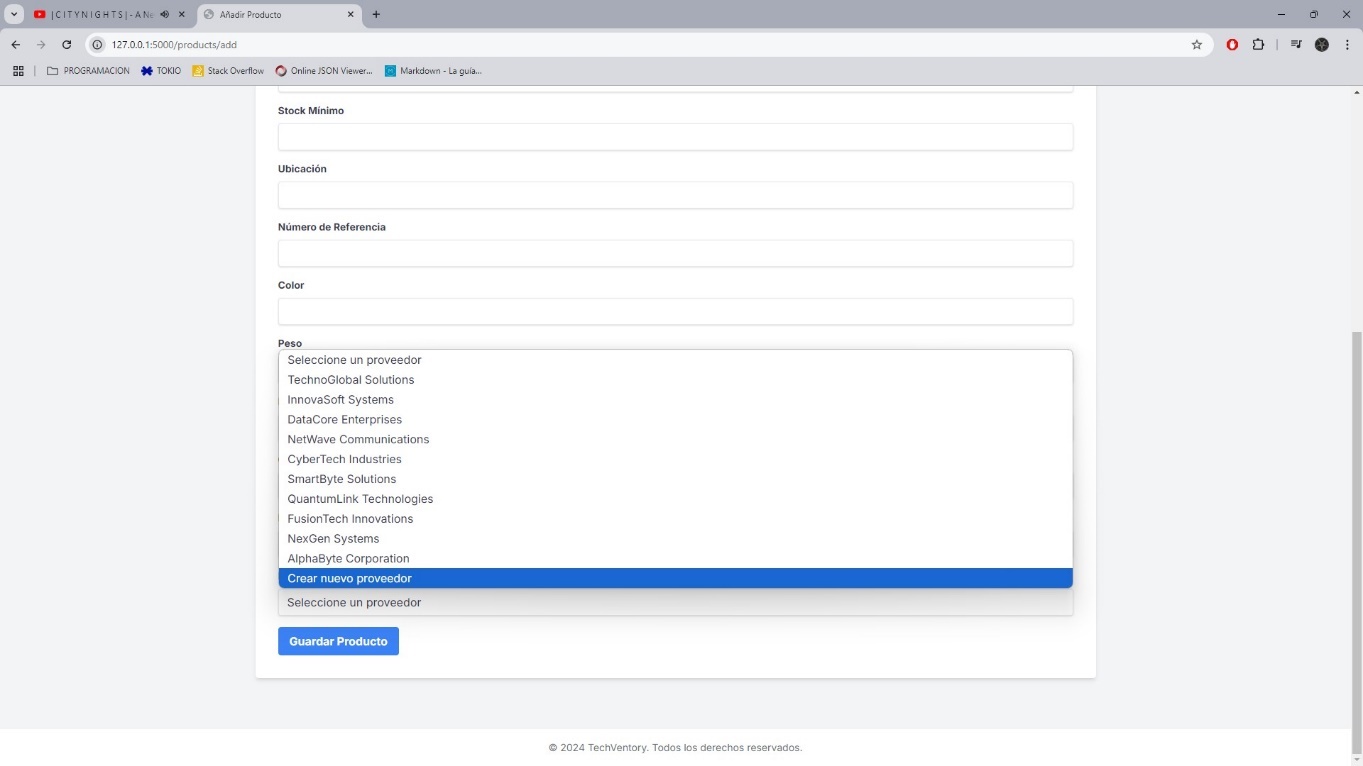
B) GESTIÓN DE PRODUCTOS

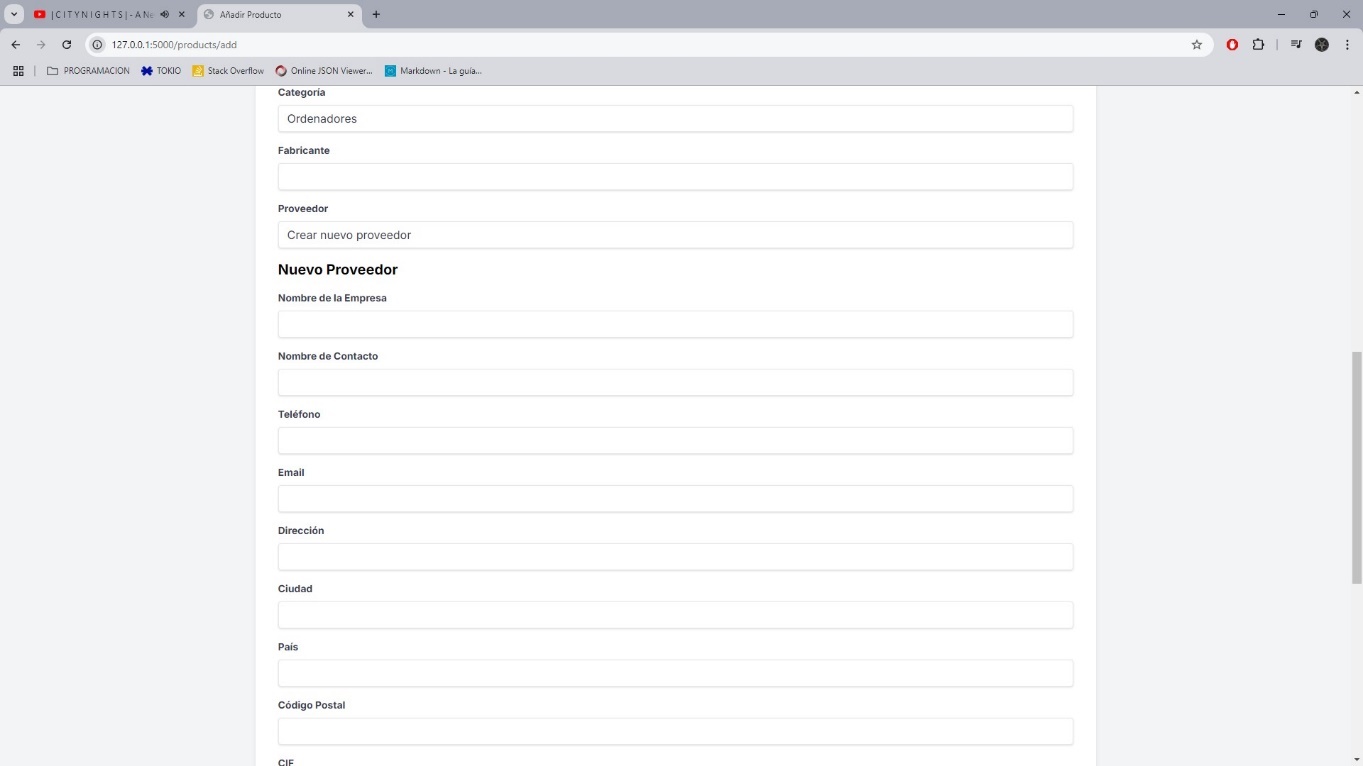
- CRUD (Crear, Leer, Actualizar, Eliminar) de productos.

- Asignación de productos a categorías.

- Seguimiento de niveles de stock.

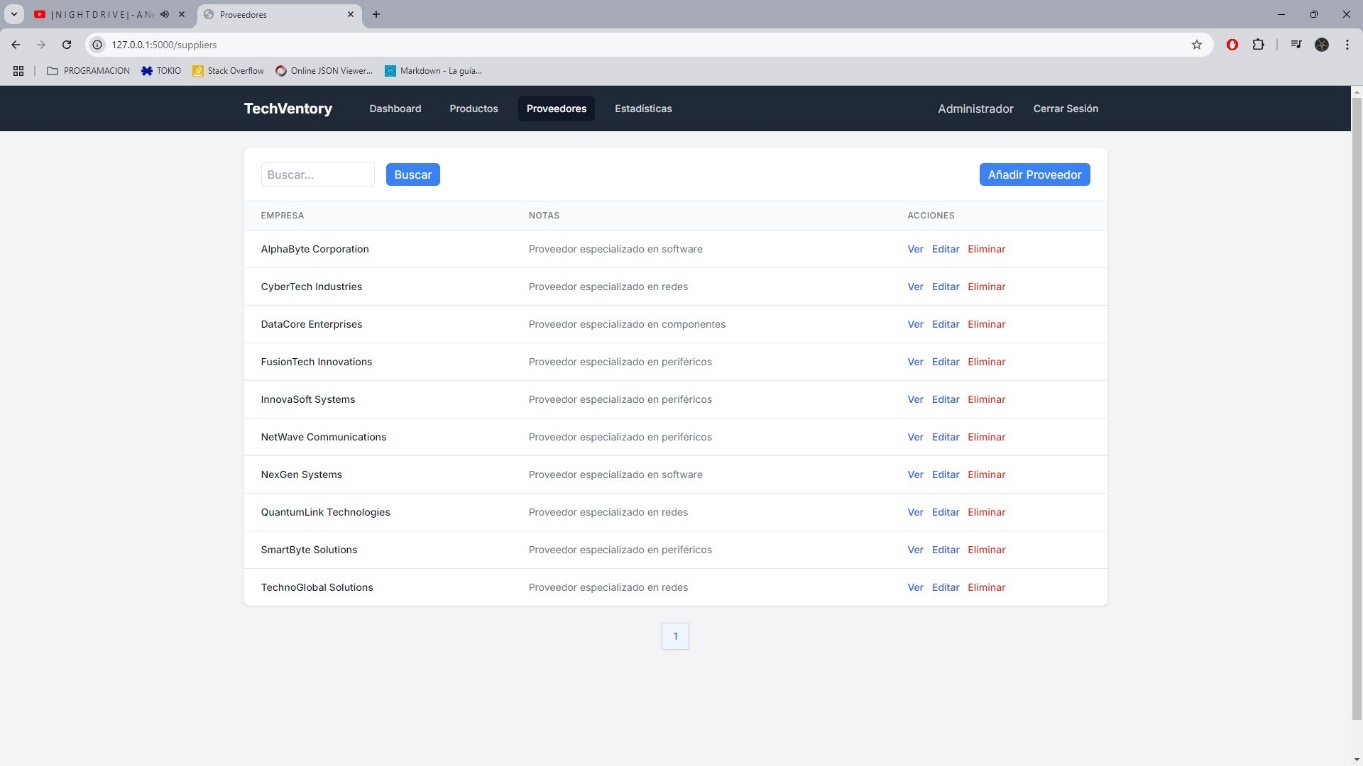
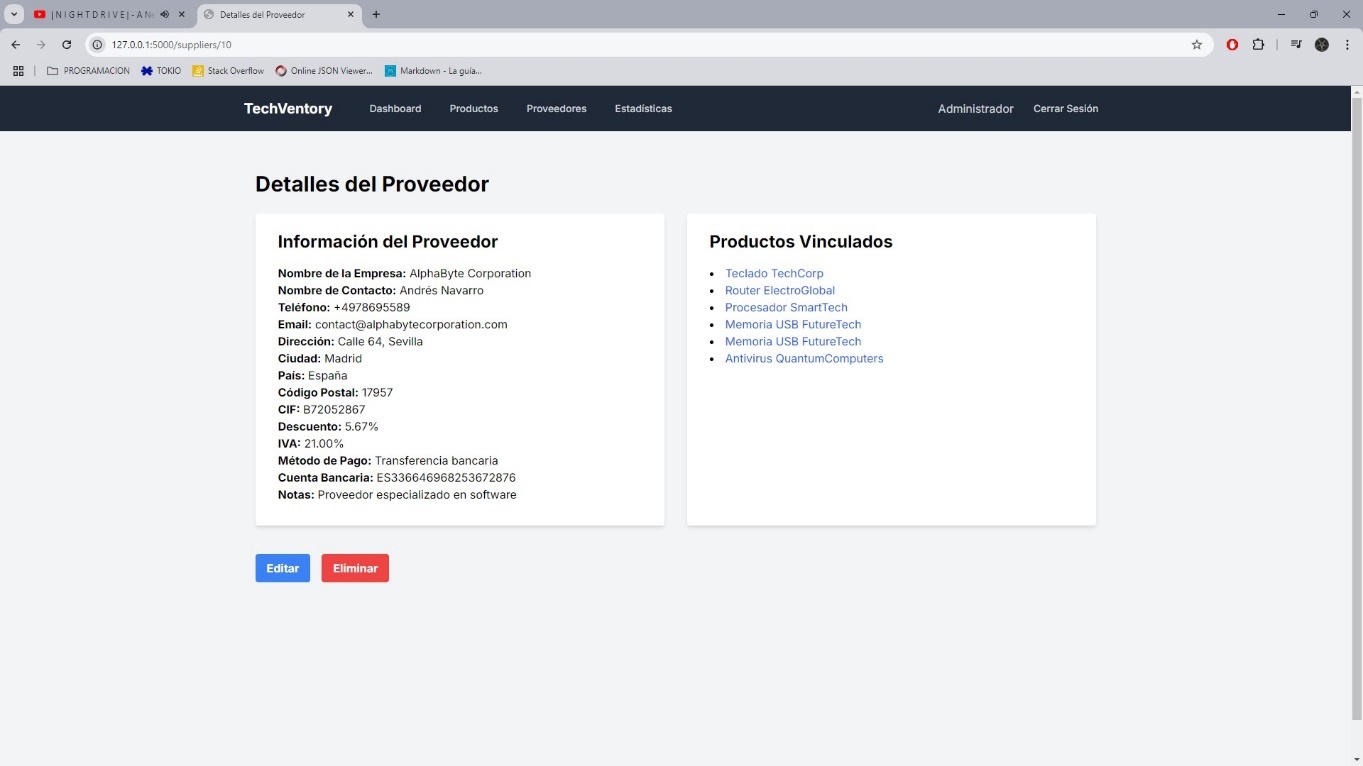
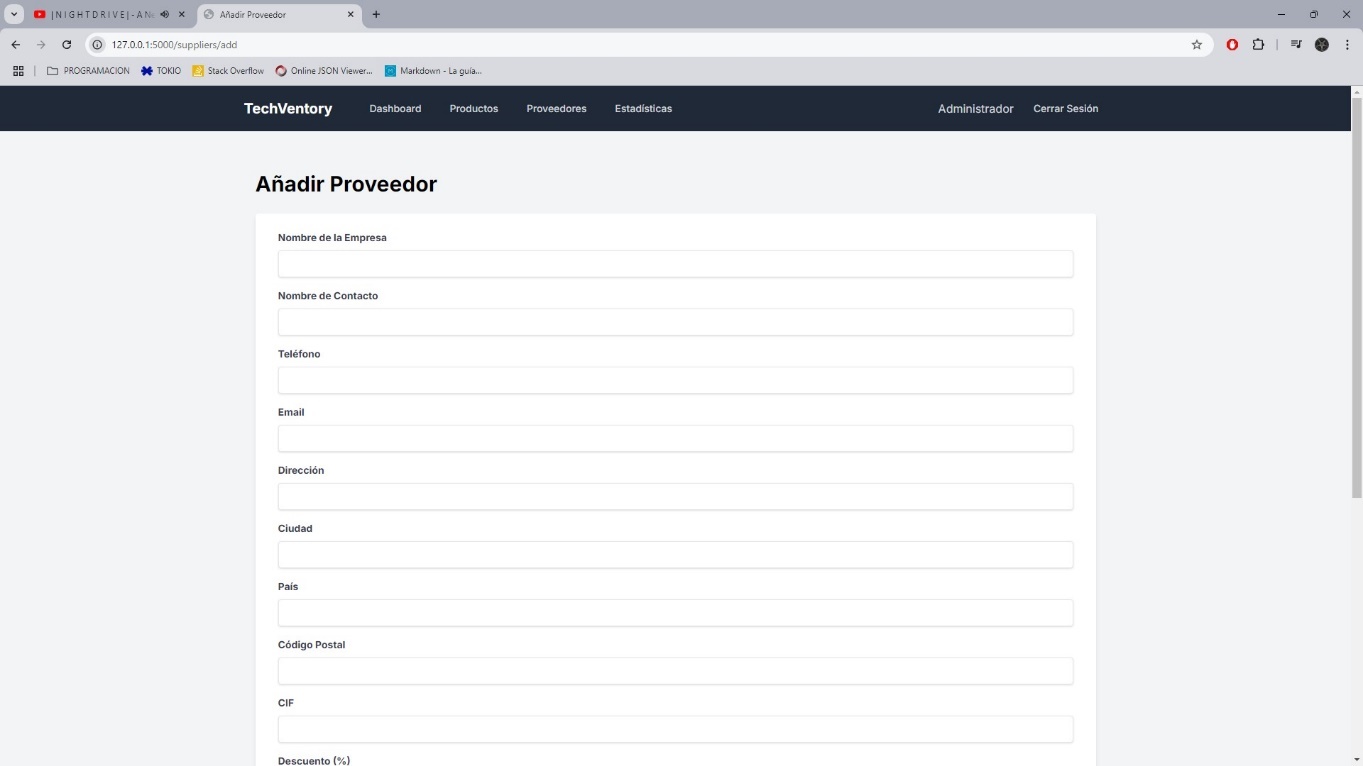
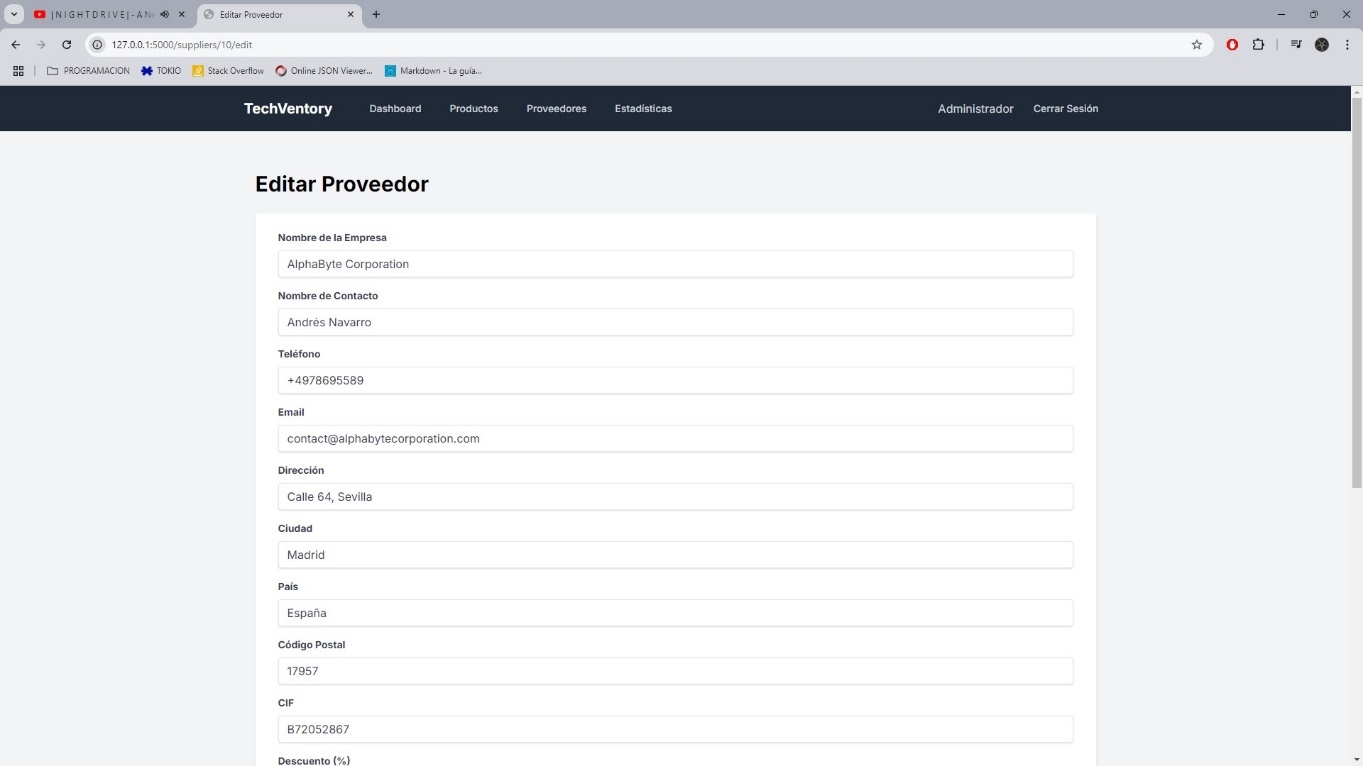
- Asociación de productos con proveedores.

* **LISTADO DE PRODUCTOS (desde el dashboard del administrador). Aquí puede hacerse un seguimiento del stock, la creación de un nuevo producto, editar un producto ya existente, o eliminarlo.**
* **VISTA DETALLADA DE LOS PRODUCTOS (desde aquí puede verse en detalle un producto, editarlo, y eliminarlo).**
* **CREACIÓN DE PRODUCTOS (sólo disponible para el administrador)**
* **EDICIÓN DE PRODUCTOS (solo disponible para el administrador)**
* **ASIGNACIÓN A PRODUCTOS DE CATEGORÍAS (tanto al editar como al crear un producto)**
* **ASOCIACIÓN DE PRODUCTOS CON PROVEEDORES (puede realizarse tanto al editar un producto como al crear uno nuevo, además, el proveedor asociado es visible desde el listado de productos y accesible desde la vista detallada de un producto). También al crear un nuevo producto se tiene la opción de asociarlo a un nuevo proveedor, de esa manera se crean ambos elementos (producto y proveedor) ya asociados entre sí.**
* **ASOCIACIÓN DE UN NUEVO PRODUCTO CON UN NUEVO PROVEEDOR**

****

C) GESTIÓN DE PROVEEDORES

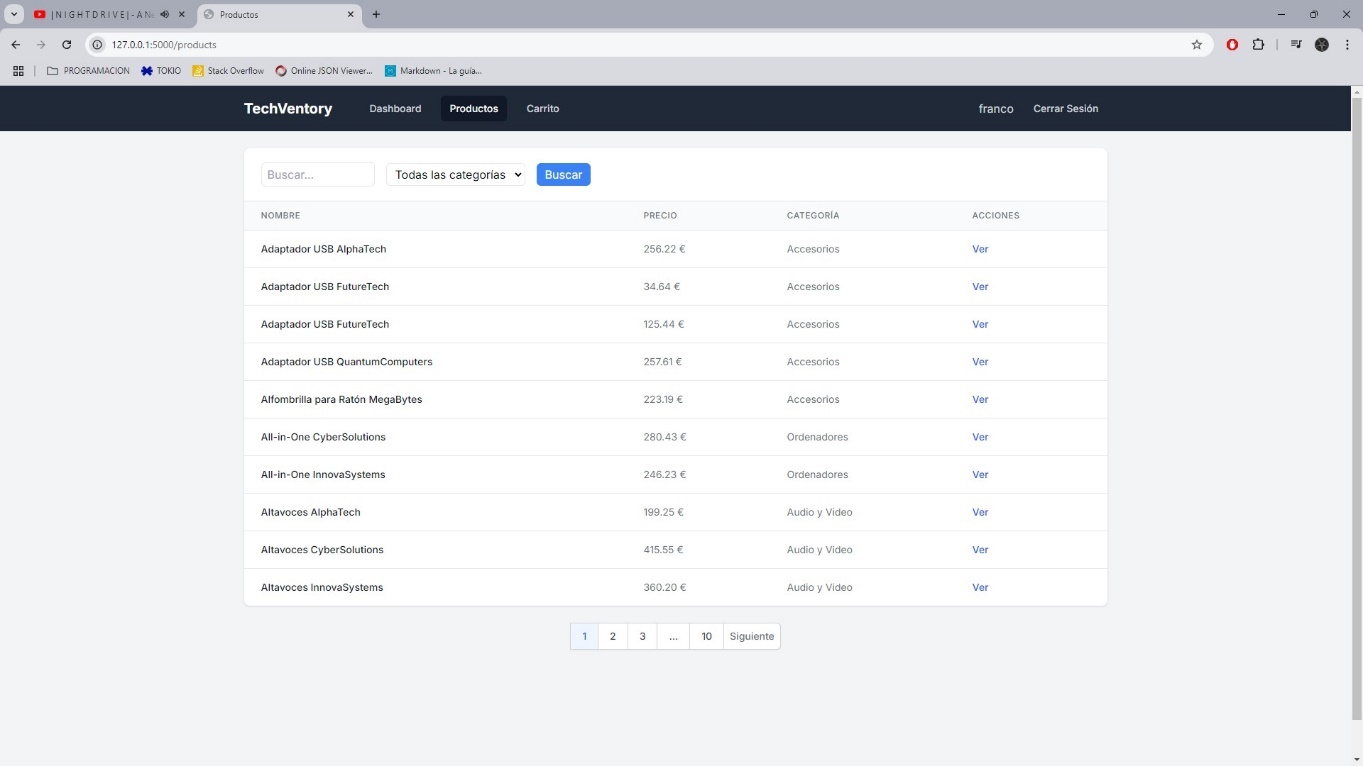
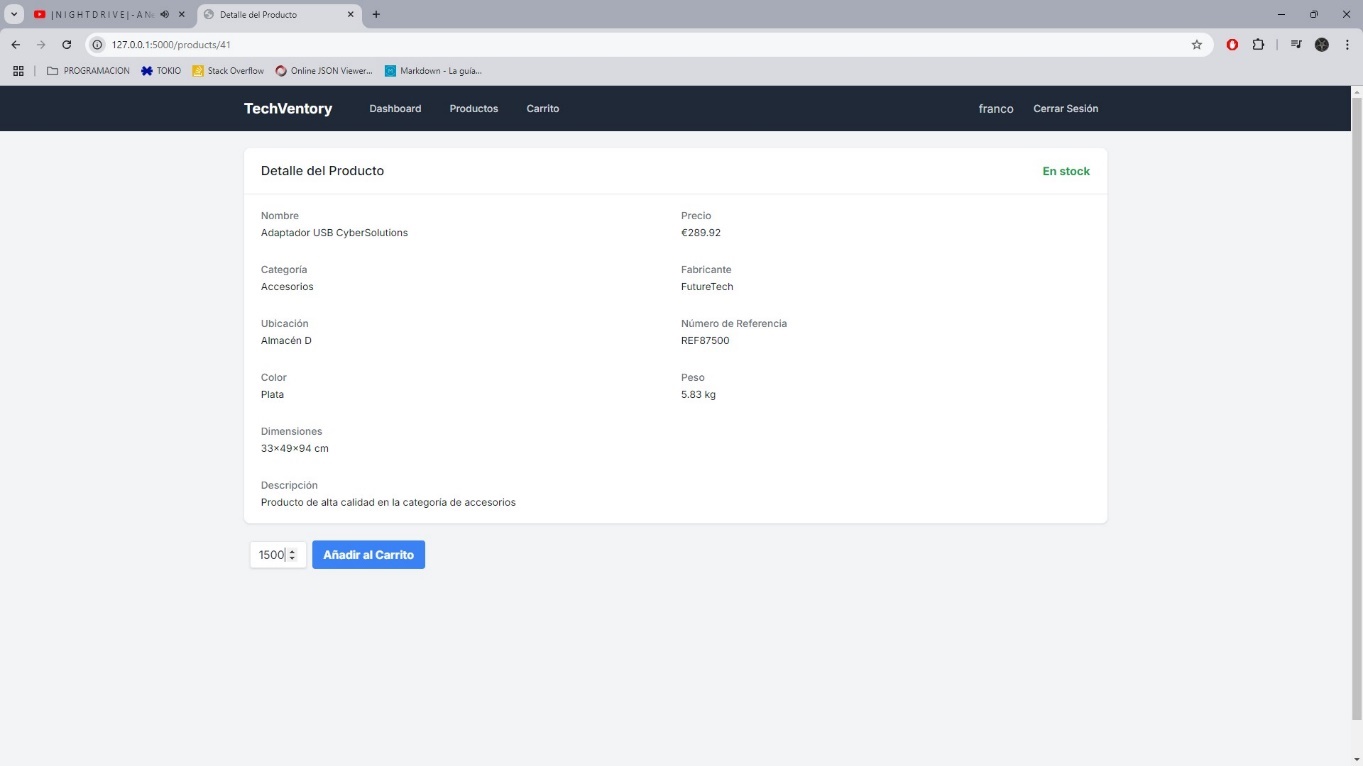
- CRUD (Crear, Leer, Actualizar, Eliminar) de proveedores.

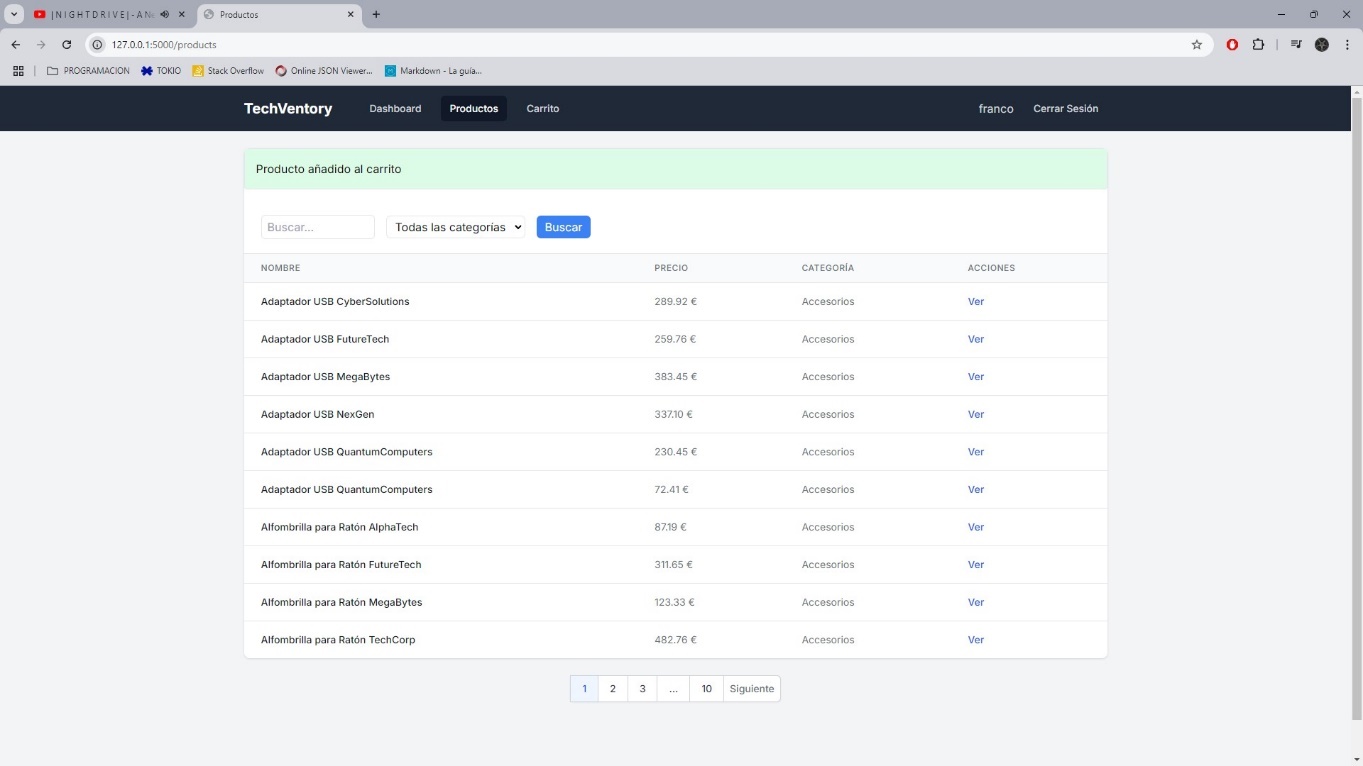
* **LISTADO DE PROVEEDORES (solo disponible para el administrador). Aquí puede realizarse un seguimiento de los proveedores, la creación de uno nuevo, editar uno ya existente, o eliminarlo.**
* **VISTA DETALLADA DEL PROVEEDOR (desde aquí puede verse en detalle un proveedor, editarlo, y eliminarlo).**
* **CREACIÓN DE UN NUEVO PROVEEDOR**
* **EDICIÓN DE PROVEEDORES**

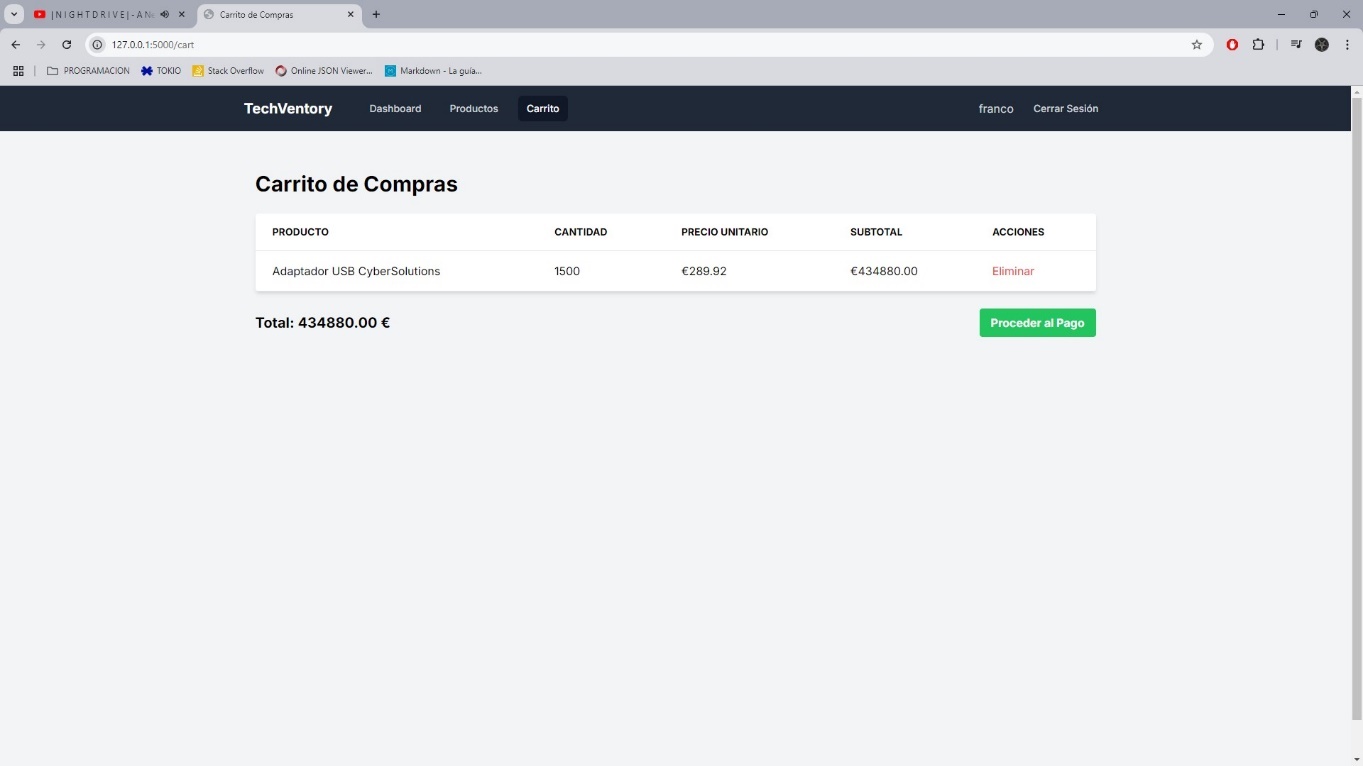
D) COMPRAS

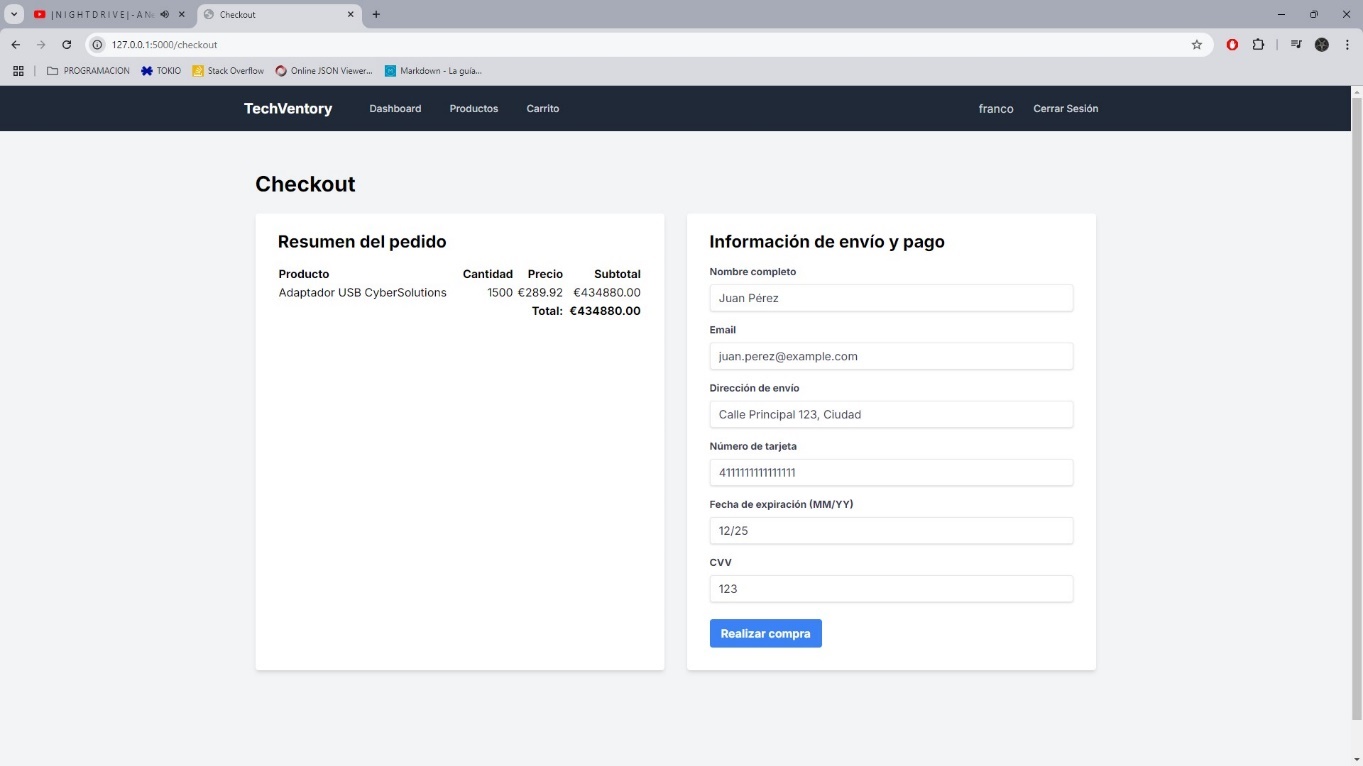
- Creación y procesamiento de órdenes de compra.

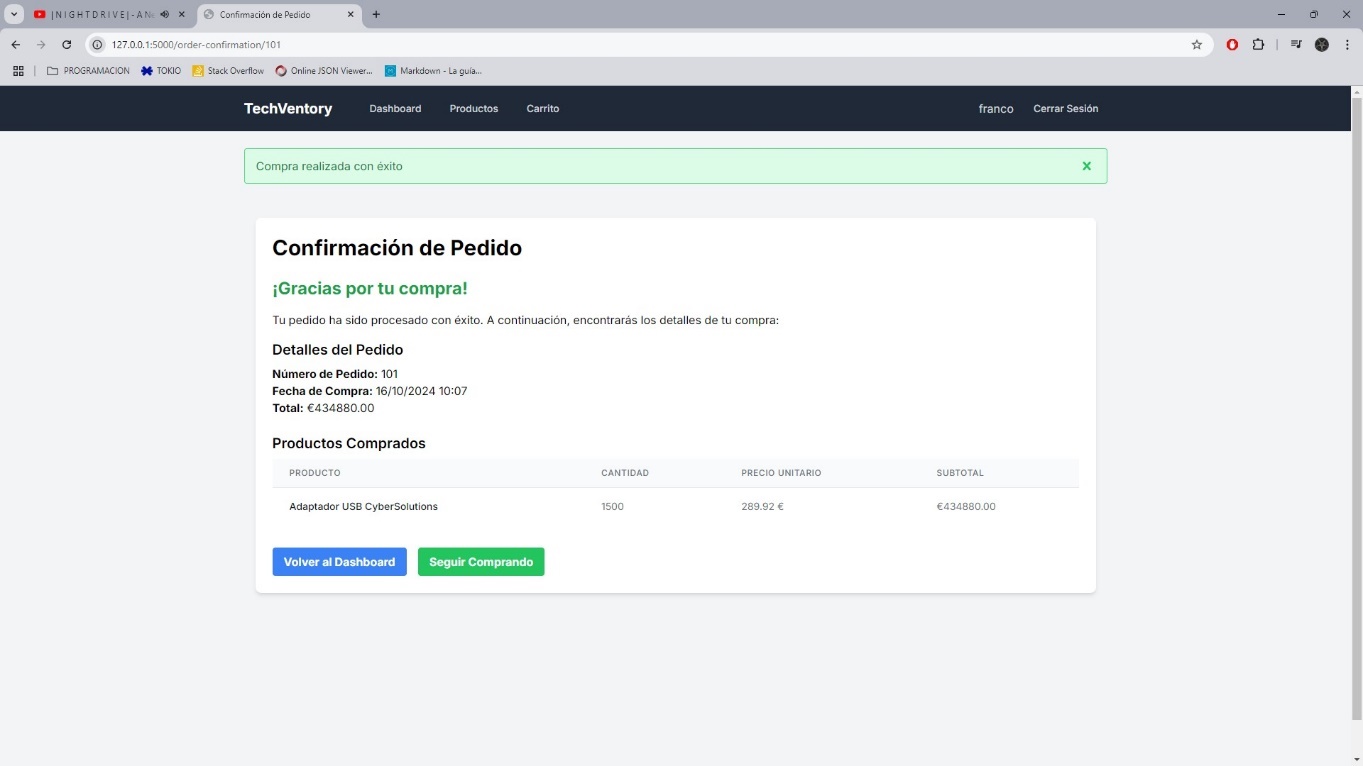
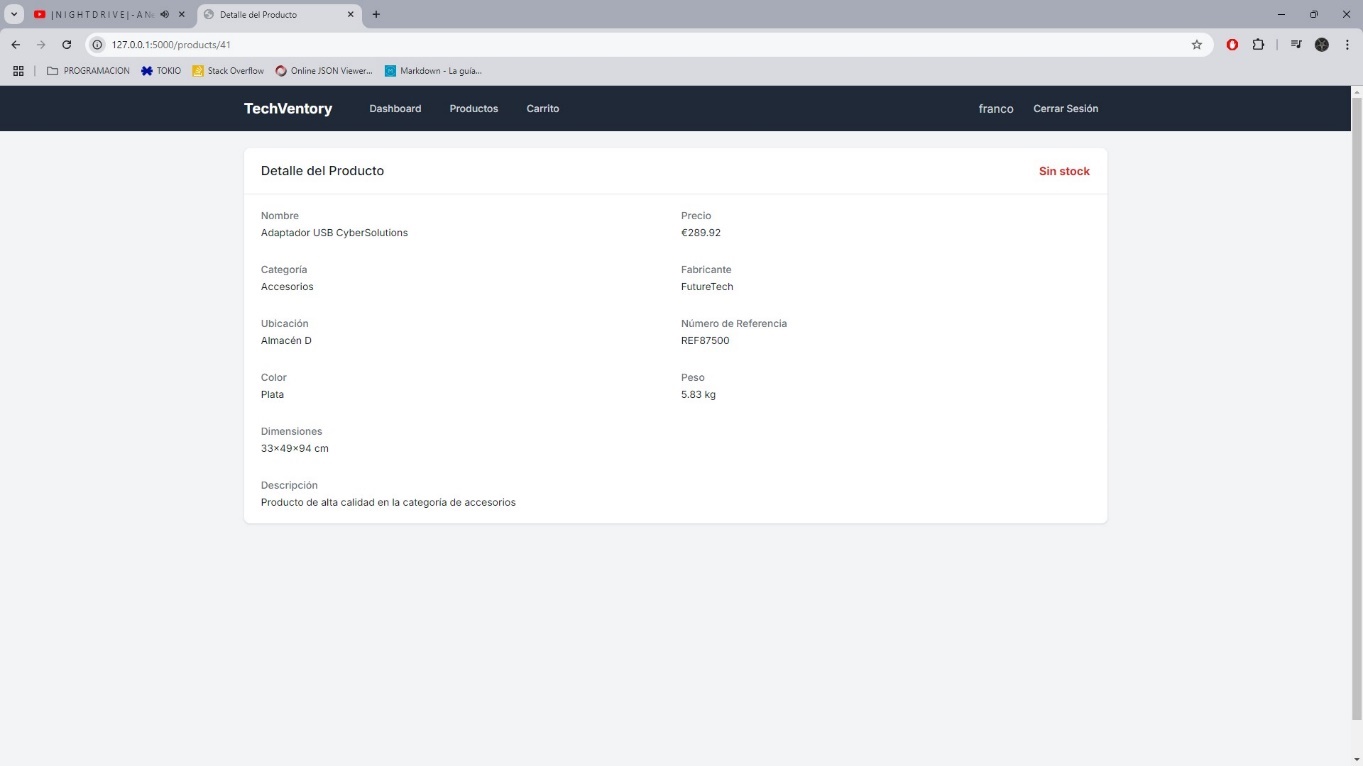
- Actualización automática de niveles de stock tras una compra.

* **PARA REALIZAR UNA COMPRA EL CLIENTE SE DIRIGIRÁ HACIA EL LISTADO DE PRODUCTOS**
* **LUEGO ACCEDERÁ AL PRODUCTO QUE QUIERA COMPRAR (en este caso seleccionamos el primer producto de la lista, y debajo introducimos la cantidad)**
* **LUEGO PRESIONARÁ EN ‘AÑADIR AL CARRITO’ PARA PREPARAR LA COMPRA (el cliente puede agregar al carrito cuantos productos y unidades quiera, siempre que haya stock)**



* **UNA VEZ FINALIZADA LA PREPARACION DE LA COMPRA, SE ACCEDERA A ‘CARRITO’ PARA PROCEDER AL PAGO (aquí pueden eliminarse productos de la lista si se desea)**
* **EL SIGUIENTE PASO ES COMPLETAR EL FORMULARIO DEL CHECKOUT PARA FINALIZAR EL PAGO (el formulario viene completo por defecto con información ficticia, puede cambiarse si se desea)**

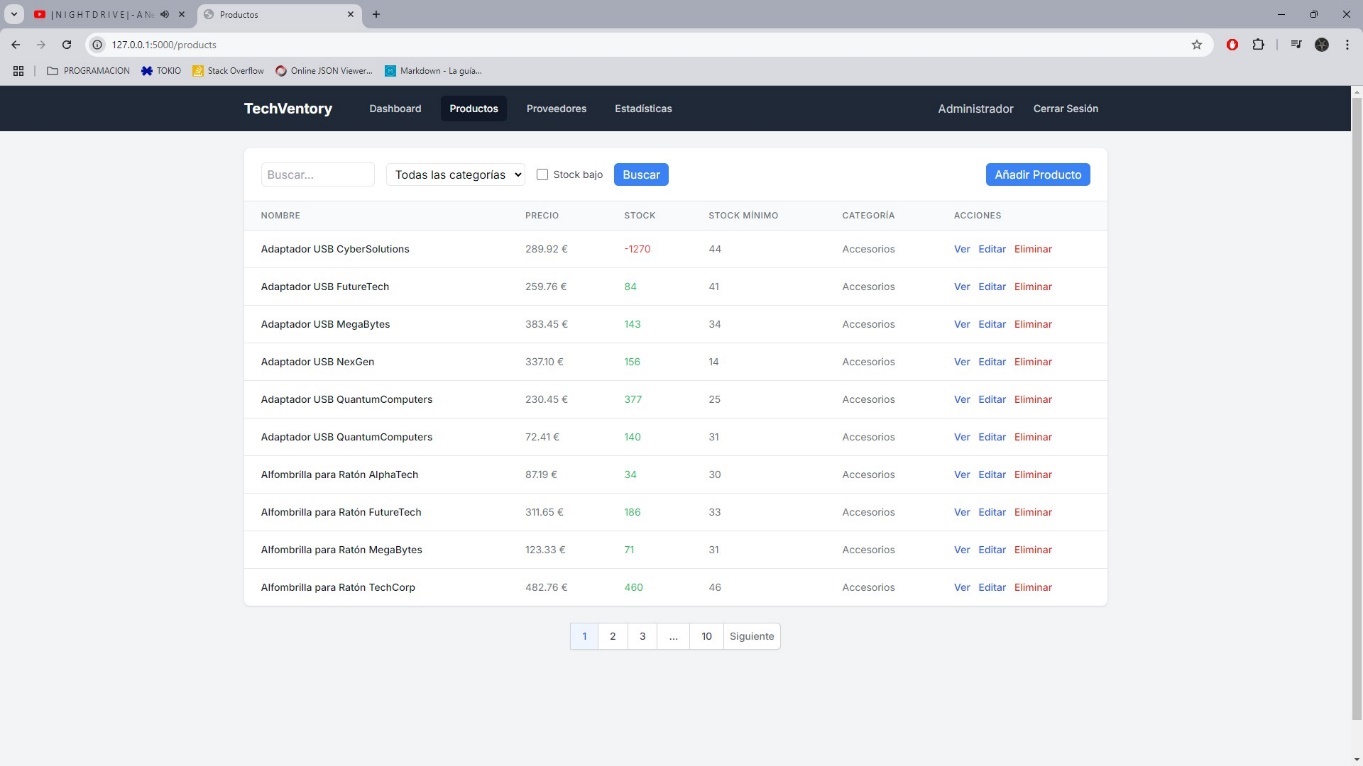
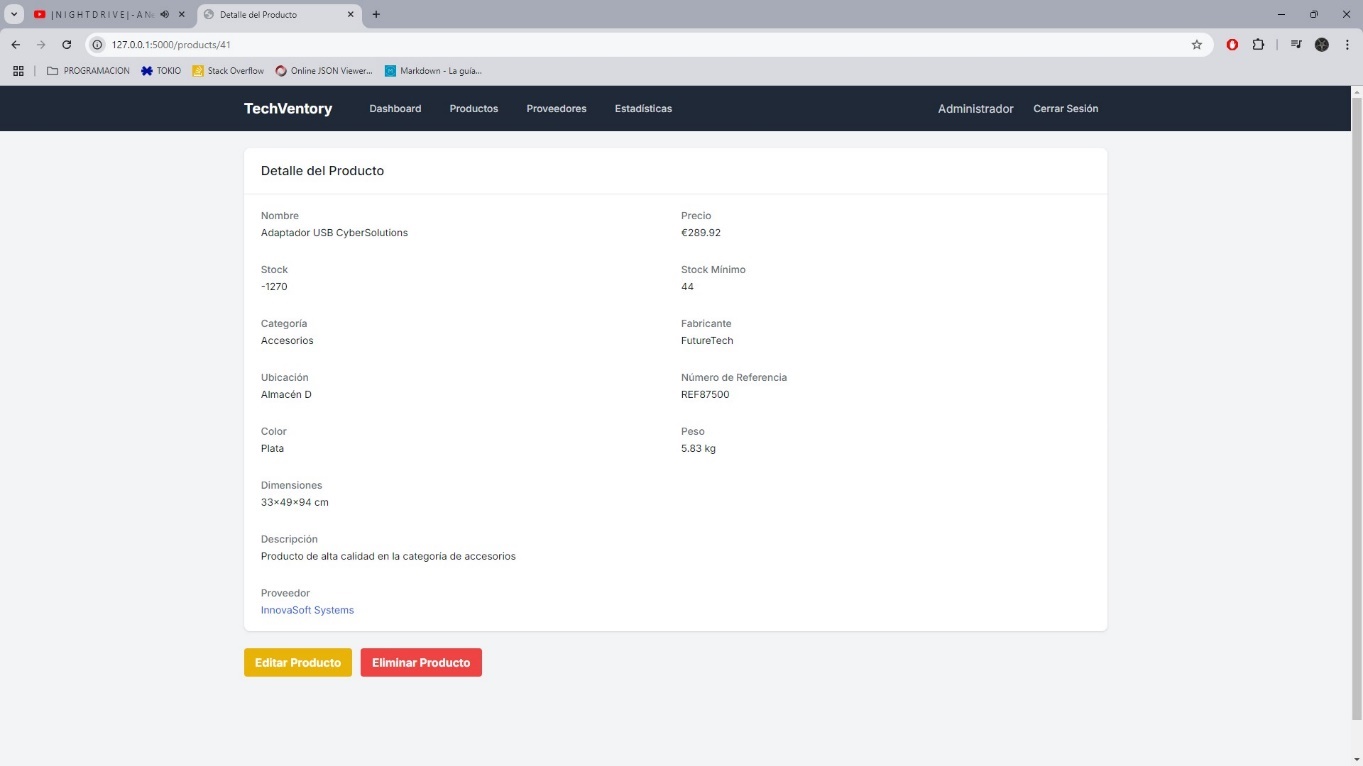


* **UNA VEZ SE HAYA COMPLETADO LA COMPRA, ACCEDEREMOS A LA CONFIRMACION DEL PEDIDO, EN DONDE PODREMOS OBSERVAR DETALLES COMO EL NUMERO DEL PEDIDO.**
* **COMO SE PUEDE OBSERVAR, AL REALIZAR LA COMPRA DE UN PRODUCTO Y LUEGO ACCEDER NUEVAMENTE A EL, VEMOS MODIFICADO EL STOCK EN EL INVENTARIO (en este caso se ha agotado el stock del producto comprado)**

E) Ventas

- Creación y procesamiento de órdenes de venta.

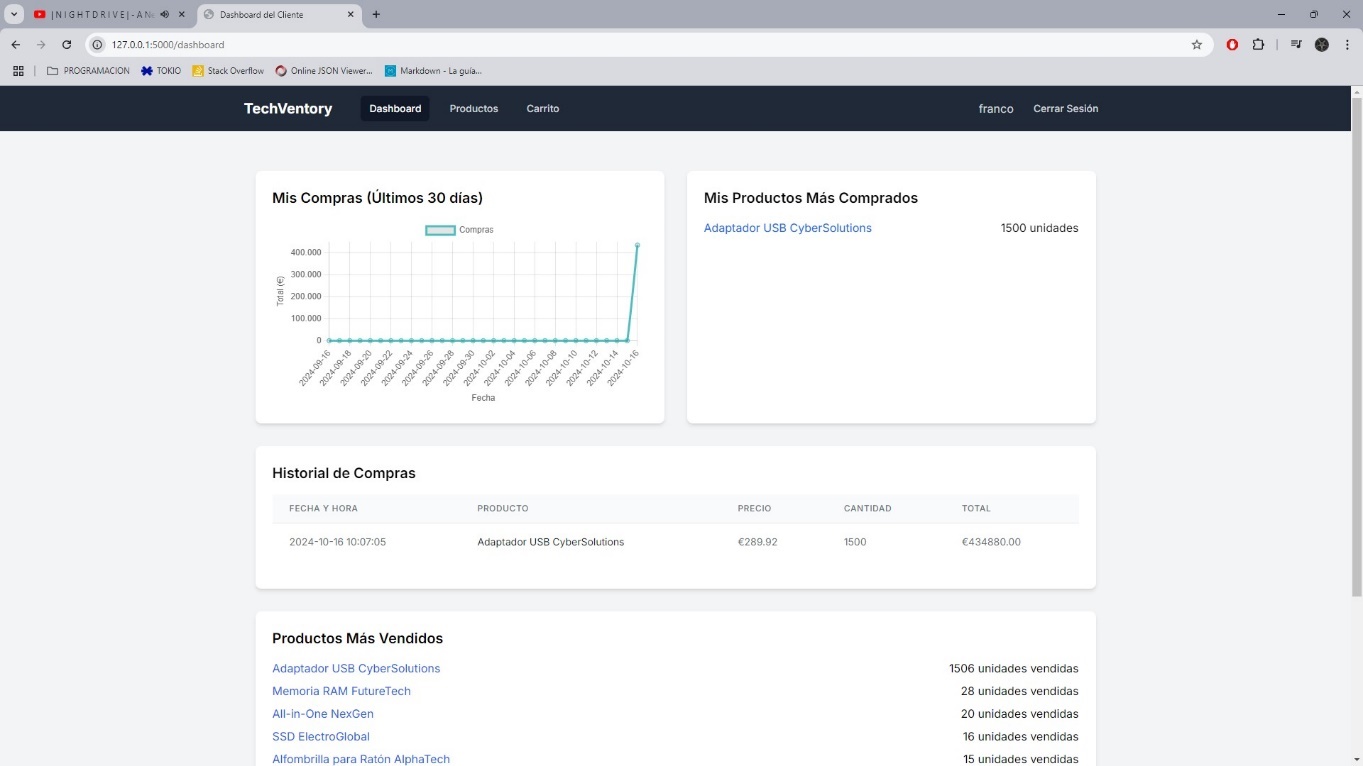
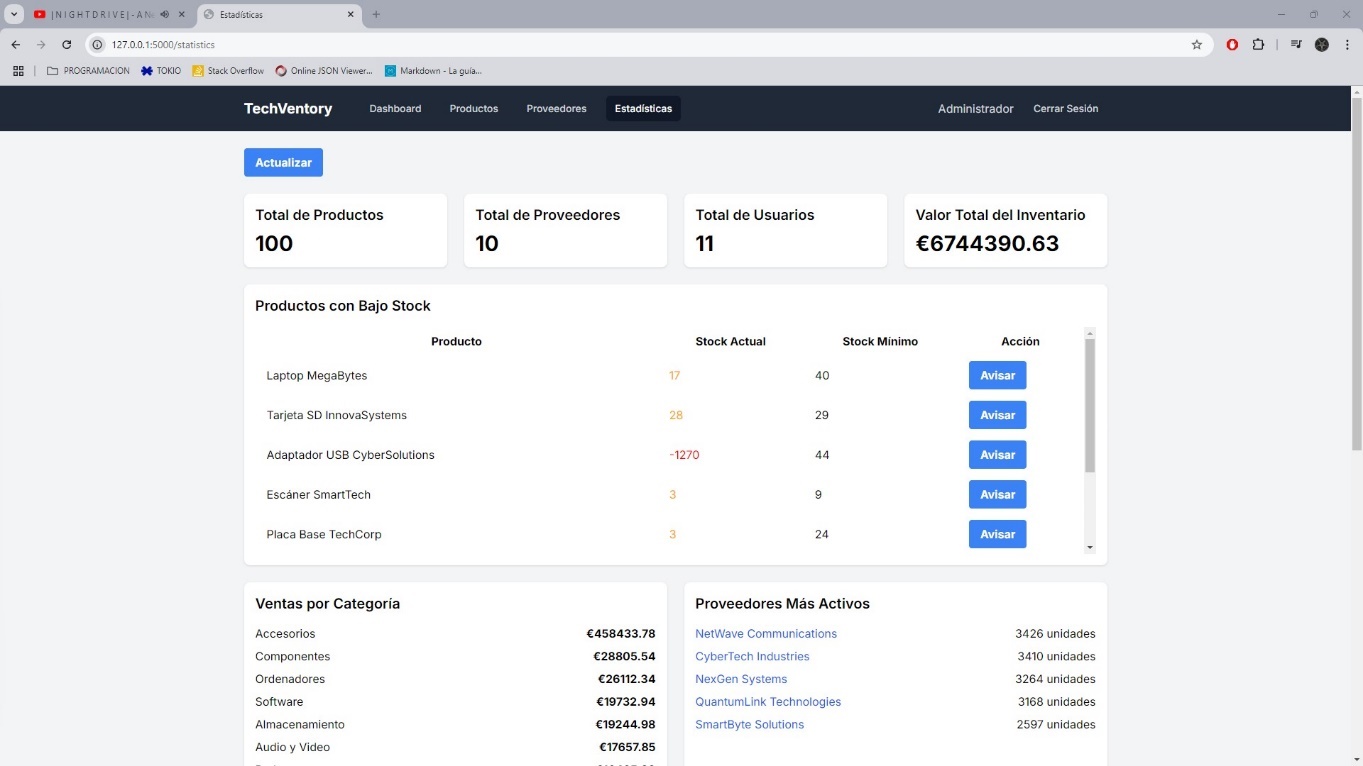
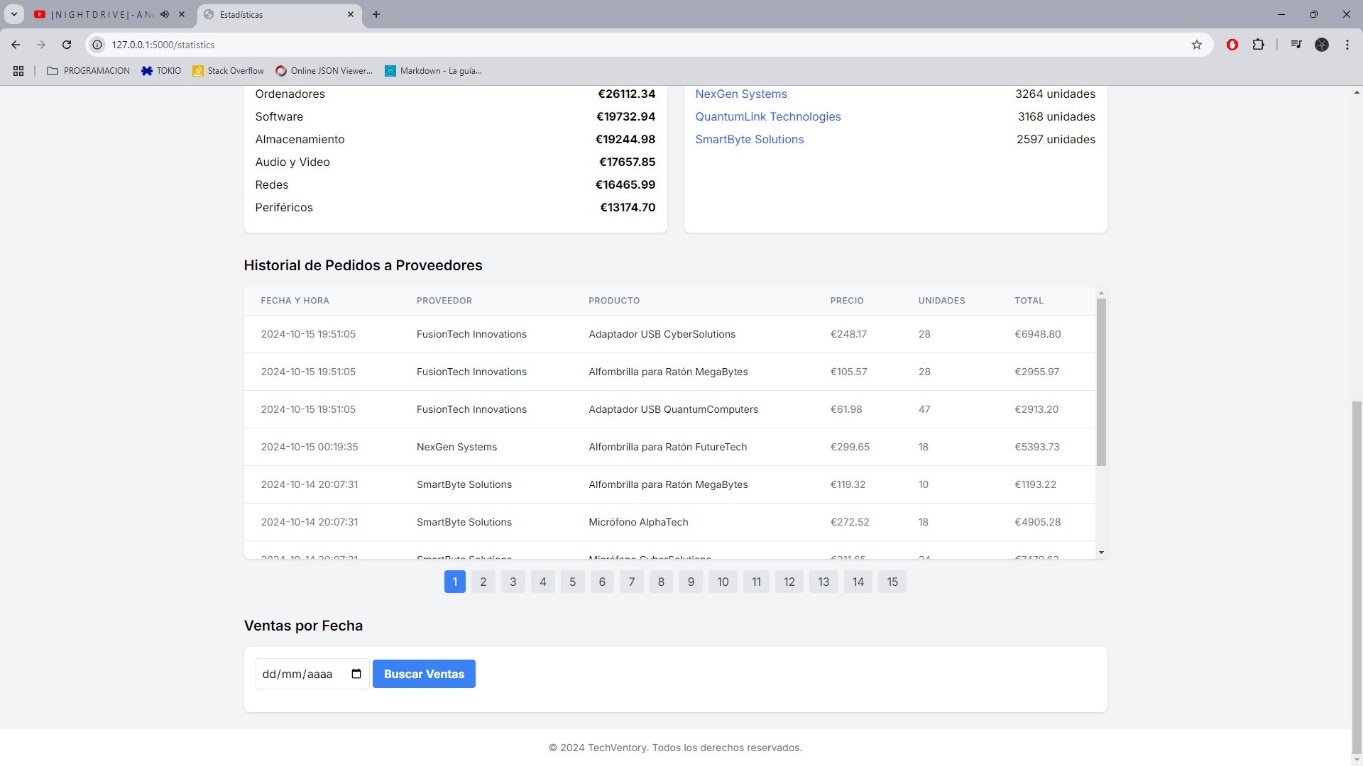
- Actualización automática de niveles de stock tras una venta.

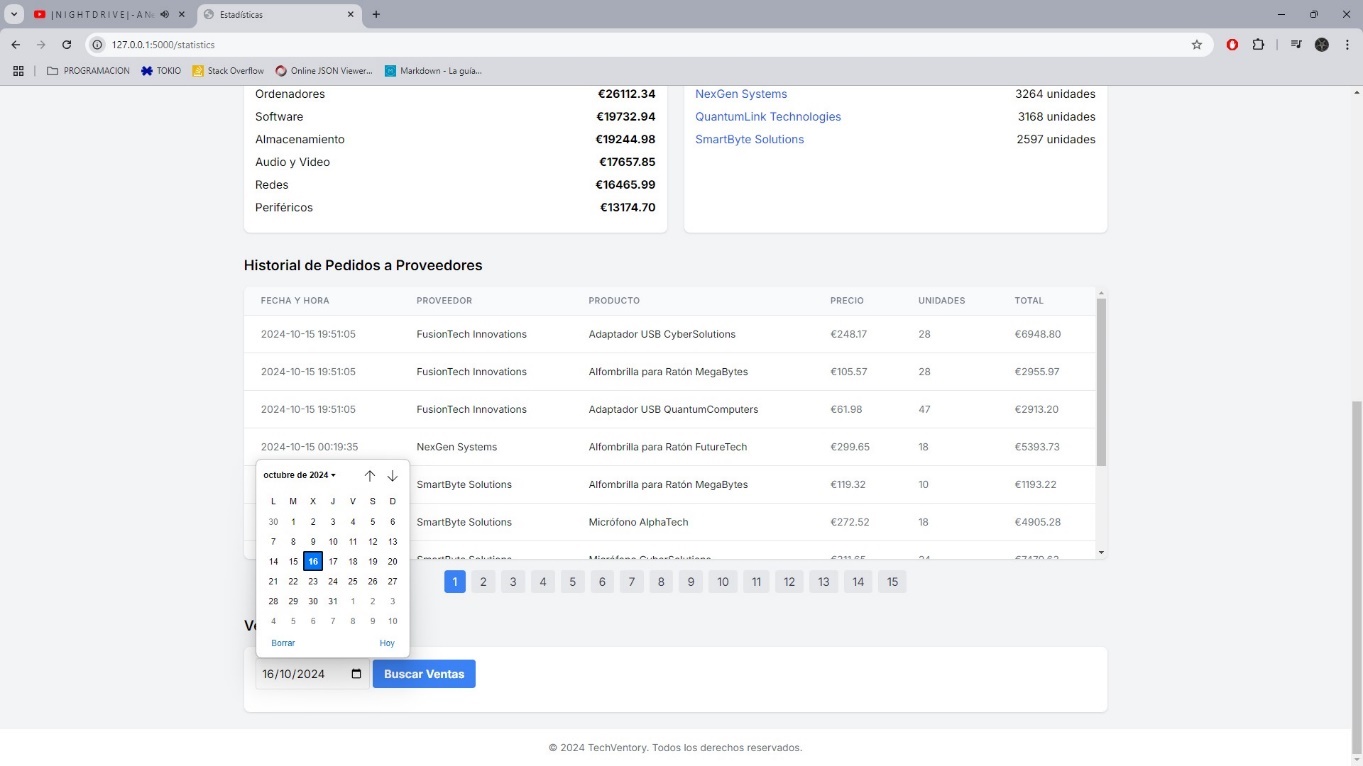
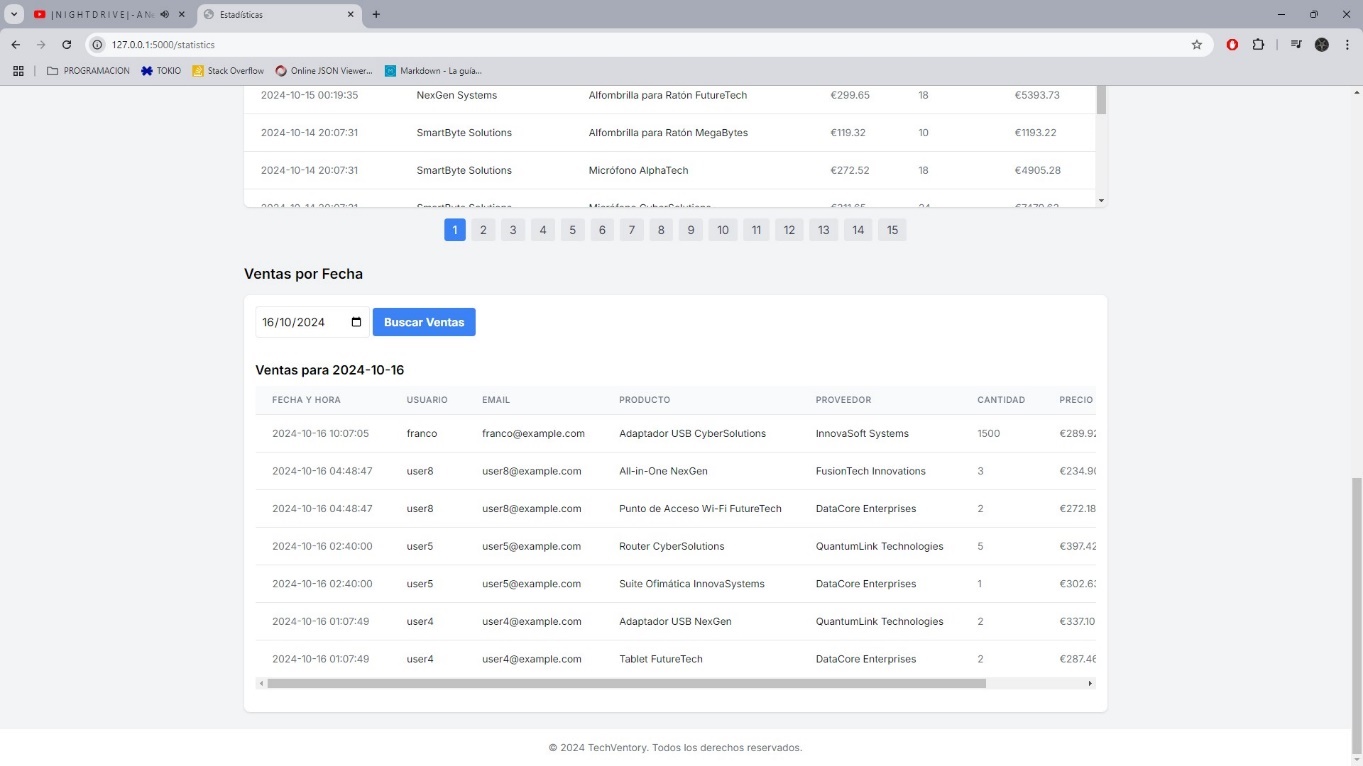
* **LUEGO DEL EJEMPLO ANTERIOR (compra) PODEMOS OBSERVAR DESDE EL DASHBOARD DEL ADMINISTRADOR UN INFORME DETALLADO DE LAS VENTAS (en este caso, el producto comprado ha pasado a ser el producto más vendido, el producto más rentable, y ha modificado las gráficas)**
* **TAMBIEN PODEMOS OBSERVAR DESDE EL LISTADO DE PRODUCTOS DEL ADMINISTRADOR LA MODIFICACION DEL STOCK EN EL INVENTARIO**
* **TAMBIEN PUEDE OBSERVARSE LA MODIFICACION DEL STOCK EN EL INVENTARIO DESDE DETALLES DEL PRODUCTO**

F) REPORTES Y ANÁLISIS

- Generación de informes de ventas, compras e inventario.

- Visualización de datos mediante gráficos.

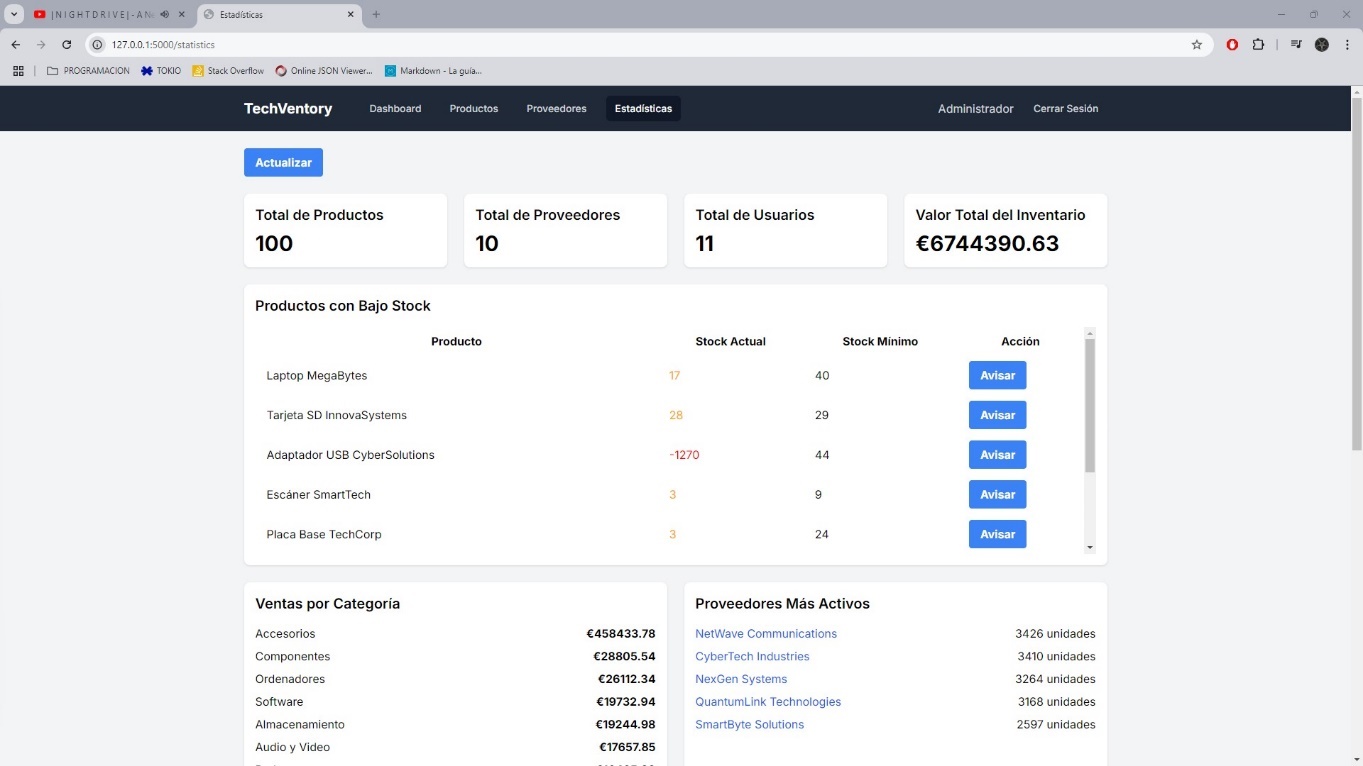
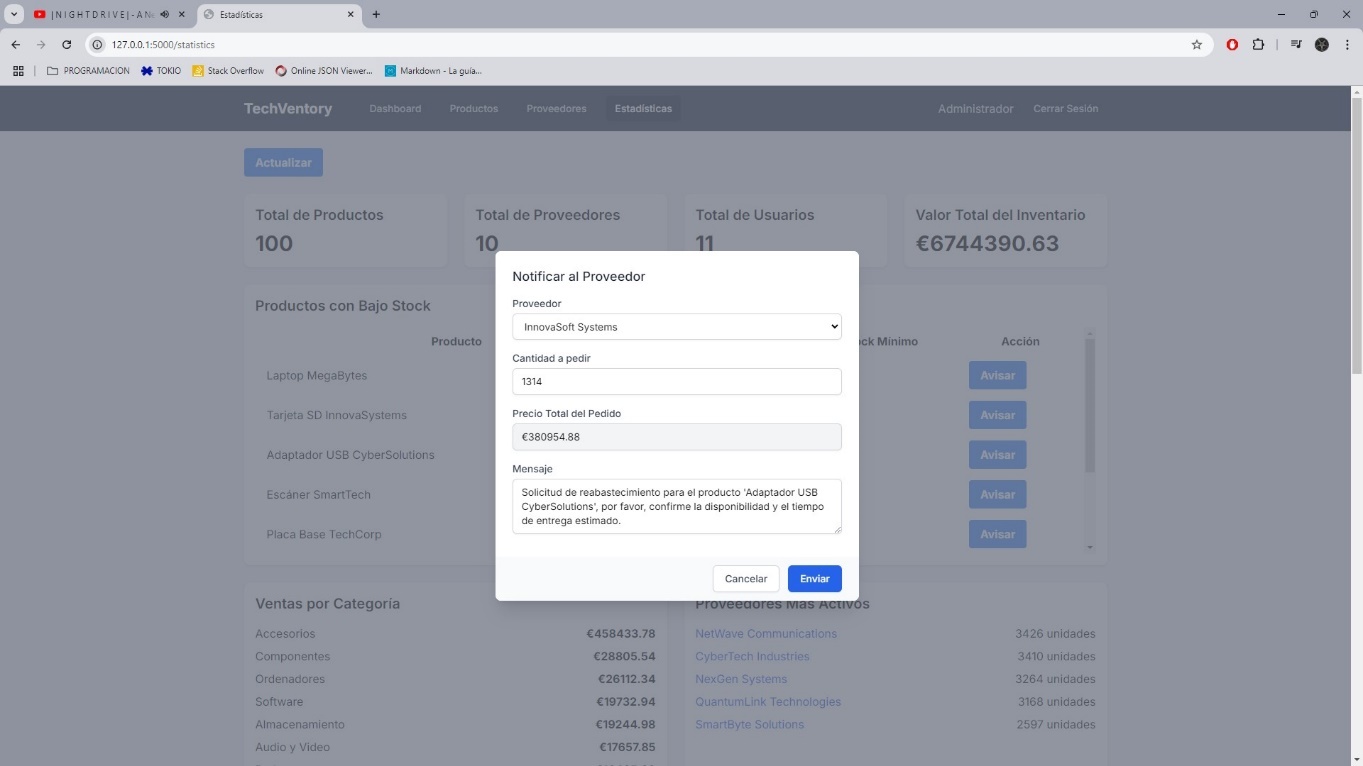
* **DESDE EL DASHBOARD DEL CLIENTE PUEDE OBSERVARSE UN REPORTE DE LOS PRODUCTOS MAS COMPRADOS POR EL USUARIO, UNA GRÁFICA INDICATIVA Y UN HISTORIAL DE COMPRAS**
* **DESDE EL DASHBOARD DEL ADMINISTRADOR PUEDE OBSERVARSE COMO HA AUMENTADO EL VALOR TOTAL DEL INVENTARIO, COMO SE HAN VISTO MODIFIICADAS LAS VENTAS POR CATEGORIA, Y COMO SE HA PRODUCIDO UN STOCK BAJO DEL PRODUCTO QUE SE HA COMPRADO**
* **TAMBIEN PODEMOS OBSERVAR UN HISTORIAL DE VENTAS EN EL QUE SE VE REGISTRADA EN UN CALENDARIO CADA VENTA EN DETALLE**



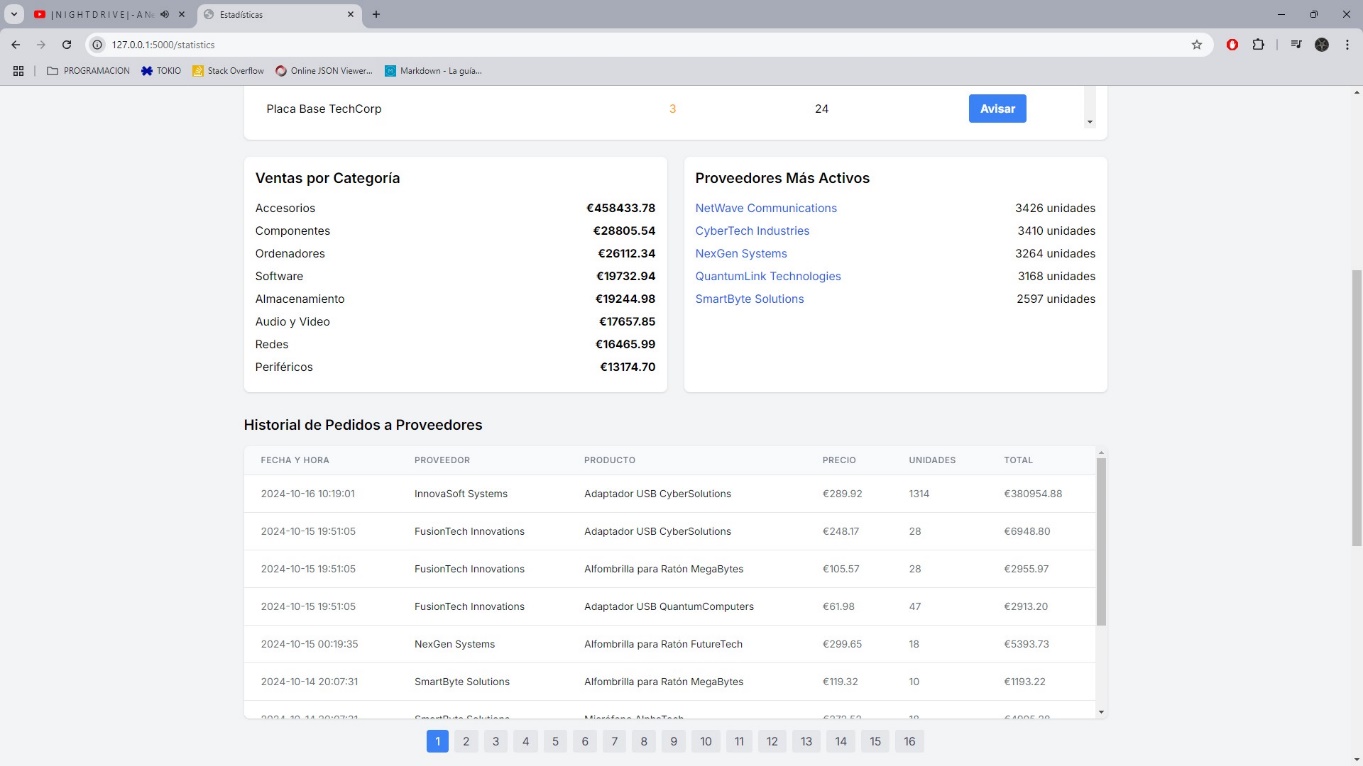
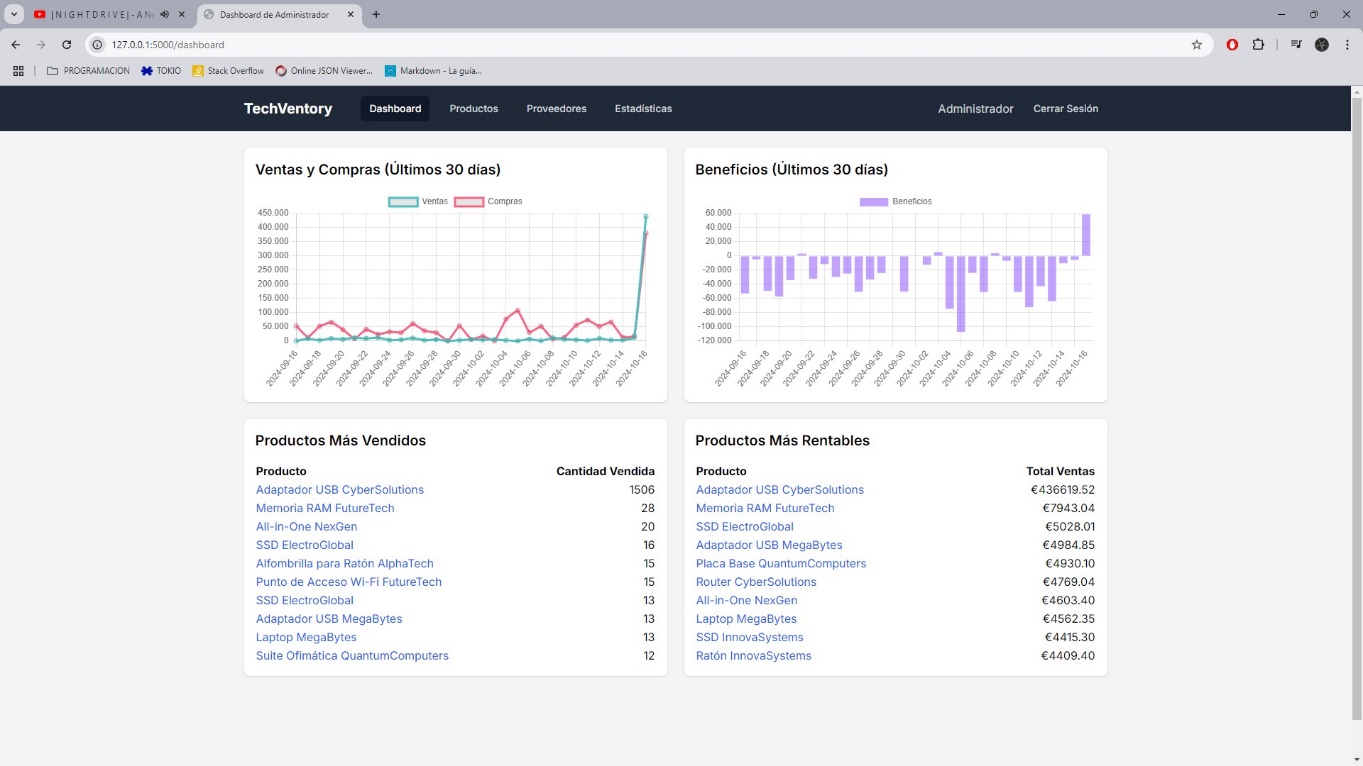
G) ALERTAS

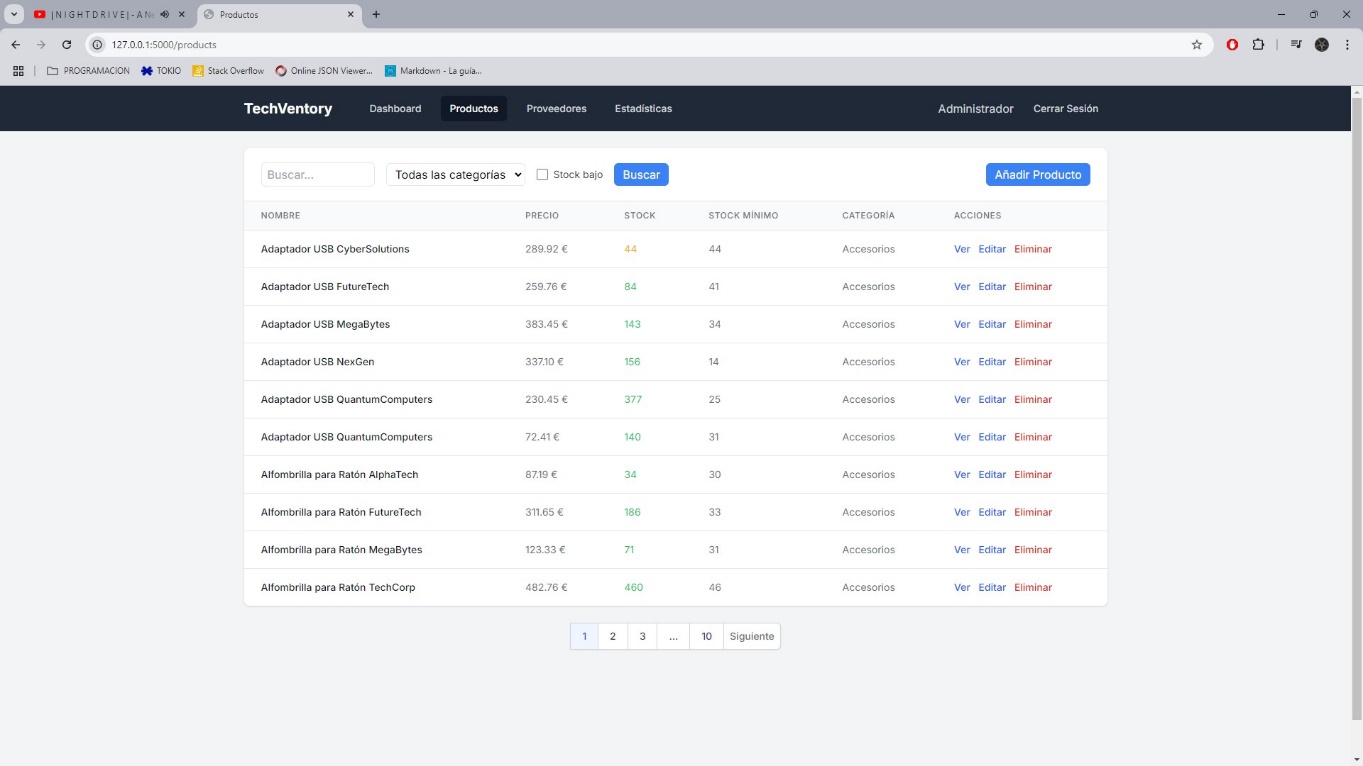
- Notificaciones de bajo stock.

* **PARA EL SISTEMA DE ALERTAS DE BAJO STOCK EN EL INVENTARIO DISPONEMOS DE UNA SECCIÓN EN ‘ESTADÍSTICAS’ PARA MANEJAR LOS AVISOS A LOS PROVEEDORES Y ASI REALIZAR LOS PEDIDOS DE LOS PRODUCTOS NECESARIOS (por defecto la cantidad de unidades a pedir son las mínimas requeridas para que el producto deje de generar la alerta, puedes introducir la cantidad que desees)**





* **TAMBIEN DISPONEMOS DE UN ‘HISTORIAL DE PEDIDOS A PROVEEDORES’ EN DONDE PODREMOS OBSERVAR EN DETALLE LOS PEDIDOS REALIZADOS**
* **UNA VEZ REALIZADO EL PEDIDO AL PROVEEDOR, VEREMOS REFLEJADA LA INFORMACION REFERENTE A LA NUEVA COMPRA DE LOS PRODUCTOS EN LOS GRÁFICOS DEL DASHBOARD DEL ADMINISTRADOR**
* **Y TAMBIEN PODREMOS VER MODIFICADO NUEVAMENTE EL STOCK DEL PRODUCTO EN EL LISTADO DE PRODUCTOS DEL ADMINISTRADOR**



REQUISITOS NO FUNCIONALES

1. Rendimiento:

º Tiempo de respuesta menor a 2 segundos para operaciones comunes.

º Capacidad para manejar al menos 100 usuarios concurrentes.

2. Seguridad:

º Autenticación segura de usuarios.

º Encriptación de datos sensibles.

º Protección contra ataques comunes (SQL injection, XSS).

3. Usabilidad:

º Interfaz intuitiva y fácil de usar.

º Diseño responsive para diferentes dispositivos.

4. Escalabilidad:

º Arquitectura que permita el crecimiento futuro de la aplicación.

5. Mantenibilidad:

º Código documentado y estructurado.

º Uso de patrones de diseño para facilitar futuras actualizaciones.

1. MANUAL DE INSTALACIÓN

REQUISITOS PREVIOS

- Python 3.8 o superior

- pip (gestor de paquetes de Python)

PASOS DE INSTALACIÓN

**1. CREAR Y ACTIVAR UN ENTORNO VIRTUAL:**

python -m venv venv

source venv/bin/activate **# En Windows: venv\Scripts\activate**

**2. INSTALAR LAS DEPENDENCIAS:**

pip install -r requirements.txt

**3. CONFIGURAR LAS VARIABLES DEL ENTORNO:**

Crea un archivo `.env` en el directorio raíz del proyecto y añade:

FLASK\_APP=main.py

FLASK\_ENV=development

SECRET\_KEY=tu\_clave\_secreta\_muy\_segura

SQLALCHEMY\_DATABASE\_URI=sqlite:///suministros.db

MAIL\_SERVER=smtp.example.com

MAIL\_PORT=587

MAIL\_USE\_TLS=True

MAIL\_USERNAME=[your-email@example.com](mailto:your-email@example.com)

MAIL\_PASSWORD=your-password

Asegúrate de reemplazar los valores de MAIL\_\* con la configuración real de tu servidor SMTP.

**4. INICIALIZAR LA BASE DE DATOS:**

flask db init

flask db migrate

flask db upgrade

**5. POBLAR LA BASE DE DATOS CON DATOS DE EJEMPLO:**

(Se crearán y añadirán automáticamente ejemplos de 100 productos, 10 proveedores, 10 usuarios, compras y ventas)

python populate\_db.py

**6. EJECUTAR LA APLICACIÓN:**

flask run

**7. ACCEDER A LA APLICACIÓN:**

Abre un navegador y visita `http://localhost:5000`

**## NOTAS ADICIONALES**

- Por defecto tendremos creada una cuenta de carácter ‘Administrador’ con el siguiente usuario y contraseña: ‘admin’ y ‘admin123’.

- Asegúrate de que el directorio 'logs' exista en la raíz del proyecto para el correcto funcionamiento del sistema de logging.

- Revisa y ajusta la configuración de seguridad antes de desplegar en un entorno de producción.

1. CONCLUSIONES

El Sistema de Gestión de Inventario desarrollado cumple con los objetivos establecidos, proporcionando una solución eficiente y fácil de usar para la gestión de inventario, ventas y compras. La aplicación demuestra la capacidad de integrar tecnologías modernas para crear una herramienta robusta y escalable.

APRENDIZAJES Y EXPERIENCIAS ADQUIRIDAS

Importancia de una planificación detallada y una definición clara de requisitos.

Valor de la elección adecuada de tecnologías para el alcance del proyecto.

Beneficios de un diseño modular y bien estructurado para la mantenibilidad.

Importancia de la seguridad y la experiencia del usuario en aplicaciones web.

1. EVOLUTIVOS DEL PROYECTO

FUTURAS MEJORAS Y DESARROLLO

1. Implementación de un sistema de notificaciones por correo electrónico.

2. Integración con sistemas de pago en línea para ventas directas.

3. Desarrollo de una API RESTful para integración con otras aplicaciones.

4. Implementación de un sistema de gestión de devoluciones.

5. Mejora de los análisis predictivos para optimización de inventario.

6. Desarrollo de una aplicación móvil complementaria.

7. Implementación de un sistema de gestión de múltiples almacenes.

8. Integración con sistemas de contabilidad externos.

9. Mejora de la internacionalización para soportar múltiples idiomas y monedas.

10. Implementación de un sistema de fidelización de clientes.

1. **DESCRIPCIÓN DE LOS COMPONENTES**

COMPONENTES PRINCIPALES

**Configuración e inicialización:**

. **(main.py):** Punto de entrada de la aplicación. Configura la aplicación Flask, inicializa las extensiones y registra las rutas.

. **(extensions.py):** Define las extensiones de Flask utilizadas (SQLAlchemy, Migrate, LoginManager, CSRFProtect, Mail).

**Modelos de Datos (models.py):**

. Define las estructuras de datos utilizando SQLAlchemy ORM.

. Principales modelos: User, Category, Product, Supplier, Sale, SaleItem, Purchase, PurchaseItem, CartItem.

. Implementa relaciones entre modelos (ej: productos y proveedores).

**Formularios (forms.py):**

. Utiliza Flask-WTF para la creación y validación de formularios.

. Incluye formularios para login, registro, productos, proveedores, carrito y checkout.

**Rutas y Lógica de Negocio (routes.py):**

. Implementa la lógica principal de la aplicación.

. Maneja autenticación, CRUD de productos y proveedores, operaciones de carrito y ventas.

. Incluye rutas para dashboard de administrador y cliente.

**Plantillas HTML:**

. Utiliza Jinja2 para el renderizado de plantillas.

. Implementa una estructura base (base.html) y plantillas específicas para cada funcionalidad.

**Manejo de Errores:**

. error\_handlers.py: Configura manejadores personalizados para diferentes códigos de error HTTP.

**Población de Base de Datos:**

. populate\_db.py: Script para poblar la base de datos con datos de ejemplo.

FUNCIONALIDADES PRINCIPALES

**Autenticación y Autorización:**

. Sistema de registro y login de usuarios.

. Diferenciación entre usuarios administradores y clientes.

**Gestión de Productos:**

. CRUD completo para productos.

. Categorización de productos.

. Control de stock y notificaciones de stock bajo.

**Gestión de Proveedores:**

. CRUD para proveedores.

. Asociación de proveedores con productos.

**Carrito de Compras:**

. Funcionalidad para añadir, actualizar y eliminar productos del carrito.

. Proceso de checkout.

**Ventas y Compras:**

. Registro de ventas a clientes.

. Registro de compras a proveedores.

**Dashboard y Estadísticas:**

. Dashboard diferenciado para administradores y clientes.

. Visualización de estadísticas de ventas, productos más vendidos, etc.

**API REST:**

. Endpoints para operaciones AJAX (ej: actualización de carrito, obtención de datos para gráficos).

CARACTERÍSTICAS TÉCNICAS

. Uso de SQLAlchemy para ORM y manejo de base de datos.

. Implementación de paginación para listados largos.

. Manejo de sesiones de usuario con Flask-Login.

. Protección CSRF en formularios.

. Uso de blueprints para organizar rutas.

. Implementación de soft delete para productos y proveedores.

. Manejo de archivos estáticos (CSS, JavaScript).

. Uso de Tailwind CSS para estilos.

SEGURIDAD

. Hashing de contraseñas.

. Protección contra CSRF.

. Validación de formularios en el servidor.

. Control de acceso basado en roles (admin vs cliente).

EXPERIENCIA DEL USUARIO

. Interfaz responsiva utilizando Tailwind CSS.

. Mensajes flash para notificaciones al usuario.

. Confirmaciones para acciones críticas (ej: eliminación de productos).

ÁREAS DE MEJORA POTENCIAL

. Implementación de tests unitarios y de integración.

. Mejora en la gestión de errores y logging.

. Posible refactorización para reducir la complejidad en algunos archivos (ej: routes.py).

. Implementación de caché para mejorar el rendimiento.

RESUMEN

En resumen, esta aplicación es un sistema completo de gestión de inventario y ventas con funcionalidades robustas tanto para administradores como para clientes. Está estructurada siguiendo prácticas de desarrollo web con Flask, y ofrece una base sólida para futuras expansiones o mejoras.