Documentación Técnica - Axion Store

Proyecto en Fase Beta

---

1. Introducción al Proyecto

Axion Store es una plataforma de comercio electrónico especializada en productos tecnológicos (tarjetas gráficas, monitores, etc.). La aplicación está construida como una tienda online completa con sistema de gestión de usuarios, carrito de compras, procesamiento de pedidos y un wallet virtual para realizar transacciones.

El sistema está desarrollado en **Python** utilizando el framework **Flask** para el backend, con **MySQL** como base de datos relacional y **SQLAlchemy** como ORM (Object-Relational Mapping) para la gestión de datos.

---

2. Arquitectura General del Sistema

2.1 Tecnologías Principales

- **Flask**: Framework web minimalista de Python que maneja las rutas, peticiones HTTP y renderizado de plantillas
- **SQLAlchemy**: ORM que permite interactuar con la base de datos MySQL mediante objetos de Python en lugar de SQL directo
- **MySQL**: Sistema de gestión de base de datos relacional donde se almacena toda la información
- **Werkzeug**: Biblioteca para seguridad (hashing de contraseñas) y manejo de archivos
- **PyMySQL**: Adaptador que permite a Python comunicarse con MySQL

2.2 Patrón de Arquitectura

El proyecto sigue el patrón **MVC (Model-View-Controller)** adaptado a Flask:

- **Modelos**: Clases de Python que representan las tablas de la base de datos
- **Vistas**: Plantillas HTML que se renderizan con datos dinámicos
- **Controladores**: Funciones decoradas con `@app.route()` que manejan la lógica de negocio

---

3. Estructura de la Base de Datos

La base de datos `axion_store` contiene 6 tablas principales que se relacionan entre sí:

3.1 Tabla `users` (Usuarios)

Almacena la información de todos los usuarios del sistema.

Campos principales:

- `ID_Users`: Identificador único del usuario
- `nombre`: Nombre completo
- `email`: Correo electrónico (único en el sistema)
- `password`: Contraseña encriptada con hash
- `es_admin`: Bandera booleana que indica si tiene privilegios de administrador
- `fecha_registro`: Fecha y hora de creación de la cuenta

3.2 Tabla `producto`

Contiene el catálogo de productos disponibles en la tienda.

Campos principales:

- `ID_Producto`: Identificador único del producto
- `nombre`: Nombre descriptivo del producto
- `precio`: Precio en unidades monetarias
- `imagen`: Ruta de la imagen del producto
- `disponible`: Indica si está activo para venta
- `stock`: Cantidad disponible en inventario

Restricción importante: El stock nunca puede ser negativo (constraint a nivel de base de datos).

3.3 Tabla `wallet`

Sistema de monedero virtual asociado a cada usuario.

Campos principales:

- `ID_Wallet`: Identificador único del wallet
- `ID_Users`: Referencia al usuario propietario
- `saldo`: Balance actual del usuario (inicia en $50.00)

Concepto clave: Cada usuario tiene UN solo wallet que se crea automáticamente al registrarse.

3.4 Tabla `carrito`

Almacena los productos que cada usuario ha agregado a su carrito de compras (temporal).

Campos principales:

- `ID_Carrito`: Identificador único del ítem en el carrito
- `ID_Users`: Usuario propietario del carrito
- `ID_Producto`: Producto agregado
- `cantidad`: Cantidad de unidades del producto

Restricción: La cantidad siempre debe ser mayor a 0.

3.5 Tabla `pedido`

Registra las compras completadas por los usuarios.

Campos principales:

- `ID_Pedido`: Identificador único del pedido
- `ID_Users`: Usuario que realizó la compra
- `total`: Monto total pagado
- `fecha`: Timestamp de cuándo se completó la compra
- `estado`: Estado del pedido (por defecto "pagado")

3.6 Tabla `detalle_pedido`

Desglosa los productos incluidos en cada pedido (relación muchos a muchos entre pedidos y productos).

Campos principales:

- `ID_Detalle_Pedido`: Identificador único
- `ID_Pedido`: Pedido al que pertenece
- `ID_Producto`: Producto incluido
- `cantidad`: Cantidad comprada
- `precio_unitario`: Precio al momento de la compra (histórico)

Concepto clave: Guardar el precio histórico es fundamental porque los precios pueden cambiar con el tiempo, pero los pedidos deben reflejar lo que se pagó en su momento.

---

4. Relaciones Entre Tablas

Diagrama Conceptual de Relaciones:

```
Usuario (1) ----< tiene >---- (1) Wallet
Usuario (1) ----< tiene >---- (*) Carritos
Usuario (1) ----< realiza >---- (*) Pedidos
Producto (1) ----< está en >---- (*) Carritos
Producto (1) ----< está en >---- (*) Detalles de Pedido
Pedido (1) ----< contiene >---- (*) Detalles de Pedido
```

Leyenda:

- `(1)`: Relación uno a uno
- `(*)`: Relación uno a muchos

---

5. Flujo de Funcionalidades Principales

5.1 Registro de Usuario

Flujo lógico:

1. Usuario completa formulario con: nombre, email, contraseña y confirmación
2. **Validaciones del sistema:**
   - Todos los campos obligatorios completos
   - Nombre mínimo 3 caracteres
   - Contraseña mínimo 6 caracteres
   - Contraseñas coinciden
   - Email no existe previamente en el sistema
3. **Si pasa validaciones:**
   - Se encripta la contraseña usando `pbkdf2:sha256` (no se guarda en texto plano)
   - Se crea el registro de Usuario en la base de datos
   - Se crea automáticamente un Wallet con $50.00 de saldo inicial
   - Se inicia sesión automáticamente
   - Se redirige a la página principal

Concepto de seguridad: Las contraseñas NUNCA se almacenan como texto plano, solo se guarda su hash. Cuando el usuario inicia sesión, se hashea la contraseña ingresada y se compara con el hash almacenado.

5.2 Inicio de Sesión

Flujo lógico:

1. Usuario ingresa email y contraseña
2. Sistema busca usuario por email en la base de datos
3. Si existe, compara el hash de la contraseña ingresada con el almacenado
4. **Si coincide:**
   - Se crea una **sesión** (objeto temporal almacenado en el servidor)
   - Se guardan en la sesión: `user_id`, `user_name`, `user_email`, `is_admin`
   - Si es admin, se redirige al panel de administración
   - Si es usuario normal, se redirige a la tienda

Concepto de sesión: Es un mecanismo que permite al servidor recordar quién es el usuario entre diferentes peticiones HTTP (que por naturaleza son stateless). Flask usa cookies firmadas para identificar la sesión.

5.3 Navegación y Búsqueda de Productos

Funcionalidades:

1. **Búsqueda:** Filtrado por texto en el nombre del producto
2. **Ordenamiento:**
   - Por nombre (alfabético)
   - Por precio ascendente/descendente
   - Por stock disponible

Flujo técnico:

- Las queries se construyen dinámicamente con SQLAlchemy según los parámetros de la URL
- Ejemplo: `/home?search=RTX&sort=precio_desc` busca "RTX" y ordena por precio descendente

5.4 Agregar Productos al Carrito

Flujo lógico:

1. **Validación de disponibilidad:**
   - Producto debe estar marcado como disponible

- Stock debe ser mayor a 0

2. **Verificación de existencia en carrito:**
   - Si el producto YA está en el carrito: incrementa cantidad (si hay stock)
   - Si NO está en el carrito: crea nuevo registro con cantidad = 1
3. Se actualiza la base de datos
4. El usuario ve un mensaje de confirmación

Concepto importante: El carrito es temporal y está ligado al usuario actual. Si cierra sesión y vuelve a entrar, su carrito persiste porque está en la base de datos.

5.5 Gestión del Carrito

Operaciones disponibles:

- **Ver carrito:** Muestra todos los productos con cálculo de subtotales y total general
- **Actualizar cantidad:** Modifica unidades de un producto (valida contra stock disponible)
- **Eliminar producto:** Quita un ítem específico del carrito
- **Vaciar carrito:** Elimina todos los productos del carrito actual

5.6 Proceso de Compra (Checkout)

Este es el proceso más crítico del sistema. Se ejecuta como una **transacción atómica** (todo o nada).

Flujo paso a paso:

1. **Validaciones iniciales:**
   - Carrito no está vacío
   - Todos los productos aún existen
   - Hay stock suficiente para cada producto
   - El wallet tiene saldo suficiente para el total
2. **Si todas las validaciones pasan:**
   - Se descuenta el total del wallet del usuario
   - Se crea un registro de Pedido con el total
   - Para cada producto en el carrito:
     - Se crea un DetallePedido (guardando precio histórico)
     - Se reduce el stock del producto

- Se vacía el carrito del usuario
- **Se hace COMMIT a la base de datos** (todo se ejecuta como una unidad)

3. **Si algo falla:**
   - Se hace ROLLBACK (se revierten TODOS los cambios)
   - Se muestra mensaje de error
   - El usuario puede intentar nuevamente

Concepto de transacción: Garantiza que o se completan TODAS las operaciones, o NINGUNA. Evita estados inconsistentes como: saldo descontado pero sin pedido creado, o pedido creado pero stock no reducido.

5.7 Historial de Pedidos

Funcionalidad:

- Muestra todos los pedidos realizados por el usuario actual
- Cada pedido incluye:
  - Número de pedido
  - Fecha de compra
  - Total pagado
  - Estado
  - Detalles de productos (nombre, cantidad, precio unitario de ese momento)

5.8 Sistema de Wallet

Operaciones:

1. **Ver saldo:** Muestra el balance actual del usuario
2. **Recargar saldo:**
   - Usuario ingresa monto a agregar
   - Validaciones: monto > 0 y monto <= $10,000
   - Se suma al saldo existente

Nota: En un sistema real, aquí se integraría una pasarela de pago (PayPal, Stripe, etc.). Actualmente es simulado.

---

6. Panel de Administración

El sistema incluye un panel completo para usuarios con privilegio `es_admin = True`.

6.1 Dashboard Principal

Métricas visualizadas:

- Total de usuarios registrados
- Total de productos en catálogo
- Total de pedidos procesados
- Suma total de ventas ($)
- Lista de últimos 5 pedidos realizados

6.2 Gestión de Usuarios

Funcionalidades:

- Ver lista completa de usuarios
- Ver información de wallets
- **Agregar saldo a usuarios:** El admin puede acreditar dinero a cualquier wallet

6.3 Gestión de Productos

Operaciones CRUD completas:

1. **Crear producto:**
   - Formulario con: nombre, precio, stock, imagen, disponibilidad
   - **Subida de imágenes:** Archivos permitidos (png, jpg, jpeg, gif, webp)
   - Tamaño máximo: 2MB
   - Se guarda en carpeta `static/img/` con timestamp para evitar colisiones
2. **Editar producto:**
   - Modifica todos los campos del producto
   - Permite cambiar imagen (subir nueva o mantener actual)
3. **Eliminar producto:**
   - **Restricción importante:** NO se puede eliminar si el producto aparece en pedidos históricos
   - Esto protege la integridad referencial de los registros de ventas
   - Si tiene pedidos asociados, se muestra mensaje de error

- Si no tiene pedidos, se elimina primero de carritos actuales y luego el producto

6.4 Gestión de Pedidos

Funcionalidad:

- Vista de todos los pedidos del sistema
- Ordenados por fecha descendente (más recientes primero)
- Acceso a detalles completos de cada pedido

---

7. Conceptos de Seguridad Implementados

7.1 Decoradores de Protección

`@login_required`:

- Protege rutas que requieren autenticación
- Si el usuario no está logueado, redirige a login
- Usado en: carrito, compras, wallet, historial

`@admin_required`:

- Protege rutas exclusivas de administradores
- Verifica que el usuario esté logueado Y sea admin
- Si no cumple, redirige a home o login según el caso
- Usado en: todo el panel de administración

7.2 Hashing de Contraseñas

Algoritmo: PBKDF2 con SHA-256

- **PBKDF2** (Password-Based Key Derivation Function 2): aplica la función hash miles de veces
- Hace extremadamente lento el ataque por fuerza bruta
- Incluye "salt" automático (valor aleatorio único por contraseña)

7.3 Validación de Archivos

Protecciones al subir imágenes:

- Lista blanca de extensiones permitidas
- Uso de `secure_filename()` para sanitizar nombres

- Límite de tamaño (2MB)
- Timestamp en nombres para evitar sobrescrituras

7.4 Sesiones Seguras

- Flask usa cookies firmadas con `SECRET_KEY`
- La `SECRET_KEY` debe cambiarse en producción (actualmente usa valor por defecto)
- Las sesiones se limpian al hacer logout

---

8. Optimizaciones de Base de Datos

8.1 Pool de Conexiones

Configuración implementada:

```
pool_size: 10          → 10 conexiones activas
max_overflow: 20       → Hasta 20 conexiones adicionales si es necesario
pool_recycle: 3600     → Recicla conexiones cada hora
pool_pre_ping: True    → Verifica conexión antes de usarla
```

Beneficio: Evita crear/destruir conexiones constantemente, mejorando el rendimiento.

8.2 Índices en Columnas Clave

Se crean índices automáticos en:

- Columnas de foreign keys (ID_Users, ID_Producto, etc.)
- Email de usuarios (búsquedas frecuentes)
- Nombre de productos (para búsquedas y ordenamiento)

Beneficio: Acelera drásticamente las consultas que filtran u ordenan por estas columnas.

8.3 Constraints de Integridad

- **Check constraints:** Garantizan reglas de negocio a nivel de base de datos
  - Stock >= 0 (no puede ser negativo)
  - Cantidad en carrito > 0 (no puede ser cero o negativa)
- **Foreign keys:** Garantizan relaciones válidas entre tablas
- **Unique constraints:** Email único por usuario

---

9. Manejo de Errores

El sistema incluye páginas personalizadas para errores HTTP comunes:

- **400 Bad Request:** Petición malformada
- **403 Forbidden:** Sin permisos para acceder
- **404 Not Found:** Página o recurso no existe
- **500 Internal Server Error:** Error del servidor

Implementación: Decoradores `@app.errorhandler()` que capturan excepciones y muestran páginas amigables en lugar de errores técnicos.

---

10. Contexto Inyectado en Plantillas

`inject_cart_count()`

Función especial: Se ejecuta automáticamente antes de renderizar CUALQUIER plantilla.

Propósito: Calcular el número total de productos en el carrito del usuario actual.

Resultado: La variable `{{ cart_count }}` está disponible en todas las plantillas HTML, típicamente usada para mostrar un badge en el ícono del carrito.

---

11. Inicialización del Sistema

Datos de Ejemplo

Al iniciar la aplicación por primera vez:

1. **Se insertan 3 productos de ejemplo:**
   - Tarjeta Gráfica GTX 1650 ($230, 15 unidades)
   - Tarjeta gráfica RTX 4090 ($1800, sin stock)
   - Monitor Xiaomi G34 ($500, 8 unidades)
2. **Se crea usuario administrador:**
   - Email: `admin@axionstore.com`
   - Password: `admin123`
   - Wallet inicial: $10,000

Nota: Solo se insertan si la base de datos está vacía, evitando duplicados.

12. Flujo Técnico de una Petición HTTP

Ejemplo: Usuario hace clic en "Agregar al carrito"

```
1. NAVEGADOR: GET /add_to_cart/5
   ↓
2. FLASK: Ejecuta función add_to_cart(producto_id=5)
   ↓
3. DECORADOR: @login_required verifica sesión activa
   ↓
4. LÓGICA DE NEGOCIO:
   - Consulta producto con ID=5
   - Verifica stock disponible
   - Busca si ya está en carrito del usuario
   - Actualiza cantidad o crea nuevo registro
   ↓
5. SQLALCHEMY: Genera y ejecuta queries SQL
   ↓
6. MYSQL: Actualiza tabla carrito
   ↓
7. FLASK: Agrega mensaje flash de confirmación
   ↓
8. NAVEGADOR: Recibe redirect a /home con mensaje
```

---

13. Consideraciones para Producción

13.1 Cambios Necesarios

Seguridad:

- [ ] Cambiar `SECRET_KEY` a un valor aleatorio y seguro
- [ ] Cambiar credenciales de base de datos
- [ ] Configurar HTTPS
- [ ] Implementar rate limiting en rutas sensibles

Configuración:

- [ ] Desactivar `debug=True`
- [ ] Configurar servidor WSGI (Gunicorn, uWSGI)
- [ ] Configurar servidor web (Nginx, Apache)

- [ ] Configurar logs apropiados

Base de datos:

- [ ] Realizar backup automático regular
- [ ] Configurar réplicas si es necesario
- [ ] Monitorear queries lentas

13.2 Mejoras Futuras Sugeridas

1. **Sistema de autenticación:**
   - Recuperación de contraseña por email
   - Verificación de email al registrarse
   - Login con redes sociales (OAuth)
2. **Carrito:**
   - Guardar para después (wishlist)
   - Cupones de descuento
   - Cálculo de envío
3. **Productos:**
   - Categorías y filtros avanzados
   - Reviews y calificaciones
   - Productos relacionados
4. **Pagos:**
   - Integración con pasarela real (Stripe, PayPal)
   - Múltiples métodos de pago
   - Facturación electrónica
5. **Administración:**
   - Dashboard con gráficas
   - Reportes de ventas
   - Gestión de inventario con alertas

---

14. Glosario de Términos Clave

ORM (Object-Relational Mapping): Técnica que permite manipular bases de datos usando objetos de programación en lugar de SQL directo.

Hash: Función matemática de un solo sentido que convierte texto en una cadena de caracteres única. No se puede revertir (descifrar).

Session: Mecanismo para mantener información del usuario entre diferentes peticiones HTTP.

Constraint: Regla de negocio implementada a nivel de base de datos que no puede ser violada.

Foreign Key: Columna que referencia la clave primaria de otra tabla, estableciendo una relación.

Transaction: Conjunto de operaciones de base de datos que se ejecutan como una unidad atómica (todo o nada).

Rollback: Revertir todos los cambios de una transacción si algo falla.

Commit: Confirmar permanentemente los cambios de una transacción en la base de datos.

Decorator: Función de Python que modifica el comportamiento de otra función (como `@login_required`).

WSGI: Estándar que define cómo un servidor web se comunica con aplicaciones Python.

---

15. Conclusión

Axion Store es una aplicación web funcional que implementa los conceptos fundamentales de una tienda online: gestión de usuarios con autenticación segura, catálogo de productos con búsqueda y filtros, sistema de carrito de compras, procesamiento de pedidos con validaciones atómicas, wallet virtual para pagos, y un panel de administración completo.

El código está estructurado de forma modular, con clara separación de responsabilidades, validaciones robustas, manejo de errores apropiado y optimizaciones de rendimiento a nivel de base de datos.

Para preguntas técnicas específicas o dudas sobre alguna funcionalidad, no duden en contactar al equipo de desarrollo.

Documento generado para: Equipo de Axion Store
Versión del proyecto: Beta
Fecha: Noviembre 2025