NovaCommerce

(SAD) Software Architecture Document

Versión 1.0

| **Identificación** |  |
| --- | --- |
| **Proyecto** |  |
| **Versión** |  |

| **Documento mantenido por** |  |
| --- | --- |
| **Fecha de última revisión** |  |
| **Fecha de próxima revisión** |  |

| **Documento aprobado por** |  |
| --- | --- |
| **Fecha de última aprobación** |  |

| **Fecha** | **Version** | **Descripcion** | **Autor** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

# 

# Tabla de contenido

[**Tabla de contenido 2**](#_heading=h.79z22tghe1r9)

[**Introducción 3**](#_heading=h.2s8eyo1)

[**Alcance 4**](#_heading=h.zet794er9cuo)

[Gestión de Usuarios 4](#_heading=h.9qii0wkkysh1)

[Administración de Productos y Usuarios 4](#_heading=h.x6dx0mfqcbzs)

[Procesamiento de Compras y Ventas 5](#_heading=h.l175vv75hzzo)

[Reportes y Métricas de Ventas 5](#_heading=h.m6tvcg2j9rae)

[Despliegue y Entorno de Ejecución 5](#_heading=h.ox3lei497jq7)

[**Referencias 6**](#_heading=h.3rdcrjn)

[**Arquitectura de Software 8**](#_heading=h.26in1rg)

[Modelo 8](#_heading=h.b113utmul4h)

[Vista 8](#_heading=h.b2postr9veky)

[Controlador 9](#_heading=h.wolw8xj8hdp)

[Api rest 9](#_heading=h.tu2jx3swetcw)

[Escalabilidad y Distribución 9](#_heading=h.whg2xtpovhsi)

[**Objetivos de la Arquitectura 10**](#_heading=h.lnxbz9)

[**Diagramas 11**](#_heading=h.35nkun2)

[**Tamaño y desempeño 12**](#_heading=h.xh4j8fds8gvf)

# 

# Introducción

El sistema NovaCommerce es una plataforma web de comercio electrónico desarrollada para optimizar la digitalización y competitividad de pequeños y medianos emprendimientos. Su propósito es ofrecer una solución integral para la gestión de ventas en línea, abarcando desde la publicación de productos hasta el análisis de métricas comerciales.

El desarrollo se realizará utilizando el framework Django, adoptando una arquitectura modular que integre de forma coherente el frontend, backend y la base de datos, asegurando escalabilidad, mantenibilidad y soporte multiplataforma.

# 

# Alcance

El sistema NovaCommerce abarca el desarrollo completo de una plataforma web de comercio electrónico destinada a facilitar la gestión y venta de productos por parte de pequeños y medianos negocios. Su alcance funcional se distribuye en cinco módulos principales, cada uno con responsabilidades claramente definidas e implementadas dentro de la arquitectura Django modular.

## Gestión de Usuarios

Este módulo se desarrollará utilizando el sistema de autenticación integrado de Django y se complementará con una API REST construida sobre el Django REST Framework.

Ubicación funcional

Backend (Django) + API REST + Base de datos.

Funciones clave

* Registro y autenticación de usuarios mediante formularios web seguros (CSRF y validación de credenciales).
* Control de roles (Administrador y Comprador) Gestionado por el sistema de permisos de Django (User, Group, Permission).
* Control de sesión y persistencia Mediante tokens o JWT para usuarios autenticados.
* Interfaz web de administración Para creación, edición y eliminación de usuarios, accesible sólo al rol Administrador.

## Administración de Productos y Usuarios

Esta función será gestionada desde un panel de control administrativo (dashboard) desarrollado en HTML, CSS y Bootstrap, con consumo de datos desde la API REST.

Ubicación funcional

Frontend (dashboard web) + API REST + Base de datos.

Funciones clave

* CRUD completo para productos (crear, leer, actualizar, eliminar) con validaciones de integridad referencial en base de datos.
* CRUD de usuarios gestionado únicamente por administradores.
* Control de inventario y actualización de precios en tiempo real.

## Procesamiento de Compras y Ventas

Este módulo representa la capa de negocio más crítica del sistema. Gestiona las operaciones de compra desde la selección de productos hasta la confirmación del pago.

Ubicación funcional

Frontend (carrito y checkout) + API REST + Backend (lógica de negocio).

Funciones clave

* Carrito de compras persistente basado en sesiones de usuario.
* Procesamiento de órdenes con validaciones de stock y actualización automática del inventario.
* Registro histórico de ventas accesible desde el panel del administrador.

## Reportes y Métricas de Ventas

Este módulo proporciona una interfaz de análisis de datos en tiempo real para los administradores.

Ubicación funcional

Frontend (panel de reportes) + Backend (servicio de estadísticas) + Base de datos.

Funciones clave

* Cálculo de métricas sobre ventas diarias, mensuales y totales.
* Identificación de productos más vendidos y clientes con mayor frecuencia de compra.
* Visualización mediante gráficos (Chart.js o similar) alimentados por endpoints REST.
* Exportación de reportes en formatos CSV.

## Despliegue y Entorno de Ejecución

El sistema será desplegado en un entorno cloud, priorizando seguridad, rendimiento y disponibilidad.

Infraestructura

* **Amazon AWS:** alojamiento del backend Django, la base de datos (PostgreSQL).
* **Cloudflare:** distribución de contenido estático y caché de recursos frontales, además de protección DDoS.
* **Nginx:** servidor intermedio de balanceo de carga y proxy inverso entre cliente y aplicación Django (uWSGI o Gunicorn).

Compatibilidad

* Accesible desde navegadores modernos en escritorio y dispositivos móviles.
* Interfaz responsive optimizada con Bootstrap.
* Conexiones cifradas bajo HTTPS con certificados gestionados por Cloudflare.

# Referencias

Framework

**Django (Python)**

Framework web principal del proyecto, utilizado para implementar la arquitectura del sistema (modelo, vistas y controladores). Proporciona un entorno estructurado para el desarrollo del backend, manejo de autenticación, control de permisos, administración de datos y exposición de endpoints mediante el Django REST Framework.

Frontend

**HTML (HyperText Markup Language)**

Lenguaje base para la estructura de las interfaces web. Se emplea para definir las páginas del sistema, como el panel del administrador, la vista de productos y el carrito de compras.

**CSS (Cascading Style Sheets)**

Usado para aplicar estilos visuales, estructurar el diseño y definir la presentación adaptable del sitio. Se integrará con Bootstrap para asegurar una experiencia uniforme en diferentes dispositivos.

**Bootstrap**

Framework de diseño frontend que facilita el desarrollo de interfaces responsive y modulares. Permite construir componentes reutilizables como formularios, menús, tarjetas de productos y paneles administrativos sin necesidad de definir estilos desde cero.

Ejemplo: la interfaz del carrito de compras se implementará con componentes de Bootstrap y formularios conectados a la API REST.

Diseño

**Figma**

Herramienta de diseño UI/UX utilizada para crear prototipos visuales y maquetas de las interfaces del sistema antes de su implementación. Facilita la colaboración y revisión del diseño entre los equipos de desarrollo y diseño.

**Canva**

Utilizada para la generación de diagramas o gráficos

Modelado de base de datos

**dbDiagram.io**

Plataforma en línea empleada para diseñar y documentar el modelo entidad–relación de la base de datos. Permite visualizar la estructura y relaciones entre las tablas (usuarios, productos, etc) antes de su implementación en Django.

Infraestructura

**Amazon AWS**

Proveedor de servicios en la nube donde se desplegará el sistema, una EC2 alojará la aplicación Django y el servidor Nginx.

**Cloudflare**

Se empleará como red de distribución de contenido (CDN) para optimizar el rendimiento y proporcionar seguridad adicional contra ataques DDoS. También gestionará los certificados SSL/TLS para conexiones HTTPS seguras.

**Nginx**

Actuará como servidor web y proxy inverso entre los clientes y el backend Django. También gestionará la compresión y el caché de archivos estáticos para reducir tiempos de carga.

Documentación y colaboración

**Google Docs**

Herramienta principal para la redacción colaborativa de documentación técnica y reportes de avance del proyecto.

**Google Sheets**

Utilizada para la gestión de cronogramas, control de versiones, asignación de tareas y registro de métricas del desarrollo.

Herramienta de desarrollo

**Visual Studio Code (VS Code)**

Entorno de desarrollo integrado (IDE) principal utilizado por el equipo. Permite la edición de código fuente, integración con Git para control de versiones, depuración del servidor Django y conexión remota a entornos AWS para despliegue y mantenimiento.

**Git – Github**

¡Falta la descripción aquí!

# Arquitectura de Software

El sistema NovaCommerce adopta una arquitectura modular basada en el patrón MVC (Model–View–Controller), implementado de forma nativa en el framework Django. Esta arquitectura permite separar las responsabilidades entre las capas de datos, presentación y control, favoreciendo la mantenibilidad, escalabilidad y reutilización de componentes.

Además, el sistema incorpora una API REST como capa intermedia de comunicación entre el frontend y el backend, permitiendo la interoperabilidad con otros servicios o clientes (por ejemplo, futuras aplicaciones móviles o integraciones externas).

## Modelo

El Modelo representa la estructura de datos y la lógica asociada a las entidades del sistema.

Está definido mediante las clases del módulo models.py de Django, que se traducen automáticamente en tablas dentro de la base de datos relacional (PostgreSQL).

**Funciones principales del modelo**

* Definir las entidades base: User, Producto, etc.
* Mantener la integridad relacional mediante claves foráneas y restricciones.
* Implementar validaciones a nivel de datos (campos obligatorios, tipos, formatos, unicidad).
* Permitir consultas optimizadas mediante el ORM de Django (Django QuerySet).
* Proveer acceso estructurado a los datos a través de la API REST, evitando acceso directo desde el frontend.

## Vista

La vista define la capa de presentación del sistema.

Está compuesta por plantillas HTML que utilizan el sistema de templates de Django junto con hojas de estilo CSS y componentes Bootstrap.

**Funciones principales de la vista**

* Renderizar la interfaz del usuario para navegadores de escritorio y móviles.
* Mostrar información dinámica proveniente del backend a través de peticiones REST.
* Adaptar el contenido de manera responsive según el dispositivo del usuario.
* Integrar componentes visuales reutilizables como formularios, tarjetas de productos y paneles informativos.

## Controlador

El Controlador se encarga de gestionar la lógica de negocio, procesar las peticiones del usuario y coordinar la comunicación entre la vista y el modelo.

En Django, esta capa se implementa a través de las views (funcionales o basadas en clases), junto con serializers y viewsets del Django REST Framework.

**Funciones principales del controlador**

* Procesar solicitudes HTTP (GET, POST, PUT, DELETE) provenientes del frontend.
* Validar datos recibidos antes de interactuar con el modelo.
* Invocar las operaciones CRUD sobre los modelos y retornar las respuestas en formato JSON a la vista.
* Gestionar autenticación, permisos y sesiones de usuario.
* Manejar la lógica de negocio, como el cálculo de totales de órdenes, descuentos o filtrado de reportes.

## Api rest

La API REST actúa como una capa de comunicación desacoplada entre el frontend y el backend, asegurando independencia tecnológica y escalabilidad futura.

Está desarrollada con Django REST Framework e implementa endpoints para todas las entidades del sistema (usuarios, productos, etc).

**Características principales**

* Endpoints estructurados bajo rutas /api/ con control de acceso por roles.
* Autenticación mediante tokens (JWT).
* Respuestas en formato JSON para compatibilidad con aplicaciones externas.
* Validaciones automáticas de datos de entrada y salida.

## Escalabilidad y Distribución

La arquitectura del sistema está diseñada para soportar componentes distribuidos y escalabilidad horizontal mediante servicios en la nube:

* El backend Django puede replicarse en múltiples instancias EC2 bajo balanceo de carga administrado por Nginx
* Los archivos estáticos y medios se podrían considerar servirse desde Amazon S3 y Cloudflare CDN a futuro.
* Los endpoints REST pueden ampliarse sin modificar la lógica del frontend gracias a la independencia de la capa de presentación.

# 

# Objetivos de la Arquitectura

La arquitectura del sistema NovaCommerce se diseña con el propósito de establecer una base tecnológica sólida, escalable y mantenible, capaz de sostener el crecimiento y la evolución funcional de la plataforma a largo plazo. Los siguientes objetivos orientan las decisiones de diseño, selección de tecnologías y organización modular del proyecto.

Garantizar modularidad y separación de responsabilidades

La arquitectura se estructura bajo el principio de separación de capas entre frontend, backend y base de datos. Cada componente cumple una función claramente definida:

* El frontend se encarga exclusivamente de la presentación y la interacción con el usuario.
* El backend gestiona la lógica de negocio, las reglas del dominio y la comunicación entre las capas.
* La base de datos almacena y mantiene la integridad de la información.

Esta modularidad permite desarrollar, mantener o escalar cada parte del sistema de forma independiente, minimizando el impacto de los cambios y facilitando la incorporación de nuevas funcionalidades. Asimismo, la API REST actúa como frontera técnica entre capas, garantizando un acoplamiento débil y favoreciendo la reutilización del backend en otros contextos (por ejemplo, aplicaciones móviles o integraciones externas).

Asegurar seguridad y fiabilidad en las transacciones

Dado que NovaCommerce operará con datos personales y financieros de los usuarios, la arquitectura prioriza la seguridad en todos los niveles del sistema. Las medidas contempladas incluyen:

* Autenticación y autorización mediante tokens JWT, controlados por el sistema de permisos de Django.
* Protección contra ataques comunes (CSRF, XSS, SQL Injection) mediante los mecanismos integrados del framework.
* Encriptación de las comunicaciones bajo protocolo HTTPS con certificados gestionados por Cloudflare.

El objetivo es garantizar la confidencialidad, integridad y disponibilidad de la información, reduciendo el riesgo operativo y fortaleciendo la confianza del usuario final en la plataforma.

# 

Facilitar la escalabilidad mediante servicios en la nube

La infraestructura del sistema se implementará sobre servicios cloud de Amazon AWS, permitiendo ajustar los recursos según la demanda.

Además, el uso de Cloudflare CDN optimiza la entrega de contenido estático y reduce la latencia global del sistema.

Este enfoque garantiza que la plataforma pueda responder a incrementos de tráfico, campañas comerciales o expansión a nuevos mercados sin comprometer su rendimiento ni requerir reestructuración arquitectónica.

Optimizar la experiencia del usuario

La arquitectura contempla una capa de presentación enfocada en la eficiencia, accesibilidad y usabilidad. El diseño del frontend se apoya en Bootstrap para asegurar compatibilidad entre dispositivos y navegadores, con tiempos de carga reducidos mediante optimización de recursos estáticos y consultas asíncronas a la API REST.

La optimización de la experiencia del usuario no solo mejora la satisfacción, sino que también incrementa la tasa de conversión y la retención dentro de la plataforma.

# 

# Diagramas

Aqui podrias agregar:  
Diagrama de Componentes (interacción entre módulos Django y servicios externos).

Diagrama de Clases (modelo de datos principal, lo mismo que da [dbDiagram.io](http://dbdiagram.io))

No se que mas

# 

# Tamaño y desempeño

El sistema NovaCommerce está diseñado para operar de manera eficiente dentro de un entorno de carga media, con capacidad de escalar conforme aumente el volumen de usuarios y transacciones.

## Tamaño estimado

Código fuente: entre 8.000 y 12.000 líneas, considerando la implementación completa del backend en Django, la API REST, los módulos de usuario, productos, ventas y reportes, así como el frontend desarrollado con HTML, CSS y Bootstrap.

Base de datos

Se proyecta un volumen inicial de hasta 50.000 registros, distribuidos principalmente entre productos, usuarios y órdenes de compra. La estructura está optimizada para escalar horizontalmente

## Desempeño esperado

* El sistema busca mantener un tiempo de respuesta promedio inferior a 300 ms por solicitud estándar, incluso en condiciones de tráfico moderado.
* Se estima soporte para aproximadamente 500 usuarios concurrentes en la configuración base del entorno cloud, con posibilidad de ampliación automática mediante balanceo de carga en AWS.
* Las consultas más frecuentes —como la carga de catálogos o métricas— se optimizarán mediante mecanismos de caché y el uso del CDN de Cloudflare, reduciendo la latencia percibida por el usuario.