Sistema de Gestión de Ventas para la Comunidad Universitaria de Icesi

Icesi Marketplace

Computación en Internet III

2024-1

Taller Nest JS

Miembros:

Cristian Felipe Perafan

Juan Sebastian Medina

Juan David Bahamon

22 de abril de 2024
Universidad Icesi

Introducción

Descripción general

La Comunidad Universitaria de Icesi actualmente enfrenta desafíos significativos en el proceso de comercialización de sus diversos productos. El uso de WhatsApp como plataforma principal ha llevado a la saturación de los grupos, lo que ha dificultado que los usuarios potenciales conozcan y accedan a los productos disponibles. Esta situación no solo afecta a los compradores, sino también a los vendedores, quienes se ven limitados en su capacidad de promover y vender sus productos de manera efectiva.

Ante esta problemática, surge la necesidad de implementar un sistema de gestión de ventas moderno y eficiente que optimice el proceso de comercialización y brinde una mejor experiencia tanto a los compradores como a los vendedores.

Objetivos

1. **Facilitar las ventas y promociones:** Desarrollar una plataforma centralizada que permita a los vendedores exhibir y promocionar sus productos de manera organizada y accesible para los compradores potenciales.
2. **Garantizar la seguridad y eficacia en la gestión de la información:** Implementar mecanismos de control de acceso y roles para garantizar la privacidad y la integridad de los datos, asegurando que solo los usuarios autorizados puedan acceder y manipular la información relevante.
3. **Generar reportes e insights útiles:** Desarrollar funcionalidades que permitan la generación de reportes basados en consultas a la base de datos, brindando información valiosa sobre ventas, tendencias, productos más populares, entre otros, para respaldar la toma de decisiones informadas.
4. **Asignar responsabilidades adecuadas:** Implementar un sistema de roles y permisos que permita asignar responsabilidades y tareas específicas a los diferentes miembros de la

comunidad, como administradores, vendedores y compradores, para garantizar una gestión eficiente del sistema.

5. **Mejorar la experiencia de usuario:** Proporcionar una interfaz intuitiva y amigable que facilite la navegación, búsqueda y compra de productos, mejorando la experiencia general de los usuarios y fomentando una mayor participación en la comunidad de ventas.

Al abordar estos objetivos, el Sistema de Gestión de Ventas para la Comunidad Universitaria de Icesi se convertirá en una herramienta valiosa que optimizará los procesos de comercialización, brindará una mayor visibilidad a los productos disponibles, mejorará la seguridad y la eficacia en la gestión de la información, y fomentará una experiencia de compra y venta más satisfactoria para todos los miembros de la comunidad.

Requerimientos

Esta sección de requerimientos define las funcionalidades principales que el sistema debe cumplir, abarcando aspectos como el registro de usuarios, la gestión de productos, el carrito de compras, la gestión de órdenes, la generación de reportes, y la autenticación y autorización de usuarios. Además, se incluyen algunos requerimientos no funcionales que son de vital importancia para la aplicación

Requerimientos Funcionales

Registro de usuarios:

- Permitir que los usuarios se registren en el sistema con información básica y credenciales de inicio de sesión.
- Permitir al administrador agregar, modificar y eliminar cuentas de usuario, incluidos vendedores y compradores.

Gestión de productos:

- Permitir a los vendedores agregar, modificar y eliminar productos de su inventario.
- El sistema debe permitir al vendedor agregar información detallada del producto, como nombre, descripción, precio.
- El sistema debe mostrar los productos con la información básica como nombre y precio.
- El sistema debe permitir al comprador seleccionar un producto, y mostrar la información detallada.
- Permitir al administrador agregar, modificar y eliminar productos de la tienda.

Carrito de compras:

Gestión de órdenes:

- Los vendedores deben poder procesar órdenes de compra recibidas.

Gestión de reportes:

- Tanto los administradores como los vendedores deben tener la capacidad de generar informes sobre ventas, inventario, clientes, etc.

Autenticación y autorización:

- El sistema debe verificar la identidad de los usuarios al iniciar sesión y garantizar que solo tengan acceso a las funciones adecuadas según su rol.
- Debe haber diferentes niveles de acceso para administradores, vendedores y compradores.

OPCIONALES (Baja prioridad)

- El comprador debe poder dejar una calificación al vendedor posterior a la completitud de la compra
- El comprador debe poder dejar un comentario al vendedor posterior a la completitud de la compra

Endpoints

Users:

- POST /users
    - Descripción: Este endpoint crea un nuevo usuario en el sistema.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera un cuerpo de solicitud (Body) en formato JSON que contiene los detalles del usuario a crear, siguiendo el esquema especificado en CreateUserDto.
    - Respuesta: Devuelve los detalles del usuario recién creado en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (Use Guards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN

- POST /users/seller
    - Descripción: Este endpoint registra un nuevo vendedor en el sistema.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera un cuerpo de solicitud (Body) en formato JSON que contiene los detalles del vendedor a registrar, siguiendo el esquema especificado en CreateCurrentUserDto.

    - Respuesta: Devuelve los detalles del vendedor registrado en formato JSON.
    - Código de Estado: 200 OK

- POST /users/buyer
    - Descripción: Este endpoint registra un nuevo comprador en el sistema.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera un cuerpo de solicitud (Body) en formato JSON que contiene los detalles del comprador a registrar, siguiendo el esquema especificado en CreateCurrentUserDto.
    - Respuesta: Devuelve los detalles del comprador registrado en formato JSON.
    - Código de Estado: 200 OK

- GET /users
    - Descripción: Este endpoint devuelve todos los usuarios registrados en el sistema.
    - Método HTTP: GET
    - Parámetros de entrada: Se pueden proporcionar parámetros de consulta (Query) para la paginación de resultados, siguiendo el esquema especificado en PaginationDto.
    - Respuesta: Devuelve una lista de usuarios en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN

- GET /users/:id
    - Descripción: Este endpoint devuelve los detalles de un usuario específico según su ID.
    - Método HTTP: GET
    - Parámetros de entrada: Se espera el ID del usuario como un parámetro de ruta (Param).
    - Respuesta: Devuelve los detalles del usuario en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN

- PATCH /users/:id
    - Descripción: Este endpoint actualiza los detalles de un usuario específico según su ID.
    - Método HTTP: PATCH
    - Parámetros de entrada: Se espera el ID del usuario como un parámetro de ruta (Param), y los nuevos detalles del usuario a actualizar en el cuerpo de la solicitud (Body) en formato JSON, siguiendo el esquema especificado en UpdateUserDto.
    - Respuesta: Devuelve los detalles actualizados del usuario en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN
    - Validación de entrada (UsePipes): Se utiliza ParseUUIDPipe para validar el ID del usuario.

- DELETE /users/:id
    - Descripción: Este endpoint elimina un usuario específico según su ID.
    - Método HTTP: DELETE
    - Parámetros de entrada: Se espera el ID del usuario como un parámetro de ruta (Param).
    - Respuesta: Devuelve un mensaje indicando el éxito de la operación en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN
    - Validación de entrada (UsePipes): Se utiliza ParseUUIDPipe para validar el ID del usuario.

Order:

- POST /order
    - Descripción: Este endpoint crea un nuevo pedido en el sistema.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera un cuerpo de solicitud (Body) en formato JSON que contiene los detalles del pedido a crear, siguiendo el esquema especificado en CreateOrderDto.
    - Respuesta: Devuelve los detalles del pedido creado en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- POST /order/:id
    - Descripción: Este endpoint crea un nuevo elemento de pedido para un pedido específico.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera el ID del pedido como un parámetro de ruta (Param), y los detalles del elemento de pedido a crear en el cuerpo de la solicitud (Body) en formato JSON, siguiendo el esquema especificado en CreateOrderItemDto.
    - Respuesta: Devuelve los detalles del elemento de pedido creado en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- GET /order
    - Descripción: Este endpoint devuelve todos los pedidos en el sistema.
    - Método HTTP: GET
    - Respuesta: Devuelve una lista de pedidos en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- GET /order/:id
    - Descripción: Este endpoint devuelve los detalles de un pedido específico según su ID.
    - Método HTTP: GET
    - Parámetros de entrada: Se espera el ID del pedido como un parámetro de ruta (Param).
    - Respuesta: Devuelve los detalles del pedido en formato JSON.

    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- PATCH /order/:id
    - Descripción: Este endpoint actualiza los detalles de un elemento de pedido específico según su ID.
    - Método HTTP: PATCH
    - Parámetros de entrada: Se espera el ID del elemento de pedido como un parámetro de ruta (Param), y los nuevos detalles del elemento de pedido a actualizar en el cuerpo de la solicitud (Body) en formato JSON, siguiendo el esquema especificado en UpdateOrderItemDto.
    - Respuesta: Devuelve los detalles actualizados del elemento de pedido en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- DELETE /order/:id
    - Descripción: Este endpoint elimina un pedido específico según su ID.
    - Método HTTP: DELETE
    - Parámetros de entrada: Se espera el ID del pedido como un parámetro de ruta (Param).
    - Respuesta: Devuelve un mensaje indicando el éxito de la operación en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

- DELETE /order/item/:id
    - Descripción: Este endpoint elimina un elemento de pedido específico según su ID.
    - Método HTTP: DELETE
    - Parámetros de entrada: Se espera el ID del elemento de pedido como un parámetro de ruta (Param).
    - Respuesta: Devuelve un mensaje indicando el éxito de la operación en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard

Products:

- GET /products
    - Descripción: Este endpoint devuelve todos los productos disponibles en el sistema.
    - Método HTTP: GET
    - Respuesta: Devuelve una lista de productos en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN, UserRole.SELLER, UserRole.BUYER

- GET /products/:id
    - Descripción: Este endpoint devuelve los detalles de un producto específico según su ID.
    - Método HTTP: GET

    - Parámetros de entrada: Se espera el ID del producto como un parámetro de ruta (Param).
    - Respuesta: Devuelve los detalles del producto en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN, UserRole.SELLER, UserRole.BUYER

- POST /products
    - Descripción: Este endpoint crea un nuevo producto en el sistema.
    - Método HTTP: POST
    - Parámetros de entrada: Se espera un cuerpo de solicitud (Body) en formato JSON que contiene los detalles del producto a crear, siguiendo el esquema especificado en CreateProductDto.
    - Respuesta: Devuelve los detalles del producto creado en formato JSON.
    - Código de Estado: 201 Created
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN, UserRole.SELLER

- PUT /products/:id
    - Descripción: Este endpoint actualiza los detalles de un producto específico según su ID.
    - Método HTTP: PUT
    - Parámetros de entrada: Se espera el ID del producto como un parámetro de ruta (Param), y los nuevos detalles del producto a actualizar en el cuerpo de la solicitud (Body) en formato JSON, siguiendo el esquema especificado en UpdateProductDto.
    - Respuesta: Devuelve los detalles actualizados del producto en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN, UserRole.SELLER
    - Validación de entrada (UsePipes): Se utiliza ParseUUIDPipe para validar el ID del producto.

- DELETE /products/:id
    - Descripción: Este endpoint elimina un producto específico según su ID.
    - Método HTTP: DELETE
    - Parámetros de entrada: Se espera el ID del producto como un parámetro de ruta (Param).
    - Respuesta: Devuelve un mensaje indicando el éxito de la operación en formato JSON.
    - Código de Estado: 200 OK
    - Guardias de uso (UseGuards): JwtAuthGuard
    - Roles requeridos: UserRole.ADMIN, UserRole.SELLER

- **Rating:**
- POST /rating
    - Descripción: Este endpoint permite crear una nueva calificación para un vendedor.
    - Parámetros de entrada:

    - rateDto (cuerpo de la solicitud): Un objeto que contiene la información de la calificación del vendedor. Este objeto sigue la estructura definida en RateSellerDto.
    - Respuesta: La respuesta depende de la lógica de la función create en RatingService.

- GET /rating
    - Descripción: Este endpoint devuelve todas las calificaciones de vendedores. Puede incluir paginación para limitar el número de resultados devueltos.
    - Parámetros de entrada:
    - paginationDto (parámetros de consulta): Un objeto que puede contener información de paginación, como limit (límite de resultados por página) y offset (desplazamiento).
    - Respuesta: La respuesta depende de la lógica de la función findAll en RatingService.

- GET /rating/:id
    - Descripción: Este endpoint devuelve la calificación de un vendedor específico identificado por su ID.
    - Parámetros de entrada:
    - id (parámetro de ruta): El ID del vendedor del que se desea obtener la calificación.
    - Respuesta: La respuesta depende de la lógica de la función getSellerRating en RatingService.

Autenticación y Autorización

En este proyecto, se implementó un sistema fuerte de autenticación y autorización utilizando JWT y Passport. El objetivo principal era garantizar la seguridad de la aplicación y controlar el acceso a los recursos protegidos.

La autenticación se maneja mediante la estrategia JWT de Passport. Cuando un usuario se autentica correctamente, se genera un token JWT firmado con una clave secreta, este token contiene información relevante del usuario, como su identificador y roles. En cada solicitud después de autenticarse,cada usuario debe incluir este token en el encabezado de autorización. La estrategia JWT de Passport se encarga de verificar la validez del token y extraer la información del usuario codificada en él.

Para la autorización, se utilizaron roles de usuario. Cada usuario tiene uno o más roles asignados (por ejemplo, administrador, vendedor, comprador). Estos roles se almacenan en la base de datos y se recuperan junto con la información del usuario durante la autenticación. Dependiendo de los roles del usuario, se pueden aplicar diferentes políticas de acceso a las rutas de la aplicación.

El módulo de autenticación (AuthModule) importa y configura los módulos necesarios, como JwtModule, PassportModule y TypeOrmModule (para interactuar con la base de datos). En el

módulo (User) Se define la entidad User que representa a un usuario en la base de datos, incluyendo campos como el correo electrónico, la contraseña y los roles.

La estrategia JWT (JwtStrategy) extiende la clase PassportStrategy y se encarga de validar los tokens JWT recibidos. Utiliza el repositorio de la entidad User para encontrar al usuario correspondiente en la base de datos y verificar que el token sea válido. Si el token no es válido, se lanza una excepción UnauthorizedException.

El servicio de autenticación (AuthService) maneja la lógica de autenticación, como verificar las credenciales del usuario y generar tokens JWT. El controlador de autenticación (AuthController) expone los endpoints necesarios para el inicio de sesión, el cierre de sesión y otras operaciones relacionadas con la autenticación.

Persistencia en la Base de Datos

La persistencia en la base de datos se implementa utilizando TypeORM para definir la estructura de las entidades y las relaciones, y mediante el uso de repositorios de TypeORM para ejecutar consultas y operaciones CRUD en la base de datos PostgreSQL.

- Cada entidad está decorada con anotaciones de TypeORM para definir su estructura y relaciones con otras entidades.
- Las anotaciones @Entity, @Column, @PrimaryGeneratedColumn, @ManyToOne, @OneToMany, entre otras, se utilizan para definir la estructura de la base de datos y las relaciones entre las entidades.

- En cuanto a los servicios contienen métodos para interactuar con su respectiva entidad como create, findAll, findOne, update, remove y remove.
- Estos métodos utilizan los repositorios de TypeORM para ejecutar consultas y operaciones CRUD en la base de datos, en caso de actualizar o eliminar en la base de datos dependiendo de la autorización del usuario
- Configuración de la Base de Datos (docker-compose.yml):
    - Se utiliza una imagen de Docker de PostgreSQL para ejecutar la base de datos.
    - Se proporcionan credenciales de acceso (POSTGRES_PASSWORD, POSTGRES_DB) y se mapea el puerto 5432 para acceder a la base de datos desde fuera del contenedor.

…

Desafíos y solución

Uno de los desafíos principales fue implementar un sistema de autenticación basado en tokens JWT y establecer roles y permisos para la autorización de rutas y funcionalidades. Necesitábamos asegurarnos de que los usuarios pudieran iniciar sesión y acceder a recursos protegidos de acuerdo con su rol asignado.

Otro de los mayores desafíos que enfrentamos durante el desarrollo de nuestra aplicación fue asegurarnos de que nuestras pruebas unitarias e integración cubrieran adecuadamente el código fuente y garantizaran la robustez y confiabilidad de la aplicación. Debido a los imports y la gestión de relaciones entre entidades de nuestras pruebas. En Nest.js, la estructura modular a veces complicaba las importaciones necesarias para las pruebas, lo que generaba errores. Para solucionarlo, revisamos la estructura de archivos y usamos mocks para simular comportamientos externos. Además, trabajar con relaciones entre entidades, especialmente con TypeORM, requería estrategias específicas para crear datos de prueba y configurar relaciones de manera eficiente.

Conclusiones y trabajo a Futuro

En conclusión, el desarrollo de esta aplicación backend con Nest.js y TypeORM nos permitió enfrentar desafíos significativos relacionados con la escritura de pruebas y la gestión de relaciones entre entidades. A través de la implementación de estrategias específicas y la revisión cuidadosa de nuestra estructura de archivos, logramos superar estos obstáculos. Para el futuro, planeamos continuar fortaleciendo nuestras pruebas, aumentando la cobertura del código y mejorando la calidad del software. Además, se va a continuar desarrollando el proyecto para incluir un frontend, lo que nos permitirá ofrecer una experiencia de usuario completa y funcional.

Requisitos mínimos:
Seed (10%):
Alimentar la base de datos con registros iniciales que permitan la realización de pruebas y/o su funcionamiento (usuario: admin).
Autenticación (10%):
Implementar un sistema de autenticación basado en tokens JWT (JSON Web Tokens).
Los usuarios deben poder iniciar sesión y cerrar sesión.
Deben haber rutas protegidas y requerir autenticación.
Autorización (20%):
Definir al menos dos roles diferentes o la cantidad que tenga su aplicación.
Establecer permisos basados en roles para restringir el acceso a ciertas rutas o funcionalidades.

Los roles deben asignarse mediante algún mecanismo de administración.

Pruebas (20%):

Implementar pruebas unitarias y de integración utilizando Jest o cualquier otra biblioteca de pruebas compatible con NestJS.

Las pruebas deben cubrir al menos el 80% del código fuente.

Persistencia en base de datos (5%):

Utilizar un ORM (por ejemplo, TypeORM) para interactuar con una base de datos relacional (puede ser MySQL, PostgreSQL, etc.).

Funcionalidades e informe (20%):

Implementar las funcionalidades necesarias en el backend para la aplicación escogida por su grupo.

Preparar un informe detallado que describa las funcionalidades implementadas en la API.

El informe debe incluir una descripción de cada endpoint, sus parámetros y respuestas. Además, debe explicar cómo se implementaron las características de autenticación, autorización y persistencia en la base de datos.

Despliegue (15%)

Se debe desplegar en algún servicio en nube.

Implementar pipelines para ejecutar las pruebas y despliegue automatizado

Entrega:

Los estudiantes deben presentar el código fuente del proyecto junto con un README que incluya instrucciones para ejecutar la aplicación y probar cada funcionalidad. Se revisarán commits para determinar el nivel de participación de los estudiantes. Además, se debe proporcionar un informe detallado que describa las funcionalidades implementadas en la API, explicando cómo se implementaron las características de autenticación, autorización y persistencia en la base de datos. Así como la ejecución de las pruebas.