Arquitectura Aplicación Comercio Electrónico

Stack Tecnológico

1. **PostgreSQL:**
    - Fiabilidad y robustez.
    - Escalabilidad.
    - Características avanzadas.
2. **Java Spring Boot:**
    - Rápido desarrollo.
    - Seguridad.
    - Soporte de la comunidad.
3. **React:**
    - Experiencia de usuario dinámica.
    - Rendimiento.
    - Ecosistema vibrante.

Backend

Descripción

La arquitectura del backend sigue un enfoque MVC (Modelo-Vista-Controlador) utilizando Spring Boot en Java. En este diseño, las entidades representan los modelos de datos de la aplicación, mientras que los DTOs manejan la transferencia de datos entre el backend y el frontend. Los repositorios JPA proporcionan una capa de acceso a datos, mientras que los servicios de negocio definen la lógica de la aplicación a través de interfaces y sus implementaciones correspondientes. Los controladores se encargan de manejar las solicitudes HTTP, conectando la lógica de negocio con el mundo exterior. Además, se incluye un paquete de seguridad para gestionar la autenticación y autorización de los usuarios, junto con un conjunto de utilidades para funciones comunes en todo el proyecto.

Patrones

Un patrón es una solución probada y generalizable para un problema recurrente en el diseño de software. Proporciona una estructura y un enfoque comunes para resolver problemas similares, promoviendo la reutilización y la consistencia en el desarrollo de software.

- Patrón MVC (Modelo-Vista-Controlador): Este patrón se refleja en la organización de los paquetes Controller, Service y Entity. Representa una forma de separar la lógica de presentación de la lógica de negocio y el acceso a los datos.
- Patrón de Seguridad con JWT (JSON Web Tokens): Representa la configuración de seguridad y autenticación de la aplicación utilizando tokens JWT para la autenticación de usuarios. Esta configuración se organiza en el paquete Security. Este patrón asegura que los datos y recursos estén protegidos y solo sean accesibles para usuarios autorizados mediante la validación de tokens JWT.

Arquitectura

1. **Entity:**
    - Crea un paquete para las clases de datos.
    - Define entidades utilizando anotaciones de JPA para mapearlas a tablas de base de datos.
2. **DTO:**
    - Crea un paquete para los DTOs.
    - Define objetos de transferencia de datos que representen las estructuras de datos utilizadas en las solicitudes y respuestas HTTP.
3. **IRepository:**
    - Crea un paquete para los repositorios de acceso a datos.
    - Utiliza Spring Data JPA para definir interfaces de repositorio que extiendan JpaRepository.
4. **IService:**
    - Crea un paquete para los servicios de negocio.
    - Define interfaces para la lógica de negocio utilizando el patrón de diseño Service.
5. **Service:**
    - Crea un paquete para las implementaciones de servicios.
    - Implementa la lógica de negocio en clases que implementen las interfaces de servicio.
6. **Controller:**
    - Crea un paquete para los controladores.
    - Define controladores utilizando anotaciones de Spring MVC para manejar las peticiones HTTP.
7. **Security:**
    - Crea un paquete para la configuración de seguridad.
    - Define clases de configuración para la seguridad utilizando Spring Security.
8. **Utils:**
    - Crea un paquete para utilidades comunes.
    - Define clases de utilidad para funcionalidades reutilizables en todo el proyecto.

Frontend

1. **Components:**
    - Organiza componentes en un directorio para mantener la modularidad y reutilización.
    - Divide los componentes según su función y responsabilidad, como `Header`, `Sidebar`, `ProductList`, etc.
2. **Services:**

- Crea servicios para manejar la lógica de negocio y las llamadas a la API.
- Utiliza servicios para encapsular la lógica de negocio y promover la reutilización de código.

3. **Pages:**
   - Crea páginas para representar las diferentes vistas de la aplicación.
   - Divide las páginas según la jerarquía de navegación y la funcionalidad, como `HomePage`, `ProductPage`, `CheckoutPage`, etc.

4. **Routing:**
   - Utiliza un enrutador para manejar la navegación entre las diferentes páginas de la aplicación.
   - Configura rutas para mapear URLs a componentes de React utilizando React Router.

5. **State Management:**
   - Utiliza un sistema de gestión de estado, como Redux o Context API, para gestionar el estado de la aplicación de manera centralizada.
   - Almacena datos globales y compartidos entre componentes para mantener la coherencia y facilitar las actualizaciones.

6. **Styling:**
   - Utiliza CSS Modules, Styled Components u otras bibliotecas de estilización para encapsular estilos y mejorar la mantenibilidad.
   - Aplica estilos de manera consistente y sigue las mejores prácticas de diseño para mejorar la experiencia del usuario.

7. **Testing:**
   - Implementa pruebas unitarias y de integración para garantizar la calidad y estabilidad del código.
   - Utiliza herramientas como Jest y React Testing Library para escribir y ejecutar pruebas de manera efectiva.

8. **Deployment:**
   - Configura un entorno de desarrollo, pruebas y producción para desplegar la aplicación de manera segura y eficiente.
   - Utiliza herramientas como Docker y Kubernetes para gestionar contenedores y orquestar el despliegue en entornos escalables.

Deployment

1. **Backend - GitHub Repository:**
   - Crea un repositorio en GitHub para el backend de la aplicación.
   - Utiliza este repositorio para almacenar y gestionar el código fuente del backend.

2. **Frontend - GitHub Repository:**
   - Crea un repositorio en GitHub para el frontend de la aplicación.
   - Utiliza este repositorio para almacenar y gestionar el código fuente del frontend.

3. **Integración Continua con Jenkins:**

    - Configura Jenkins para la integración continua del proyecto.
    - Utiliza Jenkins para automatizar el proceso de compilación, pruebas y despliegue del código.

4. **Despliegue con Docker Compose:**

    - Utiliza Docker Compose para definir y ejecutar entornos de desarrollo y producción consistentes.
    - Define servicios para el backend y frontend en un archivo `docker-compose.yml`.
    - Orquesta el despliegue de la aplicación utilizando Docker Compose para gestionar los contenedores.