1. Arquitectura de la Aplicación

La aplicación está estructurada en dos módulos principales: el **Front-End** y el **Back-End**.

- **Front-End:**
  Desarrollado en React, el Front-End se encarga de la interfaz de usuario. Se utiliza React Router para manejar la navegación y condicionalmente se muestran componentes según el estado de autenticación. Se han implementado técnicas de optimización de rendimiento como lazy loading para imágenes, minificación de archivos CSS y JavaScript (utilizando Create React App en modo producción) y optimización de recursos (por ejemplo, imágenes en formato WebP).

- **Back-End:**
  Desarrollado en Node.js con Express, el Back-End ofrece una API REST que conecta con la base de datos SQL utilizando Sequelize como ORM. Se definen rutas para la autenticación (login y registro), obtención de productos y gestión del carrito. Se implementan medidas de seguridad, tales como autenticación con JWT, sanitización de datos, protección contra CSRF y uso de middleware como Helmet para asegurar los headers.

- **Conexión Front-End / Back-End:**
  La comunicación se realiza mediante peticiones HTTP (usando Axios en el Front-End). El Front-End envía las credenciales y otras solicitudes (como agregar productos al carrito) al Back-End, que valida y responde con los datos correspondientes. Las rutas protegidas requieren un token JWT para su acceso, lo que garantiza que solo los usuarios autenticados puedan acceder a ciertas funcionalidades.

- **Base de Datos:**
  Se utiliza una base de datos SQL (MySQL o PostgreSQL) para almacenar la información. Se define un esquema con tablas para productos, usuarios y pedidos. Sequelize mapea estos modelos a las tablas y facilita la realización de operaciones CRUD y la gestión de relaciones, como que un usuario tenga múltiples pedidos.

2. Medidas de Seguridad Implementadas

Se han implementado varias medidas para garantizar la seguridad de la aplicación:

- **Autenticación y Autorización:**
  Se utiliza JSON Web Tokens (JWT) para autenticar a los usuarios. Durante el proceso de login, se genera un token que se almacena en el Front-End (en localStorage) y se utiliza para autorizar peticiones a rutas protegidas, como la gestión del carrito y la obtención de productos.

- **Protección contra Vulnerabilidades:**

  - **CSRF (Cross-Site Request Forgery):**
    Se utiliza el middleware csurf en el Back-End para proteger las rutas sensibles. Las rutas de autenticación se excluyen de esta protección, y se implementa un endpoint para obtener el token CSRF que luego se envía en las peticiones protegidas.

- **Sanitización de Datos:**
  Se aplica un middleware personalizado que recorre y limpia los datos entrantes para evitar inyecciones de código malicioso y ataques XSS (Cross-Site Scripting).
- **Helmet:**
  Se utiliza Helmet para configurar headers HTTP seguros y mitigar vulnerabilidades comunes en aplicaciones web.
- **Validación de Entradas:**
  Se emplea express-validator para asegurarse de que los datos enviados por el usuario cumplan con los formatos y restricciones establecidos.

3. Base de Datos

Esquema y Tablas

La base de datos está diseñada para gestionar la información principal de la aplicación. El esquema consta de las siguientes tablas:

- **Tabla Users:**
  - **Campos:**
    - id (clave primaria)
    - username
    - email
    - password (almacenada en forma encriptada mediante bcrypt)
    - createdAt
    - updatedAt
- **Tabla Products:**
  - **Campos:**
    - id (clave primaria)
    - name
    - description
    - price
    - stock
    - createdAt
    - updatedAt
- **Tabla Orders:**

- **Campos:**
    - id (clave primaria)
    - total
    - status (ejemplo: 'pending', 'completed')
    - userId (clave foránea referenciando a Users)
    - createdAt
    - updatedAt

Relaciones

- **Relación entre Users y Orders:**
  Un usuario puede tener múltiples pedidos. Esto se gestiona mediante una relación de uno a muchos en Sequelize, donde cada pedido está asociado a un usuario mediante el campo userId.
- **Productos:**
  Los productos se almacenan en una tabla separada, y aunque en este ejemplo simple la relación entre pedidos y productos se simula a través del manejo del carrito en memoria, en una versión más robusta se podría implementar una tabla intermedia para gestionar la relación entre pedidos y productos.

4. Desafíos Enfrentados

Durante el desarrollo de la aplicación se presentaron diversos retos, entre los cuales se destacan:

- **Integración Front-End / Back-End:**
    - **Desafío:**
      Lograr una comunicación fluida entre el Front-End y el Back-End, asegurando que las peticiones HTTP se realicen correctamente y que se manejen los tokens JWT para autenticación.
    - **Solución:**
      Se utilizó Axios para las peticiones HTTP y se implementó una lógica condicional en el Front-End (con React Router y estados) para controlar el acceso basado en la existencia del token. Se realizaron pruebas con Postman para validar el funcionamiento del Back-End.
- **Implementación de Medidas de Seguridad:**
    - **Desafío:**
      Proteger la aplicación de vulnerabilidades comunes (CSRF, XSS, inyección de SQL) sin afectar la experiencia del usuario, especialmente en el proceso de autenticación.

- **Solución:**
  Se implementó el middleware csurf y se configuraron excepciones para las rutas de autenticación. Además, se aplicó sanitización de inputs y se utilizó Helmet para asegurar que los headers HTTP sean adecuados.

- **Actualización y Sincronización del Carrito:**
  - **Desafío:**
    Lograr que la vista del carrito se actualice automáticamente cuando se agregan productos.
  - **Solución:**
    Se implementó un mecanismo de actualización de estado en el Front-End (utilizando un estado que se actualiza cada vez que se agrega un producto), el cual fuerza la recarga de los datos del carrito mediante un efecto en el componente correspondiente.

- **Depuración y Manejo de Errores:**
  - **Desafío:**
    Identificar y solucionar problemas en la comunicación entre el Front-End y el Back-End, como errores en la validación o en el flujo de autenticación.
  - **Solución:**
    Se utilizaron herramientas como la consola del navegador, logs en el Back-End y Postman para depurar y solucionar los problemas encontrados.