Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

1. Testing

Ubicación de las Pruebas

Las pruebas se encuentran en el directorio:

```
Shell
Proyecto/PosYa-app/backend/services/tests/commands/
```

Para ejecutar los tests, se encuentra un archivo shell "run-tests.sh" ubicado en la carpeta anterior.

Para verificar las funcionalidades esenciales, se crearon una serie de tests unitarios en el proyecto PosYa! Las pruebas están diseñadas para asegurar el correcto funcionamiento de las operaciones de la base de datos relacionadas con clientes, productos, vendedores y ventas.

El conjunto de pruebas fue diseñado para garantizar la confiabilidad, integridad y robustez del núcleo de la aplicación. La selección de casos de prueba se basa en los siguientes principios:

- Las pruebas de creación de clientes, productos, vendedores y ventas verifican que las **operaciones fundamentales del sistema** (CRUD) funcionen correctamente en escenarios ideales (también llamados Happy Path). Esto asegura que el flujo de trabajo principal de la aplicación sea estable.

- Pruebas como la que evita la duplicación de vendedores o la que impide crear una venta sin un cliente existente son importantes para **proteger la base de datos** contra registros corruptos o inconsistentes, asegurando que las relaciones entre entidades (como una venta y su cliente) se mantengan siempre válidas.

- El caso de prueba que diferencia entre clientes "naturales" y "jurídicos" es un ejemplo de cómo se valida la **lógica de negocio** específica de la aplicación. Estas pruebas aseguran que el sistema se comporte de acuerdo con los requisitos definidos.

- Las pruebas que esperan fallos (por ejemplo, al crear un producto con campos faltantes o eliminar uno que no existe) son esenciales. Confirman que la aplicación puede manejar **entradas inválidas o situaciones inesperadas** de forma controlada (también llamado Sad Path), devolviendo errores claros en lugar de fallar de manera abrupta. Esto hace que el sistema sea más robusto y predecible.

- La prueba de creación de una venta completa actúa como una mini-prueba de **integración**. Al requerir la creación previa de un cliente, un vendedor y un producto,

verifica que estos módulos, aunque probados de forma aislada, también funcionan correctamente en conjunto, simulando un caso de uso real.

Pruebas de Clientes

- **Prueba 1:** crear un cliente de tipo "natural"
  - *Objetivo*: asegurar que un cliente con tipoCliente: 'natural' se pueda crear exitosamente con todos sus campos (nombre, apellidos, etc.).
  - *Verificación*: comprueba que el objeto devuelto coincide con los datos de entrada esperados para un cliente natural.

- **Prueba 2:** crear un cliente de tipo "jurídica"
  - *Objetivo:* asegurar que un cliente con tipoCliente: 'juridica' se pueda crear exitosamente con su razón social.
  - *Verificación*: comprueba que el objeto devuelto coincide con los datos de entrada esperados para un cliente jurídico.

- **Prueba 3:** fallar si el tipo de cliente es desconocido
  - *Objetivo:* garantizar que el sistema rechace la creación de un cliente si el tipoCliente no es ni 'natural' ni 'juridica'.
  - *Verificación*: espera que la ejecución del comando lance una excepción con el mensaje "Tipo de cliente desconocido".

Pruebas de Productos

- **Prueba 1:** crear un producto correctamente
  - *Objetivo*: verificar que un producto se puede crear con todos sus atributos, incluyendo código, nombre, costo, precio e información de IVA.
  - Verificación: comprueba que el producto creado tiene los datos correctos y el estado "activo".

- **Prueba 2:** fallar si faltan campos obligatorios
  - *Objetivo:* asegurar que la creación de un producto falle si no se proporcionan todos los campos requeridos (por ejemplo, el nombre).
  - Verificación: espera que la ejecución del comando lance una excepción con el mensaje "Faltan campos obligatorios".

- **Prueba 3:** eliminar un producto correctamente
  - *Objetivo:* verificar que un producto existente se pueda eliminar.
  - *Verificación*: comprueba que el comando de eliminación retorna { ok: true } y que el producto ya no se encuentra en la base de datos.

- **Prueba 4:** fallar al eliminar un producto inexistente
  - O*bjetivo:* asegurar que el sistema maneje correctamente el intento de eliminar un producto que no existe.
  - *Verificación*: espera que la ejecución del comando lance una excepción con el mensaje "Producto no encontrado"

Pruebas de tipos de movimiento

- **Prueba 1:** crear un tipo de movimiento correctamente.

    - *Objetivo:* asegurar que se puede crear un nuevo tipo de movimiento con un nombre y un tipo de flujo ('Entrada' o 'Salida').
    - *Verificación*: la operación devuelve un resultado exitoso.

- **Prueba 2:** fallar si faltan campos requeridos.
    - *Objetivo:* garantizar que la creación falle si no se proporcionan campos esenciales como el nombre.
    - *Verificación:* se espera una excepción con el mensaje "Faltan campos requeridos".

- **Prueba 3:** actualizar un tipo de movimiento existente.
    - *Objetivo:* verificar que se pueden modificar los atributos de un tipo de movimiento existente.
    - *Verificación:* se actualiza un registro y se comprueba que la operación es exitosa.

- **Prueba 4:** fallar al actualizar un tipo inexistente.
    - *Objetivo:* asegurar que el sistema no permita actualizar un registro que no existe.
    - *Verificación:* se espera una excepción con el mensaje "Tipo de movimiento no encontrado".

- **Prueba 5:** eliminar un tipo de movimiento existente.
    - *Objetivo:* confirmar que un tipo de movimiento puede ser eliminado del sistema.
    - *Verificación:* se elimina un registro y se comprueba el resultado exitoso.

- **Prueba 6:** fallar al eliminar un tipo inexistente.
    - *Objetivo:* prevenir errores al intentar eliminar registros que no existen.
    - *Verificación:* se espera una excepción con el mensaje "Tipo de movimiento no encontrado".

- **Prueba 7:** consultar un tipo de movimiento por su código.
    - *Objetivo:* verificar que se puede recuperar un tipo de movimiento específico a través de su código.
    - *Verificación:* se comprueba que los datos del registro recuperado son correctos.

- **Prueba 8:** fallar si el tipo consultado no existe.
    - *Objetivo:* manejar el caso en que se consulta un código de un tipo que no existe.
    - *Verificación:* se espera una excepción con el mensaje "Tipo de movimiento no encontrado".

Pruebas de Vendedor

- **Prueba 1:** crear un vendedor y devolverlo correctamente
    - *Objetivo:* asegurar que se pueda crear un vendedor y que los datos se puedan recuperar correctamente.
    - *Verificación:* crea un vendedor y luego lo recupera, comprobando que los datos coincidan.

- **Prueba 2:** manejar la creación de un vendedor duplicado

- *Objetivo:* verificar que la aplicación no cree registros duplicados si se intenta crear el mismo vendedor dos veces. El comportamiento esperado es que la segunda operación sobreescriba a la primera.
- *Verificación:* intenta crear el mismo vendedor dos veces y confirma que solo existe un registro en la base de datos.

Pruebas de Ventas

- **Prueba 1:** crear una venta completa
  - *Objetivo:* simular y verificar la creación de una venta de extremo a extremo, incluyendo la creación previa de un cliente, un vendedor y un producto.
  - *Verificación:* ejecuta los comandos para crear las entidades necesarias y luego crea la venta. Finalmente, recupera la venta y comprueba que los datos (número de venta, cliente, total) sean correctos.

- **Prueba 2:** fallar si el cliente no existe
  - *Objetivo:* garantizar que no se pueda crear una venta si el cliente asociado no existe en la base de datos.
  - *Verificación:* intenta crear una venta con un identificacion_cliente inexistente y espera que el comando lance una excepción con el mensaje "Cliente no encontrado".

- **Prueba 3:** eliminar una venta correctamente
  - *Objetivo*: verificar que el sistema permita eliminar una venta existente y que también se eliminen sus detalles asociados.
  - *Verificación*: crea una venta válida y luego elimínala. Intenta recuperarla y espera que el comando lance una excepción con el mensaje "Venta no encontrada". Consulta los detalles de la venta y espera que retorne una lista vacía ([]).

Pruebas de Factus

- **Pruebas de Rangos de Numeración**
  - **Prueba 1:** obtener rangos de numeración correctamente
    - *Objetivo*: verificar que el sistema pueda obtener los rangos de numeración disponibles para facturas electrónicas desde Factus.
    - *Verificación*: comprueba que el comando retorne la estructura de datos esperada y que se haya llamado al servicio de Factus con los parámetros correctos.

  - **Prueba 2:** manejar errores al obtener rangos
    - *Objetivo*: garantizar que el sistema maneje adecuadamente los errores de conexión con Factus.
    - *Verificación*: espera que la ejecución del comando lance una excepción cuando el servicio de Factus falle.

- **Pruebas de Generación de Facturas**
  - **Prueba 1:** generar factura correctamente
    - *Objetivo*: verificar que una venta local pueda ser convertida en factura electrónica y enviada a Factus.
    - *Verificación*:
      - Configura una venta de prueba en la base de datos
      - Ejecuta el comando de generación
      - Comprueba que los datos transformados coincidan con el formato esperado por Factus

- Verifica que se haya llamado al servicio de envío a Factus

- **Prueba 2:** fallar si la venta no existe
  - *Objetivo*: asegurar que el sistema rechace la generación de facturas para ventas inexistentes.
  - *Verificación*: espera que la ejecución del comando lance una excepción con el mensaje "Venta no encontrada".

- **Prueba 3:** manejar errores al enviar a Factus
  - *Objetivo*: validar el manejo de errores cuando Factus rechace la factura.
  - *Verificación*: configura el mock para simular un error de validación y verifica que el comando propague el error correctamente.

- **Pruebas de Descarga de PDF**
  - **Prueba 1:** descargar PDF correctamente
    - *Objetivo*: verificar que el sistema pueda descargar el PDF de una factura generada en Factus.
    - *Verificación*: comprueba que el comando retorne el nombre del archivo y el contenido en base64.

  - **Prueba 2:** manejar errores al descargar PDF
    - *Objetivo*: asegurar el manejo adecuado cuando el PDF no esté disponible.
    - *Verificación*: configura el mock para simular un error y verifica que el comando lance la excepción correspondiente.

- **Pruebas de Descarga de XML**
  - **Prueba 1:** descargar XML correctamente
    - *Objetivo*: verificar que el sistema pueda descargar el XML de una factura generada en Factus.
    - *Verificación*: comprueba que el comando retorne el nombre del archivo y el contenido en base64.

  - **Prueba 2:** manejar errores al descargar XML
    - *Objetivo*: asegurar el manejo adecuado cuando el XML no esté disponible.
    - *Verificación*: configura el mock para simular un error y verifica que el comando lance la excepción correspondiente.

2. Análisis estático de código (linter)

2.1. Nombre de la herramienta utilizada:

ESLint: es una herramienta para analizar código JavaScript (y también TypeScript), la cual, se encarga de encontrar errores, inconsistencias y problemas de estilo. Ayuda a mantener un código limpio, legible y libre de errores comunes.

¿Para qué sirve?

- Detecta errores de sintaxis.

- Marca variables no utilizadas.

- Hacer cumplir reglas de estilo (por ejemplo: uso de comillas simples o dobles, sangrías, espacios, etc.).
- Ayuda a aplicar buenas prácticas de programación.
- Permite configuración personalizada para el equipo y el proyecto.
- Puede corregir automáticamente muchos problemas.

2.2. Configuración aplicada (hay reglas personalizadas)

Se ha implementado una estrategia equilibrada, transformando la mayoría de las reglas de estilo "error" a **"advertencia"** ('warn') y conservando las detecciones de errores críticos. El objetivo es mantener una alta calidad del código y consistencia, sin obstaculizar excesivamente el flujo de trabajo del equipo.

Visión general de la configuración de ESLint

La configuración de ESLint se basa en el formato de **Flat Config**, utilizando eslint.config.js; Incluye las configuraciones recomendadas por ESLint (js.configs.recommended), así como las reglas específicas de React Hooks y Vite para React Fast Refresh (reactHooks.configs['recommended-latest'] y reactRefresh.configs.vite).

El foco principal de esta personalización ha sido la sección rules, donde se han ajustado las directivas para encontrar un punto medio entre la estrictez total y la permisividad completa.

Análisis e impacto de las reglas modificadas

A continuación, se presenta un desglose de las reglas ajustadas y su impacto previsto:

1. 'no-unused-vars': ['warn', { varsIgnorePattern: '^[A-Z_]' }]

- **Ajuste anterior ('off'):** permisividad total, sin advertencias sobre variables, funciones o parámetros no utilizados. Esto podía llevar a código redundante.
- **Ajuste actual ('warn'):** ahora, ESLint emitirá una **advertencia** si detecta variables, funciones o parámetros que no se usan. Esto permite a los desarrolladores del equipo identificar y limpiar el código muerto sin bloquear la compilación o ejecución. La opción varsIgnorePattern: '^[A-Z_]' permite seguir ignorando variables que comienzan con mayúsculas o guiones bajos, a menudo usadas para constantes globales o componentes sin usar todavía.
- **Beneficio:** mejora la limpieza del código y la legibilidad al señalar posibles elementos superfluos, manteniendo la flexibilidad durante el desarrollo.

2. 'no-console': 'warn'

- **Ajuste anterior ('off'):** uso libre de console sin ninguna notificación.
- **Ajuste actual ('warn'):** ESLint ahora emitirá una **advertencia** cuando se detecte el uso de console.log, console.warn, etc.
- **Beneficio:** permite la depuración activa durante el desarrollo, pero proporciona un recordatorio visual para eliminar o refactorizar las sentencias console antes de un despliegue a producción, evitando la exposición de información de depuración o un impacto en el rendimiento.

3. 'semi': ['warn', 'always']

- **Ajuste anterior ('off'):** sin control sobre el uso de puntos y coma.
- **Ajuste actual (['warn', 'always']):** ESLint ahora emitirá una **advertencia** si falta un punto y coma al final de una declaración. Se ha optado por el estilo "always" (siempre requeridos).
- **Beneficio:** promueve la consistencia en el estilo de codificación. Aunque la omisión de puntos y coma es una preferencia, su uso explícito puede prevenir errores de Análisis Automático de Inserción de Punto y Coma (Automatic Semicolon Insertion - ASI) en JavaScript. Los desarrolladores son notificados, pero no se les impide compilar.

4. 'indent': ['warn', 2, { SwitchCase: 1 }]

- **Ajuste anterior ('off'):** sin reglas de indentación impuestas.
- **Ajuste actual (['warn', 2, { SwitchCase: 1 }]):** ESLint ahora emitirá una **advertencia** si la indentación no sigue la regla de **dos espacios**. La opción SwitchCase: 1 aplica una indentación de un nivel para los casos dentro de una sentencia switch.
- **Beneficio:** mejora significativamente la legibilidad y la consistencia visual del código entre diferentes desarrolladores y archivos, sin forzar una corrección inmediata que detenga el trabajo.

5. 'no-multiple-empty-lines': ['warn', { max: 1, maxEOF: 0 }]

- **Ajuste anterior ('off'):** permiso para cualquier número de líneas vacías consecutivas.
- **Ajuste actual (['warn', { max: 1, maxEOF: 0 }]):** ESLint ahora emitirá una **advertencia** si hay más de una línea vacía consecutiva dentro del código (max: 1) o si hay líneas vacías al final del archivo (maxEOF: 0).
- **Beneficio:** mantiene el código más compacto y legible, evitando bloques de espacio excesivo que pueden reducir la claridad.

6. 'curly': ['warn', 'multi-line']

- **Ajuste anterior ('off'):** permitía omitir llaves en cualquier sentencia de control de flujo.
- **Ajuste actual (['warn', 'multi-line']):** ESLint emitirá una **advertencia** si las llaves se omiten en bloques de código que abarcan **múltiples líneas**. Sin embargo, permite omitirlas para sentencias de una sola línea, lo que ofrece un compromiso entre concisión y seguridad.
- **Beneficio:** promueve una práctica de codificación que reduce la probabilidad de errores lógicos y mejora la claridad, especialmente en bloques más complejos.

7. 'react/react-in-jsx-scope': 'off'

- **Estado anterior y actual:** permanece en 'off'.
- **Justificación:** esta regla es obsoleta para proyectos que utilizan React 17 o superior con el nuevo transformador JSX, ya que no se requiere la importación explícita de React en cada archivo.

8. 'react/prop-types': 'off'

- **Estado anterior y actual:** permanece en 'off'.

- **Justificación:** esta regla de validación de props es redundante y se desactiva cuando se utiliza **TypeScript** en el proyecto, ya que TypeScript proporciona una verificación de tipos superior y en tiempo de compilación para las props.

9. 'no-var': 'warn'

- **Ajuste anterior ('off'):** uso libre de var.
- **Ajuste actual ('warn'):** ESLint ahora emitirá una **advertencia** si se utiliza la palabra clave var.
- **Beneficio:** desalienta el uso de var en favor de let y const, que son las prácticas recomendadas en el JavaScript moderno debido a su mejor manejo del alcance. Sin embargo, no lo prohíbe completamente, permitiendo flexibilidad en código heredado.

10. 'quotes': ['warn', 'single']

- **Ajuste anterior ('off'):** sin control sobre el tipo de comillas.
- **Ajuste actual (['warn', 'single']):** ESLint emitirá una **advertencia** si se utilizan comillas dobles en lugar de comillas simples para las cadenas de texto. Se establece una preferencia por las comillas simples ('single').
- **Beneficio:** promueve la consistencia estilística en las cadenas de texto, mejorando la uniformidad del código.

Nuevas reglas añadidas (en modo Advertencia)

Se han incorporado reglas adicionales para reforzar las buenas prácticas sin ser demasiado restrictivas:

- **'prefer-const': 'warn'**: ESLint advertirá si una variable declarada con let nunca se reasigna, sugiriendo que se cambie a const. Promueve el uso de constantes para mejorar la claridad y evitar reasignaciones accidentales.
- **'eqeqeq': ['warn', 'always']**: ESLint advertirá si se utiliza el operador de igualdad no estricta (== o !=) en lugar del operador de igualdad estricta (=== o !==). Fomenta el uso de la igualdad estricta para prevenir conversiones de tipo inesperadas y errores lógicos.
- **'arrow-body-style': ['warn', 'as-needed']**: ESLint advertirá si el cuerpo de una función flecha es un bloque de sentencias cuando podría ser una expresión concisa. Promueve un estilo más limpio y conciso para las funciones flecha simples.
- **'space-infix-ops': 'warn'**: ESLint advertirá si faltan espacios alrededor de los operadores (+, -, =, etc.). Mejora la legibilidad del código al asegurar un espaciado consistente.
- **'no-trailing-spaces': 'warn'**: ESLint advertirá si hay espacios en blanco al final de las líneas. Eliminar estos espacios invisibles ayuda a mantener el código más limpio y evita problemas en sistemas de control de versiones.

La configuración de ESLint actualizada representa un enfoque más maduro y equilibrado para el linting en el proyecto. Al convertir la mayoría de las reglas de estilo en advertencias en lugar de errores, se mantiene la capacidad de ESLint para señalar posibles áreas de mejora y mantener la consistencia del código, mientras se reduce la fricción en el desarrollo diario.

Este balance nos permite a los desarrolladores del proyecto trabajar de manera más fluida, aprendiendo de las sugerencias del linter sin que estas impidan la compilación. A largo plazo, esta aproximación contribuirá a un código más mantenible, legible y con menos errores sutiles.

2.3. Resultados obtenidos: (✖ 30 problems (3 errors, 27 warnings))

1. Configuración de ESLint (equilibrada)

Hemos ajustado ESLint para que la mayoría de las reglas de estilo sean **advertencias** ('warn') en lugar de errores, permitiendo flexibilidad sin sacrificar la visibilidad de posibles mejoras.

Reglas clave configuradas como warn:

- **no-unused-vars**: advierte sobre variables no utilizadas, ayudando a limpiar el código.
- **no-console**: advierte sobre sentencias console, útiles para depuración pero a eliminar en producción.
- **semi**: advierte sobre la ausencia de puntos y coma (configurado para preferir siempre).
- **indent**: advierte sobre indentación inconsistente (configurado a 2 espacios).
- **no-multiple-empty-lines**: advierte sobre exceso de líneas vacías.
- **curly**: advierte sobre la omisión de llaves en bloques de código de múltiples líneas.
- **no-var**: advierte sobre el uso de var en lugar de let/const.
- **quotes**: advierte sobre el uso inconsistente de comillas (configurado a simples).

Reglas desactivadas (off) por redundancia o conveniencia:

- **react/react-in-jsx-scope**: obsoleto en React moderno.
- **react/prop-types**: reemplazado por la verificación de tipos de TypeScript.

2. Resultados del último linting

La ejecución reciente de ESLint mostró:

✖ 30 problems (3 errors, 27 warnings)

2.1. Errores críticos (3)

Estos problemas deben resolverse de inmediato, ya que ESLint los considera críticos y podrían indicar fallos o ambigüedades en el código:

- **no-useless-escape**: dos instancias en EdicionVendedor.jsx (línea 27). Caracteres de escape innecesarios que pueden confundir o ser errores tipográficos.
- **no-empty**: una instancia en EdicionVendedor.jsx (línea 79). Un bloque de código vacío que sugiere una lógica incompleta o un error.

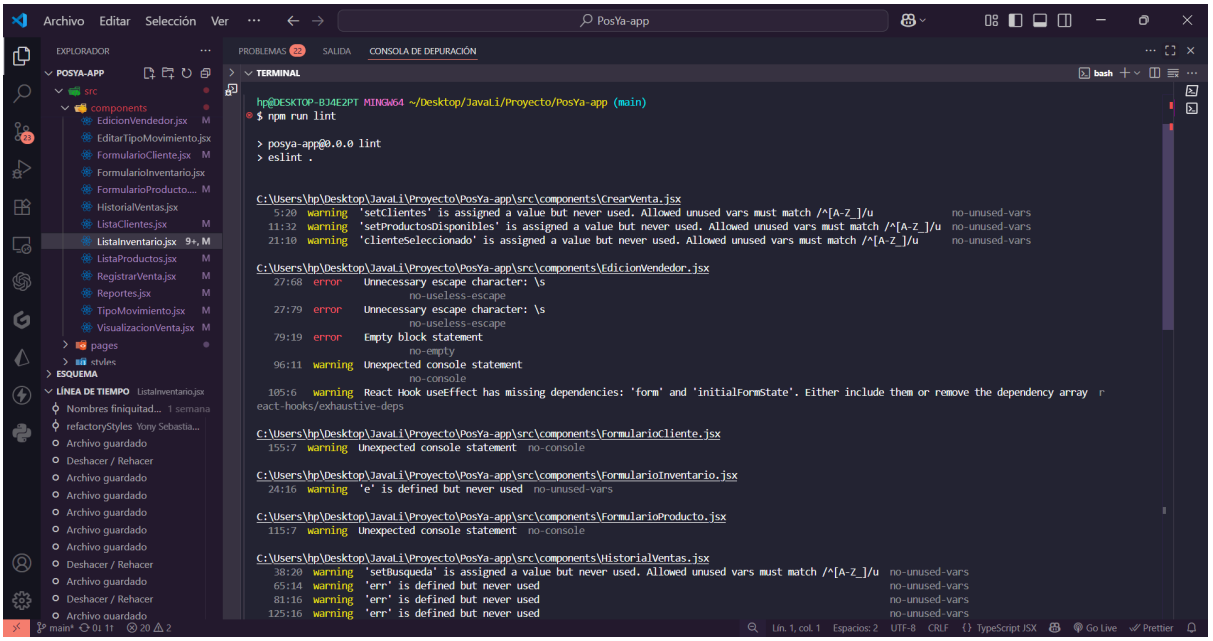
2.2. Advertencias (27)

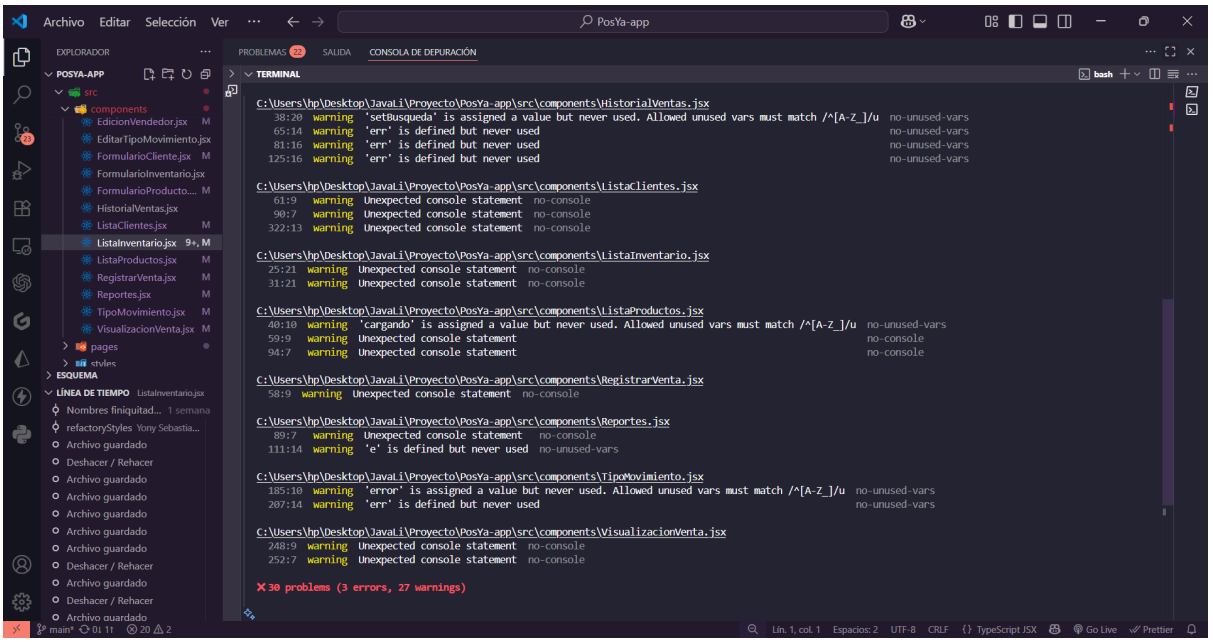
Estas son sugerencias para mejorar la calidad y la consistencia del código sin detener el proceso de desarrollo:

- **no-unused-vars (Múltiples archivos)**: variables declaradas pero no usadas, indicando posible código redundante.
- **no-console (Múltiples archivos)**: uso de sentencias console que deberían eliminarse en producción.
- **react-hooks/exhaustive-deps (EdicionVendedor.jsx, línea 105)**: dependencias faltantes en useEffect, lo cual puede llevar a bugs sutiles en los hooks de React.

2.4. Evidencia de ejecución

Capturas de pantalla de la salida en la consola de depuración.





Texto de la salida en la consola de depuración:

```
Shell
 npm run lint

> posya-app@0.0.0 lint
> eslint .


C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\CrearVenta.jsx
   5:20  warning  'setClientes' is assigned a value but never used. Allowed
unused vars must match /^[A-Z_]/u              no-unused-vars
  11:32  warning  'setProductosDisponibles' is assigned a value but never
used. Allowed unused vars must match /^[A-Z_]/u  no-unused-vars
  21:10  warning  'clienteSeleccionado' is assigned a value but never used.
Allowed unused vars must match /^[A-Z_]/u      no-unused-vars

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\EdicionVendedor
.jsx
   27:68  error    Unnecessary escape character: \s
                          no-useless-escape
   27:79  error    Unnecessary escape character: \s
                          no-useless-escape
   79:19  error    Empty block statement
                          no-empty
   96:11  warning  Unexpected console statement
                          no-console
  105:6   warning  React Hook useEffect has missing dependencies: 'form' and
'initialFormState'. Either include them or remove the dependency array
react-hooks/exhaustive-deps

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\FormularioClien
te.jsx
  155:7  warning  Unexpected console statement  no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\FormularioInven
tario.jsx
  24:16  warning  'e' is defined but never used  no-unused-vars

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\FormularioProdu
cto.jsx
  115:7  warning  Unexpected console statement  no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\HistorialVentas
.jsx
   38:20  warning  'setBusqueda' is assigned a value but never used. Allowed
unused vars must match /^[A-Z_]/u  no-unused-vars
```

```
   65:14  warning  'err' is defined but never used
no-unused-vars
   81:16  warning  'err' is defined but never used
no-unused-vars
  125:16  warning  'err' is defined but never used
no-unused-vars

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\ListaClientes.j
sx
   61:9   warning  Unexpected console statement  no-console
   90:7   warning  Unexpected console statement  no-console
  322:13  warning  Unexpected console statement  no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\ListaInventario
.jsx
  25:21  warning  Unexpected console statement  no-console
  31:21  warning  Unexpected console statement  no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\ListaProductos.
jsx
  40:10  warning  'cargando' is assigned a value but never used. Allowed
unused vars must match /^[A-Z_]/u  no-unused-vars
  59:9   warning  Unexpected console statement
no-console
  94:7   warning  Unexpected console statement
no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\RegistrarVenta.
jsx
  58:9  warning  Unexpected console statement  no-console

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\Reportes.jsx
   89:7   warning  Unexpected console statement   no-console
  111:14  warning  'e' is defined but never used  no-unused-vars

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\TipoMovimiento.
jsx
  185:10  warning  'error' is assigned a value but never used. Allowed
unused vars must match /^[A-Z_]/u  no-unused-vars
  207:14  warning  'err' is defined but never used
no-unused-vars

C:\Users\hp\Desktop\JavaLi\Proyecto\PosYa-app\src\components\VisualizacionVe
nta.jsx
  248:9  warning  Unexpected console statement  no-console
  252:7  warning  Unexpected console statement  no-console

✖ 30 problems (3 errors, 27 warnings)
```