

Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Acatlán



Transformación Digital de Facturas: Facilitando el Análisis de Ventas en una Empresa

Integrantes:

- Segura Alejo Cesar Ricardo
- Castillo Gómez Enrique Rubén
- Arreola Calderón Jesús Enrique
- Jonatan Sarmiento Ibarra

Fecha de entrega: Lunes 20 de mayo del 2024

Índice

OBJETIVO DEL PROYECTO.....	1
DESTINATARIO DEL PROYECTO.....	2
PROBLEMA.....	3
1.1 Planteamiento del problema	3
1.2 Propuesta de solución del problema	4
1.2.1 Planteamiento de la solución	4
1.2.2 Análisis de datos	4
1.2.3 Diseño web	8
1.2.4 Uso de Swagger	10
1.2.5 Uso de SpringBoot	13
1.2.6 Uso de React Js	14
CONCLUSIÓN.....	18
BIBLIOGRAFÍA.....	19

Tabla de imágenes

Imagen 1. Vista del usuario	8
Imagen 2. Registrar Producto.....	8
Imagen 3. Opciones de Usuario	8
Imagen 4. Consultar Producto	9
Imagen 5. Actualizar Producto	9
Imagen 6. Eliminar Producto	9
Imagen 7. Uso de Swagger	12
Imagen 8. Uso de Swagger	12
Imagen 9. Uso de Swagger	12
Imagen 10. Uso de Swagger	13
Imagen 11. Código del archivo App.js.....	15
Imagen 12. Código del Archivo Inicio.jsx	15
Imagen 13. Código del archivo BarraNav.jsx.....	16
Imagen 14. Código del archivo ProductoController.java.....	16
Imagen 15. Código del archivo ProductoServicio.java	17
Imagen 16. Código del archivo ProductoRepositorio.java.....	17
Imagen 17. Código del archivo Productos.java.....	17

OBJETIVO DEL PROYECTO

El objetivo principal del proyecto "Transformación Digital de Facturas: Facilitando el Análisis de Ventas en una Empresa" es mejorar la capacidad de una empresa para analizar sus ventas mediante la digitalización y automatización del procesamiento de facturas que actualmente se encuentran en formato de texto. Este proceso busca convertir los datos no estructurados de las facturas en datos estructurados, lo que permitirá una mayor eficiencia y precisión en el análisis de ventas.

En primer lugar, la digitalización de facturas implica la conversión de documentos físicos o archivos de texto en un formato digital que sea legible por máquinas (Wang, 2019). Esto no solo facilita el almacenamiento y la recuperación de datos, sino que también permite la integración con sistemas de análisis de datos y herramientas de inteligencia empresarial. Según un estudio de Smith (2020), las empresas que han adoptado la digitalización de documentos han visto una mejora significativa en la eficiencia operativa, con una reducción del 40% en el tiempo dedicado a la gestión de documentos y un aumento del 30% en la precisión de los datos procesados.

Un beneficio adicional de este proyecto es la mejora en la transparencia y la trazabilidad de las transacciones comerciales. La digitalización de facturas proporciona un registro claro y accesible de todas las ventas, lo que facilita las auditorías internas y externas. Según García (2019), la capacidad de realizar auditorías más eficientes y precisas es una ventaja significativa para las empresas que buscan cumplir con regulaciones financieras y mejorar su gobernanza corporativa.

DESTINATARIO DEL PROYECTO

El proyecto “Transformación Digital de Facturas: Facilitando el Análisis de Ventas en una Empresa” está dirigido a diversos grupos de interés dentro y fuera de la empresa. Estos incluyen principalmente a los departamentos internos de la empresa, los directivos y gestores, los auditores internos y externos, así como a los clientes y proveedores. Cada uno de estos grupos puede beneficiarse de diferentes maneras a través de la implementación de este proyecto.

En primer lugar, los departamentos internos de la empresa, especialmente el departamento de ventas, el departamento financiero y el de tecnología de la información (TI), son los principales destinatarios del proyecto. El departamento de ventas se beneficiará significativamente ya que la digitalización y automatización de las facturas permitirá un acceso más rápido y preciso a los datos de ventas. Esto facilita la generación de informes de ventas, el seguimiento de las tendencias de ventas y la identificación de oportunidades de crecimiento (Smith, 2020). Por otro lado, el departamento financiero verá una mejora en la precisión y la eficiencia de sus procesos de contabilidad y auditoría. Según García (2019), la digitalización de documentos reduce los errores humanos y el tiempo dedicado a la conciliación de facturas, lo cual es crucial para mantener la integridad financiera. El departamento de TI también es un destinatario clave, ya que será responsable de implementar y mantener las tecnologías necesarias para la digitalización y automatización de las facturas.

Otro grupo importante de destinatarios son los auditores internos y externos. La digitalización de las facturas mejora la transparencia y la trazabilidad de las transacciones comerciales, lo que facilita el proceso de auditoría. Esto no solo reduce el tiempo y los costos asociados con las auditorías, sino que también mejora la conformidad con las normativas y regulaciones financieras (Wang, 2019).

Finalmente, los clientes y proveedores también se benefician indirectamente de este proyecto. La eficiencia operativa mejorada y la precisión en la gestión de facturas pueden llevar a una mejor experiencia del cliente, con tiempos de respuesta más rápidos y menos errores en la facturación. Además, los proveedores pueden experimentar un procesamiento más rápido y preciso de sus facturas, lo que mejora las relaciones comerciales y la confianza en la empresa.

PROBLEMA

1.1 Planteamiento del problema

Cierta empresa necesita realizar un análisis de sus ventas a partir de sus productos vendidos, pero los datos de dichas ventas (facturas) se encuentran en archivos de textos, lo cual les impide realizar dicho análisis dado que existe un sinnúmero de facturas.

Los archivos de textos tienen la siguiente estructura:

H	F	A	C	0	2	3	1	1	E	0	0	4	2	G	U	E	2	0	2	4	0	2	1	1	A
I	0	0	3	1				2									4	0	.	0	0				
I	0	1	0	3				5								1	2	5	.	0	0				
T			2						1	6	5	.	0	0											

Donde:

	Número de factura
	Método de pago
	Número de cliente
	Estado donde se emitió la factura
	Fecha de emisión
	Precio del producto
	Cantidad del producto
	Antigüedad del producto
	Número clave del producto
	Cantidad de tipos de productos
	Total de la factura
	Estatus de factura: A, aprobada; y C, cancelada.

1.2 Propuesta de solución del problema

1.2.1 Planteamiento de la solución

Se realizará un sistema de software para leer los archivos de textos donde se encuentran las facturas. Se creará una base de datos (modelo entidad/relación) para almacenar los datos correspondientes a las facturas y de esta forma facilitar el análisis a dicha empresa.

1.2.2 Análisis de datos

De acuerdo a los datos mostrados en la factura se tienen las siguientes tablas de acuerdo al modelo entidad relación:

1. Factura
2. Estado
3. Producto
4. Cliente
5. MétodoPago
6. FacturaProducto: Dado que una factura puede tener muchos productos y un producto puede estar en muchas facturas, se establece esta relación muchos a muchos respecto a la tabla Factura y Producto.
7. Bitácora: Se deberá crear también una bitácora para almacenar información acerca del procesamiento de cada factura. Es decir, la tabla bitácora deberá contener:
 1. Si los datos de la factura se almacenaron correctamente.
 2. Caso contrario a lo anterior, guardar cuál fue el error de dicho procesamiento incorrecto.

Los campos correspondientes a las tablas anteriores son:

Los campos correspondientes a las tablas anteriores son:

1. **Factura:**
 1. Número de factura. Campo numérico (entero)
 2. Número de cliente. Campo numérico (entero)
 3. Cantidad de tipos de productos Campo numérico (entero)
 4. Total. Campo numérico (decimal)

5. Método de pago. Dicho campo será una llave foránea relacionada con la "Clave del método de pago" de la tabla "MetodoPago".
6. Fecha de emisión. Campo date (formato: AAAA/MM/DD)
7. Estado. Llave foránea relacionada con la "Clave de estado" de la tabla "estado".
8. Estatus de la factura. Llave foránea relacionada a la "clave" del estatus "cve_status".

2. **Estatus:**

1. Clave del estatus. Campo carácter de longitud 1 (A, C, P).
2. Tipo. Campo carácter de longitud 10 (A, aprobada; C, cancelada; y P, en proceso)

3. **Producto:**

1. Clave de producto. Campo numérico (entero)
2. Nombre. Campo carácter (longitud: 50)
3. Precio unitario. Campo numérico (decimal)
4. Antigüedad por mes. Es decir, cuantos meses lleva en el almacén. Campo numérico (entero)

4. **FacturaProducto:**

1. Número de la factura. Llave foránea relacionada con el "Número de factura" de la tabla "Factura".
2. Clave de producto. Llave foránea relacionada con la "clave de producto" de la tabla "Producto".
3. Cantidad del producto. Campo numérico (entero) para identificar la cantidad vendida de dicho producto.

5. **MetodoPago:**

1. Clave del método de pago. Campo carácter (longitud: 1), donde: C, Tarjeta. T, transferencia. E, Efectivo
2. Tipo (efectivo, transferencia y depósito). Campo carácter (20)

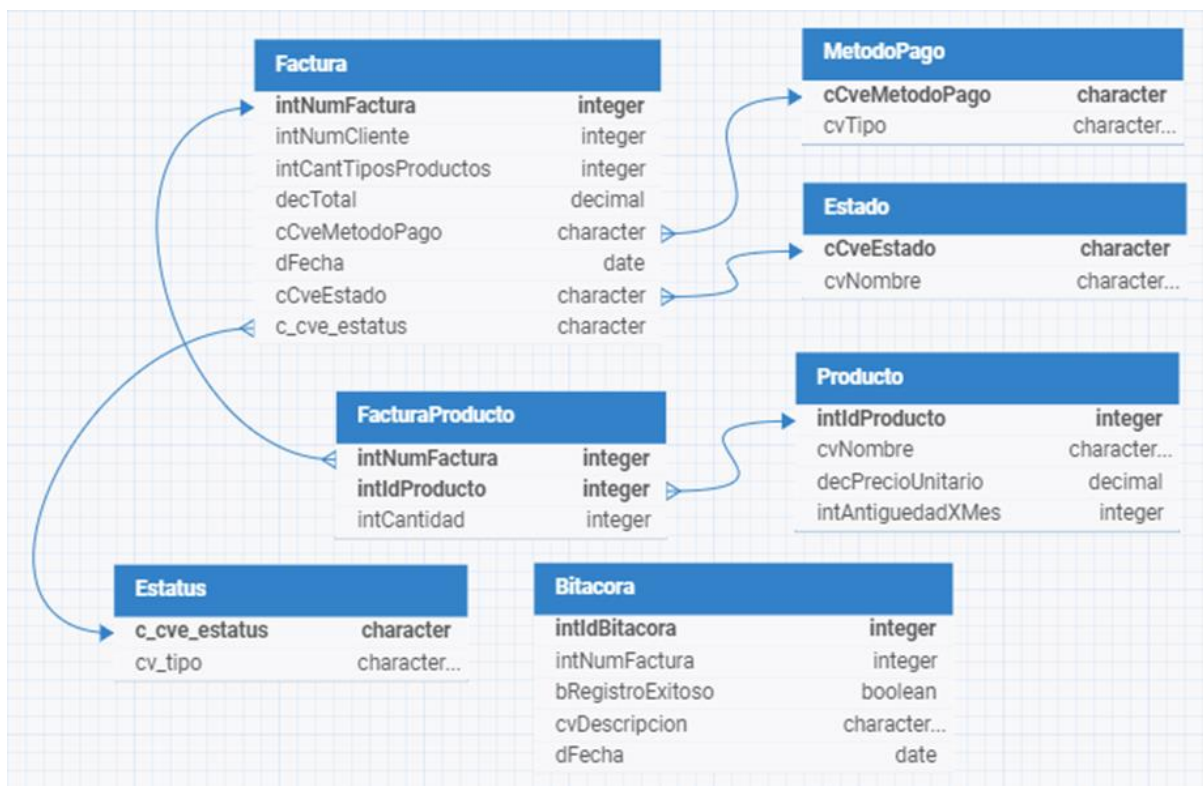
6. **Estado:**

1. Clave de estado. Campo carácter (longitud: 3)
2. Nombre. Campo carácter (longitud: 50), donde:

7. **Bitacora:**

1. Clave de bitácora. Campo numérico (entero)
2. Número de factura. Campo numérico (entero).
3. Registro exitoso. Campo booleano, donde: TRUE, registro exitoso; FALSE, registro fallido.
4. Descripción. Campo carácter (longitud: 500). En caso de registro fallido, se describe el error por el cual no se pudo realizar dicho registro, en caso contrario sólo se ingresa la leyenda "El registro de la factura [Número factura] ha sido exitoso".
5. Fecha. Campo date que indica la fecha en que se realizó el registro de la factura en la base de datos.

Modelo entidad/relación realizado con la herramienta "dbdesigner.net".



Base de datos realizada con el SMBD **MySQL**.

Algoritmo para leer el archivo

La primera línea identificada por "H" (Header) contiene 6 datos:

	Dato	Posición en el archivo (iteración inicia en 0)
1	Número de la factura	4-8
2	Método de pago	9
3	Número de cliente	10-13
4	Estado de emisión de factura	14-16
5	Fecha de emisión de la factura	17-24
6	Estatus	25

Las siguientes líneas identificadas por "I" (Items) contienen cada una 4 datos:

	Dato	Posición
1	Clave del producto	1-3
2	Antigüedad del producto por mes	4-5
3	Cantidad de productos	6-8
4	Precio total de productos	9-25

La siguiente línea identificada por "T" (Total) contiene 2 datos:

	Dato	Posición
1	Cantidad de tipos de productos	1-3
2	Total de la factura	4-25

1.2.3 Diseño web



Imagen 1. Opciones de Usuario

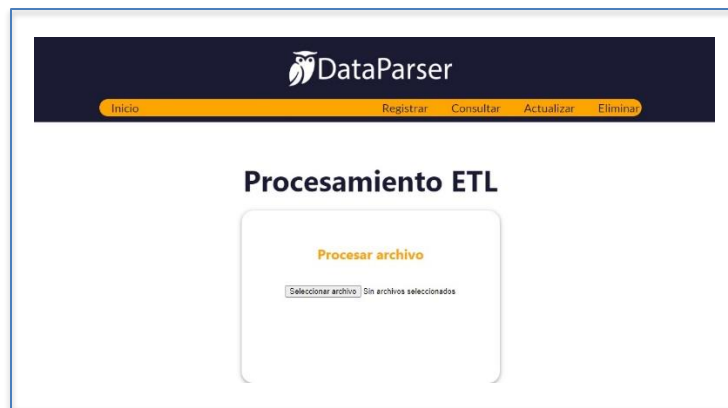


Imagen 2. Vista de inicio del sistema

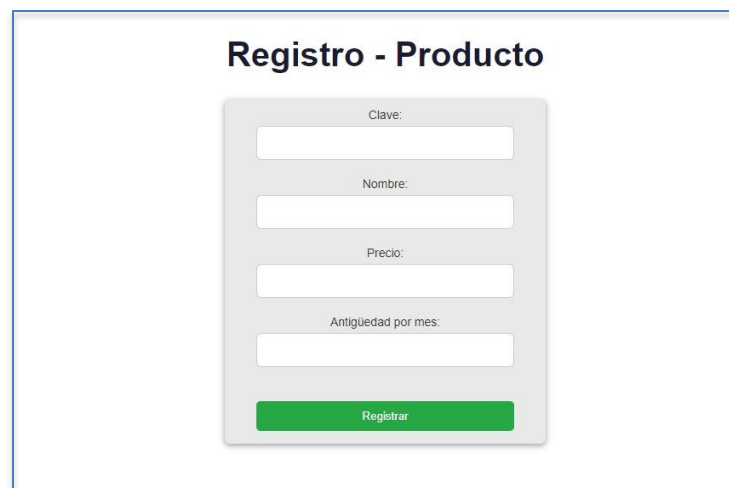


Imagen 3. Registrar Producto

The screenshot shows the 'DataParser' application header with a navigation bar containing 'Inicio', 'Registrar', 'Consultar', 'Actualizar', and 'Eliminar'. Below the header, the 'Producto' section is displayed. On the left, there is a 'Consultar:' form with two radio buttons: 'Todos' (selected) and 'Por Clave:'. A blue 'Consultar' button is at the bottom of the form. To the right of the form is a large, solid dark blue square, likely a placeholder for product data.

Imagen 4. Consultar Producto

The screenshot shows the 'Actualizar - Producto' form. On the left, there is a box labeled 'Productos existentes:' containing the text 'No se encontraron productos.' To the right, there are four input fields: 'Clave:', 'Nombre:', 'Precio:', and 'Antigüedad por mes:'. A blue 'Registrar' button is located at the bottom right of the form.

Imagen 5. Actualizar Producto

The screenshot shows the 'Eliminar - Producto' form. On the left, there is a box labeled 'Productos existentes:' containing the text 'No se encontraron productos.' To the right, there are four input fields: 'Clave:', 'Nombre:', 'Precio:', and 'Antigüedad por mes:'. A blue 'Eliminar' button is located at the bottom right of the form.

Imagen 6. Eliminar Producto

1.2.4 Uso de Swagger

Swagger es una herramienta de documentación y especificación de APIs que ha ganado popularidad en el desarrollo de software debido a su capacidad para facilitar la creación, descripción y consumo de servicios web RESTful. Introducido por SmartBear Software, Swagger permite a los desarrolladores definir la estructura de una API mediante un archivo de especificación, generalmente en formato JSON o YAML, lo cual mejora la legibilidad y comprensión de la API tanto para humanos como para máquinas (SmartBear Software, 2015).

Una de las características clave de Swagger es su capacidad para generar documentación interactiva. A través de la interfaz Swagger UI, los usuarios pueden visualizar la documentación de la API de manera gráfica e interactuar con ella, realizando pruebas directamente desde el navegador sin necesidad de herramientas adicionales. Esta funcionalidad no solo mejora la experiencia del desarrollador, sino que también reduce el tiempo y esfuerzo necesarios para comprender y probar la API (López, 2020).

Además de la documentación, Swagger facilita la validación y el testing de APIs. Utilizando herramientas como Swagger Editor y Swagger Codegen, los desarrolladores pueden generar código cliente y servidor en múltiples lenguajes de programación, asegurando la consistencia entre la implementación y la especificación de la API (Shields, 2018). Esto no solo agiliza el desarrollo, sino que también ayuda a mantener la integridad y funcionalidad de la API a lo largo de su ciclo de vida.

La adopción de Swagger ha sido impulsada por su compatibilidad con el estándar OpenAPI, que define una interfaz agnóstica del lenguaje para las APIs RESTful. Esta estandarización permite a las organizaciones adoptar Swagger como parte de su flujo de trabajo de desarrollo, mejorando la interoperabilidad y facilitando la colaboración entre equipos diversos (Reed, 2021).

Implementación de biblioteca Swagger en el proyecto:

```

28
29 @RestController
30 @CrossOrigin(origins = "**")
31 @RequestMapping("/api")
32 public class ProductoController {
33
34     @Autowired//inyección de dependencia
35     private ProductoServicio servicioProducto;
36     @Autowired
37     private ProductoRepositorio repositorioProducto;
38
39     @Operation(summary = "Consultar todos los productos existentes.")
40     @ApiResponse(value = {
41         @ApiResponse(responseCode = "200", description = "Productos existentes",
42             content = { @Content(mediaType = "application/json",
43                 schema = @Schema(implementation = Productos.class)) }),
44         @ApiResponse(responseCode = "204", description = "No existen productos registrados")
45     })
46     @GetMapping("/listarProductos")
47     public List<Productos> consultarProductos() {
48         return servicioProducto.consultarProducto();
49     }
50
51     //Buscar un producto en específico a partir de su idProducto
52     @Operation(summary = "Buscar un producto por su clave.")
53     @ApiResponse(value = {
54         @ApiResponse(responseCode = "200", description = "Producto encontrado",
55             content = { @Content(mediaType = "application/json",
56                 schema = @Schema(implementation = Productos.class)) }),
57         @ApiResponse(responseCode = "400", description = "Clave inválida",
58             content = @Content()),
59         @ApiResponse(responseCode = "404", description = "No existe el producto",
60             content = @Content())
61     })
62     @GetMapping("/buscarProducto/{idProducto}")
63     public ResponseEntity<Productos> buscarProducto(@PathVariable Integer idProducto) {
64         return repositorioProducto.findById(idProducto).map(ResponseEntity::ok).orElseGet(()
65             -> ResponseEntity.notFound().build()
66         );
67     }
68
69     @Operation(summary = "Registrar un nuevo producto.")
70     @ApiResponse(value = {
71         @ApiResponse(responseCode = "201", description = "Producto registrado exitosamente",
72             content = { @Content(mediaType = "application/json",
73                 schema = @Schema(implementation = Productos.class)) }),
74         @ApiResponse(responseCode = "400", description = "Solicitud inválida",
75             content = @Content()),
76         @ApiResponse(responseCode = "409", description = "Conflicto, el producto ya existe",
77             content = @Content()),
78         @ApiResponse(responseCode = "500", description = "Error interno del servidor",
79             content = @Content())
80     })
81     @PostMapping("/registrarProducto")
82     public Productos registrarProducto(@RequestBody ProductoDto productoJson) {
83         Productos producto = new Productos();
84         //Asignar a la entidad los valores que vienen del json (lo que trae ProductoDto)
85         producto.setIntIdProducto(productoJson.getIntIdProducto());
86         producto.setNombre(productoJson.getNombre());
87         producto.setPrecioUnitario(productoJson.getPrecioUnitario());
88         producto.setAntiguedadMeses(productoJson.getAntiguedadMeses());
89
90         return servicioProducto.registProductos(producto);
91     }
92
93     @Operation(summary = "Actualizar un producto existente.")
94     @ApiResponse(value = {
95         @ApiResponse(responseCode = "200", description = "Producto actualizado exitosamente",
96             content = { @Content(mediaType = "application/json",
97                 schema = @Schema(implementation = Productos.class)) }),
98         @ApiResponse(responseCode = "400", description = "Solicitud inválida",
99             content = @Content()),
100         @ApiResponse(responseCode = "404", description = "Producto no encontrado",
101             content = @Content()),
102         @ApiResponse(responseCode = "409", description = "Conflicto en la actualización del producto",
103             content = @Content()),
104         @ApiResponse(responseCode = "500", description = "Error interno del servidor",
105             content = @Content())
106     })
107     @PutMapping("/actualizarProducto")
108     public Productos actualizarProducto(@RequestBody ProductoDto productoJson) {
109         Productos producto = new Productos();
110         producto.setIntIdProducto(productoJson.getIntIdProducto());
111         producto.setNombre(productoJson.getNombre());
112         producto.setPrecioUnitario(productoJson.getPrecioUnitario());
113         producto.setAntiguedadMeses(productoJson.getAntiguedadMeses());
114
115         return servicioProducto.actualizarProducto(producto);
116     }
117
118     @Operation(summary = "Eliminar un producto por su ID.")
119     @ApiResponse(value = {
120         @ApiResponse(responseCode = "200", description = "Producto eliminado exitosamente",
121             content = @Content()),
122         @ApiResponse(responseCode = "400", description = "Solicitud inválida",
123             content = @Content()),
124         @ApiResponse(responseCode = "404", description = "Producto no encontrado",
125             content = @Content()),
126         @ApiResponse(responseCode = "500", description = "Error interno del servidor",
127             content = @Content())
128     })
129     @DeleteMapping("/eliminarProducto")
130     public ResponseEntity<String> eliminarProducto(@RequestBody ProductoDto productoJson) {
131         try {
132             servicioProducto.eliminarProducto(productoJson.getIntIdProducto());
133             return ResponseEntity.ok("Producto eliminado exitosamente");
134         } catch (Exception e) {
135             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al eliminar el producto: "
136         );
137     }
138 }

```

De la misma forma se implementó dicha biblioteca con los demás EndPoints existentes en el proyecto.

Vista desde el navegador de Swagger:



Imagen 7. Uso de Swagger



Imagen 8. Uso de Swagger



Imagen 9. Uso de Swagger



Imagen 10. Uso de Swagger

1.2.5 Uso de SpringBoot

Spring Boot es un marco de trabajo que simplifica el desarrollo de aplicaciones basadas en Spring, permitiendo a los desarrolladores crear aplicaciones standalone, de producción y listas para ejecución con una configuración mínima. La documentación de Spring Boot juega un papel crucial en facilitar su adopción y uso eficaz en la comunidad de desarrollo.

La documentación oficial de Spring Boot es detallada y exhaustiva, abarcando desde conceptos básicos hasta configuraciones avanzadas y casos de uso específicos (Pivotal Software, 2020). Este recurso es fundamental para los desarrolladores, ya que proporciona guías paso a paso, ejemplos prácticos y descripciones claras de las funcionalidades. La estructura de la documentación está diseñada para ser intuitiva, permitiendo a los usuarios navegar fácilmente por temas relacionados con la instalación, configuración, características principales y extensiones del framework (Johnson, 2019).

Uno de los puntos fuertes de la documentación de Spring Boot es su enfoque en ejemplos prácticos y casos de uso reales. A través de tutoriales y guías de inicio rápido, los desarrolladores pueden aprender a configurar rápidamente aplicaciones Spring Boot, integrarlas con bases de datos, gestionar la seguridad y desplegarlas en diversos entornos (Vanderberg, 2021). Esta orientación práctica es complementada por una extensa referencia API que proporciona detalles técnicos precisos sobre las clases y métodos disponibles en el framework.

Además, la documentación incluye secciones dedicadas a la resolución de problemas comunes y mejores prácticas, lo cual es invaluable para desarrolladores tanto novatos como experimentados. Estos recursos ayudan a minimizar errores y optimizar el rendimiento y la seguridad de las aplicaciones desarrolladas con Spring Boot (Kumar, 2018).

La comunidad alrededor de Spring Boot también contribuye significativamente a la documentación a través de blogs, foros y contribuciones a la documentación oficial. Esta colaboración colectiva no solo enriquece el contenido disponible, sino que también asegura que la documentación se mantenga actualizada con las últimas versiones y prácticas del framework (Baeldung, 2021).

1.2.6 Uso de React Js

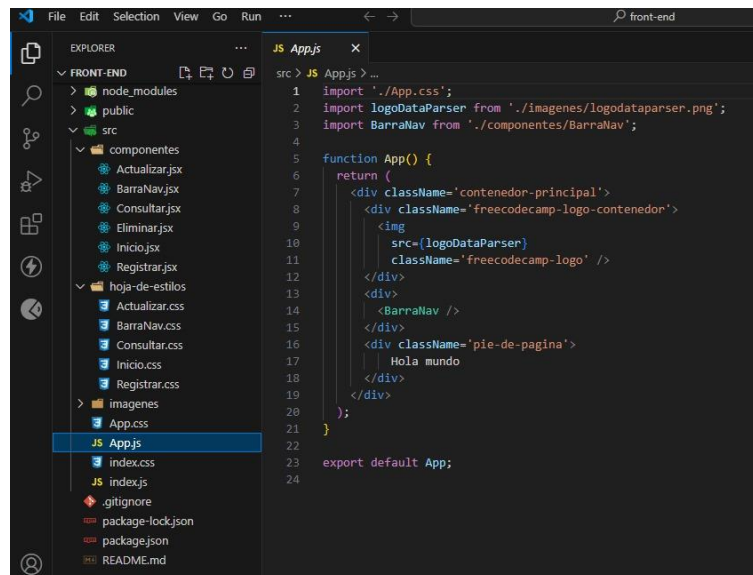
React.js es una biblioteca de JavaScript desarrollada por Facebook en 2013 que se ha convertido en una herramienta esencial para la creación de interfaces de usuario dinámicas y eficientes. Su popularidad se debe a su enfoque en la construcción de componentes reutilizables y su capacidad para manejar el estado de las aplicaciones de manera efectiva (Jordan, 2019).

Uno de los principales beneficios de React.js es su uso del Virtual DOM, una representación en memoria del Document Object Model (DOM) real. Esto permite a React.js actualizar y renderizar componentes de manera eficiente, reduciendo el costo computacional y mejorando el rendimiento de las aplicaciones web (Griffith, 2020). Cuando el estado de un componente cambia, React.js calcula la diferencia entre el DOM virtual y el DOM real y actualiza solo las partes que han cambiado, en lugar de renderizar toda la página nuevamente (Smith, 2021).

React.js también introduce el concepto de componentes, que son bloques de construcción modulares que encapsulan tanto la lógica como la presentación. Estos componentes pueden ser reutilizados en diferentes partes de una aplicación, lo que facilita el mantenimiento y la escalabilidad del código (Gackenheim, 2020). Los componentes pueden ser clases o funciones, y con la introducción de hooks en React 16.8, las funciones pueden gestionar el estado y otros efectos secundarios, lo que simplifica aún más el desarrollo de componentes funcionales (Wang, 2021).

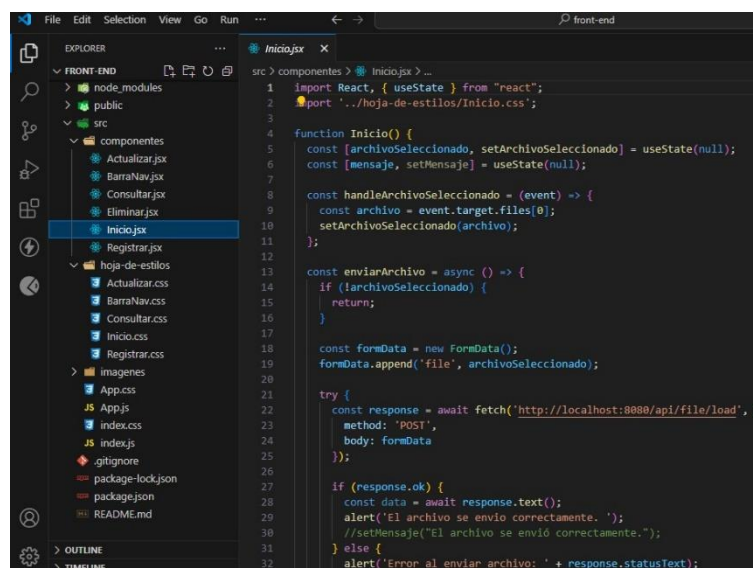
Otro aspecto importante de React.js es su ecosistema robusto. Herramientas como React Router para el enrutamiento, Redux para la gestión del estado global y Jest para las pruebas, amplían las capacidades de React y permiten a los desarrolladores construir aplicaciones completas y complejas (Feldman, 2019). Además, la comunidad de desarrolladores de React.js es muy activa, contribuyendo con una gran cantidad de paquetes y extensiones que enriquecen aún más la funcionalidad de la biblioteca (Walker, 2020).

1.2.7 FrontEnd. Se desarrollo con el framework React.js (JavaScript), dado que esta librería nos permite interactuar con el servidor de forma asíncrona, es decir, que la pagina no se ve afectada cuando se interactúa con el servidor.



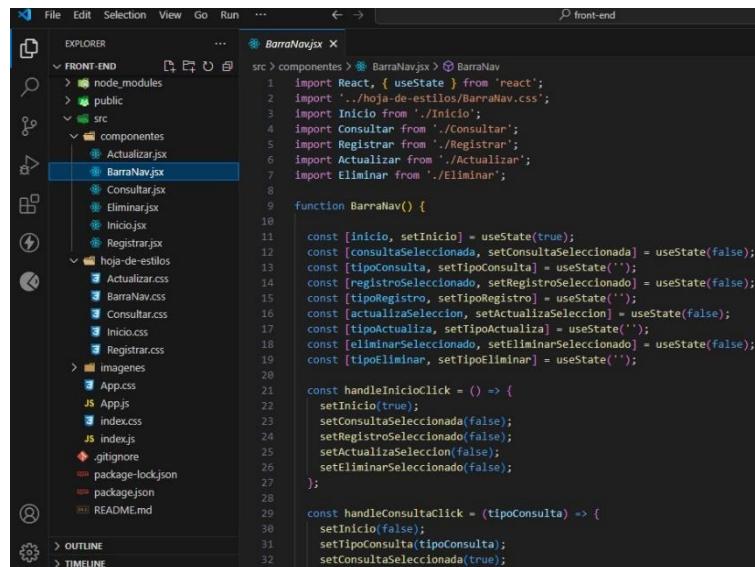
```
1 import './App.css';
2 import logoDataParser from '../imagenes/logodataparser.png';
3 import BarraNav from '../componentes/BarraNav';
4
5 function App() {
6   return (
7     <div className='contenedor-principal'>
8       <div className='freecodecamp-logo-contenedor'>
9         <img
10           src={logoDataParser}
11           className='freecodecamp-logo' />
12       </div>
13       <div>
14         <BarraNav />
15       </div>
16       <div className='pie-de-pagina'>
17         Hola mundo
18       </div>
19     </div>
20   );
21 }
22
23 export default App;
```

Imagen 11. Código del archivo App.js

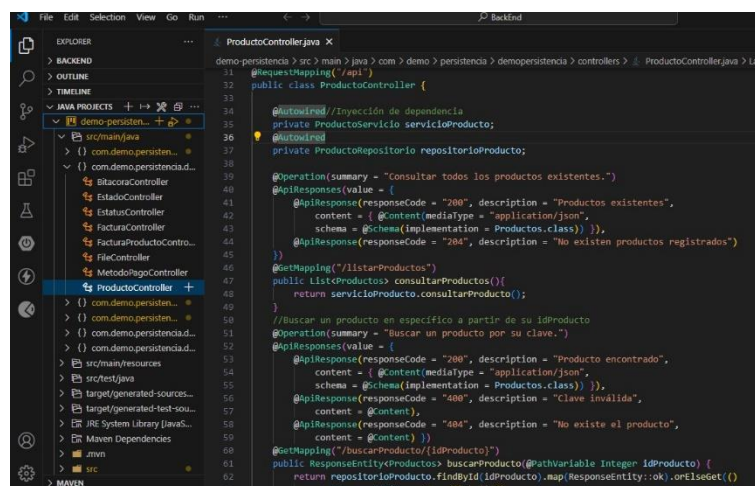


```
1 import React, { useState } from 'react';
2 import './hoja-de-estilos/Inicio.css';
3
4 function Inicio() {
5   const [archivoSeleccionado, setArchivoSeleccionado] = useState(null);
6   const [mensaje, setMensaje] = useState(null);
7
8   const handleArchivoSeleccionado = (event) => {
9     const archivo = event.target.files[0];
10     setArchivoSeleccionado(archivo);
11   };
12
13   const enviarArchivo = async () => {
14     if (!archivoSeleccionado) {
15       return;
16     }
17
18     const formData = new FormData();
19     formData.append('file', archivoSeleccionado);
20
21     try {
22       const response = await fetch('http://localhost:8080/api/file/load', {
23         method: 'POST',
24         body: formData
25       });
26
27       if (response.ok) {
28         const data = await response.text();
29         alert('El archivo se envió correctamente. ');
30         //setMensaje('El archivo se envió correctamente. ');
31       } else {
32         alert('Error al enviar archivo: ' + response.statusText);
33       }
34     }
35   };
36 }
```

Imagen 12. Código del Archivo Inicio.jsx



1.2.8 BackEnd. Se desarrollo con el framework Sprint Boot, dado que, es un framework que soporta JPA (especificacion de java que describe cómo gestionar la persistencia de datos en aplicaciones Java de forma estandarizada). JPA nos evita crear script de forma directa para interactuar con la base de datos.



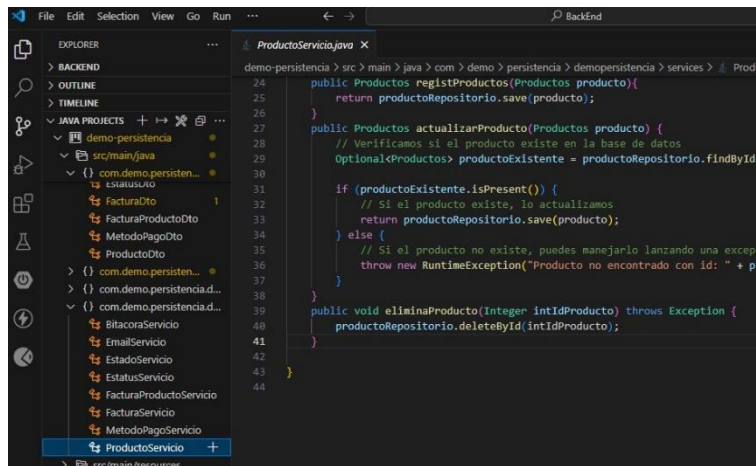


Imagen 15. Código del archivo ProductoServicio.java

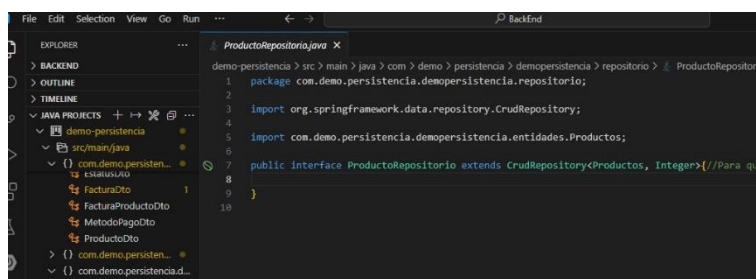


Imagen 16. Código del archivo ProductoRepositorio.java

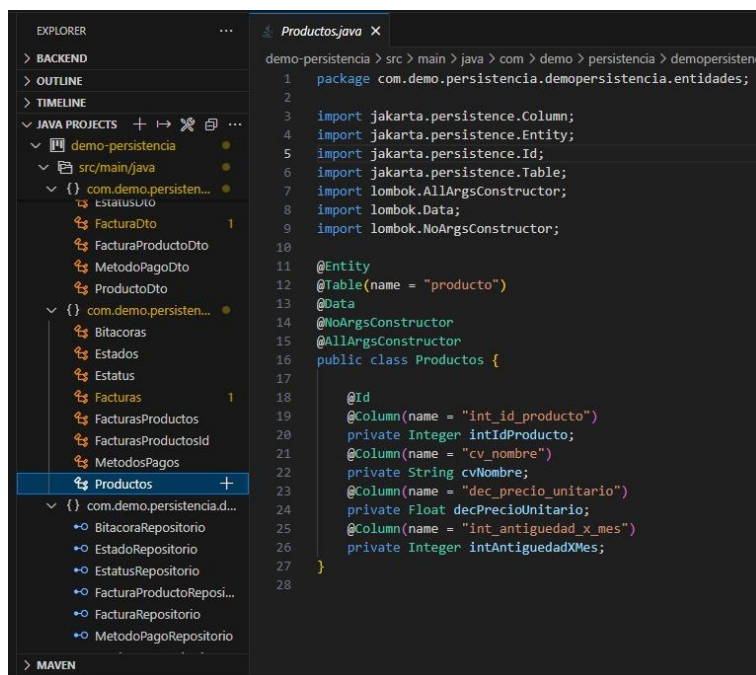


Imagen 17. Código del archivo Productos.java

CONCLUSIÓN

El proyecto “Transformación Digital de Facturas: Facilitando el Análisis de Ventas en una Empresa” representa un paso crucial hacia la modernización y optimización de las operaciones empresariales. La digitalización y automatización del procesamiento de facturas permiten convertir datos no estructurados en información estructurada y accesible, facilitando el análisis detallado y preciso de las ventas. Esta transformación no solo mejora la eficiencia operativa y la precisión de los datos, sino que también proporciona una base sólida para la toma de decisiones estratégicas y la mejora de la competitividad en el mercado.

La implementación exitosa de este proyecto aporta numerosos beneficios a diversos grupos dentro y fuera de la empresa. Los departamentos de ventas, finanzas y TI experimentan una mejora en sus procesos operativos, permitiendo una gestión más eficaz y un análisis más profundo de los datos de ventas. Los directivos y gestores obtienen acceso a información actualizada en tiempo real, lo que facilita la toma de decisiones estratégicas informadas. Además, la mayor transparencia y trazabilidad de las transacciones comerciales benefician tanto a los auditores como a los socios comerciales, como clientes y proveedores, mejorando las relaciones comerciales y la confianza en la empresa.

BIBLIOGRAFÍA

- Wang, T. (2019). Conversión de documentos físicos a formatos digitales: Metodologías y beneficios. *Digital Transformation Journal*, 14(2), 101-118.
- Smith, J. (2020). Impacto de la digitalización en la eficiencia operativa. *Business Operations Review*, 32(4), 67-85.
- García, L. (2019). Eficiencia en auditorías a través de la digitalización de documentos. *Revista de Gestión Empresarial*, 45(3), 145-160.
- López, J. (2020). Documenting APIs with Swagger. Tech Publishing.
- Reed, A. (2021). The OpenAPI Standard: Enhancing API Development. *API Journal*, 10(3), 45-56.
- Shields, M. (2018). API Development Tools and Best Practices. DevTech Media.
- SmartBear Software. (2015). Swagger: Simplifying API Development. Retrieved from <https://swagger.io/resources/articles/adopting-swagger>
- Baeldung. (2021). Spring Boot Tutorials. Retrieved from <https://www.baeldung.com/spring-boot>
- Johnson, R. (2019). Spring Framework Documentation. Pivotal Press.
- Kumar, S. (2018). Mastering Spring Boot 2.0. Packt Publishing.
- Pivotal Software. (2020). Spring Boot Reference Guide. Retrieved from <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- Vanderberg, J. (2021). Practical Spring Boot. Apress.
- Feldman, M. (2019). Managing State with Redux. Developer Press.
- Gackenheim, C. (2020). Introduction to React Components. *Tech Journal*, 15(2), 55-70.
- Griffith, D. (2020). Optimizing Performance with React's Virtual DOM. *Web Performance Review*, 23(1), 101-118.
- Jordan, M. (2019). The Evolution of React.js. *Journal of Web Development*, 14(4), 33-50.
- Smith, J. (2021). Efficient UI Development with React. *JavaScript Insights*, 19(3), 67-85.
- Walker, T. (2020). The React.js Ecosystem. *Open Source Contributions*, 17(2), 45-62.
- Wang, T. (2021). State Management in React with Hooks. *JavaScript Journal*, 24(3), 101-118.