



MANUAL DEL PROGRAMADOR

Sistema web de gestión de productos y ventas de la empresa Arimana Bazar

Versión: 1.0.0

Fecha: 07 de julio de 2025

Autores:

- Barahona Figueroa Paul Alexis
- Chipana Huamani Wilian Gabriel
- Huari Bobadilla Daniel Javier
- Muñoz Castro Olenka Del Rosario
- Torres Sacha Isaura Isabel

Índice

1. Introducción.....	3
2. Requerimientos.....	3
2.1 Requerimientos de hardware.....	3
2.2 Requerimientos de software.....	3
2.3 Requerimientos humanos.....	4
3. Instalación y configuración del entorno.....	4
3.1 Clonación del repositorio en GitHub.....	4
3.2 Configuración de Visual Studio Code.....	5
3.3 Instalación de dependencias.....	5
4. Diagramas de entidad relación.....	6
4.1 Modelo conceptual.....	7
4.2 Modelo lógico.....	8
4.3 Modelo físico.....	9
5. Estructura de directorios.....	10
6. Convenciones del proyecto.....	11
6.1 Variables.....	11
6.2 Funciones.....	12
6.3 Constantes.....	12
6.4 Comentarios en español técnico.....	12
7. Implementación de archivos estáticos.....	13
7.1 Directorio data/.....	13
7.2 Directorio lib.....	20
8. Implementación y configuración de rutas.....	21
9. Seguridad y autenticación.....	22
9.1 Gestión de usuarios y roles.....	22
9.2 Implementación de medidas de seguridad.....	22
10. Control de versiones.....	25

1. Introducción

El principal objetivo de este documento es proporcionar al programador una visión completa y detallada de todo lo desarrollado en el presente proyecto. De esta manera, cualquier desarrollador que retome el trabajo podrá comprender su estructura, lógica y funcionamiento, permitiéndole continuar con el desarrollo, mantenimiento o mejora del sistema desde el punto exacto en que fue dejado.

2. Requerimientos

2.1 Requerimientos de hardware

Para asegurar que el sistema funcione correctamente y ofrezca un rendimiento óptimo en las tareas asignadas, es esencial contar con un equipo que cumpla con ciertos requisitos técnicos. A continuación, se detallan las características del hardware utilizado, las cuales son clave para ejecutar las aplicaciones de manera eficiente y garantizar una experiencia fluida para el usuario.

- Procesador AMD Ryzen 7 5700G with Radeon Graphics
- RAM de 16,0 GB
- Disco duro de 250 GB SSD
- Monitor 27" GAME PRO GAMING GPG270 HD (1920 x 1080)

2.2 Requerimientos de software

El entorno de desarrollo y ejecución del sistema necesita una configuración de software específica para asegurar su estabilidad, compatibilidad y rendimiento. A continuación, se presentan las herramientas y versiones utilizadas, elegidas por su robustez y su capacidad para satisfacer las necesidades del proyecto. Estas aplicaciones facilitan la programación, las pruebas y el despliegue, permitiendo realizar estas tareas de manera eficiente en un entorno moderno y funcional.

- Sistema operativo Windows 11 Home Single Language (v24H2) de 64 bits (x64)

- Node JS (v20.15.0) o superiores
- NPM (v10.7.0) o superiores
- Visual Studio code
- SQL Server Management Studio 20
- Power BI Desktop

2.3 Requerimientos humanos

Para llevar a cabo el desarrollo, implementación y mantenimiento de este proyecto, se requiere un conjunto de competencias técnicas específicas. A continuación, se detallan estos conocimientos esenciales:

- Comprensión del framework Astro
- Manejo de JSX, TypeScript y JavaScript moderno
- Gestión y manejo de dependencias en JavaScript
- Control de versiones en Git y Github

3. Instalación y configuración del entorno

3.1 Clonación del repositorio en GitHub

Para iniciar el desarrollo local del proyecto, es necesario clonar el repositorio remoto alojado en GitHub. Este proceso descarga una copia del proyecto en el equipo local, manteniendo el control de versiones. A continuación, se presentan los pasos de este procedimiento:

1. Abrir la terminal o consola de comandos
2. Ubicar la carpeta donde deseas clonar el proyecto. *Por ejemplo:*

A screenshot of a terminal window with a dark background. At the top left, there are three colored window control buttons (red, yellow, green). The terminal shows a single command prompt line with the text "1 cd Documentos/proyectos/" where "cd" is highlighted in green.

```
1 cd Documentos/proyectos/
```

3. Ejecutar el comando git clone seguido de la URL del repositorio

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a single line of text:

```
1 git clone https://github.com/PaulBaFi/arimanabazarica.git
```

```
1 git clone https://github.com/PaulBaFi/arimanabazarica.git
```

4. Acceder a la carpeta del proyecto clonado

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a single line of text:

```
1 cd arimanabazarica
```

```
1 cd arimanabazarica
```

3.2 Configuración de Visual Studio Code

A continuación, se detallan las extensiones que deben instalarse en Visual Studio Code para asegurar una correcta escritura, formato y análisis del código fuente:

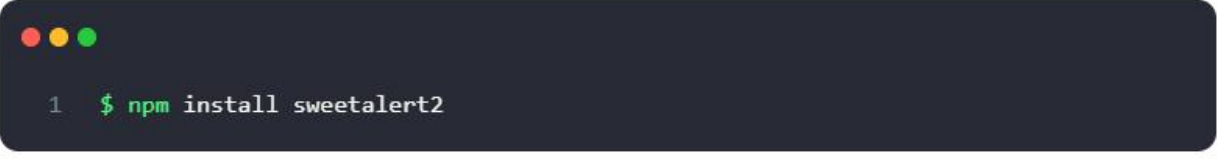
- Astro: Astro-build.astro-vscode (v2.15.4)
- ESLint: Dbaeumer.vscode-eslint (v3.0.10)
- Error Lens: usernamehw.errorlens (v3.26.0)
- Prettier — Code Formatter: Esbenp.prettier-vscode (v11.0.0)

3.3 Instalación de dependencias

El proyecto requiere una serie de dependencias que permiten añadir funcionalidades específicas y mejorar el flujo de desarrollo, interacción y despliegue. A continuación, se describen las dependencias utilizadas y su propósito dentro del sistema:

- Sweetalert2


Proporciona los recursos para mostrar ventanas emergentes adaptables y personalizables, utilizadas para mostrar retroalimentación de las funciones de accionamiento del sistema.



```
1 $ npm install sweetalert2
```

- ReactJS


Esta integración de Astro habilita el renderizado del lado del servidor y la hidratación del lado del cliente para los componentes React del proyecto.



```
1 $ npx astro add react
```

- Adaptador de Netlify

Netlify es una plataforma de despliegue que permite alojar un sitio web mediante una conexión directa al repositorio de Github. El adaptador de Netlify mejora el proceso de construcción de Astro para el despliegue del proyecto a través de Netlify.



```
1 $ npx astro add netlify
```

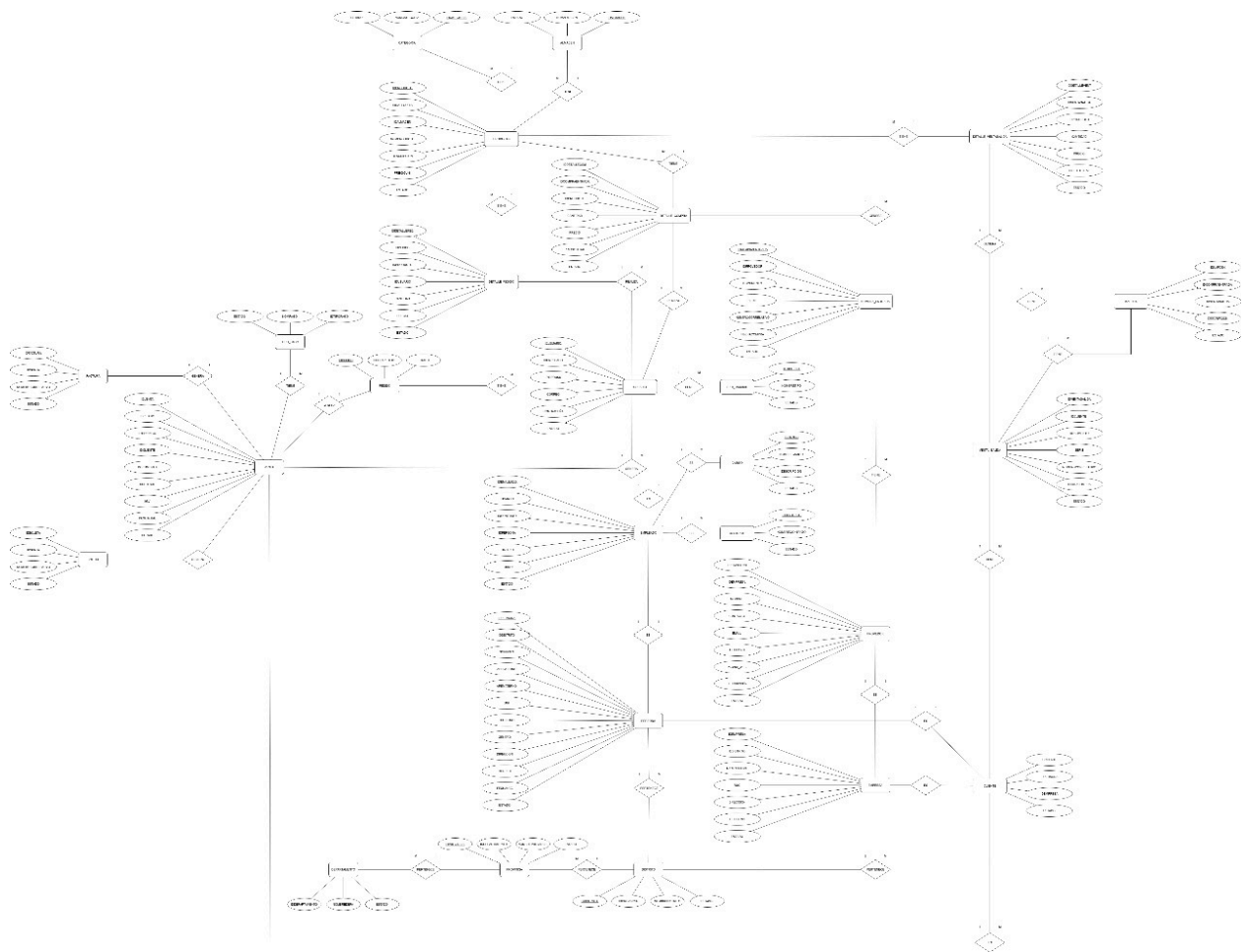
4. Diagramas de entidad relación

Para el diseño de la base de datos se tomó en consideración las primeras tres normas formales de normalización de bases de datos con el fin de evitar duplicidad de datos y prever problemas de inconsistencia. A continuación, se muestran los diferentes modelos de las tablas que componen el sistema.

4.1 Modelo conceptual

Figura 1

Imagen del modelo conceptual del sistema web de gestión de productos y ventas

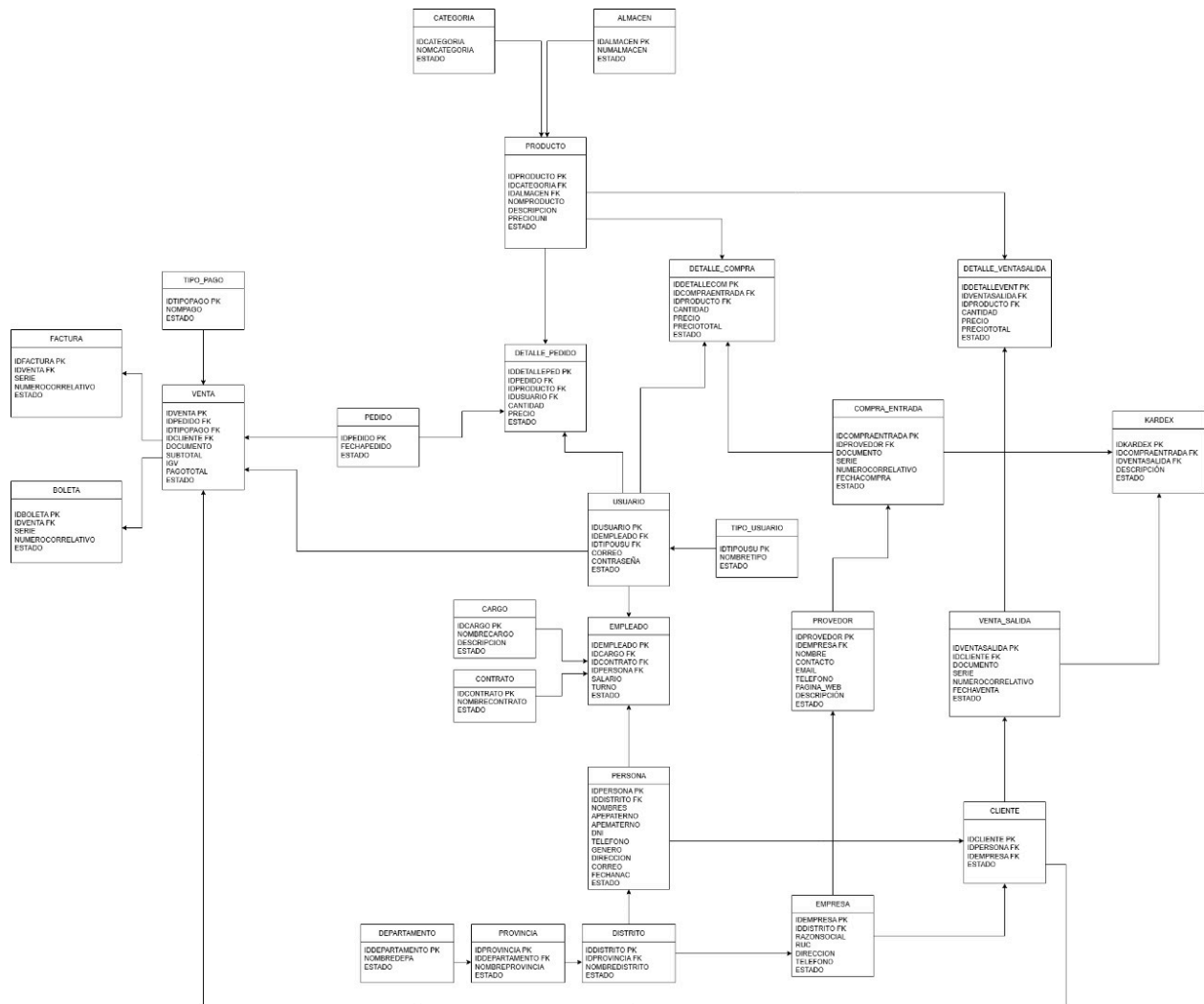


Nota. En la imagen se muestran las entidades del sistema y sus respectivos atributos, señalando las relaciones o cardinalidad entre estas.

4.2 Modelo lógico

Figura 2

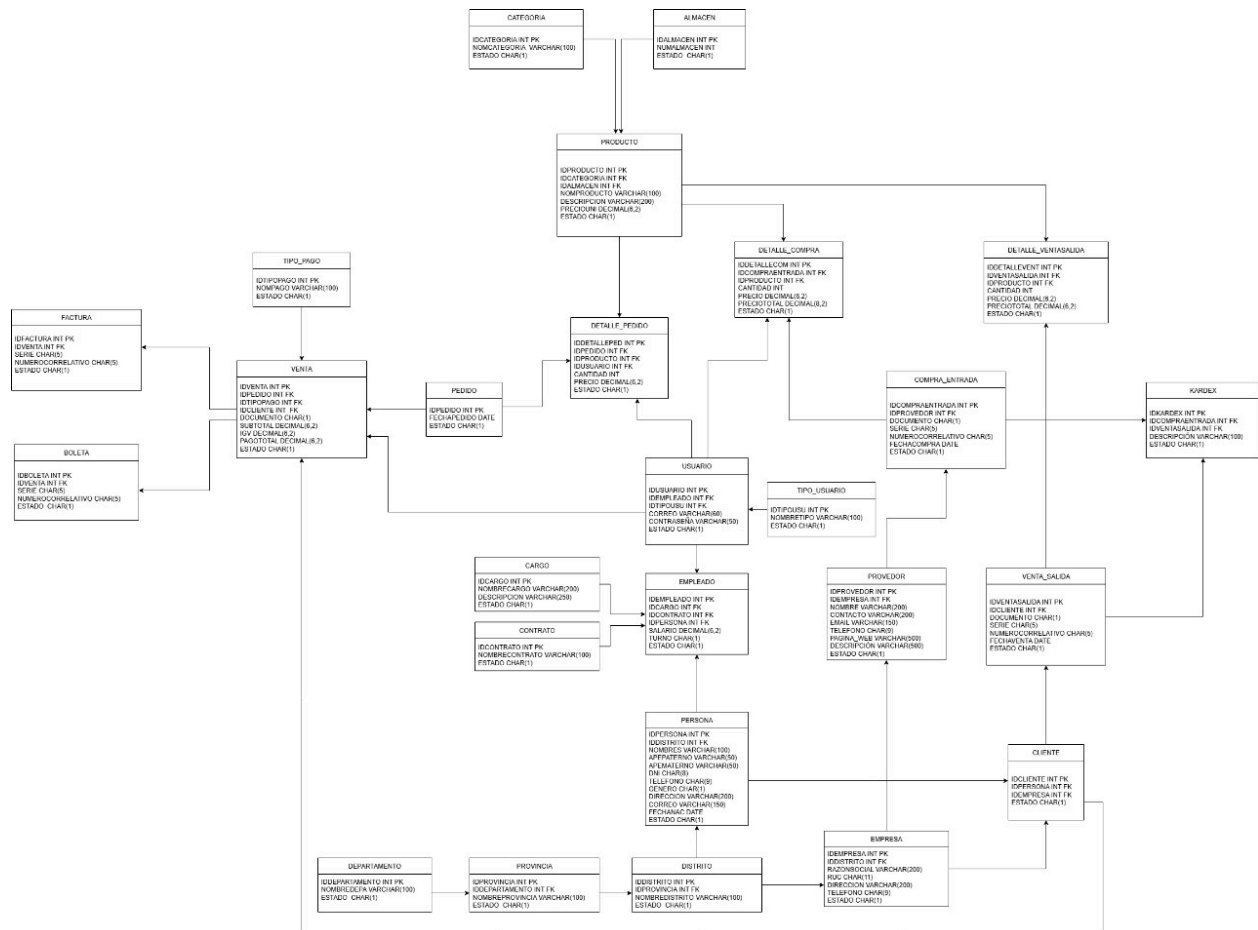
Imagen del modelo lógico del sistema web de gestión de productos y ventas



Nota. Se muestran las entidades, sus respectivos atributos y relaciones sin señalar la cardinalidad, se pretende mantener un modelo abstracto y escalable. Asimismo, se tienen en consideración atributos de auditoría como estado, por lo que se solicita obviar la duplicidad de este atributo en todas las tablas del diseño de la base de datos.

Figura 3

Imagen del modelo físico del sistema web de gestión de productos y ventas



Nota. Se utiliza como base el modelo lógico y se añade la especificación de los metadatos o tipos de datos para cada uno de los atributos de las tablas de la base de datos.

5. Estructura de directorios

Directorio public/

Contiene archivos estáticos accesibles directamente desde el navegador, esta misma contiene los directorios `fonts/` e `images/` además del `favicon.ico` o ícono de la página que aparece en la pestaña del navegador.

```
1  arimanabazarica/
2    └─ public/
3        └─ fonts/
4        └─ images/
5        └─ favicon.svg
```

Directorio src/

Directorio raíz para todo el código fuente del proyecto.

```
1  arimanabazarica/
2    └─ src/
3        └─ components/
4        └─ css/
5        └─ data/
6        └─ icons/
7        └─ layouts/
8        └─ lib/
9        └─ pages/
```

Directorio components/

Componentes reutilizables escritos en JSX y nativos de astro para representar partes de la interfaz de diferentes partes del sistema.

```
1  arimanabazarica/
2    └─ src/
3        └─ components/
4            └─ BaseHead.astro
5            └─ Categorias.astro
6            └─ Contacto.astro
7            └─ HeroSection.astro
8            └─ LandingFooter.astro
9            └─ LandingSidebar.astro
10           └─ LoginAccess.jsx
11           └─ LogoutAccess.jsx
12           └─ Navbar.astro
13           └─ Nosotros.astro
14           └─ PanelHeader.astro
15           └─ PanelMain.astro
16           └─ PanelTable.astro
17           └─ Productos.astro
18           └─ Sidebar.astro
```

Directorio css/

Archivos de estilos personalizados para cada área e interfaz del sistema web, considerando un archivo denominado global.css donde se definen estilos aplicables para todo el entorno.

```
1  arimanabazarica/
2    └─ src/
3        └─ css/
4            └─ global.css
5            └─ landing.css
6            └─ login.css
7            └─ panel.css
```

Directorio data/

Contiene una estructura de archivos estáticos que son parte clave del sistema, estos archivos almacenan información que simulan la data de una base de datos real

```
1  arimanabazarica/  
2  └─ src/  
3     └─ data/  
4         └─ Almacenes.ts  
5         └─ Cargos.ts  
6         └─ Categorias.ts  
7         └─ Cliente.ts  
8         └─ Contratos.ts  
9         └─ Empleados.ts  
10        └─ Empresas.ts  
11        └─ NavbarSections.ts  
12        └─ Perfil.ts  
13        └─ Personas.ts  
14        └─ Productos.ts  
15        └─ Proveedores.ts  
16        └─ TipoUsuario.ts  
17        └─ Usuarios.ts
```

Directorio icons/

Esta carpeta almacena componentes Astro que representan íconos personalizados utilizados en la interfaz del sistema. Su organización modular favorece la reutilización de código.

```
1  arimanabazarica/
2  └─ src/
3      └─ Icons/
4          └─ Avatar.astro
5          └─ Carpeta.astro
6          └─ Configuracion.astro
7          └─ Diamond.astro
8          └─ Empleados.astro
9          └─ Inventarios.astro
10         └─ Logout.astro
11         └─ Panel.astro
12         └─ Pedidos.astro
13         └─ Principal.astro
14         └─ Relaciones.astro
15         └─ Reporte.astro
16         └─ Target.astro
17         └─ Ventas.astro
```

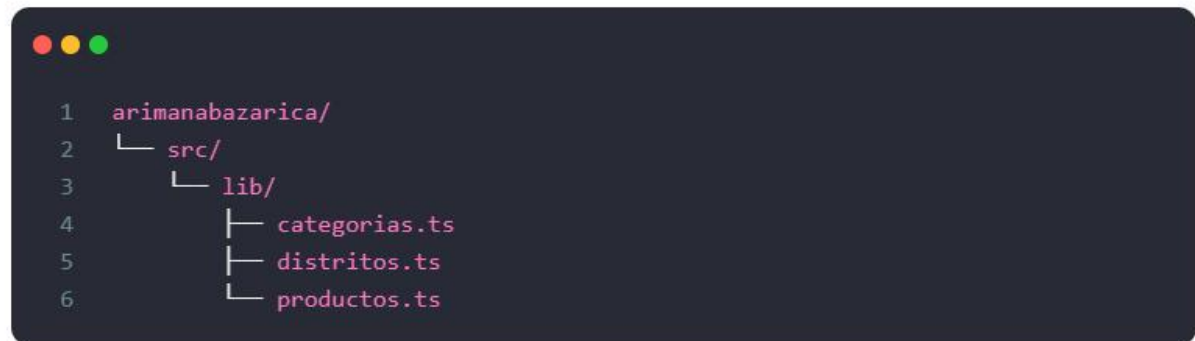
Directorio layouts/

Define estructuras base reutilizables para distintas vistas del sistema, como el panel, login y página principal. Cada diseño integra componentes compartidos y estilos específicos, promoviendo consistencia visual y mantenibilidad

```
1  arimanabazarica/
2  └─ src/
3      └─ layouts/
4          └─ LandingLayout.astro
5          └─ LoginLayout.astro
6          └─ Panellayout.astro
7          └─ SimplePanellayout.astro
```

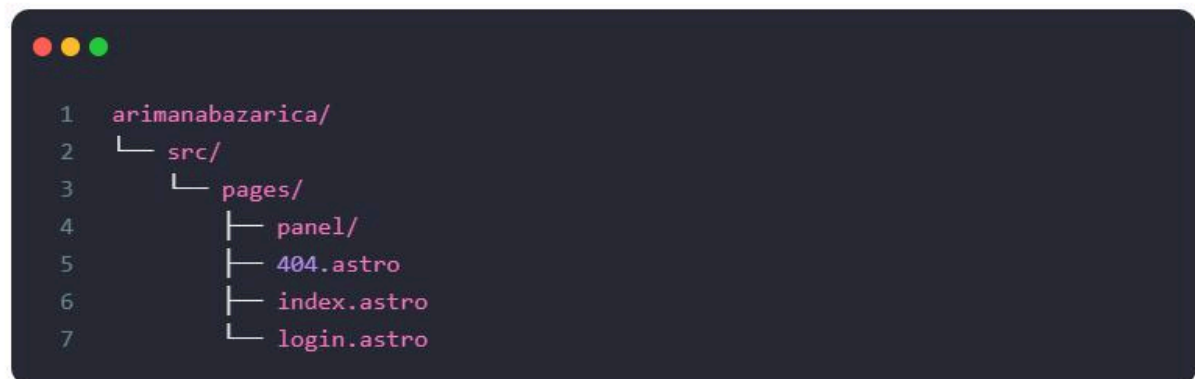
Directorio lib/

Centraliza módulos TypeScript que contienen datos estáticos y estructurados utilizados por el sistema, como categorías, productos y distritos



Directorio pages/

El directorio pages renderiza páginas de forma estática y define las rutas principales del sitio. Incluye archivos como index.astro, que actúa como punto de entrada, y 404.astro, que personaliza la página de error para rutas no encontradas.



Directorio panel/

La carpeta panel contiene todas las vistas correspondientes al dashboard administrativo de la aplicación. Incluye archivos .astro que representan secciones clave como clientes, pedidos,

productos, ventas y otros módulos de gestión, centralizando así la lógica visual del panel en un solo lugar.

```
1  arimanabazarica/
2  └─ src/
3     └─ pages/
4        └─ panel/
5           │─ almacenes.astro
6           │─ cargos.astro
7           │─ categorias.astro
8           │─ clientes.astro
9           │─ contratos.astro
10          │─ empleados.astro
11          │─ empresas.astro
12          │─ index.astro
13          │─ kardex.astro
14          │─ panel.astro
15          │─ pedidos.astro
16          │─ perfil.astro
17          │─ personas.astro
18          │─ productos.astro
19          │─ proveedores.astro
20          │─ reporte-empleados.astro
21          │─ reporte-pedidos.astro
22          │─ reporte-ventas.astro
23          │─ tipos-usuarios.astro
24          │─ usuarios.astro
25          └─ ventas.astro
```

Archivo index.astro

Entrada principal de la aplicación, sería el landing o página piloto del sistema, de acceso para todo tipo de usuarios. En este se definen las secciones principales de la aplicación en componentes reutilizables.

```
1 <LandingLayout>
2   <LandingSidebar />
3   <HeroSection />
4   <Categorias />
5   <Productos />
6   <Nosotros />
7   <Contacto />
8   <LandingFooter />
9 </LandingLayout>
```

Archivo login.astro

Ventana de inicio de sesión para acceso de los usuarios autorizados, en esta se establece el login que es validado por medio del componente `LoginAccess` cargado al final del contenido HTML, en este archivo se restringe el acceso al panel del sistema web, explicado más adelante.

La directiva `client:load` permite cargar solo esa parte del código como contenido dinámico del lado del cliente (navegador), puesto que Astro por defecto solo carga contenido estático.


```

1  <LoginLayout>
2    <div class="login-container">
3      
8
9      <h1>Iniciar sesión</h1>
10
11     <form id="loginForm" class="form" method="post" novalidate>
12       <div class="form-grupo">
13         <input type="email" name="correo" />
14         <label for="correo">Correo</label>
15       </div>
16       <div class="form-grupo">
17         <input type="password" name="clave" />
18         <label for="clave">Contraseña</label>
19       </div>
20       <button id="iniciarSesion" type="button">Ingresar</button>
21     </form>
22
23     <a class="volver-link" href="/">
24       <i class="fa-solid fa-arrow-left"></i>
25     </a>
26   </div>
27
28   <LoginAccess client:load />
29 </LoginLayout>

```

Archivo 404.astro

Ventana de error 404 para páginas de rutas no encontradas.

```

1  <html lang="en">
2    <head>
3      <BaseHead title="Página no encontrada | Arimana Bazar" />
4    </head>
5    <body>
6      <LandingLayout>
7        <div class="e404-section container">
8          
13        <h1 class="e404-title">Página no encontrada</h1>
14        <p class="e404-text">
15          Opss... parece que esta página no existe, por favor intente
16          regresar a la página principal.
17        </p>
18        <a class="e404-link" href="/">
19          <i class="fa-solid fa-home"></i>
20          Regresar
21        </a>
22        </div>
23        <LandingFooter />
24      </LandingLayout>
25    </body>
26  </html>

```

Archivo astro.config.mjs

Configuración principal del proyecto Astro, aquí se muestran las integraciones, como reactJS cuando se instala directamente desde consola, adaptadores para el despliegue, como Netlify u otros plugins.

```

1  // @ts-check
2  import { defineConfig } from "astro/config";
3
4  import react from "@astrojs/react";
5
6  import netlify from "@astrojs/netlify";
7
8  // https://astro.build/config
9  export default defineConfig({
10   output: "server",
11   integrations: [react()],
12   adapter: netlify(),
13 });

```

Archivo package.json

Define las dependencias, scripts de ejecución (dev, build, etc.) y metadatos del proyecto.

```

1  {
2    "name": "arimanabazarica",
3    "type": "module",
4    "version": "0.0.1",
5    "scripts": {
6      "dev": "astro dev",
7      "build": "astro build",
8      "preview": "astro preview",
9      "astro": "astro"
10   },
11   "dependencies": {
12     "@astrojs/netlify": "^6.4.1",
13     "@astrojs/react": "^4.3.0",
14     "astro": "^5.11.0",
15     "react": "^19.1.0",
16     "react-dom": "^19.1.0",
17     "sweetalert2": "^11.22.2"
18   }
19 }

```

Archivo tsconfig.json

Configuración de TypeScript y definición de rutas base, en este caso se define `@/*` como raíz del proyecto en la carpeta `src` donde se encuentra todo el contenido ejecutable. Además, se establece la configuración de react JSX y react.

```
1  {
2    "extends": "astro/tsconfigs/strict",
3    "include": [".astro/types.d.ts", "**/*"],
4    "exclude": ["dist"],
5    "compilerOptions": {
6      "baseUrl": ".",
7      "paths": {
8        "@/*": ["src/*"]
9      },
10     "jsx": "react-jsx",
11     "jsxImportSource": "react"
12   }
13 }
```

Archivo README.md

Figura 4

Imagen de la estructura de carpetas del sistema web de gestión de productos y ventas

```
1  arimanabazarica/
2  |  public/
3  |  |  fonts/
4  |  |  |  images/
5  |  |  |  |  favicon.svg
6  |  |
7  |  |  src/
8  |  |  |  components/
9  |  |  |  |  LoginAccess.jsx
10 |  |  |  |  LogoutAccess.jsx
11 |  |  |
12 |  |  |  css/
13 |  |  |  data/
14 |  |  |  icons/
15 |  |  |  layouts/
16 |  |  |  lib/
17 |  |  |  |  pages/
18 |  |  |  |  |  panel/
19 |  |  |  |  |  |  404.astro
20 |  |  |  |  |  |  index.astro
21 |  |  |  |  |  |  login.astro
22 |  |
23 |  |  astro.config.mjs
24 |  |  package.json
25 |  |  tsconfig.json
26 |  |  README.md
```

6. Convenciones del proyecto

6.1 Variables

Se utilizan nombres en camelCase, iniciando con minúscula y usando mayúsculas para separar palabras. *Ejemplo:*

```
1  const logoutBtn = document.getElementById("cerrarSesion");
2  const loginBtn = document.getElementById("iniciarSesion");
```

6.2 Funciones

Las funciones deben definirse usando camelCase(), reflejando claramente su propósito o acción. *Ejemplo:*

```
1 export default function LoginAccess() {  
2   return null;  
3 }
```

6.3 Constantes

Las constantes se nombran usando mayúsculas con guiones bajos, ya que no cambian su valor durante la ejecución. *Ejemplo:*

```
1 export const TIPO_USUARIO = [  
2   { id: 1, nombre: "Administrador", cantidad: 2 },  
3   { id: 2, nombre: "Usuario con acceso", cantidad: 1 },  
4   { id: 3, nombre: "Usuario sin acceso", cantidad: 4 },  
5 ];
```

6.4 Comentarios en español técnico

El texto no ejecutable o comentarios definido en el código se establece en el lenguaje español (ES) puesto que la configuración del sistema en HTML, desarrolladores y administrativos de la empresa tienen esta lengua natural. De tal manera, se pretende documentar con claridad las partes importantes del código para un mejor mantenimiento y colaboración.


7. Implementación de archivos estáticos

El directorio `data/` agrupa archivos que contienen datos simulados, diseñados para reflejar el contenido de una base de datos real. Estos se utilizan principalmente para el desarrollo, pruebas funcionales e integración de interfaces. Por otro lado, el directorio `lib/` alberga información estática y permanente, como distritos. Estos datos actúan como catálogos de referencia dentro del sistema. A diferencia de los datos en `data/`, los valores en `lib/` no cambian con frecuencia y se consideran constantes en la lógica de la aplicación.

7.1 Directorio `data/`

`Almacenes.ts`

Contiene una lista de almacenes con su ID, nombre y cantidad de productos.



```
1 export const ALMACENES = [
2   {
3     idAlmacen: 1,
4     numAlmacen: 1,
5     cantidad: "221 productos",
6   },
7   /* ... */
8 ];
```

`Cargos.ts`

Define los cargos laborales disponibles en la empresa, con información sobre su nombre, descripción y estado (activo/inactivo).

```

1  export const CARGOS = [
2    {
3      idCargo: 1,
4      nombreCargo: "Gerente",
5      descripcion:
6        "Responsable de la gestión general y toma de decisiones estratégicas",
7      cantidad: "1 empleado",
8    },
9    /* ... */
10 ];

```

Categorias.ts

Clasificación de productos en diferentes categorías para facilitar la organización y búsqueda dentro del inventario.

```

1  export const CATEGORIAS = [
2    {
3      cod: 1,
4      nombre: "Artículos de cocina",
5      cantidad: "41 productos",
6    },
7    /* ... */
8  ];

```

Cliente.ts

Representa a los clientes del sistema, que pueden ser personas físicas o empresas. Incluye referencias a las tablas correspondientes.


```
1  export const CLIENTES = [  
2    {  
3      cod: 1,  
4      identificador: "56789012",  
5      cliente: "Patricia Garcia Diaz",  
6      direccion: "Av. Libertad 678",  
7      telefono: "956123478",  
8      distrito: "Marcara",  
9    },  
10   /* ... */  
11 ];
```

Contrato.ts

Contiene los diferentes tipos de contrato laborales que pueden tener los empleados.

```
1  export const CONTRATOS = [  
2    {  
3      idContrato: 1,  
4      nombre: "Tiempo completo",  
5      cantidad: "4 emplados",  
6    },  
7    /* ... */  
8  ];
```

Empleado.ts

Relaciona personas con sus respectivos cargos y contratos. Incluye información laboral como salario, turno y datos de vinculación.

```

1  export const EMPLEADOS = [
2    {
3      idEmpleado: 1,
4      nombreCompleto: "Juan Perez Gomez",
5      nombreCargo: "Gerente",
6      nombreContrato: "Tiempo completo",
7      salario: "s/. 3,500.00",
8      turnoDescripcion: "Mañana",
9    },
10   /* ... */
11 ];

```

Empresas.ts

Información de empresas registradas (clientes o proveedores), incluyendo RUC, razón social, dirección y teléfono.

```

1  export const EMPRESAS = [
2    {
3      idEmpresa: 1,
4      nombreDistrito: "Ica",
5      razonSocial: "Distribuidora Global Cocina Sac",
6      ruc: "20100000001",
7      direccion: "Av. Los Ficus 123",
8      telefono: "987654321",
9    },
10   /* ... */
11 ];

```

NavbarSections.ts

Configuración de la barra de navegación principal del sistema. Organiza las funcionalidades en 8 módulos principales.

Perfil.ts

Lista de usuarios del sistema para pruebas, con información completa de empleados (datos personales, laborales y credenciales de acceso).

```
1  export const PERFIL = [  
2    {  
3      cod: 1,  
4      dni: "12145678",  
5      nombres: "Juan Perez Gomez",  
6      telefono: "912005678",  
7      direccion: "Av. Las Palmas 456",  
8      genero: "Masculino",  
9      salario: 3500.0,  
10     turno: "Mañana",  
11     nombreCargo: "Gerente",  
12     nombreContrato: "Tiempo Completo",  
13     nombreTipo: "Administrador",  
14     correo: "admin@gmail.com",  
15     contraseña: "123",  
16   },  
17   /* ... */  
18 ];
```

Personas.ts

Tabla principal de personas físicas. Contiene datos personales como DNI, nombres, teléfono, dirección y ubicación geográfica.

```
1  export const PERSONAS = [  
2    {  
3      idPersona: 1,  
4      distrito: "Pisquía",  
5      nombreCompleto: "Juan Perez Gomez",  
6      dni: "12145678",  
7      telefono: "912005678",  
8      genero: "Masculino",  
9      direccion: "Av. Las Palmas 456",  
10     correo: "juan.perez@gmail.com",  
11     fechaNac: "1990-06-15",  
12   },  
13   /* ... */  
14 ];
```

Producto.ts

Catálogo general de productos con información detallada: nombre, descripción, precio unitario y categoría asignada.

```
1  export const PRODUCTOS = [  
2    {  
3      idProducto: 1,  
4      nomCategoria: "Artículos de cocina",  
5      numAlmacen: "Almacén 1",  
6      nomProducto: "Juego de vasos de cristal x6",  
7      descripcion:  
8        "Vasos de cristal templado para uso diario y ocasiones especiales.",  
9      precioUni: "s/. 35.99",  
10   },  
11   /* ... */  
12 ];
```

Proveedores.ts

Información específica de proveedores, incluyendo datos de contacto, correo electrónico y descripción de servicios ofrecidos.

```
1  export const PROVEEDORES = [  
2    {  
3      cod: 1,  
4      razonSocial: "Distribuidora global cocina sac",  
5      ruc: "20100000001",  
6      direccion: "Av. los ficus 123",  
7      telefono: "987654321",  
8      contacto: "Carlos ramirez",  
9      correo: "ventas@cocinasa.com",  
10     descripcion:  
11       "Proveedor líder en artículos de cocina de alta calidad.",  
12     },  
13     /* ... */  
14   ];
```

TipoUsuario.ts

Define los distintos tipos de usuarios del sistema (por ejemplo, administrador, vendedor), con su respectivo estado (activo/inactivo).

```
1  export const TIPO_USUARIO = [  
2    { id: 1, nombre: "Administrador", cantidad: 2 },  
3    /* ... */  
4  ];
```

Usuario.ts

Contiene las cuentas de acceso al sistema, vinculadas a empleados, incluyendo credenciales y tipo de usuario asignado.

```
1  export const USUARIOS = [  
2    {  
3      cod: 1,  
4      empleado: "Juan Perez Gomez",  
5      correo: "gerente.principal@empresa.com",  
6      tipoUsuario: "ADMINISTRADOR",  
7    },  
8    /* ... */  
9  ];
```

7.2 Directorio lib/

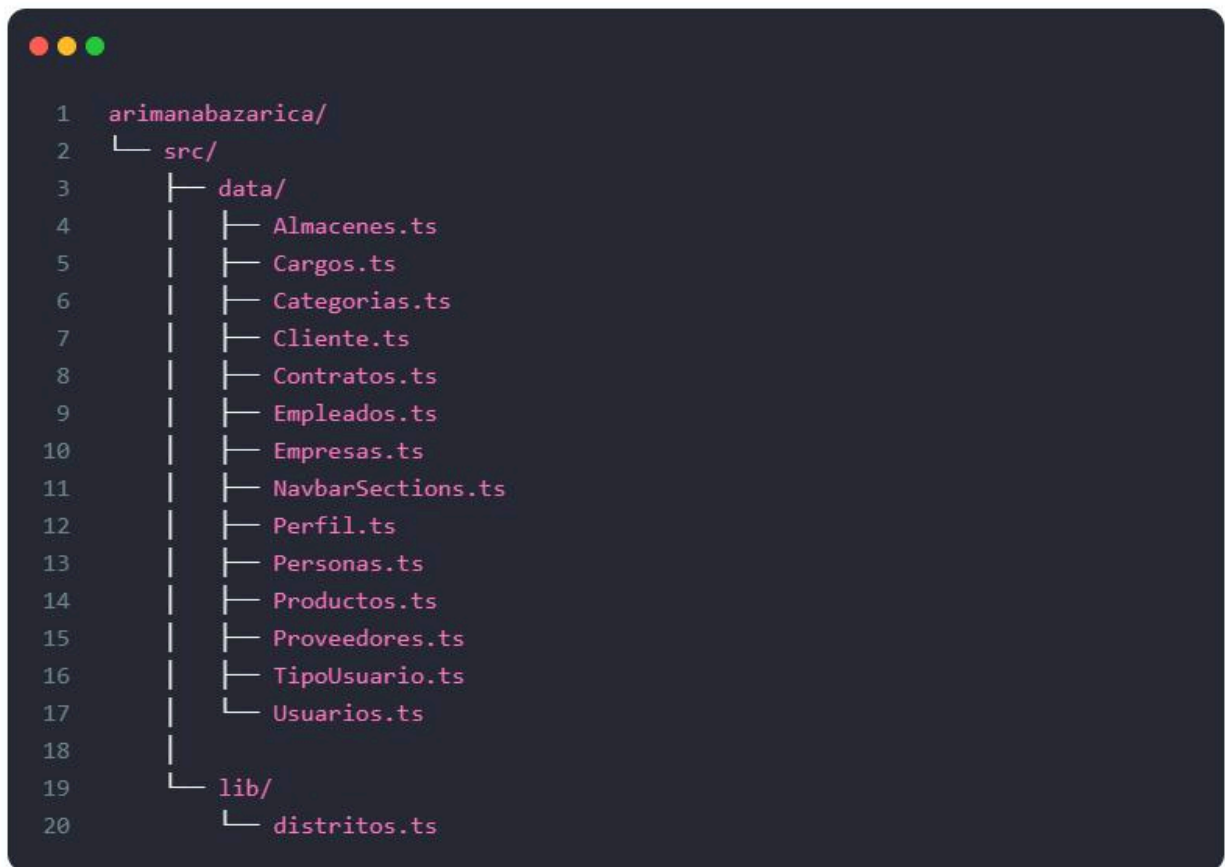
Distrito.ts

Contiene una lista estática de 14 distritos pertenecientes a la provincia de Ica, cada uno con su ID y nombre. Esta información se utiliza para referencias geográficas dentro del sistema.

```
1  export const DISTRITOS = [  
2    { id: "1", nombre: "Ica" },  
3    /* ... */  
4  ];
```

Figura 5

Imagen de las carpetas y archivos estáticos del sistema web de gestión de productos y ventas

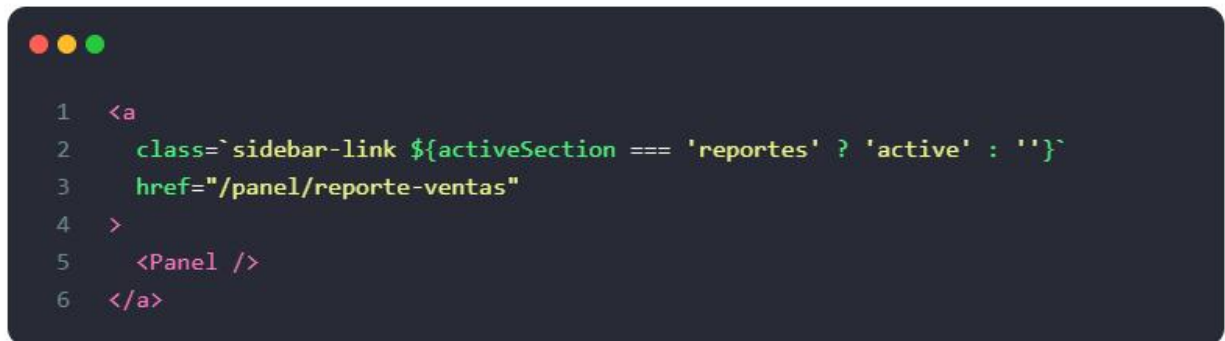


8. Implementación y configuración de rutas

Las rutas se definen dentro del componente `Sidebar.astro` que es envuelto por el elemento HTML denominado `aside`, para indicar el posicionamiento lateral de la barra de navegación principal. Las rutas extraen información desde el lugar donde se utiliza el componente, allí se establece la sección y el contenido que se mostrará en base al parámetro `activeSection`.

Figura 6

Imagen de ejemplo de enrutamiento principal del sistema web de gestión de productos y ventas

A screenshot of a code editor with a dark background and light-colored text. The code is a JSX element representing a sidebar link. It consists of six lines of code, numbered 1 through 6 on the left. Line 1: `<a`. Line 2: `class=`sidebar-link ${activeSection === 'reportes' ? 'active' : ''}``. Line 3: `href="/panel/reporte-ventas"`. Line 4: `>`. Line 5: `<Panel />`. Line 6: ``. The code is color-coded: tags are pink, strings are green, and the variable `activeSection` is blue.

9. Seguridad y autenticación

9.1 Gestión de usuarios y roles

El sistema web de ARIMANA BAZAR estará destinado exclusivamente a su personal autorizado, incluyendo empleados y administradores. Dentro de esta plataforma, el rol de Administrador (Admin) será el único con la capacidad de crear, modificar o eliminar tipos de usuarios. Cada tipo de usuario estará vinculado a un rol específico, el cual definirá de manera clara las funciones y niveles de acceso permitidos en el sistema.

Por ejemplo, algunos usuarios tendrán permisos limitados a la inserción de nuevos productos en el inventario, mientras que otros podrán editar o eliminar productos ya existentes. También se contemplarán usuarios con permisos restringidos, quienes solo podrán visualizar el stock de inventario sin la posibilidad de realizar modificaciones.

Este sistema de roles asegura un control preciso y seguro sobre las operaciones que cada usuario puede llevar a cabo, promoviendo así una gestión eficiente y organizada de los recursos y actividades internas de ARIMANA BAZAR. La asignación de permisos se realizará en función de las funciones y responsabilidades de cada empleado, garantizando la protección de información crítica

y asegurando que cada usuario tenga acceso únicamente a las herramientas necesarias para el desempeño de sus labores.

9.2 Implementación de medidas de seguridad

El sistema cuenta con una entrada que restringe el ingreso únicamente a los usuarios registrados en el sistema. Los accesos están definidos temporalmente en el archivo estático `Usuario.ts` dentro del directorio `data`. El script recorre cada registro y valida las credenciales de correo y contraseña para aceptar o rechazar el ingreso, enviando notificaciones mediante alertas para mejorar la experiencia del usuario.

Cuando las credenciales corresponden a los registros del archivo estático `Perfil.ts` se agrega una propiedad `isLoggedIn` con el valor `true` almacenada en el `localStorage`, de esta manera se restringe el acceso a la aplicación solo para los usuarios registrados.

Figura 7

Imagen del código de acceso al login del sistema web de gestión de productos ventas

```


1  export default function LoginAccess() {
2    useEffect(() => {
3      const loginBtn = document.getElementById("iniciarSesion");
4
5      loginBtn?.addEventListener("click", function () {
6        const correo = document.getElementsByName("correo")[0].value.trim();
7        const clave = document.getElementsByName("clave")[0].value.trim();
8
9        if (correo === "" || clave === "") {
10          Swal.fire({ icon: "info", title: "Campos incompletos" });
11          return;
12        }
13
14        const perfil = PERFIL.find(
15          (emp) => emp.correo === correo && emp.contraseña === clave
16        );
17
18        if (perfil) {
19          localStorage.setItem("isLoggedIn", "true");
20          localStorage.setItem("nomUsuario", perfil.nombres);
21
22          Swal.fire({
23            icon: "success",
24            title: `Bienvenido, ${perfil.nombres}`,
25            timer: 1500,
26            showConfirmButton: false,
27          }).then(() => {
28            window.location.href = "/panel";
29          });
30        } else {
31          Swal.fire({ icon: "error", title: "Credenciales incorrectas" });
32        }
33      });
34    }, []);
35
36    return null;
37  }

```

Asimismo, cada vez que se intenta ingresar directamente desde la ruta al panel del sistema, si el usuario no cuenta con una sesión activa y válida, será redirigido a la ventana del login.

Figura 8

Imagen del código de autenticación del sistema web de gestión de productos y ventas



```
1  if (localStorage.getItem("isLoggedIn") !== "true") {  
2    window.location.href = "/login";  
3  }
```

Nota. El código mostrado en la figura, se encuentra dentro del elemento script cargado en el head del componente `PanelLayout.astro` y `SimplePanelLayout.astro` para restringir el acceso a estas áreas.

Al momento de cerrar sesión por medio del botón en el apartado de perfil de usuario, se eliminó la propiedad `isLoggedIn` del `localStorage` del navegador, mostrando una notificación descriptiva y redirigiendo al usuario a la ventana de inicio de sesión.


Figura 9

Imagen del código de cerrar sesión del sistema web de gestión de productos y ventas

```
1  export default function LogoutAccess() {
2    useEffect(() => {
3      const logoutBtn = document.getElementById("cerrarSesion");
4
5      logoutBtn?.addEventListener("click", function () {
6        Swal.fire({
7          title: "¿Estás seguro?",
8          text: "Se cerrará tu sesión.",
9          icon: "warning",
10         showCancelButton: true,
11         confirmButtonText: "Cerrar sesión",
12         cancelButtonText: "Cancelar",
13       }).then((result) => {
14         if (result.isConfirmed) {
15           localStorage.removeItem("isLoggedIn");
16           localStorage.removeItem("nomUsuario");
17
18           Swal.fire({
19             icon: "info",
20             title: "Sesión cerrada",
21             timer: 1500,
22             showConfirmButton: false,
23           }).then(() => {
24             window.location.href = "/";
25           });
26         }
27       });
28     });
29   }, []);
30
31   return null;
32 }
```


10. Control de versiones

1. Abrir la consola de Visual Studio Code desde la raíz del proyecto local
2. Agregar todos los archivos modificados




```
1 $ git add .
```

3. Realizar el commit con un mensaje descriptivo claro y breve sobre los cambios



```
1 $ git commit -m "feat: Nueva actualizacion"
```

4. Sube los cambios al repositorio remoto en GitHub



```
1 $ git push origin main
```