Proyecto Final del Marketplace de Seguridad – Hito 1

by Herrera Camila & Marín Luis

Introducción

El marketplace de seguridad es una aplicación web diseñada para la venta de implementos de defensa, con roles específicos para usuarios, clientes y administradores. Este documento detalla cada componente del Hito 1, incluyendo diseño de interfaz, navegación, dependencias, base de datos y contrato de API REST.

1. Diseño de la Interfaz Gráfica (Wireframes Finales)

Vistas Públicas (Sin Login):

- **Página Principal (Home):** Lista de productos sin precios. Navbar solo con opciones de Login y Register.

Vistas Privadas (Con Login):

- **Usuarios:**
  - **Navbar:** Cambia para mostrar "Perfil" y habilitar el carrito de compras.
  - **Perfil:** Historial de compras con detalles y botón para repetir compras.
  - **Home:** Ahora muestra precios y desbloquea el carrito.
- **Cliente:**
  - **Navbar:** Muestra "Pedidos" en lugar de "Perfil".
  - **Pedidos:** Lista de todos los pedidos generados, acceso al detalle completo de cada pedido, opción de actualizar estados y deshabilitar/habilitar productos por stock.
- **Admin:**
  - **Navbar:** Acceso a gestión completa.
  - **Funciones:** Crear, editar y eliminar productos, gestionar usuarios y pedidos.

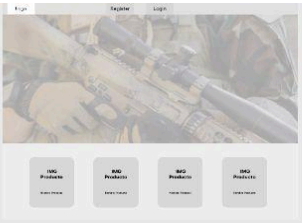
Se incluirán bocetos diseñados en **Figma** que representen cada vista: página principal, perfil de usuario, carrito, pedidos de cliente y panel de administración. Estos bocetos mostrarán la disposición de los elementos, navegación y componentes visuales.

2. Navegación

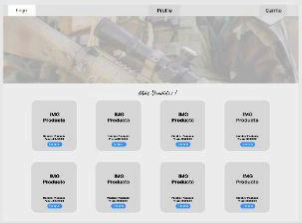
- **Público:** Solo home, login y registro.
- **Usuarios Logueados:** Home con precios, carrito, perfil con historial.
- **Cliente:** Pedidos completos y gestión.
- **Admin:** Gestión total.

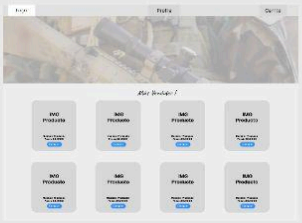
Se utilizará React Router para gestionar la navegación entre vistas públicas y privadas, respetando el flujo definido: home, login/registro, perfil, carrito, pedidos y administración.

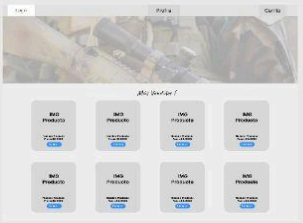
PUBLICO

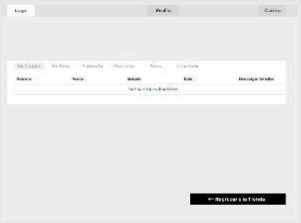


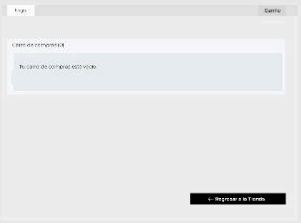
PRIVADO

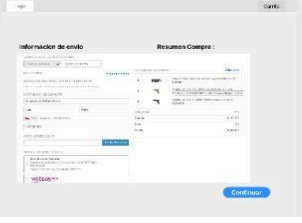












---

3. Dependencias

- **Frontend:** React, Vite, Bootstrap, React Router, Redux, Axios.
- **Backend:** Express.js, PostgreSQL, pgAdmin, JWT, bcryptjs.

---

4. Base de Datos:

Tablas principales:

- **Usuarios**:
  - `id_usuario (PK)`: INT
  - `nombre`: VARCHAR
  - `email`: VARCHAR
  - `contraseña`: VARCHAR (guardada de forma segura, encriptada)
  - `rol`: ENUM('usuario', 'cliente', 'admin') — Definir los roles según el acceso.
  - `fecha_registro`: DATETIME

- **Productos**:
  - `id_producto (PK)`: INT
  - `nombre`: VARCHAR
  - `descripcion`: TEXT
  - `precio`: DECIMAL
  - `categoria`: VARCHAR
  - `stock`: INT
  - `imagen_url`: VARCHAR
  - `fecha_agregado`: DATETIME

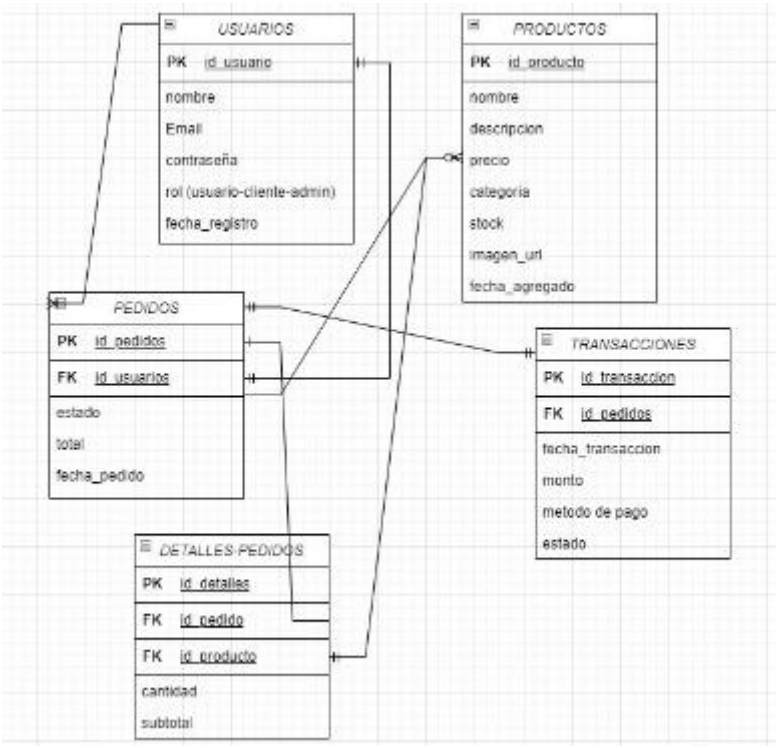
- **Pedidos**:
  - `id_pedido (PK)`: INT
  - `id_usuario (FK)`: INT (relacionado con la tabla de Usuarios)
  - `estado`: ENUM('pendiente', 'enviado', 'entregado')
  - `total`: DECIMAL
  - `fecha_pedido`: DATETIME

- **Detalle_Pedidos**:
  - `id_detalle (PK)`: INT
  - `id_pedido (FK)`: INT (relacionado con la tabla de Pedidos)
  - `id_producto (FK)`: INT (relacionado con la tabla de Productos)
  - `cantidad`: INT
  - `subtotal`: DECIMAL

- **Transacciones**:
  - `id_transaccion (PK)`: INT
  - `id_pedido (FK)`: INT
  - `fecha_transaccion`: DATETIME
  - `monto`: DECIMAL
  - `metodo_pago`: ENUM('tarjeta', 'transferencia', 'otros')
  - `estado`: ENUM('completada', 'fallida')

Relaciones entre tablas:

- Un **usuario** puede tener muchos **pedidos**.
- Un **pedido** puede tener muchos **productos** a través de la tabla de **detalle_pedidos**.
- Cada **pedido** tiene una **transacción** asociada.



---

5. Contrato de la API REST:

Se incluirá documentación en **Polacode** para mostrar ejemplos de código de los endpoints definidos:

- `POST /usuarios/registro` – Registrar usuario.
- `POST /usuarios/login` – Iniciar sesión.
- `GET /usuarios/:id/compras` – Historial de compras.
- `POST /publicaciones` – Crear publicación.
- `GET /publicaciones` – Listar publicaciones.
- `POST /compras` – Realizar compra.
- `PATCH /clientes/pedidos/:id` – Actualizar estado de pedido.
- `POST /pagos` – Registrar pago.

Cada endpoint se describirá con parámetros de solicitud y respuestas esperadas.

```
 Ruta: POST /api/auth/register
// Descripción: Registra un nuevo usuario
// Cuerpo: { "nombre": "string", "email": "string", "cor
// Respuesta: { "mensaje": "Usuario creado exitosamente'

app.post('/api/auth/register', (req, res) => {
    // Lógica para registrar al usuario
  });

  // Ruta: POST /api/auth/login
  // Descripción: Inicia sesión con email y contraseña
  // Cuerpo: { "email": "string", "contraseña": "string'
  // Respuesta: { "token": "string" }

  app.post('/api/auth/login', (req, res) => {
    // Lógica para iniciar sesión
  });

  // Ruta: GET /api/productos
  // Descripción: Obtiene la lista de productos

// Respuesta: [ { "id_producto": "int", "nombre": "strir
```

```
app.post('/api/pedidos', (req, res) => {
  // Lógica para crear un nuevo pedido
});
// Ruta: GET /api/pedidos/:id_usuario
// Descripción: Obtiene los pedidos de un usuario
// Respuesta: [ { "id_pedido": "int", "estado": "string"
app.get('/api/pedidos/:id_usuario', (req, res) => {
  // Lógica para obtener los pedidos de un usuario
});
// Ruta: GET /api/pedidos/:id_pedido
// Descripción: Obtiene detalles de un pedido
// Respuesta: { "id_pedido": "int", "productos": [{ "nom
app.get('/api/pedidos/:id_pedido', (req, res) => {
  // Lógica para obtener detalles de un pedido
});
// Ruta: POST /api/transacciones
// Descripción: Realiza un pago para un pedido
// Cuerpo: { "id_pedido": "int", "monto": "decimal", "me
// Respuesta: { "mensaje": "Transacción completada exito
app.post('/api/transacciones', (req, res) => {
  // Lógica para realizar una transacción
});
```

```
app.get('/api/productos', (req, res) => {
   // Lógica para obtener productos
 });

 // Ruta: POST /api/productos
 // Descripción: Crea un nuevo producto (solo admin)

// Cuerpo: { "nombre": "string", "descripcion": "string"
 // Respuesta: { "mensaje": "Producto creado exitosament

 app.post('/api/productos', (req, res) => {
   // Lógica para crear un nuevo producto
 });

 // Ruta: PUT /api/productos/:id_producto
 // Descripción: Actualiza un producto (solo admin)

// Cuerpo: { "nombre": "string", "descripcion": "string"
 // Respuesta: { "mensaje": "Producto actualizado exitos

 app.put('/api/productos/:id_producto', (req, res) => {
   // Lógica para actualizar un producto
 });

 // Ruta: DELETE /api/productos/:id_producto
 // Descripción: Elimina un producto (solo admin)
 // Respuesta: { "mensaje": "Producto eliminado exitosam

 app.delete('/api/productos/:id_producto', (req, res) =>
   // Lógica para eliminar un producto
 });

 // Ruta: POST /api/pedidos
 // Descripción: Crea un nuevo pedido
 // Cuerpo: { "id_usuario": "int", "productos": [{ "id_p
 // Respuesta: { "mensaje": "Pedido creado exitosamente"
```