Sistema de Seguimiento de Clientes – Prueba Técnica

Kimberly Abigail García Trujillo

Descripción general
El sistema permite que los clientes se registren, administren su información y consulten sus compras. Los administradores tienen acceso completo para gestionar clientes, visualizar sus compras, enviar ofertas y eliminar registros.

Objetivos

- **Objetivo general:** Desarrollar un sistema con roles diferenciados (cliente y administrador) que permita el seguimiento de clientes y el envío de ofertas.

Objetivos específicos:

- Permitir el registro y administración de clientes.
- Registrar compras asociadas a cada cliente.
- Restringir accesos según roles.
- Crear un frontend conectado a una API REST.

Alcance

- Módulos incluidos:
    - Registro de clientes.
    - Gestión de compras.
    - Roles (cliente y administrador).
    - CRUD de clientes y compras.
    - Envío de ofertas (simulado o implementado).

Arquitectura de la solución

- **Diagrama de arquitectura**: Cliente → Interfaz gráfica (React) → API REST (React,typescript) → Base de datos (SQL).
- Explicación de cada capa.

  Frontend: React + TypeScript + Bootstrap, maneja la interfaz de usuario.

  Backend: .NET Core 8 + Entity Framework Core, maneja la lógica de negocio y la persistencia de datos.

  Base de datos: SQL Server, almacena clientes, compras y roles.

Diseño del sistema

Modelo de datos (ERD simplificado):

- **Cliente:** id, nombre, email, contraseña (hash), teléfono, rolId
- **Compra:** id, clienteId, producto, monto, fechaCompra
- **Rol:** id, nombreRol (Cliente / Administrador)

Crear la base de datos

```
CREATE DATABASE SistemaClientes;
GO
USE SistemaClientes;
GO
-- Tabla de Roles
CREATE TABLE Roles (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    NombreRol VARCHAR(50) NOT NULL -- 'Cliente' o 'Administrador'
);
-- Tabla de Usuarios (para login)
CREATE TABLE Usuarios (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL,
    Email VARCHAR(150) UNIQUE NOT NULL,
    ContrasenaHash VARCHAR(255) NOT NULL, -- contraseña encriptada
    Telefono VARCHAR(20),
    RolId INT NOT NULL,
    CONSTRAINT FK_Usuarios_Roles FOREIGN KEY (RolId) REFERENCES Roles(Id)
);
```

```
-- Tabla de Compras

CREATE TABLE Compras (

    Id INT IDENTITY(1,1) PRIMARY KEY,

    UsuarioId INT NOT NULL,

    Producto VARCHAR(150) NOT NULL,

    FechaCompra DATETIME DEFAULT GETDATE(),

    Monto DECIMAL(10,2) NOT NULL,

    CONSTRAINT FK_Compras_Usuarios FOREIGN KEY (UsuarioId) REFERENCES
Usuarios(Id)

);



-- Insertar roles por defecto

INSERT INTO Roles (NombreRol) VALUES ('Cliente'), ('Administrador');

-- Consultas de prueba

SELECT * FROM Roles;

SELECT * FROM Usuarios;

SELECT * FROM Compras;
```

Relaciones:

- Un cliente puede tener muchas compras.
- Un administrador gestiona a todos los clientes.

Casos de uso principales:

- **Cliente:** registrarse, login, ver compras, editar perfil.
- **Administrador:** CRUD de clientes, ver compras, enviar ofertas, eliminar clientes.

Tecnologías utilizadas

- **Frontend:** React + TypeScript + Bootstrap

- **Backend:** .NET Core 8 (C#) + Entity Framework Core
- **Base de datos:** SQL Server
- **Control de versiones:** Git + GitHub
- **Testing/API:** Postman

Seguridad

- Roles y permisos implementados mediante middleware.
- Autenticación con JWT.
- Restricción de acceso a endpoints según rol.

Endpoints de la API

Método	Endpoint	Descripción
POST	`/api/auth/registro-cliente`	Registro de cliente
POST	`/api/auth/login`	Login
GET	`/api/compras/usuario/{usuarioId}`	Ver compras del cliente autenticado
POST	`/api/compras`	Registrar compra
GET	`/api/admin/usuarios`	Ver todos los clientes (solo admin)
PUT	`/api/admin/usuarios/{id}`	Actualizar información del usuario (admin o propio cliente)
DELETE	`/api/admin/usuarios/{id}`	Eliminar cliente (solo admin)

Ejemplo JSON para login:

```
{
  "Email": "aby@email.com",
  "ContrasenaHash": "123456"
}
```

Ejemplo JSON para registrar compra:

```
{
  "UsuarioId": 1,
  "Producto": "Leche",
  "Monto": 25.50
}
```

Manual de despliegue

Backend:

1. Abrir terminal en `backend/BackendClientes/`
2. Configurar cadena de conexión en `appsettings.json`
3. Ejecutar:

- `dotnet restore`
- `dotnet run`

Frontend:

1. Abrir terminal en `frontend/`
2. Instalar dependencias:
   - `npm install`
3. Ejecutar:
   - `npm start`

Variables de entorno necesarias:

- `Jwt:Key` → clave secreta para generar tokens JWT

Conclusiones

- Backend funcional y probado con endpoints de clientes y compras.
- Frontend parcialmente implementado, listo para integrarse con el backend.
- Arquitectura escalable y lista para futuras extensiones (notificaciones, reportes, mejoras de UI).