Escuela Superior de Ingeniería Mecánica y Eléctrica Unidad Zacatenco

Lenguajes del internet

Pérez Arias Julio Enrique

2020302155

Avance del proyecto (E-commerce AVtech)

Manual Técnico

(Módulos Crear Cuenta, Iniciar Sesión, Detalles de Cuenta y Agencia)

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN Y ALCANCE DEL PROYECTO

1.1 Propósito del Documento

El presente Manual Técnico tiene como objetivo documentar exhaustivamente la arquitectura, lógica de programación y estructura de datos del sistema "AVtech". Este documento está dirigido a desarrolladores, administradores de sistemas y evaluadores técnicos, proporcionando la información necesaria para el despliegue, mantenimiento y escalabilidad futura de la aplicación.

1.2 Descripción General de la Solución

AVtech es una plataforma web e-commerce B2B/B2C diseñada para la cotización de equipos audiovisuales. A diferencia de un sistema monolítico tradicional, AVtech implementa una arquitectura desacoplada donde el "Frontend" (Interfaz visual) y el "Backend" (Lógica y Datos) operan como entidades separadas que se comunican mediante protocolo HTTP/S.

1.3 Arquitectura del Sistema

El sistema sigue un patrón de **Arquitectura Limpia (Clean Architecture)** distribuida en capas:

1. **Capa de Presentación (Frontend):** Desarrollada en React JS. Es responsable de la experiencia de usuario (UX/UI), validaciones visuales y renderizado dinámico.
2. **Capa de Servicio (API Backend):** Desarrollada en ASP.NET Core 8. Actúa como el cerebro del sistema, exponiendo *Endpoints* seguros documentados con Swagger.
3. **Capa de Persistencia (Base de Datos):** Gestionada por Microsoft SQL Server, utilizando *Entity Framework Core* como ORM (Object-Relational Mapper) para abstraer las consultas SQL.

2. ENTORNO DE DESARROLLO Y TECNOLOGÍAS

2.1 Stack Tecnológico

Para la construcción de este sistema se seleccionaron tecnologías líderes en la industria por su robustez y tipado estático:

- **Backend:** C# sobre .NET Core (Framework multiplataforma de alto rendimiento).
- **Frontend:** JavaScript (ES6+) con la librería React 18.
- **Base de Datos:** SQL Server 2019/2022.
- **Seguridad:** Librería BCrypt.Net para encriptación unidireccional y System.IdentityModel.Tokens.Jwt para la generación de tokens de acceso.

2.3 Configuración del Servidor Local

El archivo appsettings.json en el Backend es vital para la conexión. Define la cadena de conexión hacia la instancia local de SQL Server:

JSON

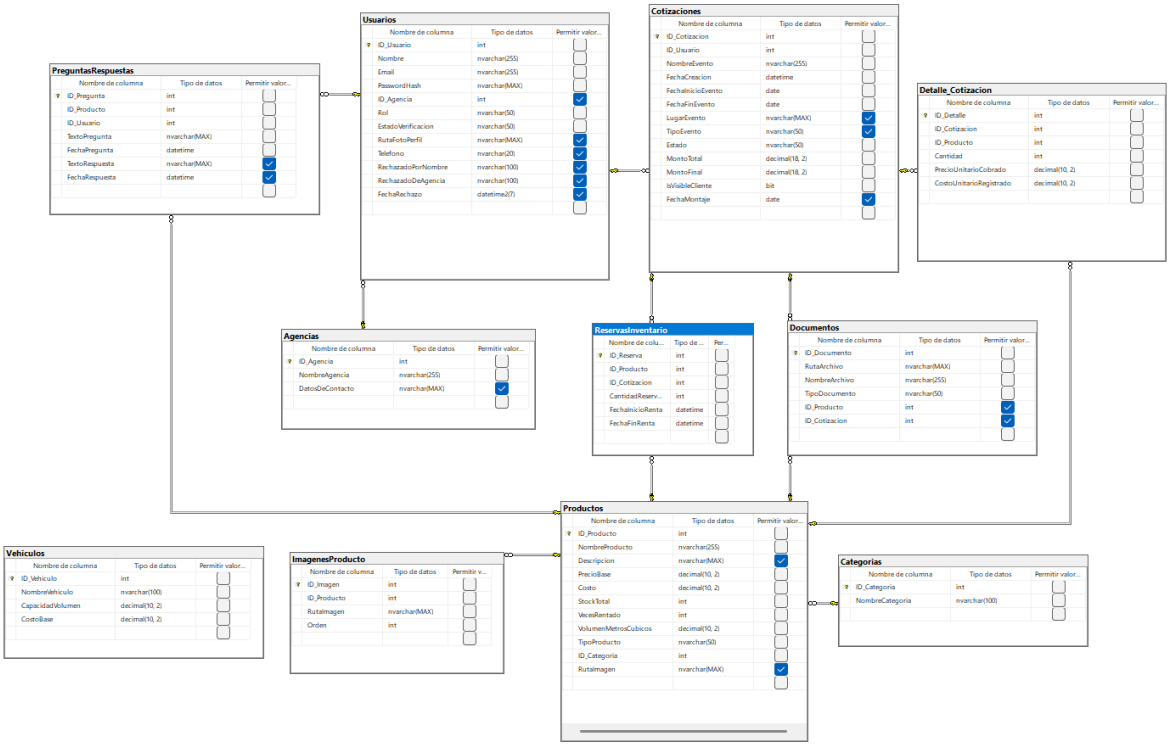
```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=LOCALHOST;Database=AvtechLenguaje;Trusted_Connection=True;Trust
ServerCertificate=True;"
  },
  "Jwt": {
    "Key": "ClaveSecretaSuperSeguraParaFirmarTokens2025",
    "Issuer": "https://localhost:7214",
    "Audience": "https://localhost:3000"
  }
}
```

Explicación: Trusted_Connection=True permite la autenticación mediante credenciales de Windows, eliminando la necesidad de exponer usuarios y contraseñas de base de datos en texto plano.

3. CAPA DE DATOS: DISEÑO DE BASE DE DATOS

3.1 Modelo Entidad-Relación (MER)

La base de datos AvtechLenguaje está normalizada hasta la tercera forma normal (3NF) para evitar redundancia.



3.2 Diccionario de Datos Detallado

Tabla: Usuarios

Fenix.Avtech... dbo.Usuarios

Nombre de columna	Tipo de datos	Permitir valores ...
ID_Usuario	int	☐
Nombre	nvarchar(255)	☐
Email	nvarchar(255)	☐
PasswordHash	nvarchar(MAX)	☐
ID_Agencia	int	☑
Rol	nvarchar(50)	☐
EstadoVerificacion	nvarchar(50)	☑
RutaFotoPerfil	nvarchar(MAX)	☑
Telefono	nvarchar(50)	☑
RechazadoPorNombre	nvarchar(100)	☑
RechazadoDeAgencia	nvarchar(100)	☑
FechaRechazo	datetime2(7)	☑
		☐

Columna	Tipo de Dato	Restricciones	Descripción Técnica
ID_Usuario	INT	PK, Identity	Llave primaria autoincremental. Identificador único interno del sistema para relaciones foráneas.
Nombre	NVARCHAR(255)	NOT NULL	Almacena el nombre completo del usuario para visualización en la interfaz (UI).
Email	NVARCHAR(255)	UNIQUE, NOT NULL	Actúa como el *Username* para el inicio de sesión. Debe ser único en todo el sistema.
PasswordHash	NVARCHAR(MAX)	NOT NULL	**Campo Crítico de Seguridad.** No almacena la contraseña real, sino un hash encriptado generado mediante el algoritmo *BCrypt* con salt incluido.
ID_Agencia	INT	FK, NULL	Llave foránea que vincula al usuario con la tabla Agencias. Si es NULL, el usuario es independiente.
Rol	NVARCHAR(50)	NOT NULL	Define el nivel de autorización (Claims): • Admin: Control total sobre su agencia. • Cliente: Permisos estándar de visualización.
EstadoVerificacion	NVARCHAR(50)	DEFAULT 'Pendiente'	Controla el flujo de aprobación en agencias. Valores: 'Pendiente', 'Verificado', 'Rechazado'.
RutaFotoPerfil	NVARCHAR(MAX)	NULL	Almacena la ruta relativa del sistema de archivos donde se aloja la imagen del usuario (ej: /Uploads/Profiles/img1.jpg).
RechazadoDeAgencia	NVARCHAR(100)	NULL	Campo de auditoría temporal utilizado por el sistema de notificaciones asíncronas para informar al usuario si fue expulsado.

Tabla: Cotizaciones

Fenix.AvtechL....Cotizaciones | Fenix.AvtechLe...- dbo.Usuarios

Nombre de columna	Tipo de datos	Permitir valores ...
ID_Cotizacion	int	☐
ID_Usuario	int	☐
NombreEvento	nvarchar(255)	☑
FechaCreacion	datetime	☑
FechaInicioEvento	date	☑
FechaFinEvento	date	☑
LugarEvento	nvarchar(MAX)	☑
TipoEvento	nvarchar(50)	☑
Estado	nvarchar(50)	☑
MontoTotal	decimal(18, 2)	☑
MontoFinal	decimal(18, 2)	☑
IsVisibleCliente	bit	☑
FechaMontaje	date	☑
		☐

Columna	Tipo de Dato	Restricciones	Descripción Técnica
ID_Cotizacion	INT	PK, Identity	Identificador único de la cotización. Utilizado en las URLs de la API para consultas (GET /api/cotizaciones/{id}).
ID_Usuario	INT	FK, NOT NULL	Llave foránea que establece la propiedad del registro. Permite filtrar las cotizaciones por usuario en el Backend.
NombreEvento	NVARCHAR(255)	NOT NULL	Título descriptivo asignado por el cliente (ej: "Boda Civil", "Conferencia Anual"). Utilizado para búsquedas por texto.
FechaCreacion	DATETIME	DEFAULT GetDate()	Marca de tiempo automática de creación. Vital para el ordenamiento cronológico en el dashboard.
FechaInicioEvento	DATE	NOT NULL	Fecha programada para el inicio del servicio.
FechaFinEvento	DATE	NOT NULL	Fecha de finalización. El sistema utiliza este rango para calcular disponibilidad (lógica futura).
MontoFinal	DECIMAL(18,2)	NOT NULL	Valor calculado resultante de la suma de los detalles (PrecioUnitario * Cantidad). Se almacena para evitar recálculos constantes.
Estado	NVARCHAR(50)	DEFAULT 'En Proceso'	Máquina de estados del documento: • En Proceso: Editable. • Finalizada: Cerrada/Pagada. • Cancelada: Anulada.
IsVisibleCliente	BIT	DEFAULT 1	**Soft Delete (Borrado Lógico).** Si es 0 (False), la cotización no se muestra en el Frontend pero permanece en la base de datos para auditoría histórica.

4. CAPA LÓGICA: ARQUITECTURA DEL BACKEND

4.1 Estructura de la Solución

El proyecto API sigue el patrón de separación de responsabilidades:

- **Controllers:** Reciben la petición HTTP y deciden qué hacer.
- **Data/Entities:** Clases que son un espejo exacto de las tablas SQL ([Table("Usuarios")]).
- **Data/DTOs:** (Data Transfer Objects) Objetos simplificados para enviar/recibir datos sin exponer la estructura interna de la BD.

4.2 Configuración del Núcleo (Program.cs)

Este archivo es el punto de entrada. Aquí se configura la **Inyección de Dependencias (DI)**.

C#

```
var builder = WebApplication.CreateBuilder(args);


// 1. Inyección del Contexto de Base de Datos
builder.Services.AddDbContext<DataContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnecti
on")));


// 2. Configuración de CORS (Permite que React consuma la API)
builder.Services.AddCors(options => {
    options.AddPolicy("AllowWebApp", policy =>
        policy.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});


// 3. Configuración del Serializador JSON
// Evita el error de "Ciclo Infinito" cuando una Entidad A referencia a B, y B
referencia a A.
```

```
builder.Services.AddControllers().AddJsonOptions(x =>
   x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
```

4.3 Patrón de Diseño: Entidades vs. DTOs

Uno de los pilares de seguridad de esta API es el uso de DTOs.

- **Problema:** Si devolvemos la entidad Usuario directa, enviaríamos el PasswordHash al navegador, lo cual es un riesgo grave de seguridad.
- **Solución:** Creamos una clase UserProfileDto que solo contiene Nombre, Email y Foto. El controlador copia los datos de la Entidad al DTO antes de responder.

5. DESGLOSE DE CONTROLADORES Y ENDPOINTS

Los controladores son clases C# que heredan de ControllerBase. Cada método público se convierte en un Endpoint accesible vía web.

5.1 Funcionamiento de los Verbos HTTP

La API utiliza estrictamente los verbos estándar:

- **GET:** Recuperar información (ej. GetMisCotizaciones). No altera la base de datos.
- **POST:** Crear nuevos recursos (ej. Registrar, CrearCotizacion).
- **PUT:** Modificar recursos existentes (ej. UpdatePerfil, CambiarAgencia).
- **DELETE:** Eliminar recursos (ej. DeleteCotizacion - solo Admins).

5.2 UsuariosController: Autenticación y Gestión

Este controlador maneja la seguridad.

Flujo de Login (Código Explicado):

C#

```
[HttpPost("login")]
public async Task<ActionResult> Login(UserLoginDto request) {
    // 1. Buscar usuario por email
    var user = await _context.Usuarios.FirstOrDefaultAsync(u => u.Email == request.Email);
    if (user == null) return BadRequest("Usuario no encontrado.");

    // 2. Verificar contraseña (comparar Hash)
    if (!BCrypt.Net.BCrypt.Verify(request.Password, user.PasswordHash)) {
        return BadRequest("Contraseña incorrecta.");
    }
```

```
    // 3. Generar Token JWT con Claims (ID, Rol, Nombre)
    string token = CrearToken(user);
    return Ok(new { token });
}
```

Lógica Inteligente de Agencia: En el método CambiarAgencia, el sistema no solo actualiza un ID. Verifica si el nombre de agencia ingresado ya existe. Si existe, vincula al usuario. Si no, crea una nueva agencia dinámicamente y asigna al usuario como administrador de la misma.

5.3 CotizacionesController: Filtros Avanzados

Este controlador implementa una lógica de filtrado dependiente del Rol del usuario.

Fragmento de Lógica de Filtrado:

C#

```
var userId = GetUserId(); // Extraído del Token
var userRole = GetUserRole();

IQueryable<Cotizacion> query = _context.Cotizaciones.AsQueryable();

// Regla de Negocio: El Admin ve todo, el Cliente solo lo suyo.
if (userRole == "Admin") {
    query = query.Include(c => c.Usuario); // Traer datos del dueño
} else {
    query = query.Where(c => c.ID_Usuario == userId); // Filtro estricto por ID
}
```

Análisis: El uso de IQueryable permite que el filtro se aplique **en la base de datos** (SQL) y no en la memoria del servidor, optimizando el rendimiento.

6. CAPA DE PRESENTACIÓN: ARQUITECTURA FRONTEND (REACT)

6.1 Estructura de Directorios

- /src/pages: Contiene las vistas completas (LoginPage, UserCotizacionesPage).
- /src/components: Elementos reutilizables (Header, Footer, NotificacionRechazo).
- /src/context: Manejo del estado global.

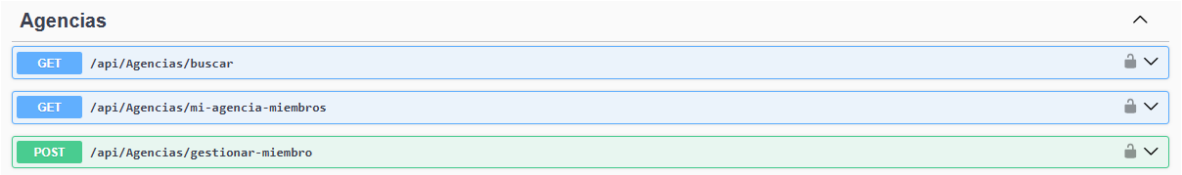
6.2 Gestión de Estado Global (Context API)

Para evitar pasar datos ("props drilling") por docenas de componentes, se utiliza AuthContext.

- Al iniciar la aplicación, AuthContext lee el localStorage, busca el Token JWT, lo decodifica y pone la información del usuario (user.id, user.role) disponible para toda la aplicación. Esto permite que el Header cambie dinámicamente (mostrar "Iniciar Sesión" o el Avatar) sin recargar la página.

6.3 Consumo de API con Axios

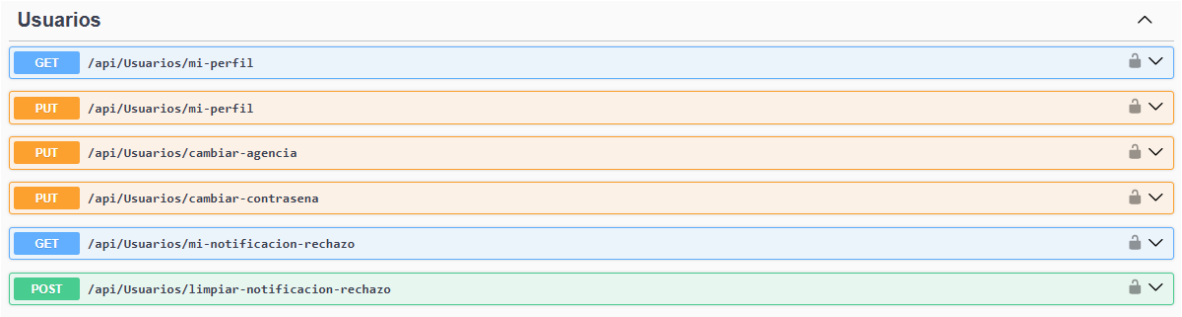
Se utiliza la librería Axios para las peticiones HTTP. **Interceptores:** Cada petición saliente es interceptada para inyectar automáticamente el encabezado Authorization: Bearer [TOKEN]. Esto autentica al usuario ante el Backend en cada clic.



Endpoints de Gestión de Agencias Este controlador implementa la lógica de negocio para los equipos de trabajo.

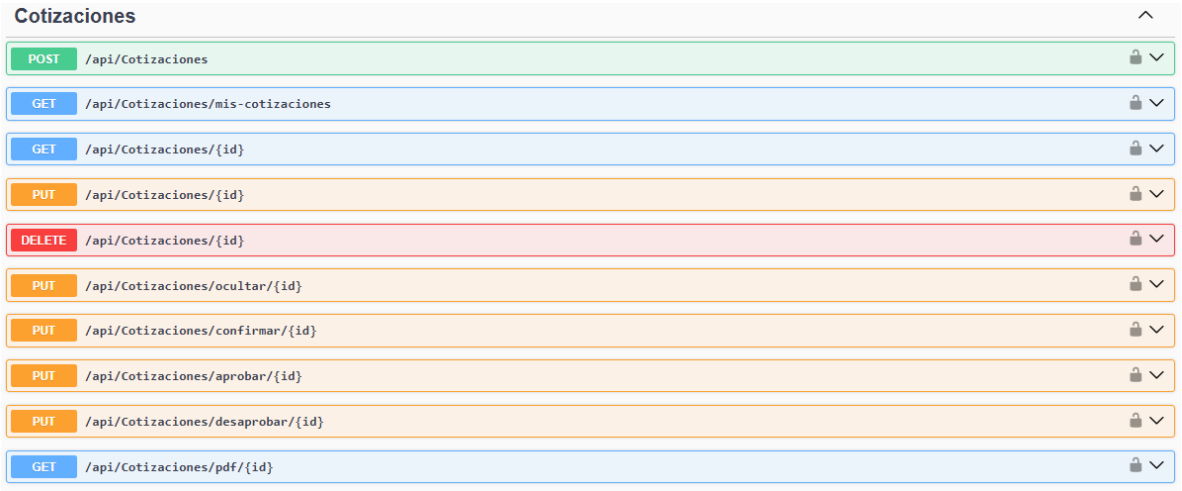
- **GET /buscar:** Endpoint público optimizado para el componente *Autocomplete* del Frontend. Retorna una lista ligera de nombres de agencias para reducir la latencia durante el registro.
- **POST /gestionar-miembro:** Endpoint crítico de seguridad. Recibe un DTO complejo que incluye el ID del usuario objetivo y la acción (Aprobar o

Rechazar). Internamente, valida si el solicitante tiene el rol de 'Admin' antes de ejecutar cualquier cambio en la base de datos.



Endpoints del Controlador de Usuarios (Swagger UI) Como se observa en la documentación autogenerada de la API, este controlador expone los métodos críticos de identidad. Destaca el uso de verbos HTTP semánticos:

- **GET /mi-perfil:** Endpoint protegido que utiliza la extracción de Claims del Token JWT para identificar al usuario sin necesidad de enviar parámetros en la URL, garantizando que un usuario solo pueda ver sus propios datos.
- **PUT /cambiar-agencia:** Permite la modificación transaccional de la relación Usuario-Agencia.
- **POST /limpiar-notificacion:** Endpoint idempotente diseñado para confirmar la lectura de alertas de expulsión.



Operaciones CRUD de Cotizaciones El controlador más extenso del sistema, manejando el ciclo de vida completo de los eventos.

- **GET /mis-cotizaciones:** Implementa parámetros de consulta (Query Parameters) para paginación y búsqueda, permitiendo al frontend solicitar solo bloques de datos específicos.
- **DELETE /{id}:** No elimina físicamente el registro. En su lugar, ejecuta un "Soft Delete" actualizando el campo IsVisibleCliente a false en la base de datos, preservando la integridad referencial.
- **GET /pdf/{id}:** Genera y retorna un stream de archivo binario (PDF) calculado al vuelo basado en los datos de la cotización.

7. FLUJO DE DATOS Y SERIALIZACIÓN

7.1 El Ciclo de Vida de una Petición

1. **Evento:** Usuario hace clic en "Guardar Cotización".
2. **Frontend:** React captura los datos, crea un objeto JSON y lo envía vía POST a /api/Cotizaciones.
3. **Red:** La petición viaja con el Token JWT en la cabecera.
4. **Backend (Middleware):** Valida que el Token sea real y no haya expirado.
5. **Backend (Controller):** Recibe el JSON, lo convierte a objeto C# (CotizacionDto), valida reglas de negocio y guarda en SQL.
6. **Respuesta:** El servidor responde 201 Created y el Frontend muestra una alerta de éxito.

7.2 Interoperabilidad JSON (CamelCase vs PascalCase)

Un desafío técnico resuelto fue la diferencia de nomenclatura.

- C# usa PascalCase (NombreAgencia).
- JavaScript usa camelCase (nombreAgencia).

Solución Implementada: En el frontend, se implementó una capa de normalización de datos robusta:

JavaScript

```
// Normalización para evitar errores undefined
const idAgencia = data.idAgencia || data.ID_Agencia || data.iD_Agencia;
```

Esto asegura que la interfaz gráfica nunca falle, independientemente de cómo el servidor serialice la respuesta.

8. MANEJO DE ERRORES Y DEPURACIÓN

El sistema cuenta con un manejo de excepciones global.

- **Backend:** Los bloques try-catch en los controladores capturan errores de base de datos (DbUpdateException) y devuelven códigos HTTP 500 con mensajes descriptivos pero seguros.
- **Frontend:** Se implementan estados de loading y error en cada página para informar al usuario si algo falla (ej. "Servidor no disponible" o "Credenciales incorrectas"), mejorando la experiencia de usuario.

9. DICCIONARIO TÉCNICO (GLOSARIO)

- **API (Application Programming Interface):** Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.
- **JWT (JSON Web Token):** Estándar abierto (RFC 7519) que define una forma compacta y autónoma para transmitir información de forma segura entre las partes como un objeto JSON.
- **ORM (Object-Relational Mapping):** Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional (Entity Framework).
- **Endpoint:** URL específica donde un servicio puede ser accedido por una aplicación cliente.
- **Soft Delete (Borrado Lógico):** Técnica donde no se eliminan los datos físicamente de la BD, sino que se marca una columna (ej. IsVisible) como falsa para ocultarlos.
- **Dependency Injection (DI):** Patrón de diseño en el que un objeto recibe otros objetos de los que depende, facilitando el desacoplamiento.

10. CONCLUSIÓN TÉCNICA

El desarrollo del sistema AVtech ha permitido la implementación práctica de una arquitectura empresarial moderna. La combinación de **React JS** para una interfaz reactiva y veloz, junto con **.NET Core** para un backend tipado, seguro y escalable, resulta en una aplicación robusta.

Se han resuelto desafíos complejos como la sincronización de estados entre cliente-servidor, la gestión de identidad mediante tokens y la integridad referencial en la base de datos. El código resultante es modular, lo que permitirá futuras expansiones (como módulos de pagos o reportes avanzados) con un impacto mínimo en la base de código actual.