

Memoria del Proyecto



Índice

Introducción.....	2
Arquitectura General del Proyecto.....	3
AWS.....	5
Despliegue de la aplicación Backend (.NET 8).....	7
Desarrollo del Backend (.NET 8 + EF Core + MySQL).....	9
Desarrollo del Frontend (React + Vite).....	15
Mecanismo de Respaldo (Fallback).....	31
Pruebas.....	43
Conclusiones.....	47

Introducción

La aplicación desarrollada en este proyecto es una **herramienta de gestión para un taller**, pensada para facilitar tareas como el control de clientes, vehículos, reparaciones y operaciones internas. Surgió de la necesidad de disponer de una solución **moderna, intuitiva y accesible**, ya que muchas herramientas actuales para talleres son antiguas, complejas o no permiten trabajar desde cualquier dispositivo.

El objetivo principal ha sido crear una aplicación **fácil de usar**, con una interfaz limpia y rápida, permitiendo a cualquier usuario que aunque no tenga conocimientos técnicos, pueda gestionar el taller o ser cliente sin complicaciones. Para ello, se ha diseñado una arquitectura con una separación clara entre la parte visual (**frontend**) y la parte lógica (**backend**), lo que mejora la escalabilidad y el mantenimiento del sistema.

El **frontend** está desarrollado con **React + Vite**, lo que aporta rapidez, modularidad y una gran experiencia de usuario. Por su parte, el **backend** utiliza **ASP.NET Core 8** junto con **Entity Framework Core**, encargándose de toda la lógica de negocio y del acceso directo a la base de datos **MySQL**.

Es fundamental destacar que **React no se comunica directamente con la base de datos**.

Todas las operaciones CRUD (crear, leer, actualizar y eliminar) se realizan exclusivamente en la API del backend, que devuelve la información al frontend en formato JSON a través de peticiones HTTP.

Para el entorno de producción, la aplicación completa se ha desplegado en **AWS**, utilizando una instancia **EC2 t2.small**, accesible mediante la IP elástica **100.26.173.76**. Esto permite que frontend, backend y base de datos funcionen de manera estable y accesible en la nube.

Esta estructura proporciona una base sólida para futuras ampliaciones, mejoras y una mayor disponibilidad del sistema.

Arquitectura General del Proyecto

La aplicación sigue una **arquitectura moderna** basada en **tres capas** bien diferenciadas, lo que permite **separar** la interfaz, la **lógica de negocio** y el **almacenamiento de datos**. Esta estructura **facilita el mantenimiento, las mejoras y la escalabilidad** del sistema.



Arquitectura de 3 capas

- Frontend (React): se encarga únicamente de la interfaz visual y de enviar peticiones HTTP al backend.
- Backend (API REST en .NET 8): procesa las peticiones, aplica reglas de negocio y se comunica con la base de datos.
- Base de datos MySQL: almacena toda la información del taller.

Flujo de funcionamiento

El funcionamiento de la aplicación sigue un **flujo sencillo y eficiente**:

1. El **usuario** interactúa con la interfaz **React**.
2. **React** realiza una **petición HTTP** mediante **axios** hacia:
[http://100.26.173.76:5000/api/...](http://100.26.173.76:5000/api/)
3. El **backend recibe** la petición y realiza:
 - Validaciones
 - Lógica correspondiente
 - Operaciones **CRUD** usando Entity Framework Core
4. La API **devuelve** una respuesta en formato **JSON**.
5. React recibe esa información y actualiza la interfaz automáticamente.

Roles bien diferenciados

- **Frontend:**
 - **Muestra la UI**
 - **Envía peticiones**
 - **No accede a la BD**
- **Backend:**
 - **Expone los endpoints**
 - **Procesa la lógica**
 - **Accede a MySQL**
- **Base de datos:**
 - **Guarda la información del taller**
 - **Responde únicamente al backend**

AWS

Para desplegar la aplicación en un entorno real, se ha utilizado una instancia **AWS EC2**, donde se ejecutan tanto el frontend como el backend. Esta máquina actúa como servidor principal y permite que la aplicación esté disponible públicamente.

Resumen de instancia de i-0f28bd6005b90e1f3 (ubuntuSmall) [Información](#)

Se ha actualizado hace less than a minute

ID de la instancia i-0f28bd6005b90e1f3	Dirección IPv4 pública 100.26.173.76 dirección abierta	Direcciones IPv4 privadas 172.31.23.21
Dirección IPv6 -	Estado de la instancia En ejecución	DNS público ec2-100-26-173-76.compute-1.amazonaws.com dirección abierta
Tipo de nombre de anfitrión Nombre de IP: ip-172-31-23-21.ec2.internal	Nombre DNS de IP privada (solo IPv4) ip-172-31-23-21.ec2.internal	Direcciones IP elásticas 100.26.173.76 (IP UBUNTU SMALL) [IP pública]
Responder al nombre DNS de recurso privado IPv4 (A)	Tipo de instancia t2.small	Hallazgo de AWS Compute Optimizer Suscribirse a AWS Compute Optimizer para recibir recomendaciones. Más información
Dirección IP asignada automáticamente -	ID de VPC vpc-09b953ff82a854007	Nombre del grupo de Auto Scaling -
Rol de IAM -	ID de subred subnet-0078507472b34d8c9	Administradas falso
IMDSv2 Required	ARN de instancia arn:aws:ec2:us-east-1:364111866549:instance/i-0f28bd6005b90e1f3	
Operador -		

Características de la instancia

- **Tipo:** t2.small
- **CPU:** 2 vCPU
- **Memoria:** 2 GB RAM
- **IP pública fija (IP Elástica: 100.26.173.76)**

Este tamaño de instancia es suficiente para alojar una aplicación pequeña o en desarrollo, garantizando buen rendimiento y un coste reducido.

Configuración del entorno en EC2

Para que la aplicación funcione correctamente, en la instancia se han instalado:

- **Runtime de .NET 8** → necesario para ejecutar la API
- **Node.js** → requerido por React para compilar y generar el build
- **MySQL Client** → útil para pruebas y gestión desde la terminal
- **Nginx** → utilizado para servir el build final del frontend

Con esto, el servidor queda preparado para hospedar tanto la API como la aplicación compilada.

React

```
ubuntu@ip-172-31-23-21:~$ dotnet --version
8.0.415
ubuntu@ip-172-31-23-21:~$ node -v
v22.21.0
ubuntu@ip-172-31-23-21:~$ nginx -v
nginx version: nginx/1.24.0 (Ubuntu)
```

Security Group (Firewall de AWS)

Se configuró un **Security Group** para permitir el acceso solo a los puertos necesarios:

- **22 → SSH** (administración del servidor)
- **80 → HTTP** (servidor web Nginx para el frontend)
- **5000 → Backend API** (.NET escucha en este puerto)
- **443 → HTTPS** (Para en un futuro desplegarlo de manera más segura)

Esto garantiza que la aplicación sea accesible desde internet pero manteniendo un nivel básico de seguridad.

Reglas de entrada							
Nombre	ID de la regla del grupo ...	Intervalo de p...	Protocolo	Origen	Grupos de seguridad	Descripción	
-	sgr-036bd93c4346b29df	3306	TCP	0.0.0.0/0	launch-wizard-2 ↗	-	
-	sgr-07ae04d9e302a4b26	5000	TCP	0.0.0.0/0	launch-wizard-2 ↗	-	
-	sgr-0e701a2d1ad8152f6	443	TCP	0.0.0.0/0	launch-wizard-2 ↗	-	
-	sgr-01722069c947c4836	22	TCP	0.0.0.0/0	launch-wizard-2 ↗	-	
-	sgr-03a29c5c76b91fffb	80	TCP	0.0.0.0/0	launch-wizard-2 ↗	-	

Despliegue de la aplicación Backend (.NET 8)

El backend se despliega mediante:

1. [dotnet publish](#)
2. Subida del resultado al servidor
3. Ejecución mediante un **servicio systemd** para que se inicie automáticamente

```
ubuntu@ip-172-31-23-21:~$ sudo systemctl status taller-backend.service
● taller-backend.service - Taller Backend API (.NET)
   Loaded: loaded (/etc/systemd/system/taller-backend.service; enabled; preset: enabled)
   Active: active (running) since Wed 2025-11-26 18:51:28 UTC; 9min ago
     Main PID: 575 (dotnet)
        Tasks: 12 (limit: 2329)
      Memory: 50.7M (peak: 51.0M)
        CPU: 594ms
      CGroup: /system.slice/taller-backend.service
              └─575 /usr/bin/dotnet /var/www/backend/TallerBackend.dll

Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]: warn: Microsoft.AspNetCore.Server.Kestrel[0]
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:   Overriding address(es) 'http://0.0.0.0:5000'. Binding to endpoints defined via IConfiguration and/or UseKestrel() instead.
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]: info: Microsoft.Hosting.Lifetime[14]
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:   Now listening on: http://0.0.0.0:5000
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]: info: Microsoft.Hosting.Lifetime[0]
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:   Application started. Press Ctrl+C to shut down.
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]: info: Microsoft.Hosting.Lifetime[0]
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:   Hosting environment: Production
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]: info: Microsoft.Hosting.Lifetime[0]
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:   Content root path: /var/www/backend
Nov 26 18:51:33 ip-172-31-23-21 taller-backend[575]:
```

Frontend (React)

Para el frontend se realiza:

1. **npm run build**
2. Copia de la carpeta **/dist** a **/var/www/frontend**
3. Configuración de **Nginx** como servidor

```
ubuntu@ip-172-31-23-21:~$ ls /var/www/frontend
assets  citas.json  coches.json  index.html  logo.png  servicios.json  usuarios.json  vite.svg
ubuntu@ip-172-31-23-21:~$ [REDACTED]
GNU nano 7.2
/etc/nginx/sites-enabled/taller-app
server {
    listen 80;
    server_name 100.26.173.76;

    # Frontend React - SPA Configuration
    location / {
        root /var/www/frontend;
        index index.html index.htm;

        # IMPORTANTE: Para SPA, siempre servir index.html para rutas no encontradas
        try_files $uri $uri/ /index.html;

        # Cabeceras de cache para SPA
        add_header Cache-Control "no-cache, no-store, must-revalidate";
        add_header Pragma "no-cache";
        add_header Expires "0";

        # Seguridad
        add_header X-Frame-Options "SAMEORIGIN";
        add_header X-Content-Type-Options "nosniff";
    }

    # Archivos estáticos - cache más largo
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2|ttf|eot)$ {
        root /var/www/frontend;
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # Backend API
    location /api/ {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;

        # CORS
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE, PATCH' always;
        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range,Authorization' always;

        # Preflight
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE, PATCH';
            add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range,Authorization';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain; charset=utf-8';
            add_header 'Content-Length' 0;
            return 204;
        }
    }
}
```

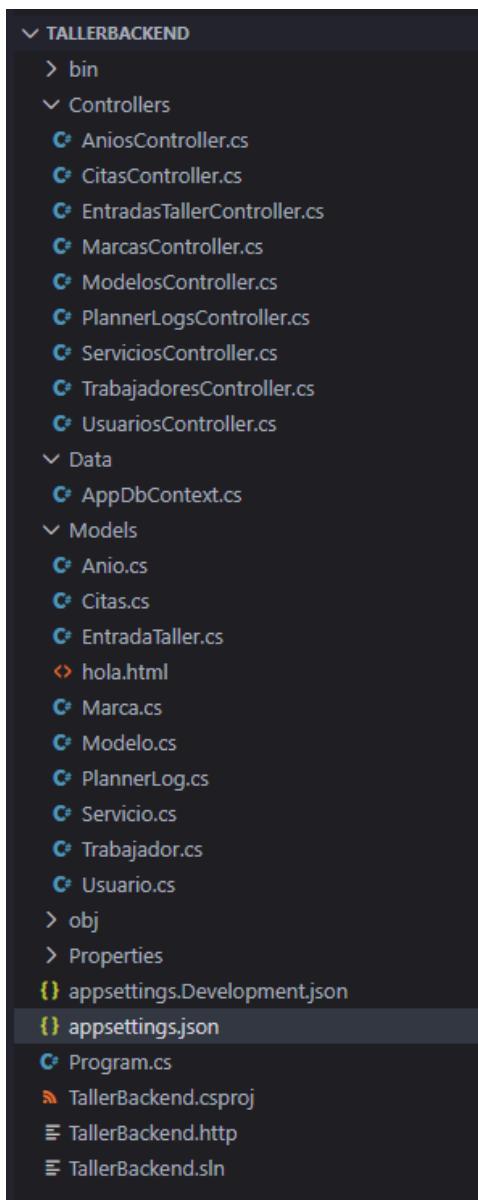
Desarrollo del Backend (.NET 8 + EF Core + MySQL)

El backend de la aplicación está construido utilizando **ASP.NET Core 8**, una tecnología moderna, rápida y multiplataforma. Su función es central en la aplicación: recibe peticiones del frontend, ejecuta la lógica de negocio, y se comunica con la base de datos MySQL mediante **Entity Framework Core**.

Estructura del proyecto

El backend tiene una estructura clara y organizada. Todos los elementos necesarios para la API se encuentran dentro de la carpeta principal **TallerBackend/**.

Elementos principales de la estructura



- **Program.cs**

Configura el servidor web, servicios, EF Core, CORS y enrutamiento.

- **Controllers/**

Contiene los controladores REST que exponen los endpoints de la API.

- **Models/**

Incluye las clases C# que representan las entidades de la base de datos.

- **Data/ApplicationDbContext.cs**

Clase que actúa como puente entre EF Core y MySQL.

- **appsettings.json**

Contiene la cadena de conexión a la base de datos MySQL.

- **Migrations/**

(Si existen) Contiene migraciones generadas por EF Core para crear tablas.

Configuración del Backend

El archivo **Program.cs** registra servicios clave como:

- Entity Framework Core
- Soporte para controladores
- Políticas CORS
- Logging
- Configuración JSON

```
"ConnectionStrings": {  
    "DefaultConnection": "server=100.26.173.76;port=3306;database=tallerdb;user=daming;password=admin;SslMode=None;ConnectionTimeout=30;"  
},  
// BASE DE DATOS  
builder.Services.AddDbContext<AppDbContext>(options =>  
    options.UseMySql(  
        builder.Configuration.GetConnectionString("DefaultConnection"),  
        new MySqlServerVersion(new Version(8, 0, 21))  
    )  
>;
```

Modelos de Datos (Models/)

En la carpeta **Models/** se encuentran las clases que representan las tablas MySQL.

EF Core transforma estas clases en tablas reales dentro de MySQL.

Acceso a datos con ApplicationDbContext

Dentro de **Data/** se encuentra **ApplicationContext.cs**, que mapea los modelos a tablas:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // Mapeo de tablas
    modelBuilder.Entity<Servicio>().ToTable("servicios");
    modelBuilder.Entity<Cita>().ToTable("citas");
    modelBuilder.Entity<Marca>().ToTable("marcas");
    modelBuilder.Entity<Modelo>().ToTable("modelos");
    modelBuilder.Entity<Anio>().ToTable("anios");
    modelBuilder.Entity<Usuario>().ToTable("usuarios");
    modelBuilder.Entity<Trabajador>().ToTable("trabajadores");
    modelBuilder.Entity<EntradaTaller>().ToTable("entradas_taller");
    modelBuilder.Entity<PlannerLog>().ToTable("planner_logs");
```

Migraciones y creación automática de tablas

EF Core permite generar migraciones para crear o actualizar la base de datos:

- `dotnet ef migrations add InitialCreate`
- `dotnet ef database update`

Esto genera una carpeta **Migrations/** con los archivos de creación de tablas.

CRUD real gestionado por la API

El backend es el **único componente** que accede directamente a la base de datos.
React **solo** le envía peticiones HTTP.

Los controladores definen endpoints como:

- **GET** → Obtener datos
- **POST** → Crear registros
- **PUT / PATCH** → Actualizar registros
- **DELETE** → Eliminar registros

Ejemplo de un controlador (servicios):

```
// GET: api/Servicios
[HttpGet]
public async Task<ActionResult<IEnumerable<Servicio>>> GetServicios()
{
    try
    {
        var servicios = await _context.Servicios.ToListAsync();
        return Ok(servicios);
    }
    catch (MySqlConnector.MySQLException mysqlEx)
    {
        _logger.LogError(mysqlEx, "Error de MySQL al obtener servicios");
        return StatusCode(500, $"Error de base de datos: {mysqlEx.Message}");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error inesperado al obtener servicios");
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

// GET: api/Servicios/5
[HttpGet("{id}")]
public async Task<ActionResult<Servicio>> GetServicio(int id)
{
    try
    {
        var servicio = await _context.Servicios.FindAsync(id);
        if (servicio == null)
        {
            _logger.LogWarning("Servicio no encontrado para ID: {Id}", id);
            return NotFound($"Servicio con ID {id} no encontrado.");
        }

        return Ok(servicio);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error al obtener servicio con ID {Id}", id);
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

// POST: api/Servicios
[HttpPost]
public async Task<ActionResult<Servicio>> PostServicio(Servicio servicio)
{
    try
    {
        _context.Servicios.Add(servicio);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetServicio), new { id = servicio.Id }, servicio);
    }
    catch (DbUpdateException dbEx)
    {
        _logger.LogError(dbEx, "Error al guardar el servicio en la base de datos");
        return StatusCode(500, $"Error al guardar en base de datos: {dbEx.InnerException?.Message ?? dbEx.Message}");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error inesperado al crear servicio");
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

// DELETE: api/Servicios/{id}
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteServicio(int id)
{
    try
    {
        var servicio = await _context.Servicios.FindAsync(id);
        if (servicio == null)
        {
            _logger.LogWarning("Servicio no encontrado para DELETE ID {Id}", id);
            return NotFound($"Servicio con ID {id} no encontrado.");
        }

        _context.Servicios.Remove(servicio);
        await _context.SaveChangesAsync();

        logger.LogInformation("Servicio eliminado exitosamente ID {Id}", id);
        return NoContent();
    }
    catch (DbUpdateException dbEx)
    {
        _logger.LogError(dbEx, "Error al eliminar servicio ID {Id}", id);
        return StatusCode(500, $"Error al eliminar en base de datos: {dbEx.InnerException?.Message ?? dbEx.Message}");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error interno al eliminar servicio ID {Id}", id);
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

private bool ServicioExists(int id)
{
    return _context.Servicios.Any(e => e.Id == id);
}
```

```

// PATCH: api/Servicios/{id}
[HttpPatch("{id}")]
public async Task<IActionResult> PatchServicio(int id, [FromBody] Dictionary<string, object> updates)
{
    try
    {
        var servicioExistente = await _context.Servicios.FindAsync(id);
        if (servicioExistente == null)
        {
            _logger.LogWarning("Servicio no encontrado para PATCH ID {Id}", id);
            return NotFound($"Servicio con ID {id} no encontrado.");
        }

        _logger.LogInformation("PATCH recibido para ID: {Id}, Updates: {Updates}",
            id, JsonSerializer.Serialize(updates));

        foreach (var update in updates)
        {
            switch (update.Key.ToLower())
            {
                case "nombre":
                    if (update.Value is string nombre)
                        servicioExistente.Nombre = nombre;
                    break;
                case "precio":
                    if (decimal.TryParse(update.Value.ToString(), out decimal precio))
                        servicioExistente.Precio = precio;
                    break;
                case "descripcion":
                    if (update.Value is string descripcion)
                        servicioExistente.Descripcion = descripcion;
                    break;
                case "Imagen":
                    if (update.Value is string imagen)
                        servicioExistente.Imagen = imagen;
                    break;
            }
        }

        await _context.SaveChangesAsync();
        _logger.LogInformation("PATCH exitoso para ID {Id}", id);
        return Ok(servicioExistente);
    }
    catch (DbUpdateException dbEx)
    {
        _logger.LogError(dbEx, "Error de base de datos al actualizar servicio con PATCH ID {Id}", id);
        return StatusCode(500, $"Error al actualizar en base de datos: {dbEx.InnerException?.Message ?? dbEx.Message}");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error interno al actualizar servicio con PATCH ID {Id}", id);
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

```

```

// PUT: api/Servicios/{id}
[HttpPut("{id}")]
public async Task<IActionResult> PutServicio(int id, Servicio servicio)
{
    _logger.LogInformation("PUT recibido para ID: {Id}", id);
    _logger.LogInformation("Datos recibidos: {Servicio}", JsonSerializer.Serialize(servicio));

    if (id != servicio.Id)
    {
        _logger.LogWarning("IDs no coinciden: URL {Id} vs Body {BodyId}", id, servicio.Id);
        return BadRequest("El ID de la URL no coincide con el ID del servicio.");
    }

    var servicioExistente = await _context.Servicios.FindAsync(id);
    if (servicioExistente == null)
    {
        _logger.LogWarning("Servicio no encontrado para ID {Id}", id);
        return NotFound($"Servicio con ID {id} no encontrado.");
    }

    servicioExistente.Nombre = servicio.Nombre;
    servicioExistente.Precio = servicio.Precio;
    servicioExistente.Descripcion = servicio.Descripcion;
    servicioExistente.Imagen = servicio.Imagen;

    try
    {
        await _context.SaveChangesAsync();
        _logger.LogInformation("PUT exitoso para ID {Id}", id);
        return NoContent();
    }
    catch (DbUpdateConcurrencyException ex)
    {
        _logger.LogError(ex, "Error de concurrencia al actualizar servicio ID {Id}", id);
        if (!ServicioExists(id))
            return NotFound();
        return StatusCode(500, "Error de concurrencia en la base de datos.");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error interno al actualizar servicio ID {Id}", id);
        return StatusCode(500, $"Error interno del servidor: {ex.Message}");
    }
}

```

Seguridad: política CORS

Como frontend y backend no están en el mismo dominio, se configuró la política:

"AllowFrontend"

Permitida para:

- o <http://100.26.173.76>

Esto evita bloqueos del navegador al enviar peticiones cross-origin.

```

app.UseCors("AllowFrontend");

app.UseAuthorization();

// ENDPOINTS
app.MapControllers();

app.Run("http://0.0.0.0:5000");

```

Pruebas del backend con Swagger

.NET genera automáticamente una interfaz Swagger donde se pueden probar todos los endpoints sin necesidad de herramientas externas.

URL:

<http://100.26.173.76:5000/swagger>

The screenshot shows the Swagger UI for the **TallerBackend** API. The interface is organized into sections based on API endpoints:

- Anios**:
 - GET /api/Anios
 - POST /api/Anios
 - GET /api/Anios/{id}
 - PUT /api/Anios/{id}
 - DELETE /api/Anios/{id}
- Citas**:
 - GET /api/Citas
 - POST /api/Citas
 - GET /api/Citas/{id}
 - PUT /api/Citas/{id}
 - PATCH /api/Citas/{id}
 - DELETE /api/Citas/{id}
 - GET /api/Citas/estado/{estado}
 - GET /api/Citas/disponibles
 - PATCH /api/Citas/{id}/estado
 - PATCH /api/Citas/{id}/revisado
 - PATCH /api/Citas/{id}/completar
- EntradasTaller**:
 - GET /api/EntradasTaller
 - POST /api/EntradasTaller
 - GET /api/EntradasTaller/{id}
 - PUT /api/EntradasTaller/{id}
 - DELETE /api/EntradasTaller/{id}
 - POST /api/EntradasTaller/desde-cita
 - PATCH /api/EntradasTaller/{id}/estado
 - PATCH /api/EntradasTaller/{id}/revisado
 - GET /api/EntradasTaller/estadisticas

Desarrollo del Frontend (React + Vite)

El frontend de la aplicación está desarrollado con **React + Vite**, lo que permite tener una interfaz rápida, ligera y muy modular. Su misión es **mostrar la información al usuario** y comunicarse con el backend mediante peticiones HTTP, sin manejar directamente ningún acceso a la base de datos.

Papel del frontend

El frontend se centra exclusivamente en:

- Renderizar los componentes de la interfaz
- Enviar peticiones HTTP al backend
- Recibir datos en formato JSON
- Actualizar la UI según la respuesta de la API

El diseño está pensado para que la aplicación sea **simple, intuitiva y usable incluso para personas sin experiencia técnica**.

Estructura del proyecto

```
✓ TALLER
  > dist
  > node_modules
  > public
  ✓ src
    ✓ axios
      JS CatalogoService.js
      JS CitaService.js
      JS EntradaTallerService.js
      JS http-axios.js
      JS index.js
      JS PayPalService.js
      JS PlannerLogService.js
      JS ServicioService.js
      JS ServicioServicios.js
      JS TrabajadorService.js
      JS UsuarioService.js
    ✓ componentes
      > auth
      > chat
      > citas
      > common
      > paypal
      > planner
      > rutas
      > servicios
      > static
      > trabajadores
      > vehiculos
      > pdfs
      > styles
      ⚙ main.jsx
    ❁ .gitignore
  ○ eslint.config.js
  ▷ index.html
  { package-lock.json
  { package.json
  ⓘ README.md
  ⚡ vite.config.js
```

La estructura del proyecto sigue el patrón estándar de Vite + React:

/components → componentes de la interfaz

/axios → funciones de llamadas a la API

main.jsx → componente principal

Esta separación ayuda a mantener el código limpio y organizado.

Componente principal main.jsx

El punto central del frontend es el archivo **main.jsx**, donde se configura por completo la estructura de la aplicación, el enrutamiento, los proveedores globales y la carga de cada página. En esta aplicación, *main.jsx sustituye al tradicional App.jsx*, actuando como el punto que define el esqueleto del frontend.

En muchas aplicaciones React, [App.jsx](#) es el componente raíz donde se definen:

- El [`<BrowserRouter>`](#)
- El [`<Routes>`](#)
- Los layouts comunes (Header, Footer)
- Los Providers globales

Sin embargo, en este proyecto **se ha decidido eliminar App.jsx** y colocar toda esta estructura directamente en `main.jsx`.

Esto ofrece varias ventajas:

- Menos archivos intermedios → estructura más limpia.
- Menos anidamiento de componentes.
- El punto de entrada queda completamente claro y centralizado.
- Alinea mejor el flujo con Vite, que ya arranca la aplicación desde [`main.jsx`](#).

```

<StrictMode>
  <BrowserRouter>
    <AuthProvider>
      <PayPalProvider>
        <Header />
        <Routes>
          {/* Públicas */}
          <Route path="/" element={<Inicio />} />
          <Route path="/sobre-nosotros" element={<SobreNosotros />} />
          <Route path="/servicios" element={<Servicios />} />
          <Route path="/inicio-sesion" element={<IniciarSesion />} />
          <Route path="/registro" element={<Registro />} />
          <Route path="*" element={<Error404 />} />
          {/* Rutas protegidas */}
          <Route path="/pedir-cita" element={
            <RutasProtegidas>
              <FormularioCita />
            </RutasProtegidas>
          }/>

```

Son las rutas accesibles sin autenticación:

```

{/* Rutas protegidas */}
<Route path="/pedir-cita" element={
  <RutasProtegidas>
    <FormularioCita />
  </RutasProtegidas>
}/>
{/* Rutas trabajador / admin */}
<Route path="/planner" element={
  <RutasTrabajador permitirAdmin={true}>
    <Planner />
  </RutasTrabajador>
}/>

```

Ciertas partes del proyecto requieren que el usuario haya iniciado sesión. Comprueba si el usuario está autenticado mediante [AuthProvider](#). Si **sí lo está**, renderiza el componente interno. Si **no lo está**, redirige al login. Los trabajadores tienen acceso a zonas del sistema que un cliente normal no debería ver.

```

{/* Rutas admin */}
<Route path="/administrar-servicios" element={
  <RutasAdmin>
    <AdministrarServicios />
  </RutasAdmin>
}/>
<Route path="/administrar-citas" element={
  <RutasAdmin>
    <Administrarcitas />
  </RutasAdmin>
}/>
<Route path="/administrar-trabajadores" element={
  <RutasAdmin>
    <AdministrarTrabajadores />
  </RutasAdmin>
}/>
<Route path="/administrar-vehiculos" element={
  <RutasAdmin>
    <AdministrarVehiculos />
  </RutasAdmin>
}/>
<Route path="/planner-log" element={
  <RutasAdmin>
    <PlannerLog />
  </RutasAdmin>
}/>
</Routes>
<Footer />
</PayPalProvider>
</AuthProvider>
</BrowserRouter>
</StrictMode>

```

Estas rutas comprueban:

- Usuario autenticado
- **Rol = admin**
- Solo entonces renderizan la página

Si no cumple, redirige automáticamente.

AuthProvider

Gestiona:

- Inicio de sesión
- Registro
- Sesión persistente
- Rol del usuario
- Estado global de autenticación

Todos los componentes dentro del provider pueden acceder al usuario actual.

Página de Servicios

Cada usuario logueado sin ser trabajador ni administrador se puede acceder a la página de servicios,a la página de pedir cita y al chat con ia.

En la página de servicio se carga con axios la lista de servicios disponibles del taller en un conjunto de botones en un scroll horizontal que al clicar en uno de ellos se carga un componente en un modal, con la información detallada del servicio

Nuestros Servicios

Seleccione un servicio para conocer todos los detalles

Cambio de aceite y filtro

Alineación y balanceo

Cambio de pastillas de freno

Revisión general

Cambio de ba

Cambio de pastillas de freno

80€



Descripción

Se reemplazan las pastillas de freno desgastadas por unas nuevas en el eje delantero o trasero, según sea necesario. Unas pastillas en buen estado son esenciales para una frenada segura y eficiente.

PayPalProvider

Envuelve toda la app para habilitar pagos.

Carga scripts, claves y lógica de interacción con PayPal.

Tras integrar el **PayPalProvider**, la aplicación incorpora un formulario avanzado para gestionar las citas del taller. Este formulario combina **validaciones inteligentes**, **carga dinámica de datos**, **control de disponibilidad** y un **sistema de pago obligatorio de 10€**, que funciona como depósito reembolsable y se descuenta del precio final de la reparación.

Para evitar reservas falsas o duplicadas, el usuario debe pagar un depósito de **10€** antes de que el sistema registre la cita.

Una vez pagado, el backend recibe la información del pago y guarda la cita como confirmada.

SOLICITUD DE CITA

Programa su visita a nuestro taller especializado

Pagar 10.00€ para la cita
Se descontará del precio final del trabajo

Datos Personales

Mario Martín García
123456747
tgwxmarioox@gmail.com

Información del Vehículo

BMW
X1
2022
2024KLX

Detalles de la Cita

Fecha: 27/11/2025 Hora: 16:00
 jueves laborable

Fallo del cambio automático

CONTINUAR AL PAGO - 10.00€ >

✓ Depósito reembolsable - Se aplica al precio final

Para evitar reservas falsas o duplicadas, el usuario debe pagar un depósito de **10€** antes de que el sistema registre la cita.

Una vez pagado, el backend recibe la información del pago y guarda la cita como confirmada.

El formulario obtiene la información del vehículo desde el backend en tiempo real:

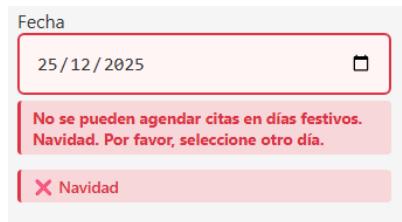
- Al abrir el formulario se cargan **todas las marcas**.
- Cuando el usuario selecciona una marca, se muestran únicamente **los modelos pertenecientes a esa marca**.
- Al seleccionar un modelo, se filtran los **años de fabricación** de ese modelo en concreto.

Todo esto evita errores y facilita que el usuario introduzca datos correctos del vehículo.

El formulario controla que los usuarios no puedan elegir días en los que el taller no abre.

El sistema aplica automáticamente:

- Bloqueo de **sábados y domingos**
- Festivos **nacionales**
- Festivos **de la Comunidad de Madrid**



Cálculo de Semana Santa

El sistema utiliza el algoritmo de Gauss (Computus) para calcular la fecha de Pascua y, a partir de ahí:

- Jueves Santo = Pascua – 3 días
- Viernes Santo = Pascua – 2 días

```
// Festivos móviles (Semana Santa)
const pascua = calcularPascua(year);
const juevesSanto = new Date(pascua);
juevesSanto.setDate(pascua.getDate() - 3);
const viernesSanto = new Date(pascua);
viernesSanto.setDate(pascua.getDate() - 2);

const festivosMoviles = [
  {
    fecha: juevesSanto.toISOString().split("T")[0],
    nombre: "Jueves Santo",
  },
  {
    fecha: viernesSanto.toISOString().split("T")[0],
    nombre: "Viernes Santo",
  },
];
festivos.push(...festivosFijos, ...festivosMadrid, ...festivosMoviles);
```

Cuando el usuario selecciona un día válido:

1. React llama al backend.
2. El backend devuelve la **lista real de horas libres**.
3. El formulario solo muestra horas que no estén ocupadas.
4. Si el día está completo, se muestra un mensaje claro.

```
useEffect(() => {
  if (form.fecha) {
    consultarHorasDisponibles(form.fecha);
  }
}, [form.fecha]);

const response = await CitaService.obtenerHorasDisponibles(fecha);

if (response.data && Array.isArray(response.data)) {
  setHorasDisponibles(response.data);
} else {
  setHorasDisponibles(todasLasHoras);
}
} catch (error) {
  console.error("Error consultando horas disponibles:", error);
  setHorasDisponibles(todasLasHoras);
} finally {
  setConsultandoHoras(false);
}
```

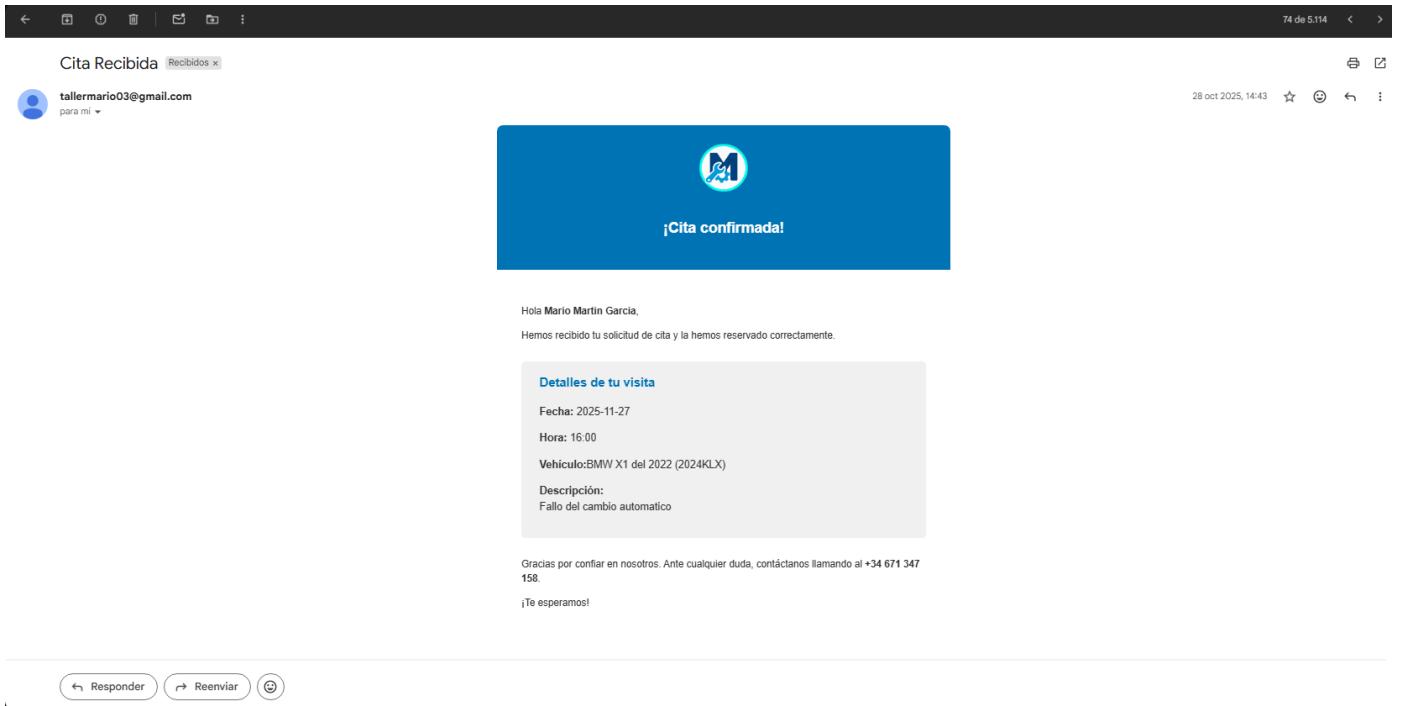
Flujo completo de reserva

El flujo del usuario es:

1. Rellena todos los datos.
2. Selecciona una fecha válida.
3. El sistema valida el día (laborable/no festivo).
4. Se cargan las horas disponibles.
5. El usuario continúa al pago de 10€.
6. Al completar el pago, la cita se registra en el backend.
7. El sistema envía un email de confirmación.
8. Se muestra en pantalla un **resumen de la cita**.

Correo de confirmación

Una vez se efectúa el pago de la cita se registra la cita en la base de datos y se envia al usuario un correo de confirmación usando la herramienta **emailjs**



Donde se usa la función **emailjs.send** con el id del servicio, el id de la plantilla y la clave pública

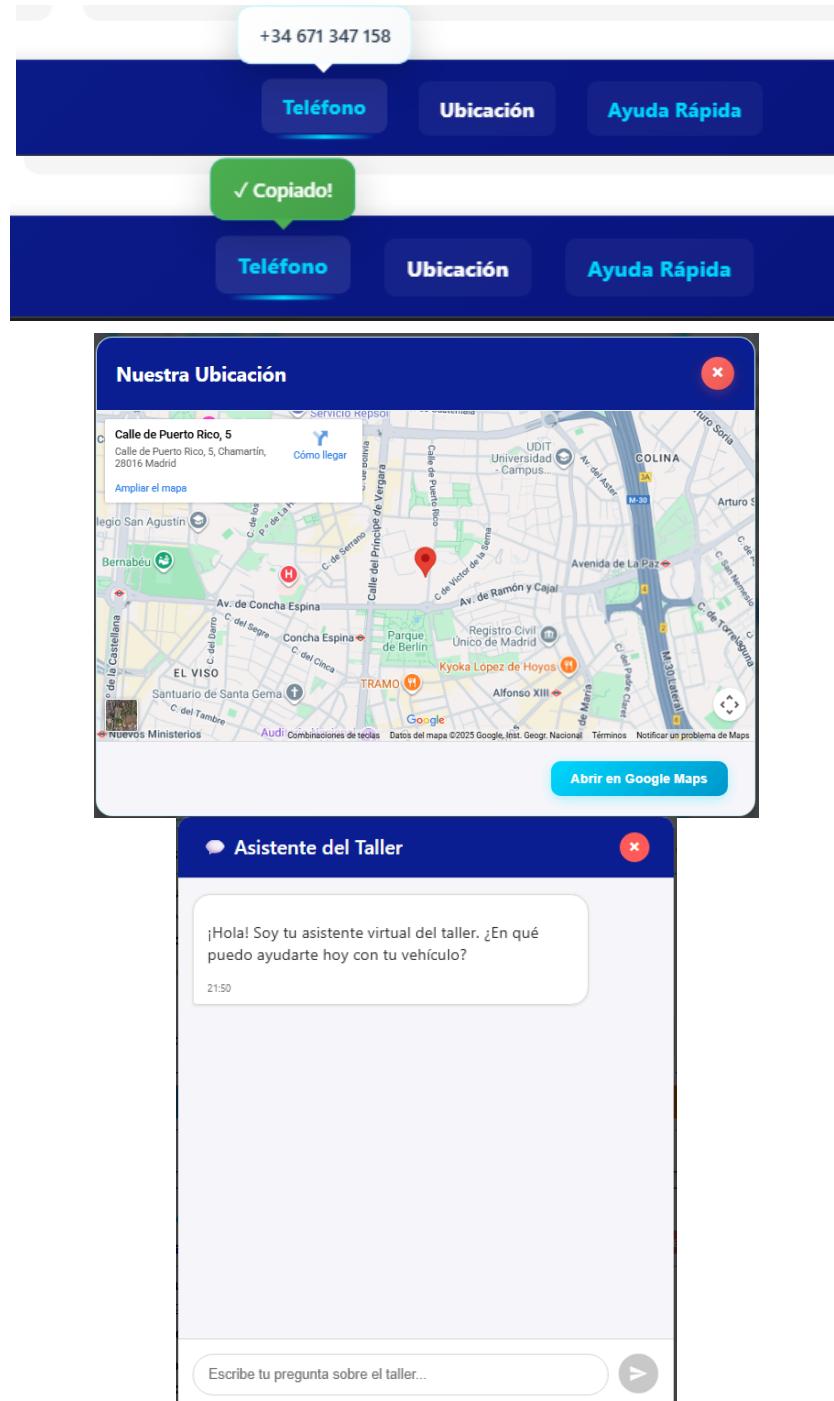
```
const serviceId = "service_ilewpu7";
const templateId = "template_prnrd2q";
const publicKey = "GeShWvojA9P2CKA5n";

const enviarEmailConfirmacion = async (templateParams) => {
  try {
    const response = await emailjs.send(
      serviceId,
      templateId,
      templateParams,
      publicKey
    );
    console.log("Email enviado exitosamente:", response);
    return response;
  } catch (error) {
    console.error("Error al enviar email:", error);
  }
};
```

Footer

El Footer contiene tres funcionalidades principales:

- Copiar número de teléfono en el portapapeles.
- Apertura del modal de ubicación mediante Google Maps.
- Apertura del chat asistente virtual.



El Footer utiliza estados internos para controlar la visibilidad de cada modal e incorpora pequeñas interacciones como tooltips y animaciones.

Modal de Ubicación

El modal de ubicación muestra la dirección del taller incrustada mediante un iframe de Google Maps.

Este modal es completamente independiente y se muestra sobre la interfaz mediante una capa oscura (overlay).

Funcionalidades incluidas

- Visualización de un mapa interactivo mediante iframe.
- Botón para “Abrir en Google Maps” que redirige al usuario a la aplicación externa o navegador.
- Botón de cierre en la parte superior derecha.
- Uso de un overlay que oscurece el resto de la página e impide interacciones fuera del modal.

Sistema de Chat Asistente Virtual

El chat integrado ofrece soporte inmediato al usuario. Se accede desde el Footer y se presenta dentro de un modal independiente.

Objetivo del chat

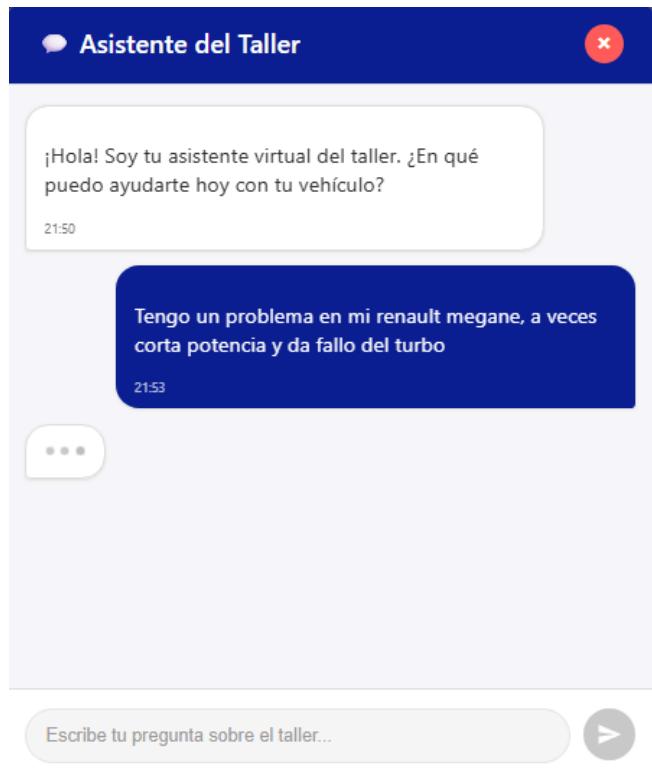
El chat proporciona asistencia automatizada sobre servicios del taller, precios orientativos, mantenimiento, horarios y dudas técnicas.

Funciona de manera completamente local en el frontend, con peticiones externas hacia modelos de lenguaje alojados en HuggingFace.

Arquitectura del componente

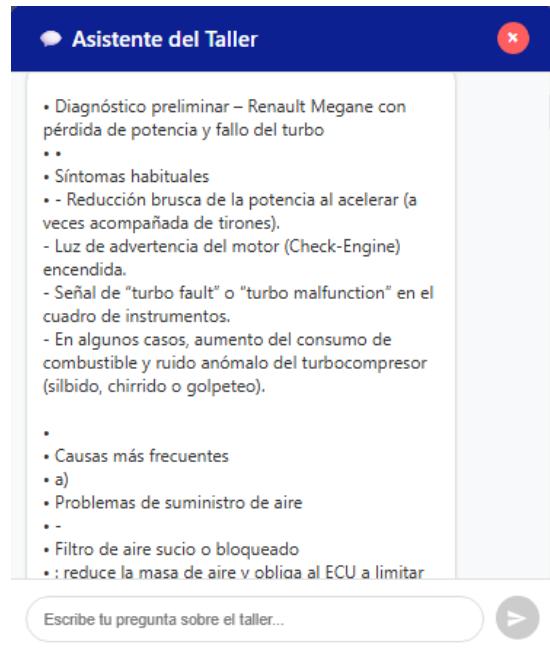
El componente mantiene tres estados principales:

- Historial de mensajes.
- Mensaje en edición.
- Indicador de actividad del bot.



Los **mensajes** se renderizan en **dos formatos** diferenciados: mensajes del usuario y mensajes generados por el asistente. Cada mensaje incluye una marca de tiempo.

El **área de mensajes** utiliza un sistema de **desplazamiento automático** para asegurar que el usuario siempre ve el último mensaje recibido.



Comunicación con la API de HuggingFace

El componente está preparado para solicitar respuestas a un modelo tipo GPT mediante el endpoint "chat/completions".

El mensaje enviado se compone de:

- Un contexto con la información del taller.
- La consulta del usuario.

El modelo responde con un texto que posteriormente se procesa para eliminar:

- Markdown.
- Etiquetas HTML.
- Formatos no compatibles con el estilo visual del chat.

```
//token de Hugging Face
const HF_TOKEN = "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";
```

```
const getAIResponseFromAPI = async (message) => {
  try {
    console.log("Enviando mensaje a la API...");
    const response = await fetch(
      "https://router.huggingface.co/v1/chat/completions",
      {
        method: "POST",
        headers: {
          Authorization: `Bearer ${HF_TOKEN}`,
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          model: "openai/gpt-oss-120b:groq",
          messages: [
            {
              role: "system",
              content: `Eres un asistente especializado en taller mecánico.  
Información del taller:  
- Dirección: Calle Puerto Rico 5, Chamartín, 28016 Madrid  
- Teléfono: +34 671 347 158  
- Horario: Lunes a Viernes 8:00-18:00, Sábados 9:00-13:00  
  
Responde de manera útil, profesional y detallada.  
Proporciona información completa sin límites de longitud.  
Usa solo texto plano sin formato markdown ni tablas.  
Sé muy específico y técnico cuando sea necesario.`,
            },
            {
              role: "user",
              content: message,
            },
          ],
          temperature: 0.7,
          stream: false,
        }),
      }
    );
    console.log("Status de respuesta:", response.status);
    if (!response.ok) {
      const errorText = await response.text();
      console.error(`Error detallado: ${errorText}`);
    }
  } catch (error) {
    console.error(`Error en la solicitud: ${error}`);
  }
};
```

```
let processed = text
  // Convertir tablas a formato legible
  .replace(/\|([^\|]+)\|([^\|]+)\|([^\|]+)/g, "\n• $1: $2 | $3")
  .replace(/\|-/g, "") // Remover separadores de tabla
  // Convertir listas markdown
  .replace(/d+\.\s+/g, "\n• ")
  .replace(/\*\*\s*/g, "\n• ")
  // Limpiar formato markdown pero mantener énfasis
  .replace(/\*\*([^\*]+)\*\*/g, "$1")
  .replace(/\*([^\*]+)\*/g, "$1")
  .replace(/\`([^\`]+)\`/g, "$1")
  .replace(/\#{1,6}\s?/g, "")
  // Convertir saltos de línea HTML
  .replace(/<br\s*\//gi, "\n")
  .replace(/<[^>]+(>|$)/g, "") // Quitar otras etiquetas HTML
  // Limpiar espacios múltiples
  .replace(/\n\s*\n/g, "\n\n")
  .replace(/\ +/g, " ")
  .trim();

return processed;
```

Esto asegura que todas las respuestas se representan como texto plano legible

Mecanismo de Respaldo (Fallback)

En caso de que la API de HuggingFace no esté disponible, el sistema recurre automáticamente a un conjunto de respuestas predefinidas.

Estas respuestas abarcan los temas más frecuentes, como mantenimiento, precios, neumáticos, frenos o problemas de motor.

Este sistema garantiza que el chat pueda seguir proporcionando información útil incluso sin depender del servicio externo.

```
const getFallbackResponse = (message) => {
  const lowerMessage = message.toLowerCase();

  if (lowerMessage.includes("hola") || lowerMessage.includes("buenas")) {
    return "Hola! 🚗 Soy tu asistente virtual del taller mecánico. Estoy aquí para ayudarte con cualquier consulta sobre tu vehículo. Puedo asistirte con información sobre servicios :)";
  } else if (
    lowerMessage.includes("precio") ||
    lowerMessage.includes("cuánto") ||
    lowerMessage.includes("cuesta")
  ) {
    return "💡 **Información detallada de precios aproximados:**\n\n* Cambio de aceite y filtro: Desde 50€ (dependiendo del tipo de aceite y vehículo)\n* Pastillas de freno delanteras";
  } else if (
    lowerMessage.includes("horario") ||
    lowerMessage.includes("hora") ||
    lowerMessage.includes("abierto")
  ) {
    return "🕒 **Horarios de atención detallados:**\n\nLunes a Viernes: Mañanas: 8:00 - 14:00\nTardes: 15:00 - 18:00\nSábados: Solo mañanas: 9:00 - 13:00\nDomingos y festivos: Cerrado";
  } else if (
    lowerMessage.includes("aceite") ||
    lowerMessage.includes("cambio")
  ) {
    return "💡 **Información completa sobre cambio de aceite:**\n\n* Frecuencia recomendada:\n  - Vehículos gasolina: Cada 15,000 km o 1 año\n  - Vehículos diésel: Cada 10,000-12,000 km";
  } else if (
    lowerMessage.includes("frenos") ||
    lowerMessage.includes("freno")
  ) {
    return "💡 **Sistema de frenos - Guía completa:**\n\n* Señales de alerta que indican revisión urgente:\n  - Ruido metálico o chirrido al frenar\n  - Vibración en el pedal o volante";
  } else if (
    lowerMessage.includes("motor") ||
    lowerMessage.includes("falla") ||
    lowerMessage.includes("problema")
  ) {
    return "💡 **Diagnóstico de problemas de motor - Asistencia profesional:**\n\n* Problemas comunes y sus posibles causas:\n  - Motor se para o falla:\n    - Problemas de combustible (bomba de combustible, inyector, etc.)";
  } else if (
    lowerMessage.includes("neumático") ||
    lowerMessage.includes("llanta") ||
    lowerMessage.includes("rueda")
  ) {
    return "💡 **Neumáticos - Información completa:**\n\n* Presiones recomendadas (consultar manual del vehículo):\n  - Normal: 2.2-2.5 bares (32-36 PSI)\n  - Carga pesada: +0.2-0.3 bares";
  } else if (
    lowerMessage.includes("batería") ||
    lowerMessage.includes("arranque")
  ) {
    return "💡 **Baterías - Guía completa:**\n\n* Duración media:\n  - Baterías estándar: 3-5 años\n  - Baterías premium: 4-6 años\n  - Baterías AGM/Gel: 5-7 años\n\n* Síntomas de batería débil:\n  - Encendido lento\n  - Arranque lento\n  - Luces intermitentes\n  - Pantalla de información de la batería";
  } else {
    return "💡 **Asistencia completa para tu taller mecánico:**\n\nEntiendo tu consulta y estoy aquí para ayudarte de manera integral. Como asistente especializado en taller mecánico, ¡estoy aquí para resolver tus dudas y ofrecerte información precisa!";
  }
};
```

Planner

El componente Planner es un **sistema de gestión** que permite **administrar citas y entradas directas** de vehículos mediante un sistema de columnas tipo Kanban. Solo los **trabajadores y administradores** pueden acceder al planner, pero **solo los administradores** pueden **ver los logs** de actividad.

The screenshot shows the 'Planner de Taller' interface. At the top, there's a navigation bar with 'Inicio', 'Servicios', 'Pedir Cita', 'Planner', a user icon, and 'Cerrar Sesión'. Below the navigation is the title 'Planner de Taller' and a subtitle 'Gestiona el estado de citas y vehículos arrastrando las tarjetas entre columnas'. A purple button 'Ver Logs de Actividad' is visible. The main area has three columns: 'Pendiente' (orange), 'En Proceso' (pink), and 'Finalizado' (blue). The 'Pendiente' column contains two items: one for 'Mario Martín' (Audi TT, 1998, matrícula 1234ABC) and another for 'Agustín Aguilera' (Citroën C5, N/A, matrícula 9802HHX). The 'En Proceso' column also contains two items for 'Mario Martín' (BMW M3, 2006, matrícula 2839NAD) and 'Laura Gómez' (Citroën C4, 2007, matrícula 1234ACB). The 'Finalizado' column has a message 'No hay items finalizados'. At the bottom, there are buttons for 'Teléfono', 'Ubicación', and 'Ayuda Rápida', and a blue '+' button.

Funcionalidades Principales

Sistema de Columnas Kanban

- **Pendiente:** Vehículos que acaban de llegar al taller
- **En Proceso:** Vehículos en reparación
- **Finalizado:** Trabajos completados

Gestión de Items

- **Citas:** Procedentes del formulario de reserva online
- **Entradas Directas:** Vehículos que llegan sin cita previa

Generación de documentos mediante JSPDF

23/12/2025 - 10:00

Mario Martin

Vehículo: Audi TT

Año: 1998

Matrícula: 1234ABC

Teléfono: 123456789

Email: tgwxmarioox@gmail.com

Descripción: Fallo en el freno de mano

 Entrada  Presupuesto  Seguro

↔ Arrastrar para mover

Se utiliza la librería **jsPDF** para generar automáticamente documentos PDF profesionales para la gestión del taller mecánico. jsPDF funciona creando documentos **PDF** directamente en el navegador mediante **JavaScript**, permitiendo generar de forma dinámica cuatro tipos esenciales de documentos: el justificante de entrada del vehículo que sirve como comprobante de recepción en el taller, el documento para el seguro con información técnica detallada para las aseguradoras, el presupuesto desglosado con trabajos y repuestos para aprobación del cliente, y finalmente la factura oficial que se genera automáticamente cuando el cliente acepta el presupuesto.

Generación de documentos de entrada y seguro

JUSTIFICANTE DE ENTRADA

Taller Mecánico AutoPro
Calle Puerto Rico 5, Chamartín, 28016 Madrid
Tel: +34 671 347 158
Email: info@autopro.com

Fecha: 26/11/2025
Nº Referencia: 5

INFORMACIÓN DEL CLIENTE

Nombre: Mario Martín
Teléfono: 123456789
Email: tgwxmarioox@gmail.com

INFORMACIÓN DEL VEHÍCULO

Marca: Audi
Modelo: TT
Matrícula: 1234ABC
Año: 1998

DESCRIPCIÓN DEL TRABAJO

Fallo en el freno de mano

ESTADO ACTUAL

Estado: Pendiente

JUSTIFICANTE PARA SEGURO

Taller Mecánico AutoPro
Calle Puerto Rico 5, Chamartín, 28016 Madrid
Tel: +34 671 347 158
CIF: B12345678

Fecha: 26/11/2025

Por medio del presente, Taller Mecánico AutoPro con CIF B12345678, certifica que el vehículo con las siguientes características se encuentra actualmente en nuestras instalaciones para su reparación:

Propietario: Agustín Aguilera
Vehículo: Citroën C5
Matrícula: 9802HHX
Año: No especificado

El vehículo ingresó en nuestras instalaciones el día 26/11/2025 y permanecerá en el taller hasta la finalización de las reparaciones.

Este documento puede ser presentado a compañías de seguros como justificante de la estancia del vehículo en el taller.

Firma y sello del taller:

Documento emitido el 26/11/2025 a las 23:50

Presupuesto

Generar Presupuesto

Cliente: Agustín Aguilera
Vehículo: Citroën C5 - 9802HHX

Descripción del Trabajo *

Revision Pre ITV

Mano de Obra

Mano de obra general	1	45	45,00 €
----------------------	---	----	---------

Repuestos y Materiales

Filtro de aceite	1	15	15,00 €
------------------	---	----	---------

Resumen del Presupuesto

Subtotal (sin IVA):	60,00 €
IVA (21%):	12,60 €
TOTAL:	72,60 €

El **presupuesto** se **genera** con un **formulario** donde se ponen el trabajo, las **horas** de ese trabajo y el **coste**, además de si hiciese falta **materiales** necesarios se añadiría el **nombre de la pieza** y el **precio** y hace el **cálculo total** con y sin **IVA**

Si fuese una **cita**, se **resta automáticamente** el coste de la misma en el presupuesto

Resumen del Presupuesto	
Descuento por reserva:	- 10,00 €
Se ha aplicado el descuento del depósito de reserva pagado	
Subtotal (sin IVA):	50,00 €
IVA (21%):	10,50 €
TOTAL:	60,50 €

PRESUPUESTO DE REPARACIÓN

Taller Mecánico AutoPro
Calle Puerto Rico 5, Chamartín, 28016 Madrid
Tel: +34 671 347 158
CIF: B12345678

Fecha: 27/11/2025
Nº Presupuesto: PS-6-2294
Válido hasta: 12/12/2025

Fecha: 27/11/2025
Nº Presupuesto: PS-6-3742
Válido hasta: 12/12/2025

DATOS DEL CLIENTE Y VEHÍCULO

Cliente: Agustín Aguilera
Teléfono: 671247158
Email: tgwxmarioox@gmail.com

Vehículo: Citroën C5
Matrícula: 9802HHX
Año: No especificado

DESCRIPCIÓN DEL TRABAJO

Revisión Pre ITV

MANO DE OBRA

Mano de obra general 1 horas 45,00 €/h 45,00 €

REPUESTOS Y MATERIALES

Filtro de aceite 1 uds. 15,00 € 15,00 €

RESUMEN DEL PRESUPUESTO

Subtotal (sin IVA): 60,00 €
IVA (21%): 12,60 €
TOTAL: 72,60 €

PRESUPUESTO DE REPARACIÓN

Taller Mecánico
Calle Puerto Rico 5, Chamartín, 28016 Madrid
Tel: +34 671 347 158
CIF: B12345678

Fecha: 27/11/2025
Nº Presupuesto: PS-6-3742
Válido hasta: 12/12/2025

DATOS DEL CLIENTE Y VEHÍCULO

Cliente: Agustín Aguilera
Teléfono: 671247158
Email: tgwxmarioox@gmail.com

Vehículo: Citroën C5
Matrícula: 9802HHX
Año: No especificado

DESCRIPCIÓN DEL TRABAJO

Revisión Pre ITV

MANO DE OBRA

Mano de obra general 1 horas 45,00 €/h 45,00 €

REPUESTOS Y MATERIALES

Filtro de aceite 1 uds. 15,00 € 15,00 €

RESUMEN DEL PRESUPUESTO

Subtotal original: 50,00 €
Descuento por reserva: - 10,00 €

Subtotal (sin IVA): 40,00 €
IVA (21%): 8,40 €
TOTAL: 48,40 €

* Se ha aplicado el descuento del depósito de reserva de 10,00€

Presupuesto de una entrada directa

Presupuesto de cita

Si el **cliente acepta** el presupuesto, mediante un **botón** y se **genera** otro **pdf** con la **factura**

FACTURA	
Taller Mecánico AutoPro Calle Puerto Rico 5, Chamartín, 28016 Madrid Tel: +34 671 347 158 CIF: B12345678	Fecha: 27/11/2025 Factura N°: F-F-6-4023 Nº Cliente: CL-6
DATOS DEL CLIENTE	
Nombre: Agustín Aguilera Teléfono: 671247158 Vehículo: Citroën C5 Matrícula: 9802HHX	
CONCEPTO	
Reparación y mantenimiento vehículo Citroën C5 - 9802HHX	
Mano de obra general - 1 horas	45,00 €
Filtro de aceite - 1 uds.	15,00 €
Base Imponible:	60,00 €
IVA (21%):	12,60 €
TOTAL:	72,60 €
Formas de pago aceptadas: Efectivo, Transferencia, Tarjeta Garantía: 12 meses en mano de obra y repuestos instalados Factura emitida por sistema automatizado	

Sistema de Logs del Planner

- Registro automático de todas las acciones
- Solo es visible para administradores

Logs de Actividad - Planner

Registro de todas las acciones realizadas en el planner

Usuario: Filtrar por usuario... Fecha: dd/mm/aaaa Acción:

52TOTAL LOGS 22MOVIMIENTOS 1NUEVAS ENTRADAS 15PDFS GENERADOS

28/11/2023, 23:01:34 Mario Martín PRESUPUESTO_ACEPTADO

Aceptó presupuesto y generó factura para Citroën C5

Citroën C5 - 9802HHX Agustín Aguilera Tipo: cita (ID: 6)

28/11/2023, 23:00:23 Mario Martín GENERAR_PRESUPUESTO

Generó presupuesto para Citroën C5

Citroën C5 - 9802HHX Agustín Aguilera Tipo: cita (ID: 6)

28/11/2023, 22:54:42 Mario Martín GENERAR_PDF_ENTRADA

Generó justificante de entrada para Audi TT

Audi TT - 1234ABC Mario Martín Tipo: cita (ID: 5)

28/11/2023, 22:50:52 Mario Martín GENERAR_PDF_SEGURO

Generó justificante para seguro de Citroën C5

Citroën C5 - 9802HHX Agustín Aguilera Tipo: cita (ID: 6)

28/11/2023, 22:50:44 Mario Martín GENERAR_PDF_ENTRADA

Generó justificante de entrada para Audi TT

Audi TT - 1234ABC Mario Martín Tipo: cita (ID: 5)

El **Log del Planner** es un diario donde se registran todas las acciones importantes que ocurren en el taller. Solo los **administradores** pueden verlo.

Cada vez que sucede algo en el sistema, se guarda automáticamente en el PlannerLog:

- Cuando un **mecánico mueve una cita o entrada** entre columnas del planner.
- Cuando se genera un **presupuesto**.
- Cuando se crea un **documento PDF**.
- Cuando se **acepta una factura**.

Los administradores pueden buscar información específica usando **filtros**, por ejemplo:

- Ver lo que hizo un trabajador en un día determinado.
- Encontrar todos los presupuestos de la semana pasada.

La información es muy **visual**:

- Los **movimientos** entre columnas son **azules**.
- Los **documentos PDF** son **morados**.
- Los **pagos** son **verdes**.
- También hay números que muestran **resúmenes de actividad**: cuántos movimientos hubo, cuántos documentos se generaron y cuántas entradas nuevas se crearon.

Además, los administradores pueden:

- **Descargar** toda la información en un archivo de texto.
- **Eliminar** registros muy antiguos para mantener el sistema limpio.

SweetAlert2

SweetAlert2 es una biblioteca JavaScript que **reemplaza los diálogos nativos del navegador** (alert, confirm, prompt) con **modales modernos y personalizables**. Proporciona una experiencia de usuario mejorada con diseños atractivos y funcionalidades avanzadas que los diálogos nativos no ofrecen.

Para qué se usa

Se utiliza principalmente para **comunicar estados importantes** al usuario de forma elegante y profesional. En aplicaciones web como un taller mecánico, SweetAlert2 es ideal para:

- **Notificaciones de éxito** cuando una operación se completa correctamente
- **Confirmaciones de acción** antes de realizar cambios irreversibles
- **Mensajes de error** cuando ocurren problemas en las operaciones
- **Advertencias** para alertar sobre situaciones que requieren atención
- **Diálogos de carga** durante procesos asíncronos

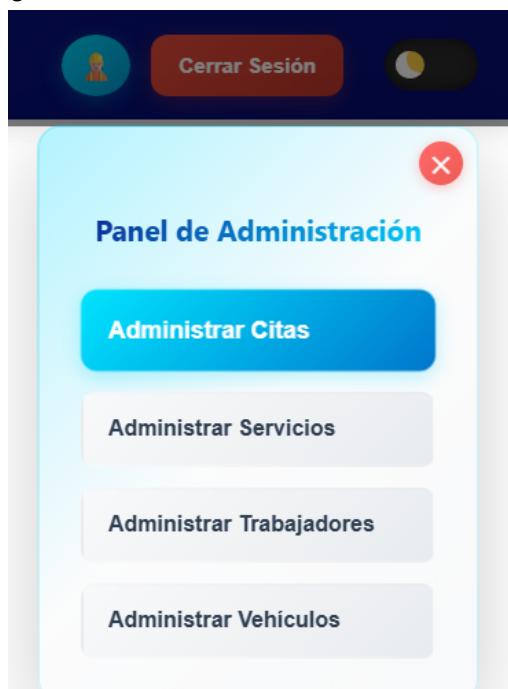
Ventajas Principales

- Mejora la experiencia del usuario
- Puedes personalizarlos completamente
- Incluye funciones como timers automáticos, formularios integrados, soporte para imágenes...
- Mejora la accesibilidad



Páginas de Administración

Cuando el usuario es administrador aparecerá en el Header un botón con un modal con links para las diferentes páginas de administración



Administrador Servicios

The image displays a grid of service cards. Each card contains a small image, the service name, its price, and two buttons: 'Editar' (blue) and 'Eliminar' (red). The services listed are:

Imagen	Servicio	Precio	Editar	Eliminar
	Cambio de aceite y filtro	60€	Editar	Eliminar
	Alineación y balanceo	45€	Editar	Eliminar
	Cambio de pastillas de freno	80€	Editar	Eliminar
	Revisión general	40€	Editar	Eliminar
	Cambio de batería	90€	Editar	Eliminar
	Carga de aire acondicionado	60€	Editar	Eliminar
	Diagnóstico con escáner	35€	Editar	Eliminar
	Cambio de correa de distribución	250€	Editar	Eliminar
	Limpieza de inyectores	70€	Editar	Eliminar
	Cambio de amortiguadores	200€	Editar	Eliminar
	Restauración de vehículos	5000€	Editar	Eliminar
	Cambio de bujías	45€	Editar	Eliminar

Página de Administración de Servicios

Comunicación con el Backend (Axios)

El frontend se comunica con la API del backend usando Axios, con la URL en una variable poder manejarlo mejor en caso de que haya algún cambio.

```
import axios from 'axios';

const URL = 'http://100.26.173.76:5000/api';

export const httpServicios = axios.create({
  baseURL: `${URL}/Servicios`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpUsuarios = axios.create({
  baseURL: `${URL}/Usuarios`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpMarcas = axios.create({
  baseURL: `${URL}/Marcas`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpModelos = axios.create({
  baseURL: `${URL}/Modelos`,
  headers: { 'Content-Type': 'application/json' },
});
}

export const httpAnios = axios.create([
  baseURL: `${URL}/Anios`,
  headers: { 'Content-Type': 'application/json' },
]);

export const httpCitas = axios.create({
  baseURL: `${URL}/Citas`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpTrabajadores = axios.create({
  baseURL: `${URL}/Trabajadores`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpEntradasTaller = axios.create({
  baseURL: `${URL}/EntradasTaller`,
  headers: { 'Content-Type': 'application/json' },
});

export const httpPlannerLogs = axios.create({
  baseURL: `${URL}/PlannerLogs`,
  headers: { 'Content-Type': 'application/json' },
});
```

Este patrón se repite para todas las operaciones CRUD.

Manejo del Estado en React

Para gestionar la información mostrada en pantalla se utilizan hooks como:

useState

Permite almacenar datos como formularios, listas, mensajes, etc.

useEffect

Se ejecuta cuando el componente se renderiza para cargar datos desde la API:

Gracias a estos hooks, la interfaz se actualiza automáticamente cada vez que se reciben datos nuevos del backend.

Actualización de la UI

Una vez que llega la respuesta de la API, la interfaz se actualiza en tiempo real:

- Listas que se refrescan
- Formularios que se limpian
- Mensajes de éxito o error
- Estados de carga o espera

Todo esto se hace sin recargar la página, lo que mejora mucho la experiencia del usuario.

Pruebas

Pruebas del Backend (API) – Swagger

El backend desarrollado en **ASP.NET Core 8** incluye por defecto **Swagger**, una herramienta que permite probar la API de forma visual y sin necesidad de utilizar programas externos como Postman o CURL. Esta interfaz es muy útil durante el desarrollo, ya que ofrece una vista completa de todos los endpoints disponibles del backend.

¿Qué permite hacer Swagger?

Swagger facilita varias tareas esenciales:

- **Visualizar todos los endpoints** expuestos por la API REST.
- **Realizar pruebas** de llamadas **GET, POST, PUT, PATCH y DELETE** directamente desde el navegador.
- **Enviar cuerpos JSON** para probar inserciones o actualizaciones.
- Validar **modelos**, campos obligatorios y formatos.
- Comprobar **códigos de estado HTTP**, como 200, 400 o 404.

Cómo se realizaron las pruebas

Todas las pruebas del backend se realizaron accediendo a:

- <http://100.26.173.76:5000/swagger>

Desde ahí se comprobó que:

- El backend recibe correctamente los datos enviados por el frontend.
- Es capaz de crear, modificar y eliminar registros en **MySQL** mediante **Entity Framework Core**.
- Las respuestas devueltas por la API están siempre en **formato JSON**, lo cual permite al frontend interpretarlas fácilmente.

POST /api/Servicios

Parameters

No parameters

Request body application/json

```
{
  "id": 0,
  "nombre": "Chapa y pintura",
  "descripcion": "Arreglo de aboyaduras, arañazos y cualquier tipo de desperfecto",
  "imagen": "http://imagenes/chapaypintura.png",
  "precio": 120
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://100.26.173.76:5000/api/Servicios' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 0,
    "nombre": "Chapa y pintura",
    "descripcion": "Arreglo de aboyaduras, arañazos y cualquier tipo de desperfecto",
    "imagen": "http://imagenes/chapaypintura.png",
    "precio": 120
}'
```

Request URL http://100.26.173.76:5000/api/Servicios

Server response

Code	Details
500 Undocumented	Error: Internal Server Error Response body <pre>Error interno del servidor: An exception has been raised that is likely due to a transient failure. Consider enabling transient error resiliency by adding 'EnableRetryOnFailure()' to the 'UseMySql' call.</pre> Download Response headers <pre>content-type: text/plain; charset=utf-8 date: Wed, 26 Nov 2025 19:41:35 GRT server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK Media type text/plain Control Accept header: Example Value Schema <pre>{ "id": 0, "nombre": "string", "descripcion": "string", "imagen": "string", "precio": 0 }</pre>	No links

Ejemplo de un POST desde swagger

Swagger también permitió validar que la API gestionaba correctamente:

- Errores de validación
- Respuestas vacías
- Datos mal formados
- Intentos de operaciones sobre registros inexistentes

Esto garantiza que el backend es estable y que responde adecuadamente incluso ante casos no previstos.

Pruebas del Frontend (React) – React Testing Library (RTL) + Jest

Propósito de las Pruebas

El sistema de pruebas para los componentes valida el comportamiento funcional mediante simulaciones controladas. Las pruebas verifican que las interacciones del usuario producen los resultados esperados en la interfaz, independientemente del estado del backend.

Herramientas Principales

- Jest: Motor de ejecución de pruebas con entorno DOM virtual
- React Testing Library: Utilidades para renderizado y simulación de interacciones
- JSON Server: Servidor de desarrollo para simular APIs REST

Enfoque de Testing

Para probar el componente `Servicios.jsx`, implementaría un sistema de pruebas que simula el comportamiento real del usuario sin depender del backend.

Estrategia de Implementación

Configuración de Mocks

- Mock de la API: Simularía ServicioService.obtenerServicios() para devolver datos de prueba predefinidos
- Mock de SweetAlert: Interceptaría las notificaciones para verificar mensajes de error
- Mock de componentes hijos: Simplificaría DetalleServicio para focalizarme en el componente principal

Casos de Prueba Principales

Prueba de Carga Inicial:

- Verificar que se llama a la API al montar el componente
- Comprobar que se renderizan todas las tarjetas de servicios
- Confirmar que se muestra el mensaje de selección inicial

Prueba de Interacción de Usuario:

- Simular clics en diferentes servicios
- Validar que la tarjeta seleccionada recibe estilos activos
- Verificar que se renderiza el detalle del servicio correcto

Prueba de Manejo de Errores:

- Forzar errores en la petición API
- Comprobar que se muestra la notificación de error adecuada
- Validar que la interfaz permanece funcional

TAREAS	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13
Análisis y definición de requisitos													
Diseño preliminar de la base de datos													
Preparación del entorno y creación de proyectos (React + .NET + MySQL)													
Modelado de base de datos y migraciones													
Implementación del backend: CRUD básicos													
Sistema de autenticación: Login, registro y JWT													
Estructura inicial del frontend y rutas base													
Implementación frontend del login y registro con roles													
Gestión de servicios en frontend (CRUD + panel admin)													
Formulario de cita (marcas, modelos, años)													
Validación avanzada de fechas (festivos, fines de semana, Semana Santa)													
Panel de administración de citas (listar, filtrar, editar)													
Implementación del Planner y Planner Log													
Integración del pago con PayPal													
Integración del chat inteligente (HuggingFace)													
Modal de ubicación + mapa de Google Maps													
Ajuste de UX/UI y estilos finales													
Despliegue del backend en AWS EC2 (Linux + Nginx + Kestrel)													
Despliegue del frontend en AWS (Nginx estático)													
Configuración de CORS, seguridad y pruebas finales													
Realización de Pruebas													
Documentación final del proyecto													

Conclusiones

El desarrollo de este proyecto ha permitido construir una aplicación moderna y bien estructurada, basada en la separación completa entre **frontend** y **backend**, lo que ha aportado múltiples beneficios tanto a nivel técnico como de organización.

Ventajas de separar frontend y backend

La separación entre React (frontend) y ASP.NET Core (backend) ha resultado muy positiva:

- **Mayor claridad en la arquitectura:** cada parte tiene su responsabilidad bien definida.
- **Escalabilidad:** se puede modificar o mejorar el frontend sin tocar el backend, y viceversa.
- **Reutilización de la API:** otros sistemas o aplicaciones móviles podrán consumir la misma API sin modificaciones.
- **Mantenibilidad:** el código está más ordenado, modular y fácil de evolucionar con el tiempo.
- **Despliegue flexible:** cada parte puede actualizarse, reiniciarse o cambiarse de servidor por separado.

Uso de tecnologías modernas y actuales

El proyecto me ha permitido trabajar con herramientas y estándares actuales:

- **React + Vite** para un frontend rápido y modular.
- **ASP.NET Core 8** para construir un backend rápido, robusto y escalable.
- **Entity Framework Core** para automatizar el acceso a la base de datos.
- **MySQL** como sistema accesible y ampliamente utilizado.
- **React Router con rutas protegidas**, permitiendo control de roles (usuario, trabajador, admin).
- **Context API con AuthProvider** para gestionar autenticación a nivel global.
- **PayPalProvider** para integrar pagos de forma profesional.

Esto convierte el proyecto en una base sólida para funcionalidades futuras o ampliaciones.

Seguridad y control de acceso correctamente implementados

El taller incorpora varias capas de control:

- **CORS configurado correctamente** para permitir solo acceso desde el dominio/IP autorizada.
- **Rutas protegidas** para impedir acceso a usuarios no autenticados.
- **Rutas específicas por rol** (admin, trabajador), aumentando la seguridad interna.
- **Validaciones en backend** antes de realizar operaciones CRUD.

Gracias a esto, los datos del taller se mantienen protegidos y solo se muestran a quien corresponde.