

**Tema**

EXAMEN DE LA AEROLINA VUELO CON DOTNET

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU. CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

02/06/2025

Tabla de contenido

[PROYECTO EN .NET CON RESTFUL 5](#_Toc199754973)

[SERVICIO EN RESTFULL 5](#_Toc199754974)

[CONFIGURACIÓN DEL PROYECTO 5](#_Toc199754975)

[CREAR LOS MODELOS 6](#_Toc199754976)

[CREAR EL CONTEXTO DE BASE DE DATOS 7](#_Toc199754977)

[CONFIGURAR EL CONTEXTO EN PROGRAM.CS 8](#_Toc199754978)

[CREAR LOS CONTROLADORES REST 9](#_Toc199754979)

[EVIDENCIA DE LA EJECUCIÓN DEL SERVICIO 15](#_Toc199754980)

[CLIENTE CONSOLA 17](#_Toc199754981)

[CREAR EL PROYECTO DE CONSOLA 17](#_Toc199754982)

[INSTALACIÓN DE LAS DEPENDENCIAS 17](#_Toc199754983)

[CREACIÓN DE LAS CARPETAS CON SUS ARCHIVOS 17](#_Toc199754984)

[CONFIGURACIÓN DE CONTROLADORES 18](#_Toc199754985)

[CONFIGURACIÓN DE LOS MODELOS 20](#_Toc199754986)

[CLIENTE ESCRITORIO 25](#_Toc199754987)

[1️ CREAR EL PROYECTO 25](#_Toc199754988)

[2️ CREAR LAS CARPETAS DEL MVC 26](#_Toc199754989)

[3️ CREAR LAS CLASES DEL MODELO 26](#_Toc199754990)

[4️ CREAR LA CLASE APICONTROLLER 28](#_Toc199754991)

[5️ CREAR LAS VISTAS (FORMS) 31](#_Toc199754992)

[CLIENTE MÓVIL 36](#_Toc199754993)

[CREACIÓN DEL PROYECTO MÓVIL 36](#_Toc199754994)

[CREACIÓN CAPA MODELO 37](#_Toc199754995)

[EJECUCION 40](#_Toc199754996)

[CLIENTE WEB 42](#_Toc199754997)

[1️ CREAR EL PROYECTO 42](#_Toc199754998)

[2️ CREAR LAS CARPETAS MVC PERSONALIZADAS 42](#_Toc199754999)

[3️ MODELOS (MODELS) 43](#_Toc199755000)

[4 CONTROLADORES 45](#_Toc199755001)

[5 VISTAS 48](#_Toc199755002)

[CONCLUSIONES 48](#_Toc199755003)

[RECOMENDACIONES 48](#_Toc199755004)

[BIBLIOGRAFÍA 48](#_Toc199755005)

**ÍNDICE DE IMÁGENES**

[Figura 1. Creación del proyecto. 6](#_Toc199754939)

[Figura 2. Estructura del proyecto. 7](#_Toc199754940)

[Figura 3. Carpeta de la Data. 9](#_Toc199754941)

[Figura 4. Carpeta de los controladores. 11](#_Toc199754942)

[Figura 5. API Boletos. 16](#_Toc199754943)

[Figura 6. API Compras. 17](#_Toc199754944)

[Figura 7. API Usuarios. 17](#_Toc199754945)

[Figura 8. API Vuelos. 17](#_Toc199754946)

[Figura 9. Carpetas del cliente Consola. 18](#_Toc199754947)

[Figura 10. Menú de la aerolínea. 26](#_Toc199754948)

[Figura 11. Funcionalidad. 26](#_Toc199754949)

[Ilustración 12. Carpetas del Cliente Escritorio. 27](#_Toc199754950)

[Ilustración 13. Clases del modelo. 27](#_Toc199754951)

[Figura 14. Controlador. 29](#_Toc199754952)

[Figura 15. Login. 32](#_Toc199754953)

[Figura 16. Menú principal. 33](#_Toc199754954)

[Figura 17. Iniciar sesión. 33](#_Toc199754955)

[Figura 18. Crear Cuenta. 34](#_Toc199754956)

[Figura 19. Buscar Vuelos. 34](#_Toc199754957)

[Figura 20. Comprar Boletos. 35](#_Toc199754958)

[Figura 21. Mostrar Boletos. 35](#_Toc199754959)

[Figura 22. Mostrar Vuelos. 36](#_Toc199754960)

Figura 23. Vistas.

[Figura 24. Creación de proyecto. 37](#_Toc199754962)

[Figura 25. Estructura. 37](#_Toc199754963)

[Modelo 26. Modelo. 38](#_Toc199754964)

[Figura 27. Creación clases de servicio 39](#_Toc199754965)

[Figura 28. Creación clases de controlador 40](#_Toc199754966)

[Figura 29. Ejecución Pantalla Login 42](#_Toc199754967)

[Figura 30. Menú Principal. 42](#_Toc199754968)

[Figura 31. Estructura. 44](#_Toc199754969)

[Figura 32. modelos. 44](#_Toc199754970)

[Figura 33. Controladores. 46](#_Toc199754971)

[Figura 34. Vistas. 49](#_Toc199754972)

# PROYECTO EN .NET CON RESTFUL

## SERVICIO EN RESTFULL

Se crea primero el proyecto, se usa la plantilla de **API ASP.NET Core** (Plantilla: "ASP.NET Core WebAPI"), Asigna un nombre: ViajecitosAPI.

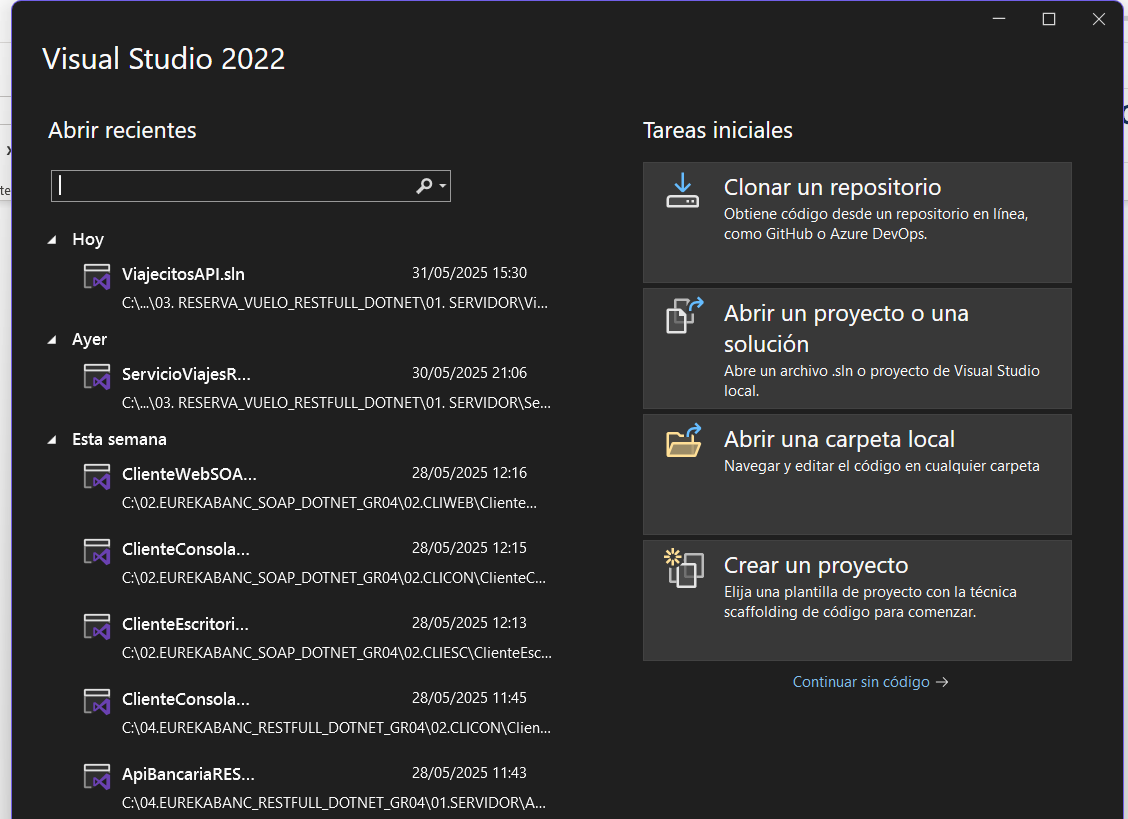


Figura 1. Creación del proyecto.

### CONFIGURACIÓN DEL PROYECTO

1. Instala los paquetes NuGet necesarios:
   * Microsoft.EntityFrameworkCore
   * Microsoft.EntityFrameworkCore. SqlServer
   * Microsoft.EntityFrameworkCore. Tools
2. En appsettings. json, agrega la cadena de conexión:

Seguido de estos pasos, tenemos la estructura del proyecto de la siguiente manera, donde se divide en modelo, controlador y Data.

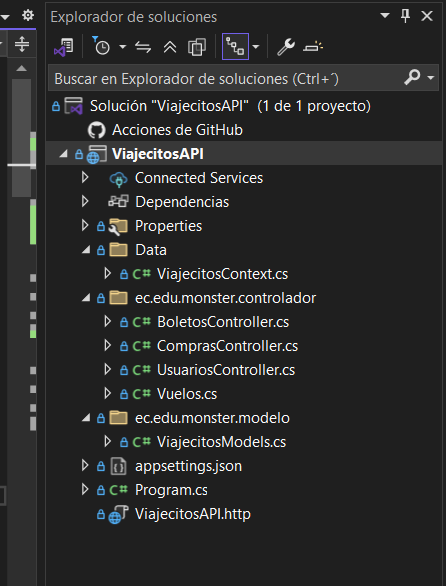


Figura 2. Estructura del proyecto.

### CREAR LOS MODELOS

En la carpeta de ec.edu.monster.modelo se creara dentro un archivo ViajecitosModels.cs.

Que contendrá el siguiente código. Esta clase sirve como modelo de datos para la aplicación ViajecitosAPI, definiendo las entidades principales del sistema: Vuelo representa los vuelos disponibles con información de origen, destino, precio, hora de salida y asientos; Usuario representa a los usuarios con datos personales y credenciales; Boleto almacena los detalles de cada boleto comprado, como el usuario, el vuelo y el número de asiento; y Compra registra las transacciones realizadas por los usuarios, incluyendo el monto total y la fecha de compra. Estas clases permiten estructurar la información en la base de datos, realizar operaciones CRUD y comunicar datos en la API RESTful de manera coherente.

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ViajecitosAPI.ec.edu.monster.modelo

{

public class Vuelo

{

[Key]

public int id\_vuelo { get; set; }

public string ciudad\_origen { get; set; }

public string ciudad\_destino { get; set; }

public decimal valor { get; set; }

public DateTime hora\_salida { get; set; }

public int asientos\_disponibles { get; set; }

}

public class Usuario

{

[Key]

public int id\_usuario { get; set; }

public string nombre\_usuario { get; set; }

public string apellido\_usuario { get; set; }

public string cedula { get; set; }

public string celular { get; set; }

public string email { get; set; }

public string contrasena { get; set; }

}

public class Boleto

{

[Key]

public int id\_boleto { get; set; }

[ForeignKey("Usuario")]

public int id\_usuario { get; set; }

[ForeignKey("Vuelo")]

public int id\_vuelo { get; set; }

public DateTime fecha\_compra { get; set; }

public string numero\_asiento { get; set; }

}

public class Compra

{

[Key]

public int id\_compra { get; set; }

[ForeignKey("Usuario")]

public int id\_usuario { get; set; }

[ForeignKey("Boleto")]

public int id\_boleto { get; set; }

public DateTime fecha\_compra { get; set; }

public decimal monto\_total { get; set; }

}

}

### CREAR EL CONTEXTO DE BASE DE DATOS

La clase ViajecitosContext es el contexto de base de datos para la aplicación ViajecitosAPI, y sirve como el puente entre la base de datos y el código en C#. Se crea dentro de la carpeta Data y hereda de DbContext de Entity Framework Core, lo que permite mapear las clases del modelo (Vuelo, Usuario, Boleto y Compra) a tablas en la base de datos. Esta clase define cuatro propiedades DbSet (una por cada entidad del modelo), que representan las tablas y permiten realizar operaciones como consultas, inserciones, actualizaciones y eliminaciones en la base de datos. Además, en el constructor recibe opciones de configuración (como la cadena de conexión) que permiten establecer cómo conectarse a la base de datos. En resumen, ViajecitosContext es el motor que conecta el código con la base de datos y permite trabajar con las entidades del proyecto.

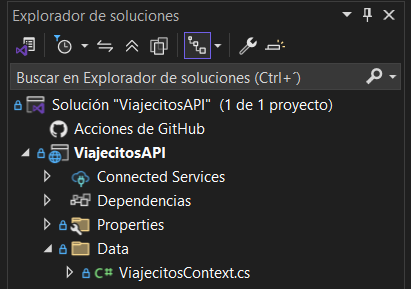


Figura 3. Carpeta de la Data.

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.ec.edu.monster.modelo;

namespace ViajecitosAPI.Data

{

public class ViajecitosContext : DbContext

{

public ViajecitosContext(DbContextOptions<ViajecitosContext> options) : base(options) { }

public DbSet<Vuelo> Vuelos { get; set; }

public DbSet<Usuario> Usuarios { get; set; }

public DbSet<Boleto> Boletos { get; set; }

public DbSet<Compra> Compras { get; set; }

}

}

### CONFIGURAR EL CONTEXTO EN PROGRAM.CS

En esta parte del código se configura el contexto de la base de datos en Program.cs, que es el archivo de inicio del proyecto ViajecitosAPI. Se utiliza inyección de dependencias para registrar el servicio ViajecitosContext dentro del contenedor de servicios de la aplicación, permitiendo que se pueda utilizar el contexto en cualquier parte del proyecto. Esto se hace con el método AddDbContext, donde se indica que se usará SQL Server y se obtiene la cadena de conexión llamada ViajecitosConnection desde el archivo de configuración (appsettings.json). Además, se configuran los servicios de controladores, la documentación con Swagger, el HTTPS y el mapeo de rutas de la API. En resumen, esta configuración permite que la aplicación esté conectada a la base de datos y tenga disponibles los servicios para exponer los endpoints REST.

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.ec.edu.monster.controlador;

using ViajecitosAPI;

using ViajecitosAPI.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddDbContext<ViajecitosContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("ViajecitosConnection")));

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

### CREAR LOS CONTROLADORES REST

Este código crea un controlador REST llamado VuelosController, que permite exponer los endpoints de la API para manejar las operaciones CRUD (crear, leer, actualizar y eliminar) sobre la entidad Vuelo. Se utiliza el atributo [ApiController] para marcarlo como controlador de API y [Route("api/[controller]")] para definir la ruta base como /api/vuelos. El controlador tiene un constructor que recibe el contexto de base de datos ViajecitosContext mediante inyección de dependencias, permitiendo acceder a la base de datos. Se definen cinco métodos: GetVuelos devuelve la lista de todos los vuelos, GetVuelo busca un vuelo por su ID, PostVuelo crea un nuevo vuelo, PutVuelo actualiza un vuelo existente y DeleteVuelo elimina un vuelo por su ID. Cada método es asincrónico (async) y utiliza las herramientas de Entity Framework Core para interactuar con la base de datos. En resumen, este controlador permite a la API recibir y procesar solicitudes HTTP para la gestión de los vuelos.

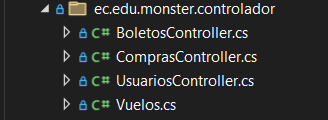


Figura 4. Carpeta de los controladores.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.Data;

using ViajecitosAPI.ec.edu.monster.modelo;

namespace ViajecitosAPI.ec.edu.monster.controlador

{

[ApiController]

[Route("api/[controller]")]

public class VuelosController : ControllerBase

{

private readonly ViajecitosContext \_context;

public VuelosController(ViajecitosContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Vuelo>>> GetVuelos()

{

return await \_context.Vuelos.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Vuelo>> GetVuelo(int id)

{

var vuelo = await \_context.Vuelos.FindAsync(id);

if (vuelo == null) return NotFound();

return vuelo;

}

[HttpPost]

public async Task<ActionResult<Vuelo>> PostVuelo(Vuelo vuelo)

{

\_context.Vuelos.Add(vuelo);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetVuelo), new { id = vuelo.id\_vuelo }, vuelo);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutVuelo(int id, Vuelo vuelo)

{

if (id != vuelo.id\_vuelo) return BadRequest();

\_context.Entry(vuelo).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteVuelo(int id)

{

var vuelo = await \_context.Vuelos.FindAsync(id);

if (vuelo == null) return NotFound();

\_context.Vuelos.Remove(vuelo);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

El controlador UsuariosController es una clase que gestiona las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para los usuarios de la aplicación. Se marca como un controlador de API ([ApiController]) y se expone bajo la ruta api/usuarios. En su constructor, recibe el contexto de base de datos ViajecitosContext por inyección de dependencias para acceder a la base de datos. Los métodos definidos son: GetUsuarios para obtener la lista de todos los usuarios, GetUsuario para obtener un usuario específico por su ID, PostUsuario para crear un nuevo usuario, PutUsuario para actualizar un usuario existente y DeleteUsuario para eliminar un usuario. Cada método es asíncrono (async) y utiliza Entity Framework Core para interactuar con la base de datos, facilitando así la gestión de datos de los usuarios mediante peticiones HTTP.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.Data;

using ViajecitosAPI.ec.edu.monster.modelo;

namespace ViajecitosAPI.ec.edu.monster.controlador

{

[ApiController]

[Route("api/[controller]")]

public class UsuariosController : ControllerBase

{

private readonly ViajecitosContext \_context;

public UsuariosController(ViajecitosContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Usuario>>> GetUsuarios()

{

return await \_context.Usuarios.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Usuario>> GetUsuario(int id)

{

var usuario = await \_context.Usuarios.FindAsync(id);

if (usuario == null) return NotFound();

return usuario;

}

[HttpPost]

public async Task<ActionResult<Usuario>> PostUsuario(Usuario usuario)

{

\_context.Usuarios.Add(usuario);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetUsuario), new { id = usuario.id\_usuario }, usuario);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutUsuario(int id, Usuario usuario)

{

if (id != usuario.id\_usuario) return BadRequest();

\_context.Entry(usuario).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteUsuario(int id)

{

var usuario = await \_context.Usuarios.FindAsync(id);

if (usuario == null) return NotFound();

\_context.Usuarios.Remove(usuario);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

El controlador BoletosController permite gestionar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para los boletos de la aplicación. Se configura como un controlador API ([ApiController]) y expone sus rutas bajo api/boletos. Usa el contexto de base de datos ViajecitosContext para acceder a la información de los boletos en la base de datos. Define métodos para obtener todos los boletos (GetBoletos), obtener un boleto específico por su ID (GetBoleto), crear un nuevo boleto (PostBoleto), actualizar un boleto existente (PutBoleto) y eliminar un boleto (DeleteBoleto). Cada método es asíncrono (async) y usa Entity Framework Core para manipular los datos, facilitando así la gestión de los boletos a través de solicitudes HTTP en la API.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.Data;

using ViajecitosAPI.ec.edu.monster.modelo;

namespace ViajecitosAPI.ec.edu.monster.controlador

{

[ApiController]

[Route("api/[controller]")]

public class BoletosController : ControllerBase

{

private readonly ViajecitosContext \_context;

public BoletosController(ViajecitosContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Boleto>>> GetBoletos()

{

return await \_context.Boletos.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Boleto>> GetBoleto(int id)

{

var boleto = await \_context.Boletos.FindAsync(id);

if (boleto == null) return NotFound();

return boleto;

}

[HttpPost]

public async Task<ActionResult<Boleto>> PostBoleto(Boleto boleto)

{

\_context.Boletos.Add(boleto);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetBoleto), new { id = boleto.id\_boleto }, boleto);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutBoleto(int id, Boleto boleto)

{

if (id != boleto.id\_boleto) return BadRequest();

\_context.Entry(boleto).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteBoleto(int id)

{

var boleto = await \_context.Boletos.FindAsync(id);

if (boleto == null) return NotFound();

\_context.Boletos.Remove(boleto);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

El ComprasController es un controlador REST que permite manejar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para las compras realizadas en la aplicación. Este controlador está configurado como un API ([ApiController]) y expone sus rutas bajo api/compras. Usa el contexto de base de datos ViajecitosContext para acceder y manipular la información de las compras. Tiene métodos para obtener la lista completa de compras (GetCompras), buscar una compra por su ID (GetCompra), registrar una nueva compra (PostCompra), actualizar una compra existente (PutCompra) y eliminar una compra (DeleteCompra). Los métodos son asíncronos (async) y utilizan Entity Framework Core para interactuar con la base de datos, asegurando la persistencia de datos y la funcionalidad de la API.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ViajecitosAPI.Data;

using ViajecitosAPI.ec.edu.monster.modelo;

namespace ViajecitosAPI.ec.edu.monster.controlador

{

[ApiController]

[Route("api/[controller]")]

public class ComprasController : ControllerBase

{

private readonly ViajecitosContext \_context;

public ComprasController(ViajecitosContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Compra>>> GetCompras()

{

return await \_context.Compras.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Compra>> GetCompra(int id)

{

var compra = await \_context.Compras.FindAsync(id);

if (compra == null) return NotFound();

return compra;

}

[HttpPost]

public async Task<ActionResult<Compra>> PostCompra(Compra compra)

{

\_context.Compras.Add(compra);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetCompra), new { id = compra.id\_compra }, compra);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutCompra(int id, Compra compra)

{

if (id != compra.id\_compra) return BadRequest();

\_context.Entry(compra).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteCompra(int id)

{

var compra = await \_context.Compras.FindAsync(id);

if (compra == null) return NotFound();

\_context.Compras.Remove(compra);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

En el archivo appsettings.json se configura la cadena de conexión a la base de datos SQL Server utilizada por la aplicación ViajecitosAPI. La propiedad "ViajecitosConnection" contiene la información necesaria para conectarse: el servidor (localhost), el nombre de la base de datos (viajecitos\_sa), el usuario (sa) y la contraseña (Adali020). También incluye TrustServerCertificate=True para aceptar certificados no seguros en el entorno de desarrollo. Además, el archivo configura el nivel de logging (registro de eventos) de la aplicación, estableciendo el nivel predeterminado en Information y limitando los mensajes de Microsoft a Warning. Finalmente, "AllowedHosts": "\*" indica que la API acepta solicitudes de cualquier host. En resumen, este archivo centraliza las configuraciones clave del proyecto, como la conexión a la base de datos y los ajustes de log.

{

"ConnectionStrings": {

"ViajecitosConnection": "Server=DESKTOP-GM48FCK;Database=viajecitos\_sa;User Id=sa;Password=Adali020;TrustServerCertificate=True;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

### EVIDENCIA DE LA EJECUCIÓN DEL SERVICIO

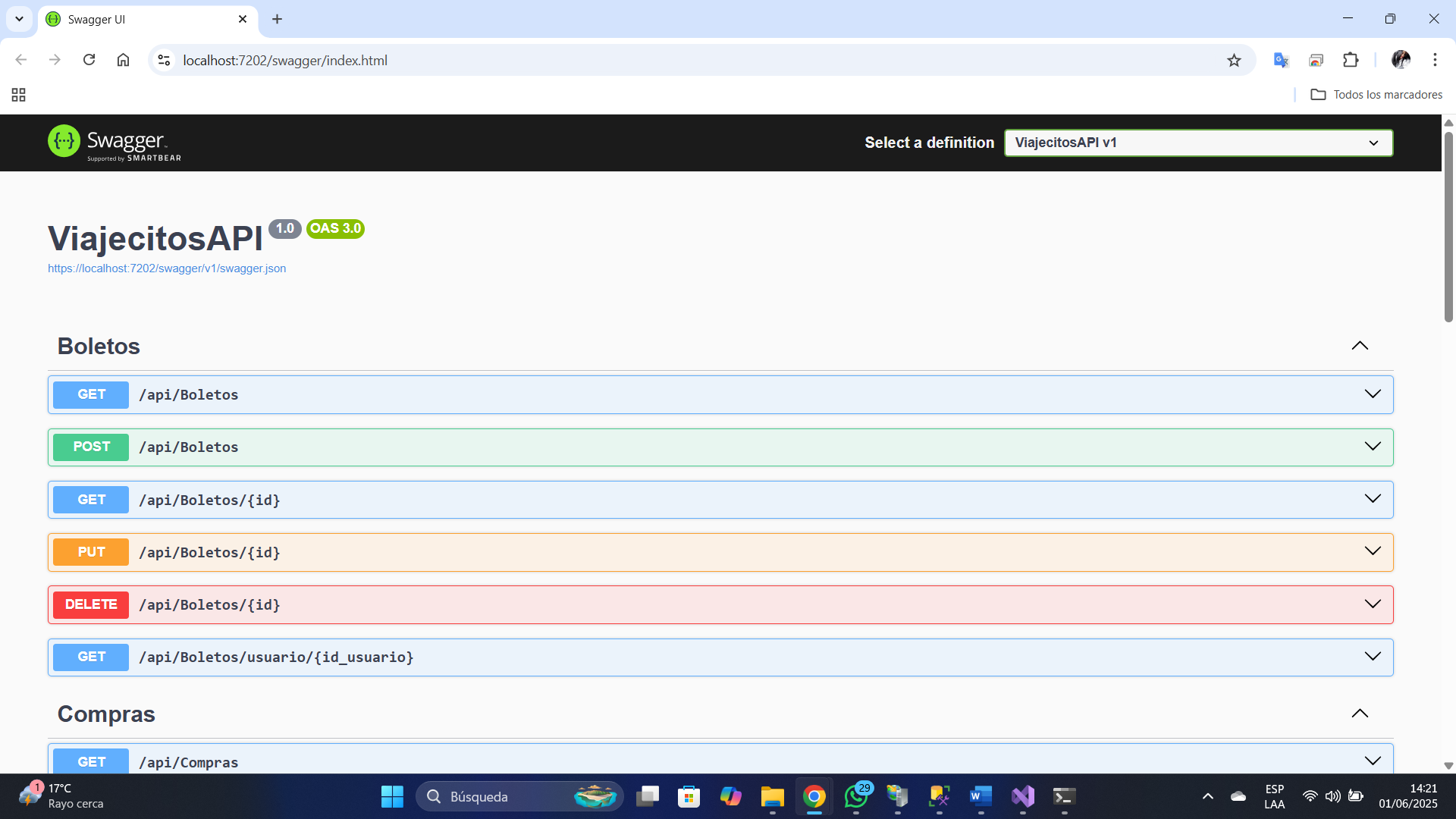


Figura 5. API Boletos.

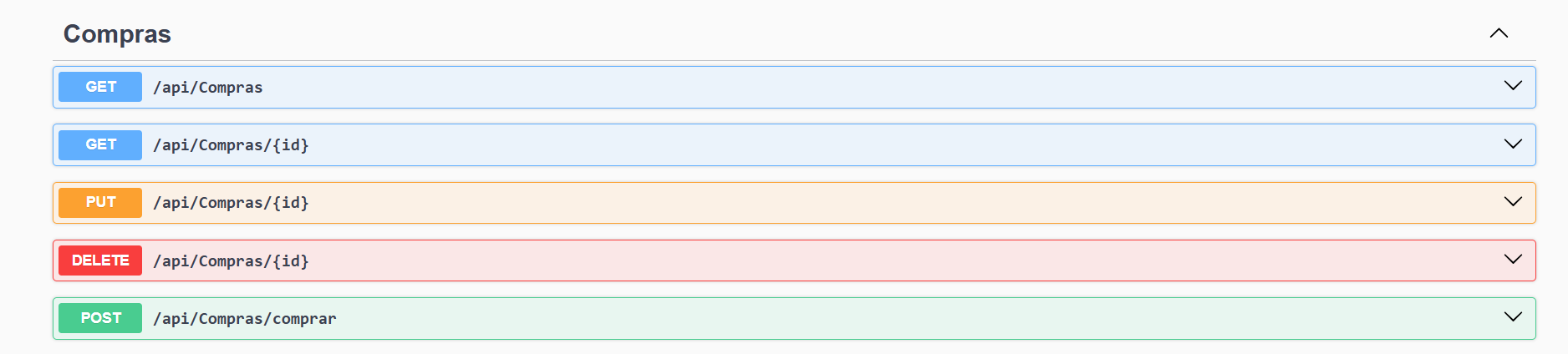


Figura 6. API Compras.

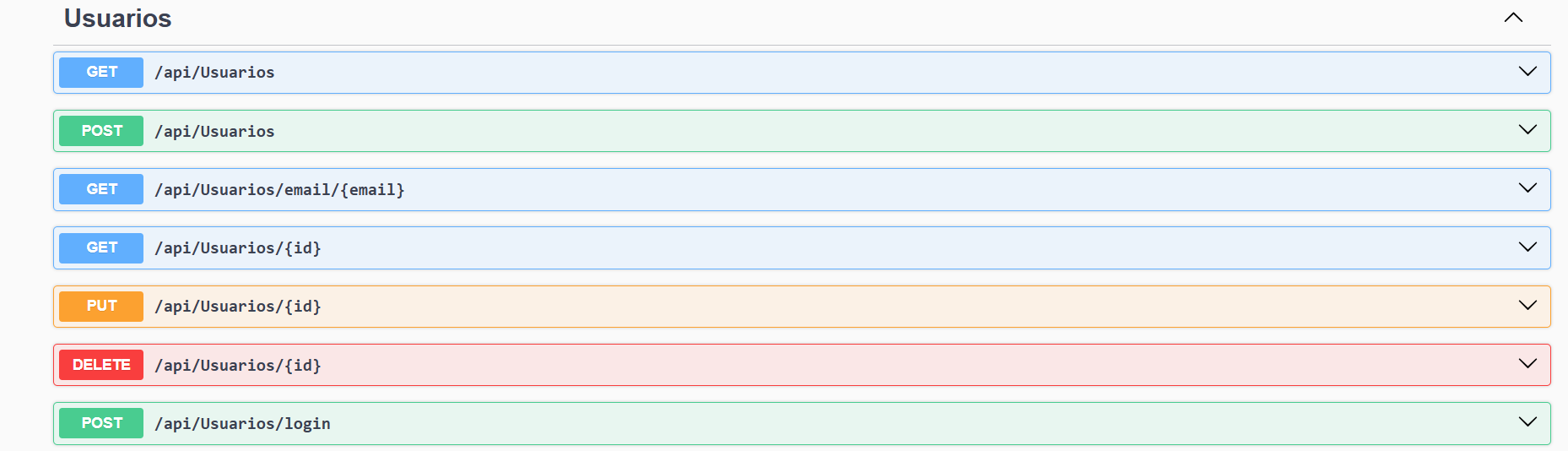


Figura 7. API Usuarios.

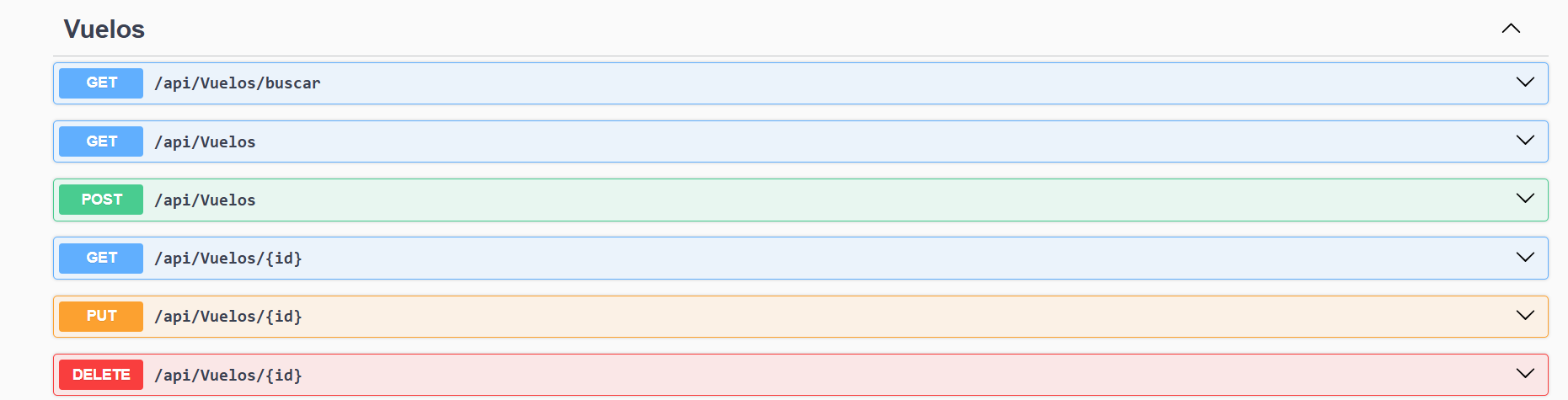


Figura 8. API Vuelos.

## CLIENTE CONSOLA

### CREAR EL PROYECTO DE CONSOLA

1️Abre **Visual Studio.**  
2️Crea un nuevo proyecto:

* Selecciona **Aplicación de Consola (.NET Core o .NET 6/7/8).**
* Nombre del proyecto: ClienteConsolaVuelos.

3️Configura la versión de .NET (usa la misma que tu API, por ejemplo .NET 6/7/8).

### INSTALACIÓN DE LAS DEPENDENCIAS

En la Consola de Administrador de Paquetes o en el Administrador de Paquetes NuGet:

Instala:

Install-Package Newtonsoft.Json

Install-Package System.Net.Http.Json

### CREACIÓN DE LAS CARPETAS CON SUS ARCHIVOS

Se procede a crear las carpetas con los respectivos nombres ec.edu.monster.modelo y los diferentes archivos

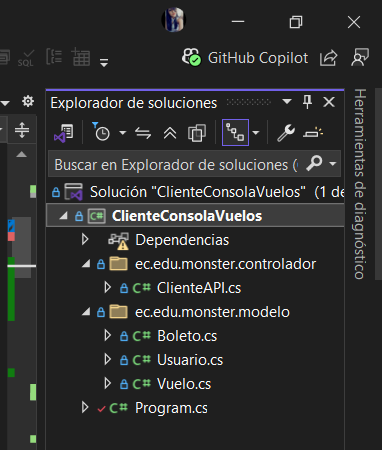


Figura 9. Carpetas del cliente Consola.

### CONFIGURACIÓN DE CONTROLADORES

Este código implementa una clase llamada ClienteAPI que actúa como un cliente REST para consumir un servicio web expuesto en la dirección http://192.168.10.247/. Utiliza HttpClient para realizar solicitudes HTTP al servidor, configurando el encabezado para aceptar respuestas en formato JSON. La clase contiene varios métodos asíncronos: BuscarVuelos busca vuelos según el origen y destino proporcionados; ComprarBoletos permite comprar boletos para un vuelo específico mediante una solicitud POST; MostrarBoletosUsuario recupera los boletos de un usuario específico; MostrarTodosVuelos devuelve la lista de todos los vuelos; RegistrarUsuario registra un nuevo usuario enviando sus datos en formato JSON; Login valida las credenciales de un usuario enviadas en el cuerpo de la solicitud; y ObtenerIdUsuarioPorEmail obtiene el ID de un usuario a partir de su correo electrónico. Todos estos métodos deserializan las respuestas JSON en objetos del modelo correspondiente (Vuelo, Boleto, Usuario).

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using ClienteConsolaVuelos.ec.edu.monster.modelo;

using Newtonsoft.Json;

namespace ClienteConsolaVuelos.ec.edu.monster.controlador

{

public class ClienteAPI

{

private readonly HttpClient client;

public ClienteAPI()

{

client = new HttpClient();

client.BaseAddress = new Uri("http://192.168.10.247/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public async Task<List<Vuelo>> BuscarVuelos(string origen, string destino)

{

var response = await client.GetAsync($"api/Vuelos/buscar?origen={origen}&destino={destino}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

public async Task<string> ComprarBoletos(int idUsuario, int idVuelo, int numeroAsientos)

{

var response = await client.PostAsync($"api/Compras/comprar?idUsuario={idUsuario}&idVuelo={idVuelo}&numeroAsientos={numeroAsientos}", null);

return await response.Content.ReadAsStringAsync();

}

public async Task<List<Boleto>> MostrarBoletosUsuario(int idUsuario)

{

var response = await client.GetAsync($"api/Boletos/usuario/{idUsuario}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Boleto>>(json);

}

public async Task<List<Vuelo>> MostrarTodosVuelos()

{

var response = await client.GetAsync("api/Vuelos");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

public async Task<Usuario> RegistrarUsuario(Usuario usuario)

{

var jsonContent = new StringContent(JsonConvert.SerializeObject(usuario), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

public async Task<Usuario> Login(string email, string contrasena)

{

var datos = new { email = email, contrasena = contrasena };

var jsonContent = new StringContent(JsonConvert.SerializeObject(datos), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios/login", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

public async Task<int> ObtenerIdUsuarioPorEmail(string email)

{

var response = await client.GetAsync($"api/Usuarios/email/{email}");

if (!response.IsSuccessStatusCode) return -1;

var idStr = await response.Content.ReadAsStringAsync();

return int.Parse(idStr);

}

}

}

### CONFIGURACIÓN DE LOS MODELOS

Este código define una clase llamada Boleto en el espacio de nombres ClienteConsolaVuelos.ec.edu.monster.modelo, que representa un modelo de datos para almacenar información de un boleto de vuelo. La clase tiene propiedades públicas con métodos get y set para encapsular los datos: id\_boleto es el identificador del boleto, id\_usuario identifica al usuario que compró el boleto, id\_vuelo corresponde al vuelo asociado, fecha\_compra almacena la fecha y hora de la compra, numero\_asiento indica el asiento asignado, y valor almacena el precio del boleto (opcional si la API lo devuelve). Esta clase sirve como estructura para mapear objetos JSON recibidos del servicio web en objetos de C# o para construir objetos que se enviarán a la API.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteConsolaVuelos.ec.edu.monster.modelo

{

public class Boleto

{

public int id\_boleto { get; set; }

public int id\_usuario { get; set; }

public int id\_vuelo { get; set; }

public DateTime fecha\_compra { get; set; }

public string numero\_asiento { get; set; }

public decimal valor { get; set; } // Si la API lo devuelve

}

}

Este código define una clase llamada Usuario en el espacio de nombres ClienteConsolaVuelos.ec.edu.monster.modelo, que sirve como modelo de datos para representar a un usuario dentro del sistema. La clase contiene propiedades públicas con métodos get y set para encapsular los datos del usuario: id\_usuario es el identificador único del usuario (marcado con el atributo [Key] para indicar que es la clave primaria), nombre\_usuario y apellido\_usuario almacenan el nombre y apellido del usuario, cedula y celular guardan la identificación y el número de teléfono, email contiene la dirección de correo electrónico, y contrasena guarda la contraseña de acceso

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteConsolaVuelos.ec.edu.monster.modelo

{

public class Usuario

{

[Key]

public int id\_usuario { get; set; }

public string nombre\_usuario { get; set; }

public string apellido\_usuario { get; set; }

public string cedula { get; set; }

public string celular { get; set; }

public string email { get; set; }

public string contrasena { get; set; }

}

}

Este código define la clase Vuelo dentro del espacio de nombres ClienteConsolaVuelos.ec.edu.monster.modelo, la cual sirve como modelo de datos para representar un vuelo en el sistema. La clase contiene propiedades públicas con métodos get y set que encapsulan la información de cada vuelo: id\_vuelo es el identificador único del vuelo, ciudad\_origen y ciudad\_destino indican las ciudades de salida y llegada del vuelo, valor almacena el precio del vuelo, hora\_salida registra la fecha y hora programada de salida, y asientos\_disponibles indica cuántos asientos quedan libres en el vuelo. Esta clase permite estructurar los datos de un vuelo que se reciben o se envían al servicio API, facilitando la gestión de vuelos en la aplicación.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteConsolaVuelos.ec.edu.monster.modelo

{

public class Vuelo

{

public int id\_vuelo { get; set; }

public string ciudad\_origen { get; set; }

public string ciudad\_destino { get; set; }

public decimal valor { get; set; }

public DateTime hora\_salida { get; set; }

public int asientos\_disponibles { get; set; }

}

}

Este código implementa una aplicación de consola que permite a los usuarios iniciar sesión, registrarse, buscar vuelos, comprar boletos, mostrar sus boletos, ver todos los vuelos y obtener el ID de usuario por correo electrónico, utilizando un menú interactivo y consumiendo un servicio REST a través de la clase ClienteAPI.

using ClienteConsolaVuelos.ec.edu.monster.controlador;

using ClienteConsolaVuelos.ec.edu.monster.modelo;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace ClienteConsolaVuelos

{

internal class Program

{

static Usuario usuarioLogueado = null;

static async Task Main(string[] args)

{

// Login inicial quemado

if (!LoginInicial())

{

Console.WriteLine("Acceso denegado. El programa se cerrará.");

return;

}

var api = new ClienteAPI();

while (true)

{

if (usuarioLogueado == null)

{

Console.WriteLine("\n=== Inicio de Sesion Aerolinea ===");

Console.WriteLine("1. Iniciar sesión");

Console.WriteLine("2. Registrarse");

Console.WriteLine("0. Salir");

Console.Write("Opción: ");

var opcion = Console.ReadLine();

switch (opcion)

{

case "1":

await Login(api);

break;

case "2":

await Registro(api);

Console.WriteLine("\nPor favor, inicia sesión para continuar.");

await Login(api);

break;

case "0":

return;

default:

Console.WriteLine("Opción inválida.");

break;

}

}

else

{

Console.WriteLine("\n=== Menú Principal ===");

Console.WriteLine("3. Buscar vuelos");

Console.WriteLine("4. Comprar boletos");

Console.WriteLine("5. Mostrar mis boletos");

Console.WriteLine("6. Mostrar todos los vuelos");

Console.WriteLine("7. Obtener ID de usuario por email");

Console.WriteLine("0. Salir");

Console.Write("Opción: ");

var opcion = Console.ReadLine();

switch (opcion)

{

case "3":

await BuscarVuelos(api);

break;

case "4":

await ComprarBoletos(api);

break;

case "5":

await MostrarBoletos(api);

break;

case "6":

await MostrarTodosVuelos(api);

break;

case "7":

await ObtenerIdPorEmail(api);

break;

case "0":

return;

default:

Console.WriteLine("Opción inválida.");

break;

}

}

}

}

static bool LoginInicial()

{

Console.WriteLine("\n=== Login de Seguridad ===");

Console.Write("Usuario: ");

string usuario = Console.ReadLine();

Console.Write("Contraseña: ");

string contrasena = Console.ReadLine();

if (usuario.ToUpper() == "MONSTER" && contrasena == "MONSTER9")

{

Console.WriteLine("¡Login exitoso!");

return true;

}

else

{

Console.WriteLine("Credenciales incorrectas.");

return false;

}

}

static async Task Login(ClienteAPI api)

{

Console.Write("Email: ");

var email = Console.ReadLine();

Console.Write("Contraseña: ");

var pass = Console.ReadLine();

var user = await api.Login(email, pass);

if (user != null)

{

usuarioLogueado = user;

Console.WriteLine($"Bienvenido {user.nombre\_usuario} {user.apellido\_usuario}");

}

else

{

Console.WriteLine("Login fallido.");

}

}

static async Task Registro(ClienteAPI api)

{

var u = new Usuario();

Console.Write("Nombre: "); u.nombre\_usuario = Console.ReadLine();

Console.Write("Apellido: "); u.apellido\_usuario = Console.ReadLine();

Console.Write("Cédula: "); u.cedula = Console.ReadLine();

Console.Write("Celular: "); u.celular = Console.ReadLine();

Console.Write("Email: "); u.email = Console.ReadLine();

Console.Write("Contraseña: "); u.contrasena = Console.ReadLine();

var result = await api.RegistrarUsuario(u);

if (result != null)

{

Console.WriteLine($"Usuario registrado correctamente:");

Console.WriteLine($"ID: {result.id\_usuario} | Nombre: {result.nombre\_usuario} {result.apellido\_usuario} | Email: {result.email}");

}

else

{

Console.WriteLine("Error al registrar usuario.");

}

}

static async Task BuscarVuelos(ClienteAPI api)

{

Console.Write("Origen: ");

var o = Console.ReadLine();

Console.Write("Destino: ");

var d = Console.ReadLine();

var vuelos = await api.BuscarVuelos(o, d);

foreach (var v in vuelos)

Console.WriteLine($"{v.id\_vuelo} | {v.ciudad\_origen}->{v.ciudad\_destino} | ${v.valor} | Asientos: {v.asientos\_disponibles}");

}

static async Task ComprarBoletos(ClienteAPI api)

{

if (usuarioLogueado == null) { Console.WriteLine("Debe iniciar sesión."); return; }

Console.Write("ID de Vuelo: ");

var idVuelo = int.Parse(Console.ReadLine());

Console.Write("Número de asientos: ");

var asientos = int.Parse(Console.ReadLine());

var result = await api.ComprarBoletos(usuarioLogueado.id\_usuario, idVuelo, asientos);

Console.WriteLine(result);

}

static async Task MostrarBoletos(ClienteAPI api)

{

if (usuarioLogueado == null) { Console.WriteLine("Debe iniciar sesión."); return; }

var boletos = await api.MostrarBoletosUsuario(usuarioLogueado.id\_usuario);

foreach (var b in boletos)

Console.WriteLine($"{b.id\_boleto} | Vuelo: {b.id\_vuelo} | Asiento: {b.numero\_asiento} | Fecha: {b.fecha\_compra}");

}

static async Task MostrarTodosVuelos(ClienteAPI api)

{

var vuelos = await api.MostrarTodosVuelos();

foreach (var v in vuelos)

Console.WriteLine($"{v.id\_vuelo} | {v.ciudad\_origen}->{v.ciudad\_destino} | ${v.valor} | Asientos: {v.asientos\_disponibles}");

}

static async Task ObtenerIdPorEmail(ClienteAPI api)

{

Console.Write("Email: ");

var email = Console.ReadLine();

var id = await api.ObtenerIdUsuarioPorEmail(email);

Console.WriteLine($"ID del usuario: {id}");

}

}

}

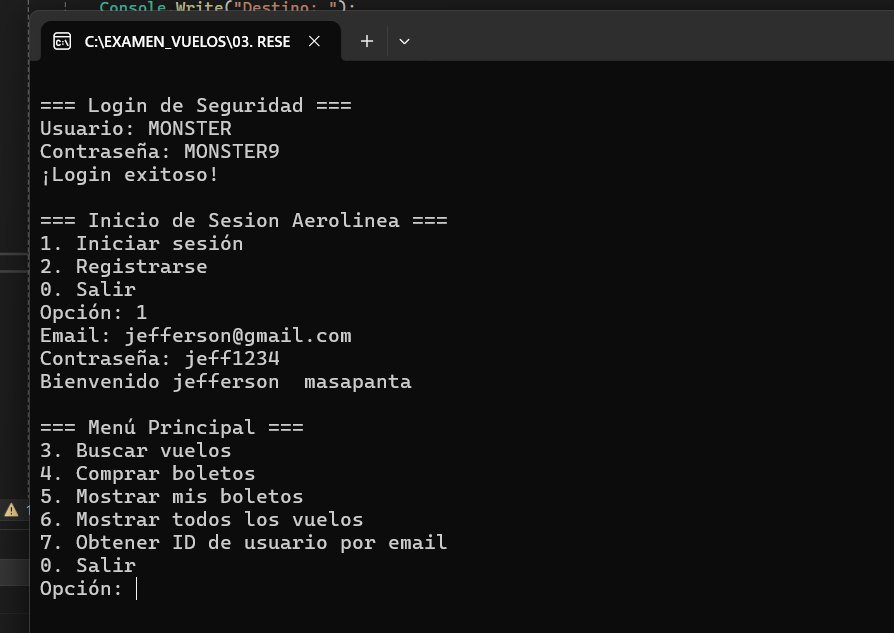


Figura 10. Menú de la aerolínea.

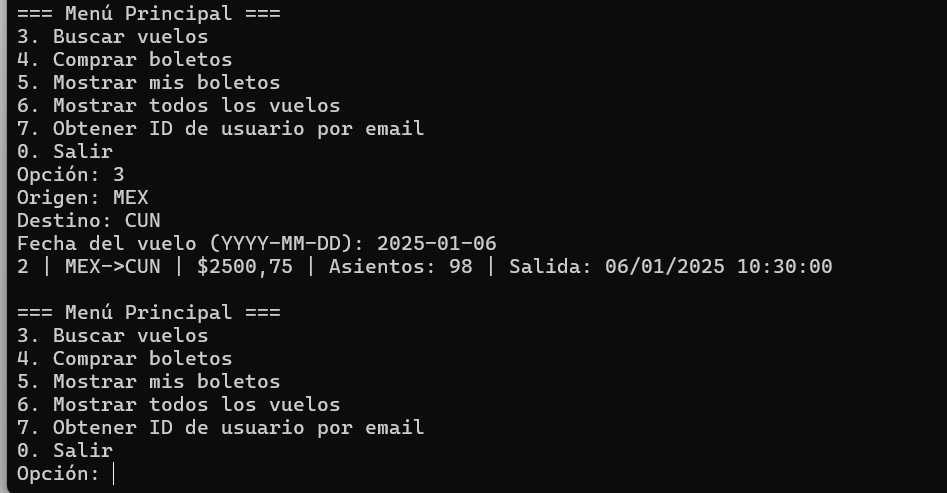


Figura 11. Funcionalidad.

## CLIENTE ESCRITORIO

### 1️ CREAR EL PROYECTO

* Abre **Visual Studio**.
* Selecciona **Nuevo proyecto** → **Aplicación de Windows Forms (.NET Framework)**.
* Ponle un nombre: ClienteEscritorioVuelos.
* Elige la carpeta de ubicación.
* Click en **Crear**.

### 2️ CREAR LAS CARPETAS DEL MVC

En el Explorador de soluciones:

1. Clic derecho sobre el proyecto → **Agregar** → **Nueva carpeta**.
2. Crea:
   * ec.edu.monster.modelo
   * ec.edu.monster.controlador
   * ec.edu.monster.vista

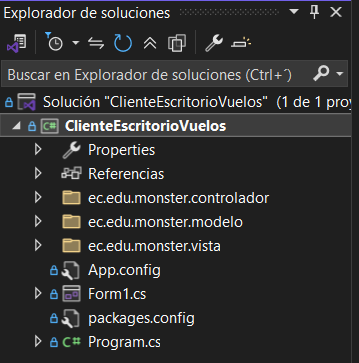


Ilustración 12. Carpetas del Cliente Escritorio.

### 3️ CREAR LAS CLASES DEL MODELO

En la carpeta ec.edu.monster.modelo:

* Crea las clases:
  + Usuario.cs
  + Vuelo.cs
  + Boleto.cs

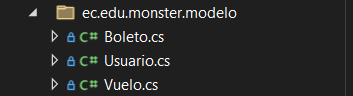


Ilustración 13. Clases del modelo.

Esta clase Boleto es un modelo de datos que representa la información de un boleto de avión en el sistema. Contiene las propiedades necesarias para almacenar el ID del boleto,

ID del usuario, ID del vuelo, la fecha de compra, el número de asiento y el valor del boleto (en caso de que la API devuelva este dato). Estas propiedades permiten trabajar con la información del boleto en el cliente de escritorio.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Boleto

{

public int id\_boleto { get; set; }

public int id\_usuario { get; set; }

public int id\_vuelo { get; set; }

public DateTime fecha\_compra { get; set; }

public string numero\_asiento { get; set; }

public decimal valor { get; set; } // Si la API lo devuelve

}

}

Esta clase Usuario es un modelo de datos que representa a un usuario registrado en el sistema. Contiene las propiedades necesarias para almacenar su información: ID de usuario (clave primaria), nombre, apellido, cédula, celular, correo electrónico y contraseña. Estas propiedades permiten identificar y gestionar la información del usuario en la aplicación.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Usuario

{

[Key]

public int id\_usuario { get; set; }

public string nombre\_usuario { get; set; }

public string apellido\_usuario { get; set; }

public string cedula { get; set; }

public string celular { get; set; }

public string email { get; set; }

public string contrasena { get; set; }

}

}

Esta clase Vuelo es un modelo de datos que representa la información de un vuelo en el sistema. Define propiedades como el ID del vuelo, la ciudad de origen, ciudad de destino, el valor del boleto, la hora de salida y la cantidad de asientos disponibles. Estas propiedades permiten trabajar con los datos de los vuelos dentro del cliente de escritorio para funciones como búsqueda, reserva y visualización.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Vuelo

{

public int id\_vuelo { get; set; }

public string ciudad\_origen { get; set; }

public string ciudad\_destino { get; set; }

public decimal valor { get; set; }

public DateTime hora\_salida { get; set; }

public int asientos\_disponibles { get; set; }

}

}

### 4️ CREAR LA CLASE APICONTROLLER

En ec.edu.monster.controlador:

* Crea ApiController.cs con la lógica de consumo API (ya la tienes implementada correctamente).
* Recuerda: Aquí va la conexión con tu API REST (BuscarVuelos, ComprarBoletos, etc.).



Figura 14. Controlador.

using ClienteEscritorioVuelos.ec.edu.monster.modelo;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.controlador

{

public class ApiController

{

private readonly HttpClient client;

public ApiController()

{

client = new HttpClient();

client.BaseAddress = new Uri("http://10.40.20.28/"); // Cambia según tu IP/URL

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

// 🔐 Login con API

public async Task<Usuario> LoginApi(string email, string contrasena)

{

var datos = new { email = email, contrasena = contrasena };

var jsonContent = new StringContent(JsonConvert.SerializeObject(datos), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios/login", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

// 📋 Registrar usuario

public async Task<Usuario> RegistrarUsuario(Usuario usuario)

{

var jsonContent = new StringContent(JsonConvert.SerializeObject(usuario), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

// ✈️ Buscar vuelos

public async Task<List<Vuelo>> BuscarVuelos(string origen, string destino)

{

var response = await client.GetAsync($"api/Vuelos/buscar?origen={origen}&destino={destino}");

if (!response.IsSuccessStatusCode) return new List<Vuelo>();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

// 🛒 Comprar boletos

public async Task<string> ComprarBoletos(int idUsuario, int idVuelo, int numeroAsientos)

{

var response = await client.PostAsync($"api/Compras/comprar?idUsuario={idUsuario}&idVuelo={idVuelo}&numeroAsientos={numeroAsientos}", null);

return await response.Content.ReadAsStringAsync();

}

// 🎟️ Mostrar boletos por usuario

public async Task<List<Boleto>> MostrarBoletosUsuario(int idUsuario)

{

var response = await client.GetAsync($"api/Boletos/usuario/{idUsuario}");

if (!response.IsSuccessStatusCode) return new List<Boleto>();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Boleto>>(json);

}

// ✈️ Mostrar todos los vuelos

public async Task<List<Vuelo>> MostrarTodosVuelos()

{

var response = await client.GetAsync("api/Vuelos");

if (!response.IsSuccessStatusCode) return new List<Vuelo>();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

// 🔍 Obtener ID usuario por email

public async Task<int> ObtenerIdUsuarioPorEmail(string email)

{

var response = await client.GetAsync($"api/Usuarios/email/{email}");

if (!response.IsSuccessStatusCode) return -1;

var idStr = await response.Content.ReadAsStringAsync();

return int.Parse(idStr);

}

}

}

### 5️ CREAR LAS VISTAS (FORMS)

En ec.edu.monster.vista:

* Crea los Forms para cada funcionalidad:
  + LoginForm.cs

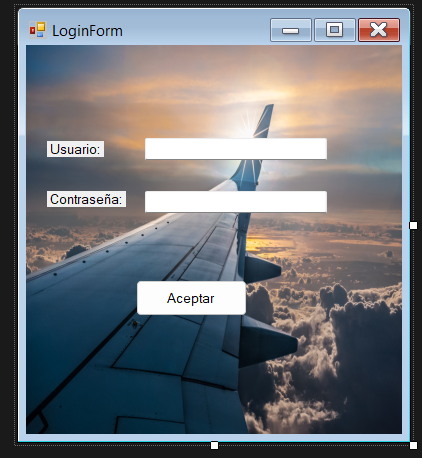


Figura 15. Login.

* + MenuForm.cs



Figura 16. Menú principal.

* + IniciarSesionForm.cs

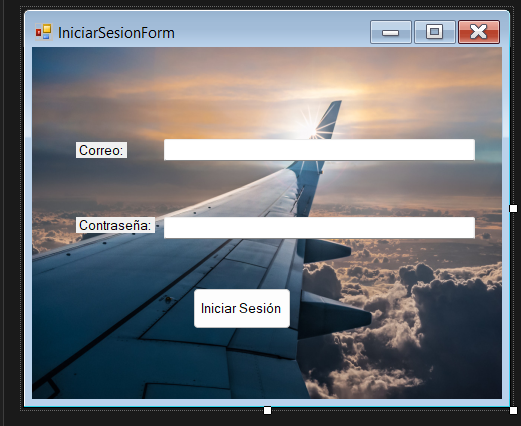


Figura 17. Iniciar sesión.

* + CrearCuentaForm.cs

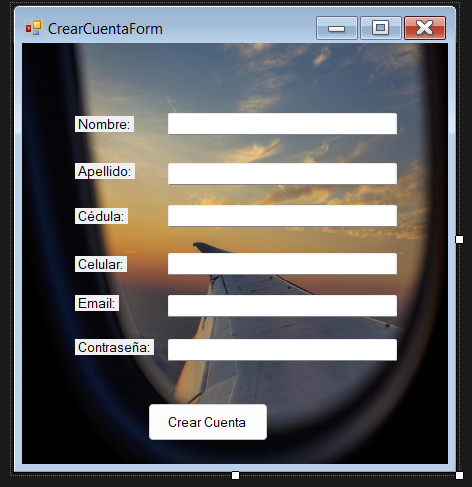


Figura 18. Crear Cuenta.

* + BuscarVuelosForm.cs

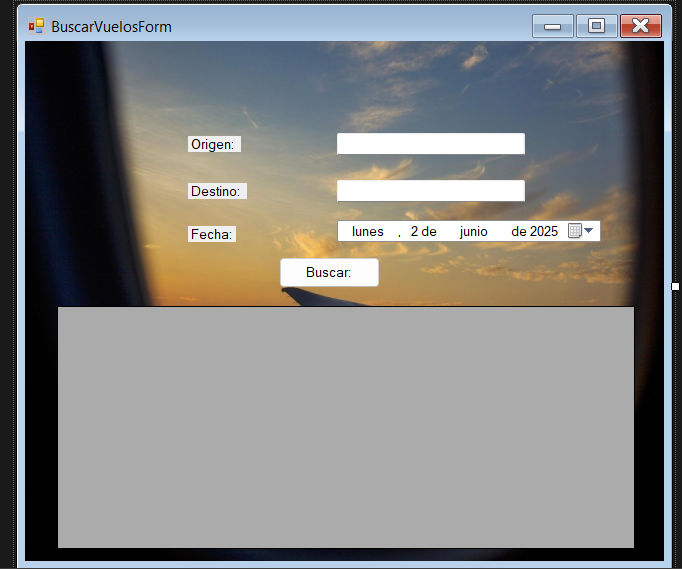


Figura 19. Buscar Vuelos.

* + ComprarBoletosForm.cs

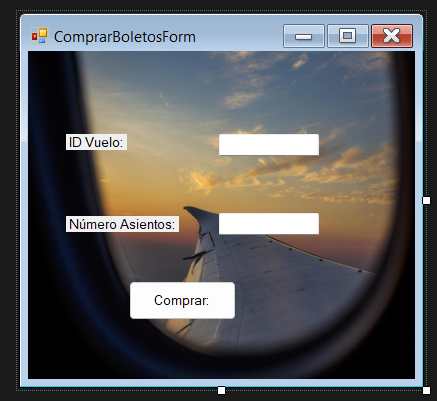


Figura 20. Comprar Boletos.

* + MostrarBoletosForm.cs

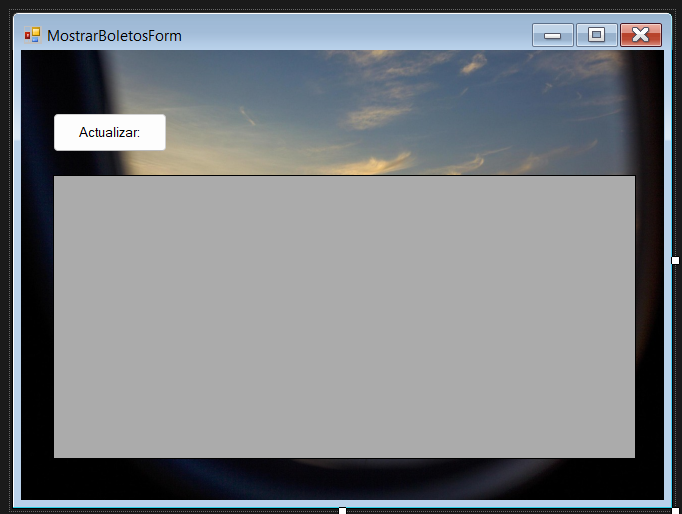


Figura 21. Mostrar Boletos.

* + MostrarVuelosForm.cs

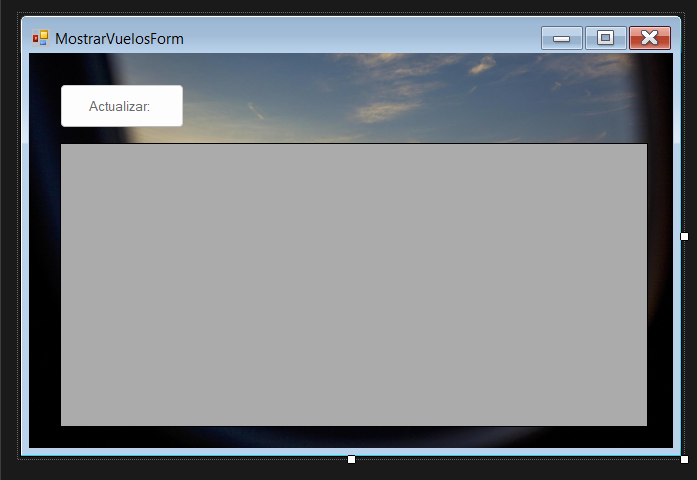


Figura 22. Mostrar Vuelos.

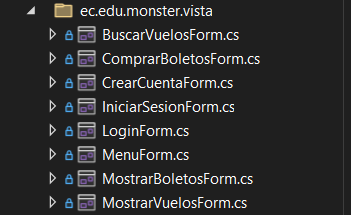
Cada Form tiene su diseño con controles (Labels, TextBox, Buttons, DataGridView según el caso).

Figura 23. Vistas.

## CLIENTE MÓVIL

### CREACIÓN DEL PROYECTO MÓVIL

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 24. Creación de proyecto.

Texto

El contenido generado por IA puede ser incorrecto.

Figura 25. Estructura.

### CREACIÓN CAPA MODELO

**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

Modelo 26. Modelo.

package ec.edu.monster.model  
  
import com.google.gson.annotations.SerializedName  
import java.util.Date  
  
data class Boleto(  
 @SerializedName("id\_boleto") val idBoleto: Int,  
 @SerializedName("id\_usuario") val idUsuario: Int,  
 @SerializedName("id\_vuelo") val idVuelo: Int,  
 @SerializedName("fecha\_compra") val fechaCompra: Date,  
 @SerializedName("numero\_asiento") val numeroAsiento: String  
)

package ec.edu.monster.model  
  
import com.google.gson.annotations.SerializedName  
import java.util.Date  
  
data class Compra(  
 @SerializedName("id\_compra") val idCompra: Int,  
 @SerializedName("id\_usuario") val idUsuario: Int,  
 @SerializedName("id\_boleto") val idBoleto: Int,  
 @SerializedName("fecha\_compra") val fechaCompra: Date,  
 @SerializedName("monto\_total") val montoTotal: Double  
)

package ec.edu.monster.model  
  
import com.google.gson.annotations.SerializedName  
  
data class Usuario(  
 @SerializedName("id\_usuario") val idUsuario: Int,  
 @SerializedName("nombre\_usuario") val nombreUsuario: String,  
 @SerializedName("apellido\_usuario") val apellidoUsuario: String,  
 @SerializedName("cedula") val cedula: String,  
 @SerializedName("celular") val celular: String,  
 @SerializedName("email") val email: String,  
 @SerializedName("contrasena") val contrasena: String  
)

package ec.edu.monster.model  
  
import com.google.gson.annotations.SerializedName  
import java.util.Date  
  
data class Vuelo(  
 @SerializedName("id\_vuelo") val idVuelo: Int,  
 @SerializedName("ciudad\_origen") val ciudadOrigen: String,  
 @SerializedName("ciudad\_destino") val ciudadDestino: String,  
 @SerializedName("valor") val valor: Double,  
 @SerializedName("hora\_salida") val horaSalida: Date,  
 @SerializedName("asientos\_disponibles") val asientosDisponibles: Int  
)

*Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.*

Figura 27. Creación clases de servicio

*Texto

El contenido generado por IA puede ser incorrecto.*

Figura 28. Creación clases de controlador

package ec.edu.monster.service  
  
import ec.edu.monster.model.Boleto  
import ec.edu.monster.model.CompraResponse  
import ec.edu.monster.model.LoginRequest  
import ec.edu.monster.model.Usuario  
import ec.edu.monster.model.Vuelo  
import retrofit2.Call  
import retrofit2.http.Body  
import retrofit2.http.GET  
import retrofit2.http.POST  
import retrofit2.http.Path  
import retrofit2.http.Query  
  
interface ApiService {  
 @GET("api/Vuelos/buscar")  
 fun buscarVuelos(  
 @Query("origen") origen: String,  
 @Query("destino") destino: String,  
 @Query("fecha") fecha: String  
 ): Call<List<Vuelo>>  
  
 @POST("api/Compras/comprar")  
 fun comprarBoletos(  
 @Query("idUsuario") idUsuario: Int,  
 @Query("idVuelo") idVuelo: Int,  
 @Query("numeroAsientos") numeroAsientos: Int  
 ): Call<CompraResponse> // Cambiado de String a CompraResponse  
  
 @GET("api/Boletos/usuario/{idUsuario}")  
 fun mostrarBoletosUsuario(@Path("idUsuario") idUsuario: Int): Call<List<Boleto>>  
  
 @GET("api/Vuelos")  
 fun mostrarTodosVuelos(): Call<List<Vuelo>>  
  
 @POST("api/Usuarios")  
 fun registrarUsuario(@Body usuario: Usuario): Call<Usuario>  
  
 @POST("api/Usuarios/login")  
 fun login(@Body loginRequest: LoginRequest): Call<Usuario>  
  
 @GET("api/Usuarios/idPorEmail")  
 fun obtenerIdUsuarioPorEmail(@Query("email") email: String): Call<Int>  
}

package ec.edu.monster.service  
  
import com.google.gson.GsonBuilder  
import retrofit2.Retrofit  
import retrofit2.converter.gson.GsonConverterFactory  
  
object RetrofitClient {  
 private const val BASE\_URL = "http://10.0.2.2:5166/"  
  
 private val gson = GsonBuilder()  
 .setDateFormat("yyyy-MM-dd'T'HH:mm:ss") // Formato esperado por el backend  
 .setLenient() // Hace que Gson sea más tolerante con formatos no estrictos  
 .create()  
  
 val apiService: ApiService by *lazy* **{** Retrofit.Builder()  
 .baseUrl(BASE\_URL)  
 .addConverterFactory(GsonConverterFactory.create(gson))  
 .build()  
 .create(ApiService::class.*java*)  
 **}**}

### EJECUCION

*Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.*

Figura 29. Ejecución Pantalla Login

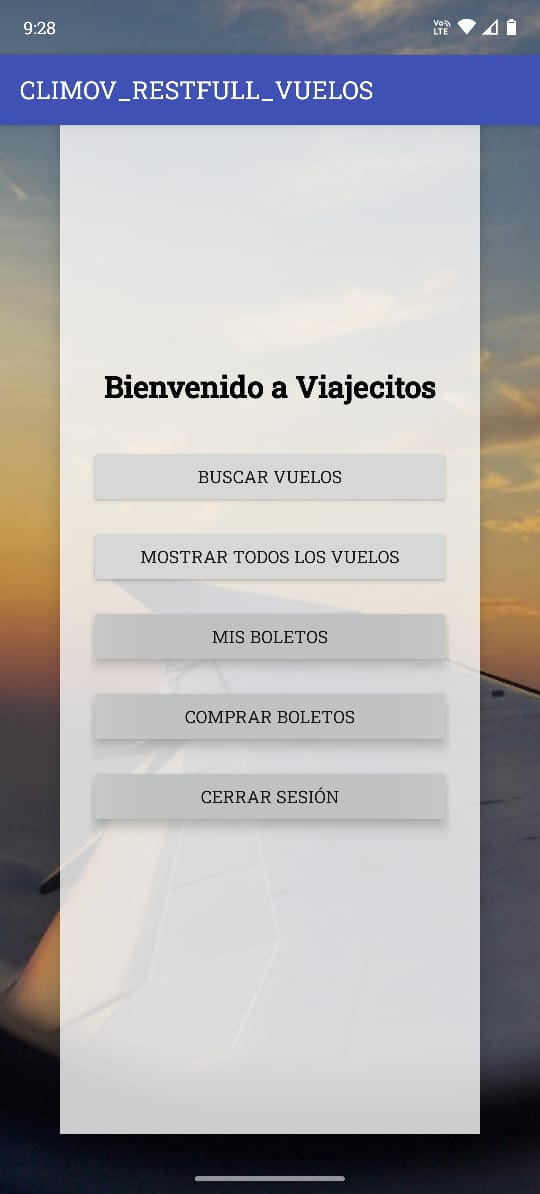


Figura 30. Menú Principal.

## CLIENTE WEB

### 1️ CREAR EL PROYECTO

* Abre **visual studio**.
* Selecciona **nuevo proyecto**.
* Elige: **aplicación web asp.net core mvc** (puede ser .net core o framework según tu caso).
* Nombre del proyecto: clientewebvuelos.
* Clic en **Crear**.

### 2️ CREAR LAS CARPETAS MVC PERSONALIZADAS

En el **Explorador de soluciones**:

1. Clic derecho sobre el proyecto → **Agregar** → **Nueva carpeta**:
   * Models → Para clases Usuario.cs, Vuelo.cs, Boleto.cs.
   * Controllers → Para controladores (UsuariosController.cs, etc.).
   * Views → Dentro de Views, crea carpetas:
     + Login
     + Usuarios
     + Vuelos
     + Shared (si no existe ya).

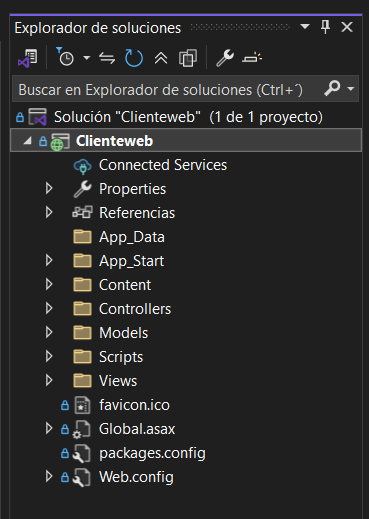


Figura 31. Estructura.

### 3️ MODELOS (MODELS)

Dentro de Models, crea:

* Usuario.cs
* Vuelo.cs
* Boleto.cs

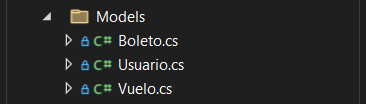


Figura 32. modelos.

Esta clase Boleto es un modelo de datos que representa la información de un boleto de avión en el sistema. Contiene las propiedades necesarias para almacenar el ID del boleto,

ID del usuario, ID del vuelo, la fecha de compra, el número de asiento y el valor del boleto (en caso de que la API devuelva este dato). Estas propiedades permiten trabajar con la información del boleto en el cliente de escritorio.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Boleto

{

public int id\_boleto { get; set; }

public int id\_usuario { get; set; }

public int id\_vuelo { get; set; }

public DateTime fecha\_compra { get; set; }

public string numero\_asiento { get; set; }

public decimal valor { get; set; } // Si la API lo devuelve

}

}

Esta clase Usuario es un modelo de datos que representa a un usuario registrado en el sistema. Contiene las propiedades necesarias para almacenar su información: ID de usuario (clave primaria), nombre, apellido, cédula, celular, correo electrónico y contraseña. Estas propiedades permiten identificar y gestionar la información del usuario en la aplicación.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Usuario

{

[Key]

public int id\_usuario { get; set; }

public string nombre\_usuario { get; set; }

public string apellido\_usuario { get; set; }

public string cedula { get; set; }

public string celular { get; set; }

public string email { get; set; }

public string contrasena { get; set; }

}

}

Esta clase Vuelo es un modelo de datos que representa la información de un vuelo en el sistema. Define propiedades como el ID del vuelo, la ciudad de origen, ciudad de destino, el valor del boleto, la hora de salida y la cantidad de asientos disponibles. Estas propiedades permiten trabajar con los datos de los vuelos dentro del cliente de escritorio para funciones como búsqueda, reserva y visualización.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClienteEscritorioVuelos.ec.edu.monster.modelo

{

public class Vuelo

{

public int id\_vuelo { get; set; }

public string ciudad\_origen { get; set; }

public string ciudad\_destino { get; set; }

public decimal valor { get; set; }

public DateTime hora\_salida { get; set; }

public int asientos\_disponibles { get; set; }

}

}

### 4 CONTROLADORES

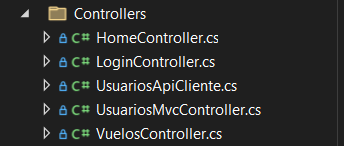


Figura 33. Controladores.

La clase UsuariosApiCliente es un cliente HTTP que permite interactuar con la API REST del sistema de vuelos. Utiliza HttpClient para realizar peticiones como buscar vuelos, comprar boletos, registrar usuarios, iniciar sesión y obtener datos de usuarios o vuelos. Esta clase se encarga de convertir las respuestas JSON de la API en objetos C# (Vuelo, Boleto, Usuario) para que el cliente web pueda utilizarlos de manera sencilla.

using ClienteConsolaVuelos.ec.edu.monster.modelo;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using System.Web;

namespace Clienteweb.Controllers

{

public class UsuariosApiCliente

{

private readonly HttpClient client;

public UsuariosApiCliente()

{

client = new HttpClient();

client.BaseAddress = new Uri("http://10.40.20.28/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public async Task<List<Vuelo>> BuscarVuelos(string origen, string destino)

{

var response = await client.GetAsync($"api/Vuelos/buscar?origen={origen}&destino={destino}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

public async Task<string> ComprarBoletos(int idUsuario, int idVuelo, int numeroAsientos)

{

var response = await client.PostAsync($"api/Compras/comprar?idUsuario={idUsuario}&idVuelo={idVuelo}&numeroAsientos={numeroAsientos}", null);

return await response.Content.ReadAsStringAsync();

}

public async Task<List<Boleto>> MostrarBoletosUsuario(int idUsuario)

{

var response = await client.GetAsync($"api/Boletos/usuario/{idUsuario}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Boleto>>(json);

}

public async Task<List<Vuelo>> MostrarTodosVuelos()

{

var response = await client.GetAsync("api/Vuelos");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<List<Vuelo>>(json);

}

public async Task<Usuario> RegistrarUsuario(Usuario usuario)

{

var jsonContent = new StringContent(JsonConvert.SerializeObject(usuario), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

public async Task<Usuario> Login(string email, string contrasena)

{

var datos = new { email = email, contrasena = contrasena };

var jsonContent = new StringContent(JsonConvert.SerializeObject(datos), Encoding.UTF8, "application/json");

var response = await client.PostAsync("api/Usuarios/login", jsonContent);

if (!response.IsSuccessStatusCode) return null;

var json = await response.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<Usuario>(json);

}

public async Task<int> ObtenerIdUsuarioPorEmail(string email)

{

var response = await client.GetAsync($"api/Usuarios/email/{email}");

if (!response.IsSuccessStatusCode) return -1;

var idStr = await response.Content.ReadAsStringAsync();

return int.Parse(idStr);

}

}

}

Esta clase UsuariosMvcController es un **controlador MVC** que permite gestionar las vistas y acciones relacionadas con el inicio de sesión y el registro de usuarios en una aplicación web. Usa UsuariosApiCliente para comunicarse con la API REST y realizar las operaciones de **login** e **insertar un nuevo usuario**. También permite mostrar las vistas correspondientes (IniciarSesion y Registrar). Si el usuario se autentica correctamente, se guarda en la sesión (Session["UsuarioLogueado"]) para que pueda acceder al menú principal. Si hay errores, los muestra en la vista.

using System.Threading.Tasks;

using System.Web.Mvc;

using ClienteConsolaVuelos.ec.edu.monster.modelo;

namespace Clienteweb.Controllers

{

public class UsuariosMvcController : Controller

{

private readonly UsuariosApiCliente api = new UsuariosApiCliente();

public ActionResult IniciarSesion()

{

return View();

}

### 5 VISTAS

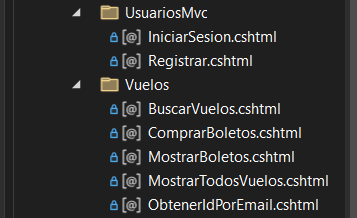


Figura 34. Vistas.

## CONCLUSIONES

* La implementación de una API RESTful facilita la comunicación entre diferentes tipos de clientes (consola, web, móvil y escritorio), permitiendo el acceso a los mismos servicios mediante estándares HTTP, lo que promueve la escalabilidad y el mantenimiento del sistema.
* El uso de una arquitectura basada en REST mejora la modularidad del sistema: cada cliente es independiente, y el servidor expone sus funcionalidades mediante endpoints bien definidos, permitiendo futuras expansiones o integraciones sin afectar la lógica de negocio.
* Desarrollar clientes para distintos entornos tecnológicos (consola, escritorio, web y móvil) demuestra la versatilidad y la adaptabilidad de las API REST, destacando la importancia de un backend robusto para soportar múltiples plataformas de consumo.

## RECOMENDACIONES

* Establecer una documentación clara para la API REST, utilizando herramientas como Swagger/OpenAPI, que facilite el consumo de los servicios por los distintos clientes.
* Implementar validaciones y manejo de errores adecuado en la API y los clientes, para mejorar la experiencia de usuario y evitar caídas inesperadas ante entradas inválidas o fallos de red.
* Mantener actualizados los frameworks y librerías utilizadas en cada cliente (ASP.NET MVC, Windows Forms, Xamarin/MAUI, .NET Console), asegurando la compatibilidad y la seguridad de las aplicaciones.

## BIBLIOGRAFÍA

* Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California.
* Microsoft Docs. (2024). *Create a Web API with ASP.NET Core*. Disponible en: <https://learn.microsoft.com/en-us/aspnet/core/web-api/>
* Villamizar, M., et al. (2015). *Evaluating the Performance of Web Applications using REST and SOAP Architectures*. IEEE Latin America Transactions.
* Xamarin Docs. (2024). *Create your first mobile app with .NET MAUI*. Disponible en: <https://learn.microsoft.com/en-us/dotnet/maui/>