

**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**

01/06/2025

Tabla de contenido

[JAVA ENTERPRISE EDITION 6](#_Toc134613096)

[INTRODUCCIÓN 6](#_Toc134613097)

[OBJETIVO 9](#_Toc134613098)

[INSTALACIÓN DE LAS HERRAMIENTAS 9](#_Toc134613099)

[INSTALACIÓN DE POWER DESIGNER 9](#_Toc134613100)

[INSTALACIÓN DE NETBEANS 16](#_Toc134613101)

[INSTALACIÓN DE PAYARA SERVER 24](#_Toc134613102)

[INSTALACIÓN DE MYSQL 32](#_Toc134613103)

[POOL DE CONEXIÓN, BASE DE DATOS MYSQL – PAYARA 39](#_Toc134613104)

[Comandos útiles dentro de Java 42](#_Toc134613105)

[CONCLUSIONES 43](#_Toc134613106)

[RECOMENDACIONES 43](#_Toc134613107)

[BIBLIOGRAFÍA 44](#_Toc134613108)

[CONCLUSIONES 46](#_Toc134613109)

[RECOMENDACIONES 46](#_Toc134613110)

[BIBLIOGRAFÍA 46](#_Toc134613111)

**ÍNDICE DE IMÁGENES**

[Figura 1. Logotipo identificativo de Java Enterprise Edition 6](#_Toc134613134)

[Figura 2. Especificaciones de APIs de Java Enterprise Edition. 6](#_Toc134613135)

[Figura 3. Integración de JEE con Base de Datos mediante JPA. 7](#_Toc134613136)

[Figura 4. Integración de JEE Session Beans y capas con el API CDI. 7](#_Toc134613137)

[Figura 5. Mensajería en Java Enterpise Edition. 8](#_Toc134613138)

[Figura 6. Arquitectura SOA con JEE. 8](#_Toc134613139)

[Figura 7. Página principal de instalación de Power Designer 9](#_Toc134613140)

[Figura 8. Página de registro Power Designer 9](#_Toc134613141)

[Figura 9. Página descarga Power Designer 10](#_Toc134613142)

[Figura 10. Ventana Guardar archivo 10](#_Toc134613143)

[Figura 11. Selección del idioma 10](#_Toc134613144)

[Figura 12. Ventana instalación SAP 11](#_Toc134613145)

[Figura 13. Selección tipo de licencia SAP 11](#_Toc134613146)

[Figura 14 Acuerdo licencia SAP 12](#_Toc134613147)

[Figura 15 Selección carpeta destino SAP 12](#_Toc134613148)

[Figura 16. Selección características a instalar 13](#_Toc134613149)

[Figura 17. Ventana selección características a importar 13](#_Toc134613150)

[Figura 18. Selección nombre de la carpeta SAP 14](#_Toc134613151)

[Figura 19. Ventana revisión de las opciones seleccionadas 14](#_Toc134613152)

[Figura 20. Instalación de Power Designer 15](#_Toc134613153)

[Figura 21. Página inicio Power Designer 15](#_Toc134613154)

[Figura 22. Página descarga NetBeans 16](#_Toc134613155)

[Figura 23. Página descarga imagen Java 16](#_Toc134613156)

[Figura 22 Guardar archivo en computador 17](#_Toc134613157)

[Figura 23 Ventana instalador java 17](#_Toc134613158)

[Figura 24. Selecciones características JDK, JRE 17](#_Toc134613159)

[Figura 25 Selección carpeta destino Java 18](#_Toc134613160)

[Figura 26 Finalizar la instalación Java 18](#_Toc134613161)

[Figura 27 Instalación NetBeans 19](#_Toc134613162)

[Figura 28 Guardar el archivo 19](#_Toc134613163)

[Figura 29 Configuración del instalador NetBeans 20](#_Toc134613164)

[Figura 30 Características a instalar NetBeans 20](#_Toc134613165)

[Figura 31 Selección de paquetes y servidores a instalar 21](#_Toc134613166)

[Figura 32 Aceptación de términos 21](#_Toc134613167)

[Figura 33 Selección de carpeta de instalación 22](#_Toc134613168)

[Figura 34 Instalación de paquetes seleccionados 22](#_Toc134613169)

[Figura 35 Finalización de la instalación 23](#_Toc134613170)

[Figura 36. Servidor de Aplicaciones de NetBeans. 23](#_Toc134613171)

[Figura 37. Servidor de Aplicaciones de NetBeans y BDD que pueden trabajar con NetBeans 24](#_Toc134613172)

[Figura 38. Logo Identificativo de Payara 24](#_Toc134613173)

[Figura 39. Instalación Drivers Payara Server 25](#_Toc134613174)

[Figura 40. Instalación Drivers Payara Server 25](#_Toc134613175)

[Figura 41. Descarga de Payara. 26](#_Toc134613176)

[Figura 42. Pegar los archivos del Servidor Aplicaciones Payara en el Disco C 26](#_Toc134613177)

[Figura 43. Dirigirse al Directorio Program Files 27](#_Toc134613178)

[Figura 44. Add Server 27](#_Toc134613179)

[Figura 45. Insertar Payara 28](#_Toc134613180)

[Figura 46. Ubicación de Payara en Netbeans 28](#_Toc134613181)

[Figura 47. Dominio Payara 29](#_Toc134613182)

[Figura 48. Verificación instalación 29](#_Toc134613183)

[Figura 49. Ejecución Payara Server 29](#_Toc134613184)

[Figura 50. Sitio web de Payara 30](#_Toc134613185)

[Figura 51. Admin Console de Payara 30](#_Toc134613186)

[Figura 52. Ejecución Payara Server 31](#_Toc134613187)

[Figura 53. Ejecución Payara Server 31](#_Toc134613188)

[Figura 54. Ejecución Payara Server 31](#_Toc134613189)

[Figura 55. Integrar un driver para un RDBMS en NetBeans. 32](#_Toc134613190)

[Figura 56. Integrar un driver para un RDBMS en NetBeans. 32](#_Toc134613191)

[Figura 57. Selección del sistema operativo 33](#_Toc134613192)

[Figura 58. Acuerdo de licencia MySQL. 33](#_Toc134613193)

[Figura 59. Selección del tipo de instalación MySQL. 34](#_Toc134613194)

[Figura 60. Paquetes MySQL a instalarse. 34](#_Toc134613195)

[Figura 61. Configuración del producto. 35](#_Toc134613196)

[Figura 62. Configurar la disponibilidad MySQL. 35](#_Toc134613197)

[Figura 63. Configuraciones de conectividad. 36](#_Toc134613198)

[Figura 64. Selección del método de autenticación. 36](#_Toc134613199)

[Figura 65. Agregar usuarios y contraseñas. 37](#_Toc134613200)

[Figura 66. Ventana cambiar configuraciones del servicio. 37](#_Toc134613201)

[Figura 67. Ventana ejecución de las configuraciones realizadas. 38](#_Toc134613202)

[Figura 68. Ventana de conexión con el servidor. 38](#_Toc134613203)

[Figura 69. Ventana de instalación completa. 39](#_Toc134613204)

[Figura 70. Configurar base de datos 39](#_Toc134613205)

[Figura 71. Configurar base de datos 40](#_Toc134613206)

[Figura 72. Configurar base de datos 40](#_Toc134613207)

[Figura 73. Configurar base de datos 40](#_Toc134613208)

[Figura 74. Creando Pool de conexiones 41](#_Toc134613209)

[Figura 75. Connection Pool Properties 41](#_Toc134613210)

[Figura 76. Propiedades del Pool de Conexiones. 42](#_Toc134613211)

[Figura 77. Comandos útiles dentro de Java 43](#_Toc134613212)

**ÍNDICE DE TABLAS**

[Tabla 1. Resumen comandos útiles dentro de Java 42](#_Toc134613431)

# 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.

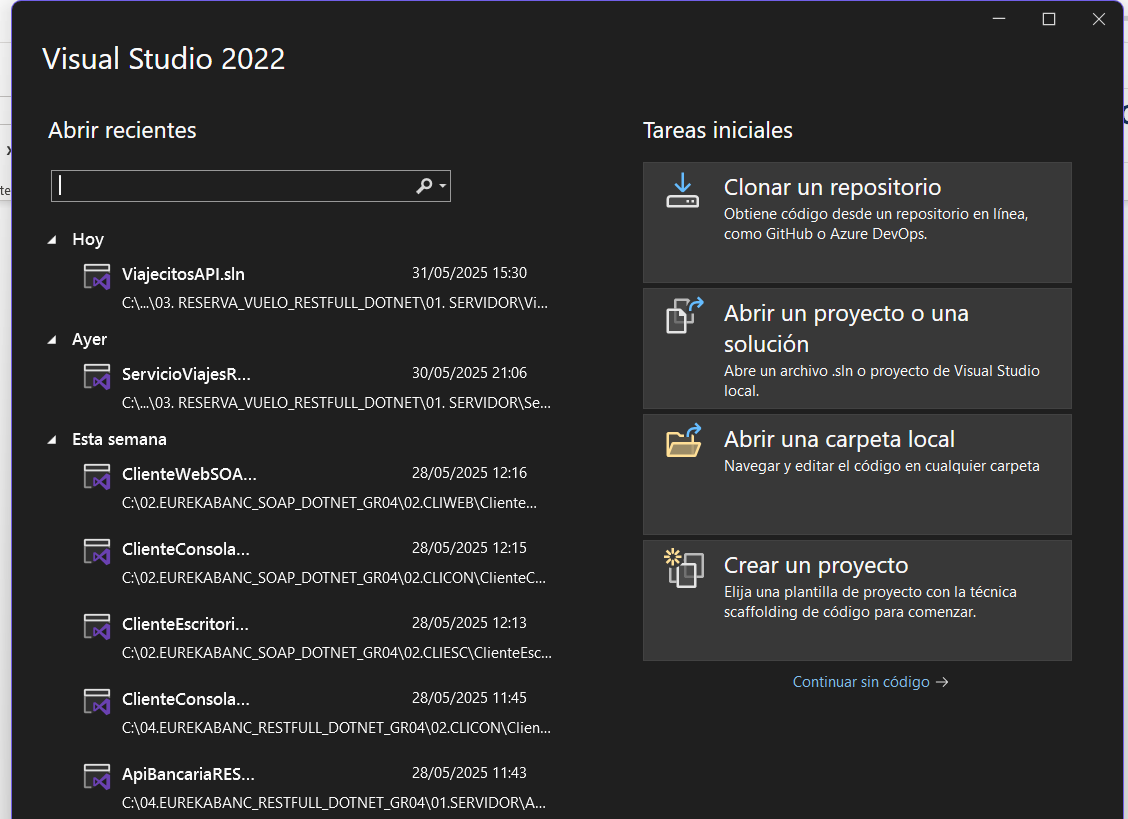


Figura 1. Creación del proyecto.

### 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.

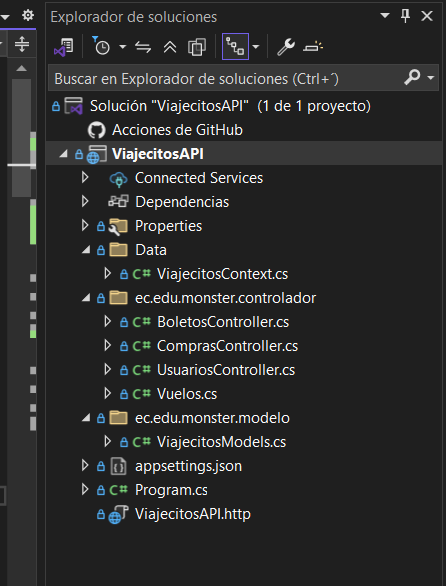


Figura 2. Estructura del proyecto.

### 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.

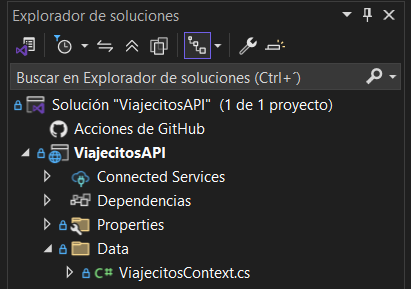


Figura 3. Carpeta de la Data.

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.

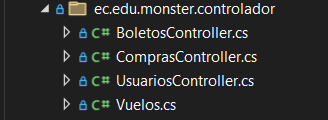


Figura 4. Carpeta de los controladores.

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

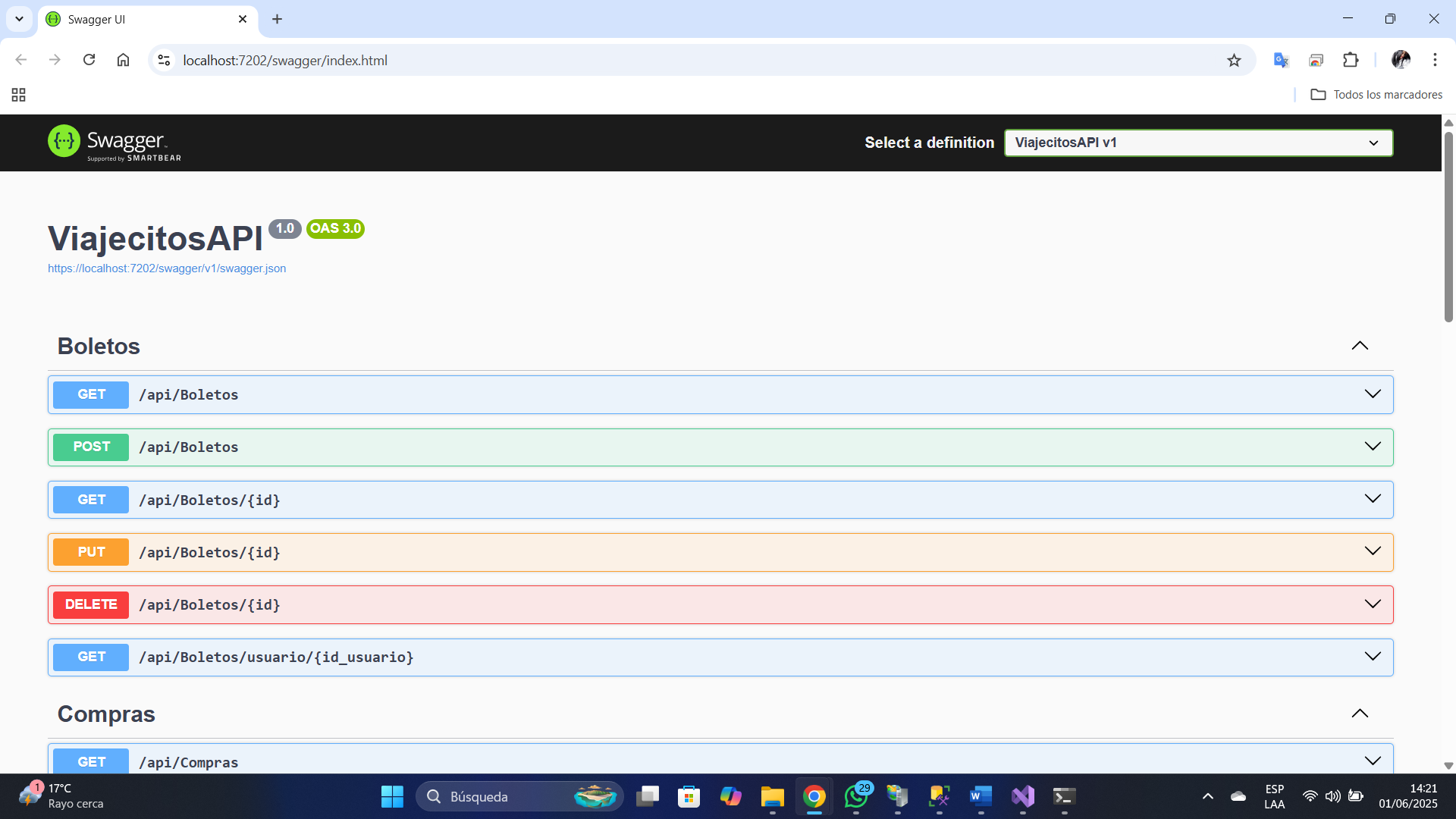


Figura 5. API Boletos.

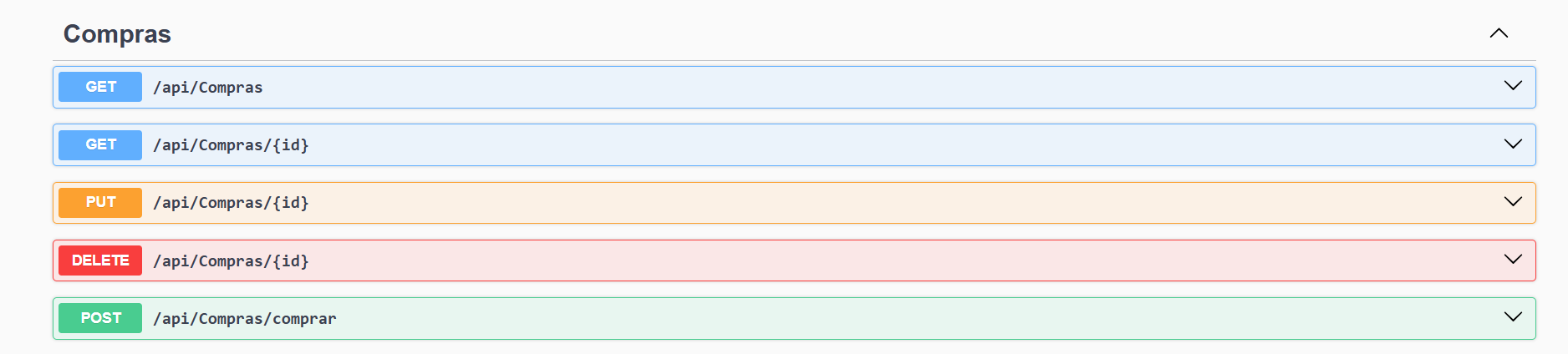


Figura 6. API Compras.

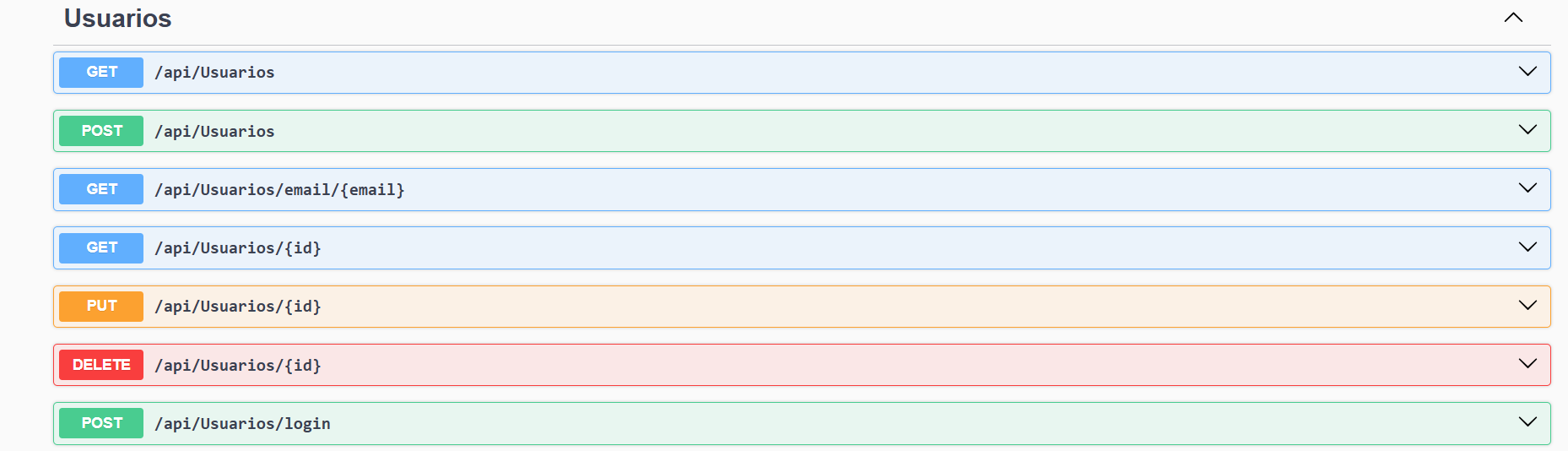


Figura 7. API Usuarios.

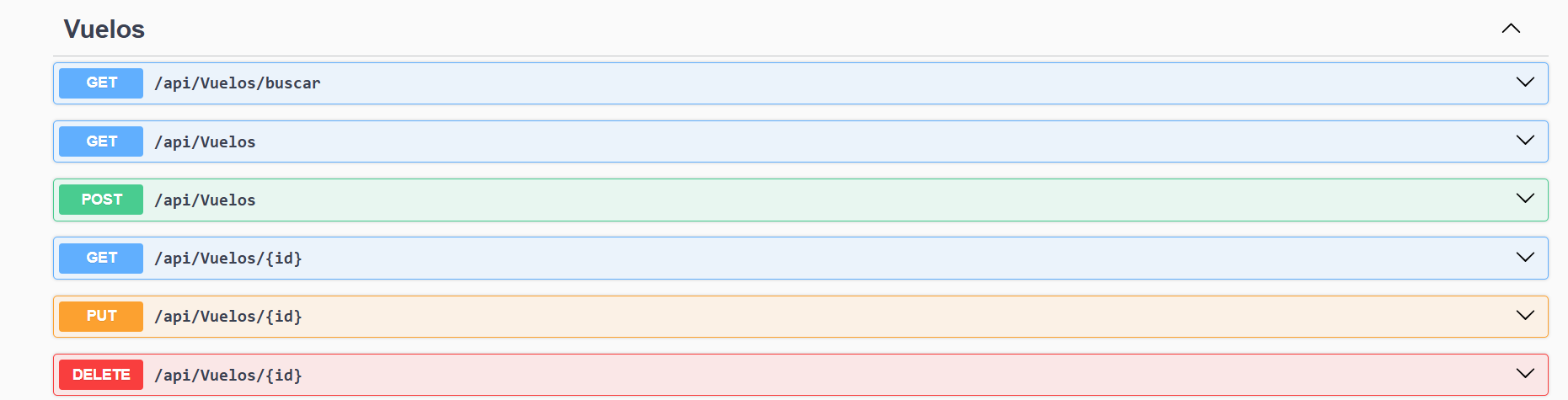


Figura 8. API Vuelos.

## 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

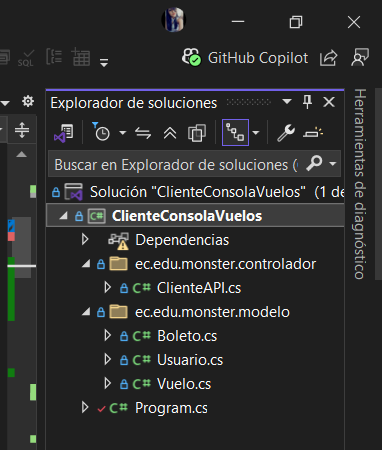


Figura 9. Carpetas del cliente Consola.

### 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}");

}

}

}

## CONCLUSIONES

* El servidor de aplicaciones GlassFish dentro del desarrollo en Java, ha sido reemplazado por Payara, ofreciendo mayor robustez, soporte y actualizaciones, permitiendo así desarrollar y testear aplicativos eficientemente.
* Java EE en sus versiones más recientes busca simplificar el desarrollo en J2EE que tradicionalmente ha sido muy complejo.

## RECOMENDACIONES

* Antes de realizar la instalación de Power Designer se debe tomar en cuenta ciertas características que debe poseer el equipo para que el software a ser instalado funcione de una manera óptima y no presente inconvenientes a la hora de su utilización, como requisitos mínimos de sistema:
  + Microsoft Windows 7, 8, 8.1 o 10, o en el caso de Microsoft Windows Server 2008 o 2012 (32 bits o 64 bits)
  + Procesador de 2 GHz
  + 3 GB de RAM
  + Adaptador gráfico SVGA o de mayor resolución y monitor de color compatible (800x600)
  + 1 GB de espacio en disco para la instalación de todos los módulos principales (y vídeos de demostración).
* NetBeans es uno de los entornos de programación más utilizados a la hora de construir programas en JAVA. Para descargar e instalar NetBeans es necesario haber instalado primero el JDK.
* MySQL es una de las bases de datos de código abierto más populares y utilizadas en todo el mundo, especialmente para el desarrollo de entornos web, aunque actualmente pertenece a Oracle, fue una empresa llamada MySQL AB la que inició el desarrollo de MySQL. Más tarde esta empresa fue adquirida por Sun Microsystem que a su vez fue comprada por Oracle, la actual propietaria. Este juego de compras de empresas seguro que te puede dar una idea de la importancia estratégica de MySQL para las compañías dedicadas a las bases de datos. MySQL, aunque puede utilizarse en una amplia gama de aplicaciones, se asocia más a menudo con las aplicaciones web y la publicación en línea.

## BIBLIOGRAFÍA

* Payara for Beginners—Adding Payara Server to NetBeans. (s. f.). Recuperado 24 de mayo de 2020, de <https://blog.payara.fish/adding-payara-server-to-netbeans>
* 01.- Curso Java EE 6. Configuración del Entorno de desarrollo (2012). Rcuperado el 24 de mayo de 2020 de: <https://www.youtube.com/watch?v=V23GNjnf438&t=73s&pbjreload=10>