

Universidad Mariano Gálvez de Guatemala
Ingeniería en Sistemas de Información y Ciencias de la computación



Link del video: https://drive.google.com/file/d/1a_wEaJTNzLbo2i6erd06K5fOXzrrpx3Y/view?usp=drivesdk

Programación III

Ing. José Luis Xiloj

Dayli Yadira Santos Herrarte	1590-23-6686
Jaqueline Yajaira Lorenzana Cisneros	1590-23-12507
Kimberli Yanet Hernández Rodríguez	1590-23-12625
Robin Elías García Santos	1590-21-7722
Alexander Jossimar García de León	1590-23-18540
José Manuel Monterroso Avalos	1590-23-14680

Sección: B

Nombre	Participación
Dayli Yadira Santos Herrarte	100%
Jaqueline Yajaira Lorenzana Cisneros	100%
Kimberli Yanet Hernández Rodríguez	100%
Robin Elías García Santos	100%
Alexander Jossimar García de León	100%
José Manuel Monterroso Avalos	100%

Introducción

El crecimiento constante de las ciudades ha provocado un aumento significativo en el número de vehículos que circulan diariamente por las vías urbanas. Este fenómeno ha generado nuevos desafíos en materia de movilidad, tales como la congestión vehicular, el aumento en los tiempos de desplazamiento, la contaminación ambiental y la necesidad de sistemas eficientes de control del tráfico. En este contexto, la informática y la programación juegan un papel fundamental al permitir el diseño y la implementación de herramientas tecnológicas que simulen, analicen y optimicen el comportamiento del tránsito en una red vial.

El presente proyecto se enmarca en la asignatura de Programación III y tiene como propósito el desarrollo de una aplicación web interactiva que represente el funcionamiento de una red vial urbana utilizando estructuras dinámicas de dato. El sistema ha sido desarrollado bajo el entorno de Blazor Server, aprovechando la flexibilidad de C# para el diseño de estructuras eficientes y adaptables.

Una de las principales restricciones académicas de este proyecto consiste en no utilizar estructuras de datos integradas del lenguaje, como List, Array, Dictionary o similares. En su lugar, se diseñaron desde cero clases que representan nodos y listas enlazadas, las cuales permiten recorrer, insertar y enlazar dinámicamente intersecciones, rutas y conexiones entre ellas. Esta decisión fortalece el conocimiento del estudiante sobre estructuras de bajo nivel y mejora su capacidad para manipular directamente la memoria lógica de los objetos. Se incluye una estimación del tiempo de recorrido basado en los semáforos y la cantidad de vehículos en espera, representando así un modelo de tráfico urbano simplificado pero funcional.

Este informe detalla el diseño, desarrollo, implementación y pruebas del sistema, así como los aprendizajes adquiridos a lo largo del proceso. También se incluyen las capturas de pantalla, el código fuente comentado, los diagramas de estructura y el análisis de funcionamiento. A través de este proyecto se demuestra cómo es posible aplicar conceptos fundamentales de estructuras de datos en un entorno real, resolviendo un problema común en las ciudades y desarrollando habilidades clave para la ingeniería en sistemas.

Proyecto Red Vial Urbana



AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using Proyecto1.Models;

namespace Proyecto1.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
        }

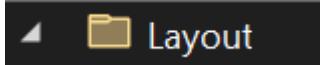
        public DbSet<Nodo> Nodos { get; set; }
    }
}
```

RedVialDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using Proyecto1.Models;

namespace Proyecto1.Data
{
    public class RedVialContext : DbContext
    {
        public RedVialContext(DbContextOptions<RedVialContext> options) : base(options) {}

        public DbSet<Nodo> Nodos { get; set; }
        public DbSet<ReporteRedVial> Reportes { get; set; }
    }
}
```



MainLayout.razor

```
@inherits LayoutComponentBase
```

```
@using Proyecto1.Shared
```

```
<div class="d-flex" style="min-height: 100vh;">
    <div class="bg-light border-end p-3" style="width: 250px;">
        <NavMenu />
    </div>

    <main class="flex-grow-1 p-4 bg-white">
        @Body
    </main>
</div>
```



20250509001021_CrearReporteTabla.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;
```

```
#nullable disable
```

```
namespace Proyecto1.Migrations
```

```
{
    /// <inheritdoc />
    public partial class CrearReporteTabla : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Reportes",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    NodosTexto = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    InterseccionMasCongestionada = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CuellosBotellaTexto = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    FechaGeneracion = table.Column<DateTime>(type: "datetime2", nullable: false)
                },
            );
        }
}
```

```

constraints: table =>
{
    table.PrimaryKey("PK_Reportes", x => x.Id);
};

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Reportes");
}
}
}

```

20250509003501_AgregarTablaNodos.cs

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

```

namespace Proyecto1.Migrations
{
    /// <inheritdoc />
    public partial class AgregarTablaNodos : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Nodos",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    VehiculosEnEspera = table.Column<int>(type: "int", nullable: false),
                    EstadoSemaforo = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    TiempoPromedioCruce = table.Column<int>(type: "int", nullable: false),
                    TipoViaNorte = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    TipoViaSur = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    TipoViaEste = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    TipoViaOeste = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>

```

```
        {
            table.PrimaryKey("PK_Nodos", x => x.Id);
        });
    }

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Nodos");
}
}
}

20250509005404_CrearTablaNodos.cs
using Microsoft.EntityFrameworkCore.Migrations;
```

```
#nullable disable
```

```
namespace Proyecto1.Migrations
{
    /// <inheritdoc />
    public partial class CrearTablaNodos : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
        }
    }
}
```

20250509005729_AgregarReferenciasNodos.cs

```
using Microsoft.EntityFrameworkCore.Migrations;
```

```
#nullable disable
```

```
namespace Proyecto1.Migrations
{
    /// <inheritdoc />
    public partial class AgregarReferenciasNodos : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<string>(
                name: "IdNodoEste",
                table: "Nodos",
                type: "nvarchar(max)",
                nullable: true);

            migrationBuilder.AddColumn<string>(
                name: "IdNodoNorte",
                table: "Nodos",
                type: "nvarchar(max)",
                nullable: true);

            migrationBuilder.AddColumn<string>(
                name: "IdNodoOeste",
                table: "Nodos",
                type: "nvarchar(max)",
                nullable: true);

            migrationBuilder.AddColumn<string>(
                name: "IdNodoSur",
                table: "Nodos",
                type: "nvarchar(max)",
                nullable: true);
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "IdNodoEste",
                table: "Nodos");

            migrationBuilder.DropColumn(

```

```

        name: "IdNodoNorte",
        table: "Nodos");

    migrationBuilder.DropColumn(
        name: "IdNodoOeste",
        table: "Nodos");

    migrationBuilder.DropColumn(
        name: "IdNodoSur",
        table: "Nodos");
    }
}
}

20250509011102_CrearTablaNodos2.cs
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Proyecto1.Migrations
{
/// <inheritdoc />
public partial class CrearTablaNodos2 : Migration
{
/// <inheritdoc />
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.RenameColumn(
        name: "IdNodoSur",
        table: "Nodos",
        newName: "IdSur");

    migrationBuilder.RenameColumn(
        name: "IdNodoOeste",
        table: "Nodos",
        newName: "IdOeste");

    migrationBuilder.RenameColumn(
        name: "IdNodoNorte",
        table: "Nodos",
        newName: "IdNorte");
}
}

```

```

        migrationBuilder.RenameColumn(
            name: "IdNodoEste",
            table: "Nodos",
            newName: "IdEste");
    }

/// <inheritDoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.RenameColumn(
        name: "IdSur",
        table: "Nodos",
        newName: "IdNodoSur");

    migrationBuilder.RenameColumn(
        name: "IdOeste",
        table: "Nodos",
        newName: "IdNodoOeste");

    migrationBuilder.RenameColumn(
        name: "IdNorte",
        table: "Nodos",
        newName: "IdNodoNorte");

    migrationBuilder.RenameColumn(
        name: "IdEste",
        table: "Nodos",
        newName: "IdNodoEste");
}

}
}

}

RedVialContextModelSnapshot.cs
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Proyecto1.Data;

#nullable disable

```

```

namespace Proyecto1.Migrations
{
    [DbContext(typeof(RedVialContext))]
    partial class RedVialContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {

#pragma warning disable 612, 618

        modelBuilder
            .HasAnnotation("ProductVersion", "9.0.4")
            .HasAnnotation("Relational:MaxIdentifierLength", 128);

       SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

        modelBuilder.Entity("Nodo", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("EstadoSemaforo")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("IdEste")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("IdNorte")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("IdOeste")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("IdSur")
                .HasColumnType("nvarchar(max)");

            b.Property<int>("TiempoPromedioCruce")
                .HasColumnType("int");

            b.Property<string>("TipoViaEste")
                .IsRequired();
        });
    }
}

```

```

        .HasColumnType("nvarchar(max)");

    b.Property<string>"("TipoViaNorte")
        .IsRequired()
        .HasColumnType("nvarchar(max)");

    b.Property<string>"("TipoViaOeste")
        .IsRequired()
        .HasColumnType("nvarchar(max)");

    b.Property<string>"("TipoViaSur")
        .IsRequired()
        .HasColumnType("nvarchar(max)");

    b.Property<int>"("VehiculosEnEspera")
        .HasColumnType("int");

    b.HasKey("Id");

    b.ToTable("Nodos");
};

modelBuilder.Entity("Proyecto1.Models.ReporteRedVial", b =>
{
    b.Property<int>"("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("int");

   SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>"("Id"));

    b.Property<string>"("CuellosBotellaTexto")
        .IsRequired()
        .HasColumnType("nvarchar(max)");

    b.Property<DateTime>"("FechaGeneracion")
        .HasColumnType("datetime2");

    b.Property<string>"("InterseccionMasCongestionada")
        .IsRequired()
        .HasColumnType("nvarchar(max)");
}

```

```

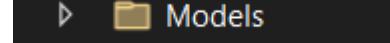
        b.Property<string>("NodosTexto")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.HasKey("Id");

        b.ToTable("Reportes");
    });

#pragma warning restore 612, 618
}
}
}

```



CaminoNodos.css

```

namespace Proyecto1.Models
{
    public class CaminoNodos
    {
        public class Paso
        {
            public Nodo Nodo { get; set; }
            public Paso? Siguiente { get; set; }

            public Paso(Nodo nodo)
            {
                Nodo = nodo;
                Siguiente = null;
            }
        }
    }

    private Paso? inicio;

    public CaminoNodos()
    {
        inicio = null;
    }

    public void AgregarPaso(Nodo nodo)
    {

```

```

var nuevo = new Paso(nodo);
if (inicio == null)
    inicio = nuevo;
else
{
    Paso actual = inicio;
    while (actual.Siguiente != null)
        actual = actual.Siguiente;
    actual.Siguiente = nuevo;
}
}

public void EliminarUltimoPaso()
{
    if (inicio == null) return;

    if (inicio.Siguiente == null)
    {
        inicio = null;
        return;
    }

    Paso actual = inicio;
    while (actual.Siguiente?.Siguiente != null)
        actual = actual.Siguiente;

    actual.Siguiente = null;
}

public IEnumerable<Nodo> ObtenerRuta()
{
    Paso? actual = inicio;
    while (actual != null)
    {
        yield return actual.Nodo;
        actual = actual.Siguiente;
    }
}

public Nodo? ObtenerUltimo()
{
    Paso? actual = inicio;

```

```

        if (actual == null) return null;

        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }

        return actual.Nodo;
    }

public bool EstaVacio()
{
    return inicio == null;
}

public double TiempoTotal()
{
    double total = 0;
    foreach (var nodo in ObtenerRuta())
    {
        total += nodo.TiempoPromedioCruce;
    }
    return total;
}
}

```

CuelloBotella.cs

```

using System.ComponentModel.DataAnnotations;

namespace Proyecto1.Models
{
    public class CuelloBotella
    {
        [Key]
        public int Id { get; set; }
        public string Nodold { get; set; } = "";
        public int VehiculosEnEspera { get; set; }
        public string EstadoSemaforo { get; set; } = "";
        public int TiempoPromedioCruce { get; set; }
    }
}

```

```
}
```

```
}
```

ListaCuellos.cs

```
namespace Proyecto1.Models
{
    public class ListaCuellos
    {
        private class Elemento
        {
            public Nodo Dato { get; set; }
            public Elemento? Siguiente { get; set; }

            public Elemento(Nodo nodo)
            {
                Dato = nodo;
            }
        }

        private Elemento? cabeza;

        public void Agregar(Nodo nodo)
        {
            var nuevo = new Elemento(nodo);
            if (cabeza == null)
                cabeza = nuevo;
            else
            {
                var actual = cabeza;
                while (actual.Siguiente != null)
                    actual = actual.Siguiente;
                actual.Siguiente = nuevo;
            }
        }

        public IEnumerable<Nodo> ObtenerTodos()
        {
            var actual = cabeza;
            while (actual != null)
            {
                yield return actual.Dato;
```

```
        actual = actual.Siguiente;
    }
}

public bool EstaVacia() => cabeza == null;
}
}
```

ListaNodos.cs

```
using System.Collections.Generic;
```

```
namespace Proyecto1.Models
```

```
{
    public class ListaNodos
    {
        public class Elemento
        {
            public Nodo Valor { get; set; }
            public Elemento? Siguiente { get; set; }

            public Elemento(Nodo valor)
            {
                Valor = valor;
                Siguiente = null;
            }
        }
    }
}
```

```
private Elemento? cabeza;
```

```
public ListaNodos()
{
    cabeza = null;
}
```

```
public void Agregar(Nodo nodo)
{
    var nuevo = new Elemento(nodo);
    if (cabeza == null)
        cabeza = nuevo;
    else
    {
```

```

        Elemento actual = cabeza;
        while (actual.Siguiente != null)
            actual = actual.Siguiente;
        actual.Siguiente = nuevo;
    }
}

public IEnumerable<Nodo> ObtenerTodos()
{
    Elemento? actual = cabeza;
    while (actual != null)
    {
        yield return actual.Valor;
        actual = actual.Siguiente;
    }
}

public bool EstaVacia()
{
    return cabeza == null;
}

public Nodo? BuscarPorId(string? id)
{
    if (id == null) return null;
    var actual = cabeza;
    while (actual != null)
    {
        if (actual.Valor.Id == id)
            return actual.Valor;
        actual = actual.Siguiente;
    }
    return null;
}
}

```

ListaReportes.cs

```

using System.Collections.Generic;
using Proyecto1.Models;

```

```

namespace Proyecto1.Models
{
    public class ListaReportes
    {
        public class Elemento
        {
            public ReporteRedVial Valor { get; set; }
            public Elemento? Siguiente { get; set; }

            public Elemento(ReporteRedVial valor)
            {
                Valor = valor;
                Siguiente = null;
            }
        }
    }

    private Elemento? cabeza;

    public void Agregar(ReporteRedVial reporte)
    {
        var nuevo = new Elemento(reporte);
        if (cabeza == null)
            cabeza = nuevo;
        else
        {
            Elemento actual = cabeza;
            while (actual.Siguiente != null)
                actual = actual.Siguiente;
            actual.Siguiente = nuevo;
        }
    }

    public IEnumerable<ReporteRedVial> ObtenerTodos()
    {
        var actual = cabeza;
        while (actual != null)
        {
            yield return actual.Valor;
            actual = actual.Siguiente;
        }
    }
}

```

```
    public bool EstaVacia() => cabeza == null;
}
}
```

Nodo.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
```

```
public class Nodo
{
    [Key]
    [Required]
    public string Id { get; set; } = "";

    public int VehiculosEnEspera { get; set; }

    [Required]
    public string EstadoSemaforo { get; set; } = "Rojo";

    public int TiempoPromedioCruce { get; set; }

    public string TipoViaNorte { get; set; } = "Ninguna";
    public string TipoViaSur { get; set; } = "Ninguna";
    public string TipoViaEste { get; set; } = "Ninguna";
    public string TipoViaOeste { get; set; } = "Ninguna";

    public string? IdNorte { get; set; }
    public string? IdSur { get; set; }
    public string? IdEste { get; set; }
    public string? IdOeste { get; set; }

    [NotMapped] public Nodo? Norte { get; set; }
    [NotMapped] public Nodo? Sur { get; set; }
    [NotMapped] public Nodo? Este { get; set; }
    [NotMapped] public Nodo? Oeste { get; set; }
}
```

NodoMasCongestionado.cs

```
using System.ComponentModel.DataAnnotations;

namespace Proyecto1.Models
{
    public class NodoMasCongestionado
    {
        [Key]
        public int Id { get; set; }
        public string Nodold { get; set; } = "";
        public int VehiculosEnEspera { get; set; }
    }
}
```

ReporteRedVial.cs

```
using System.ComponentModel.DataAnnotations;

namespace Proyecto1.Models
{
    public class ReporteRedVial
    {
        [Key]
        public int Id { get; set; }

        public string NodosTexto { get; set; } = "";
        public string InterseccionMasCongestionada { get; set; } = "";
        public string CuellosBotellaTexto { get; set; } = "";
        public DateTime FechaGeneracion { get; set; } = DateTime.Now;
    }
}
```



_Host.cshtml

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Proyecto1.Pages
{
    public class _HostModel : PageModel
    {
```

```
public void OnGet()
{
}
}

}

}

@page "/agregar-nodo"
```

AgregarNodo.razor

```
@using Proyecto1.Models
@using Proyecto1.Data
@inject RedVialContext Db

<PageTitle>Aregar Nodo</PageTitle>
```

```
<div class="container p-4 mt-3 rounded shadow" style="background-color: #fdfdfd;">
    <h3 class="mb-4 text-center text-primary">Aregar Nueva Intersección</h3>

    <EditForm Model="@nuevoNodo" OnValidSubmit="AgregarNodoCentral">
        <DataAnnotationsValidator />
        <ValidationSummary />

        <div class="mb-3">
            <label class="form-label">ID de Intersección</label>
            <InputText class="form-control" @bind-Value="nuevoNodo.Id" />
        </div>

        <div class="mb-3">
            <label class="form-label">Vehículos en Espera</label>
            <InputNumber class="form-control" @bind-Value="nuevoNodo.VehiculosEnEspera" />
        </div>

        <div class="mb-3">
            <label class="form-label">Estado del Semáforo</label>
            <InputSelect class="form-select" @bind-Value="nuevoNodo.EstadoSemaforo">
                <option value="">Seleccione...</option>
                <option value="Verde">Verde</option>
                <option value="Amarillo">Amarillo</option>
                <option value="Rojo">Rojo</option>
                <option value="Desactivado">Desactivado</option>
            </InputSelect>
        </div>
    </EditForm>
</div>
```

```

</div>

<div class="mb-3">
    <label class="form-label">Tiempo Promedio de Tránsito (segundos)</label>
    <InputNumber class="form-control" @bind-Value="nuevoNodo.TiempoPromedioCruce" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de Vía al Norte</label>
    <InputText class="form-control" @bind-Value="nuevoNodo.TipoViaNorte" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de Vía al Sur</label>
    <InputText class="form-control" @bind-Value="nuevoNodo.TipoViaSur" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de Vía al Este</label>
    <InputText class="form-control" @bind-Value="nuevoNodo.TipoViaEste" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de Vía al Oeste</label>
    <InputText class="form-control" @bind-Value="nuevoNodo.TipoViaOeste" />
</div>

<button type="submit" class="btn btn-success">Agregar Nodo Central</button>
</EditForm>

@if (!string.IsNullOrEmpty(mensaje))
{
    <div class="alert alert-info mt-3">@mensaje</div>
}
</div>

@code {
    private Nodo nuevoNodo = new Nodo();
    private string mensaje = "";

    private void AgregarNodoCentral()
}

```

```

{
    Db.Nodos.Add(nuevoNodo);
    Db.SaveChanges();
    mensaje = $"Nodo '{nuevoNodo.Id}' guardado en la base de datos.";
    nuevoNodo = new Nodo(); // Limpiar
}
}

ConecrtarNodo.razor

@page "/conectar-nodos"
@using Proyecto1.Models
@using Proyecto1.Data
@inject RedVialContext Db

<PageTitle>Conectar Intersecciones</PageTitle>

<div class="container mt-4 p-4 shadow rounded bg-light">
    <h3 class="text-center text-primary mb-4">Conectar Intersecciones</h3>

    <EditForm Model="@nodoDestino" OnValidSubmit="Conecrtar">
        <DataAnnotationsValidator />
        <ValidationSummary />

        <div class="mb-3">
            <label class="form-label">ID de Intersección Origen</label>
            <InputText class="form-control" @bind-Value="idOrigen" />
        </div>

        <div class="mb-3">
            <label class="form-label">Dirección a Conectar</label>
            <InputSelect class="form-select" @bind-Value="direccion">
                <option value="">Seleccione una dirección</option>
                <option value="norte">Norte</option>
                <option value="sur">Sur</option>
                <option value="este">Este</option>
                <option value="oeste">Oeste</option>
            </InputSelect>
        </div>

        <div class="form-check mb-3">
            <InputCheckbox class="form-check-input" @bind-Value="esBidireccional" />
        </div>
    </EditForm>

```

```
<label class="form-check-label">Conexión bidireccional</label>
</div>

<hr />
<h5>Datos de la Nueva Intersección Destino</h5>

<div class="mb-3">
    <label class="form-label">ID</label>
    <InputText class="form-control" @bind-Value="nodoDestino.Id" />
</div>

<div class="mb-3">
    <label class="form-label">Vehículos en espera</label>
    <InputNumber class="form-control" @bind-Value="nodoDestino.VehiculosEnEspera" />
</div>

<div class="mb-3">
    <label class="form-label">Estado del semáforo</label>
    <InputSelect class="form-select" @bind-Value="nodoDestino.EstadoSemaforo">
        <option value="">Seleccione...</option>
        <option value="Verde">Verde</option>
        <option value="Amarillo">Amarillo</option>
        <option value="Rojo">Rojo</option>
        <option value="Desactivado">Desactivado</option>
    </InputSelect>
</div>

<div class="mb-3">
    <label class="form-label">Tiempo promedio de tránsito (seg)</label>
    <InputNumber class="form-control" @bind-Value="nodoDestino.TiempoPromedioCruce" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de vía al Norte</label>
    <InputText class="form-control" @bind-Value="nodoDestino.TipoViaNorte" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de vía al Sur</label>
    <InputText class="form-control" @bind-Value="nodoDestino.TipoViaSur" />
</div>
```

```

<div class="mb-3">
    <label class="form-label">Tipo de vía al Este</label>
    <InputText class="form-control" @bind-Value="nodoDestino.TipoViaEste" />
</div>

<div class="mb-3">
    <label class="form-label">Tipo de vía al Oeste</label>
    <InputText class="form-control" @bind-Value="nodoDestino.TipoViaOeste" />
</div>

<button class="btn btn-primary mt-3" type="submit">Conectar</button>
</EditForm>

@if (!string.IsNullOrEmpty(mensaje))
{
    <div class="alert alert-info mt-3">@mensaje</div>
}
</div>

@code {
    private string idOrigen = "";
    private string direccion = "";
    private bool esBidireccional = false;
    private Nodo nodoDestino = new Nodo();
    private string mensaje = "";

    private void Conectar()
    {
        var origen = Db.Nodos.FirstOrDefault(n => n.Id == idOrigen);
        if (origen == null)
        {
            mensaje = $"No se encontró el nodo con ID '{idOrigen}'";
            return;
        }

        // Asignar conexión y tipo de vía
        switch (direccion.ToLower())
        {
            case "norte":
                origen.IdNorte = nodoDestino.Id;

```

```

nodoDestino.TipoViaSur = origen.TipoViaNorte;
if (esBidireccional) nodoDestino.IdSur = origen.Id;
break;
case "sur":
    origen.IdSur = nodoDestino.Id;
    nodoDestino.TipoViaNorte = origen.TipoViaSur;
    if (esBidireccional) nodoDestino.IdNorte = origen.Id;
    break;
case "este":
    origen.IdEste = nodoDestino.Id;
    nodoDestino.TipoViaOeste = origen.TipoViaEste;
    if (esBidireccional) nodoDestino.IdOeste = origen.Id;
    break;
case "oeste":
    origen.IdOeste = nodoDestino.Id;
    nodoDestino.TipoViaEste = origen.TipoViaOeste;
    if (esBidireccional) nodoDestino.IdEste = origen.Id;
    break;
default:
    mensaje = "Dirección inválida.";
    return;
}

```

```

Db.Nodos.Update(origen);
Db.Nodos.Add(nodoDestino);
Db.SaveChanges();

```

```

mensaje = $"Nodo '{nodoDestino.Id}' conectado con éxito.";
nodoDestino = new Nodo();
direccion = "";
esBidireccional = false;
}
}

```

HistorialReportes.razor

```

@page "/historial-reportes"
@using Proyecto1.Models
@using Proyecto1.Data
@inject RedVialContext Db
@inject NavigationManager Nav

```

```
<PageTitle>Historial de Reportes</PageTitle>
```

```
<div class="container mt-4 p-4 shadow rounded bg-white">
    <h3 class="text-center text-primary mb-4">  Historial de Reportes Guardados</h3>

    @if (historial.EstaVacia())
    {
        <div class="alert alert-info text-center">No hay reportes guardados aún.</div>
    }
    else
    {
        <table class="table table-bordered table-striped">
            <thead class="table-dark text-center">
                <tr>
                    <th>ID</th>
                    <th>Fecha</th>
                    <th>Intersección Más Congestionada</th>
                    <th>Cuellos de Botella</th>
                    <th>Nodos</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var r in historial.ObtenerTodos())
                {
                    <tr class="text-center">
                        <td>@r.Id</td>
                        <td>@r.FechaGeneracion.ToString("dd/MM/yyyy HH:mm")</td>
                        <td>@r.InterseccionMasCongestionada</td>
                        <td>@r.CuellosBotellaTexto</td>
                        <td>@r.NodosTexto</td>
                    </tr>
                }
            </tbody>
        </table>

        <div class="d-flex gap-2 mt-4">
            <button class="btn btn-danger" @onclick="EliminarHistorial">  Eliminar Historial</button>
            <button class="btn btn-success" @onclick="DescargarPdf">  Descargar PDF</button>
        </div>
    }
</div>
```

```

@code {
    private ListaReportes historial = new();

    protected override void OnInitialized()
    {
        foreach (var reporte in Db.Reportes.OrderBy(r => r.FechaGeneracion))
        {
            historial.Agregar(reporte);
        }
    }

    private void EliminarHistorial()
    {
        Db.Reportes.RemoveRange(Db.Reportes);
        Db.SaveChanges();
        Nav.NavigateTo(Nav.Uri, forceLoad: true);
    }

    private void DescargarPdf()
    {
        Nav.NavigateTo("/descargar-historial", forceLoad: true);
    }
}

```

Home.razor

```

@page "/"
@using Proyecto1.Models
@using Proyecto1.Data
@using Proyecto1.Services
@inject RedVialContext Db
@inject ReporteService Reportes
@inject PdfGeneratorService Pdf
@inject NavigationManager Nav

<PageTitle>Red Vial</PageTitle>

<div class="container mt-4 p-4 shadow rounded bg-white">
    <h3 class="text-center text-primary mb-4">Estado Actual de la Red Vial</h3>

    @if (listaNodos.EstaVacia())
    {

```

```

<p class="text-muted">No hay intersecciones agregadas todavía.</p>
}
else
{
    <table class="table table-bordered table-hover mt-3">
        <thead class="table-dark text-center">
            <tr>
                <th>ID</th>
                <th>Vehículos</th>
                <th>Semáforo</th>
                <th>Tiempo (s)</th>
                <th>Norte</th>
                <th>Sur</th>
                <th>Este</th>
                <th>Oeste</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var nodo in listaNodos.ObtenerTodos())
            {
                <tr class="text-center">
                    <td>@nodo.Id</td>
                    <td>@nodo.VehiculosEnEspera</td>
                    <td>@nodo.EstadoSemaforo</td>
                    <td>@nodo.TiempoPromedioCruce</td>
                    <td>@(nodo.Norte?.Id ?? "-")</td>
                    <td>@(nodo.Sur?.Id ?? "-")</td>
                    <td>@(nodo.Este?.Id ?? "-")</td>
                    <td>@(nodo.Oeste?.Id ?? "-")</td>
                </tr>
            }
        </tbody>
    </table>

    @if (nodoMasCongestionado != null)
    {
        <div class="alert alert-warning mt-4">
            <strong>❗ Intersección más congestionada:</strong>
            @nodoMasCongestionado.Id con @nodoMasCongestionado.VehiculosEnEspera vehículos en
            espera.
        </div>
    }
}

```

```

    }

    @if (!cuellosBotella.EstaVacia())
    {
        <div class="alert alert-danger mt-4">
            <h5>⚠️ Cuellos de Botella Detectados:</h5>
            <ul>
                @foreach (var nodo in cuellosBotella.ObtenerTodos())
                {
                    <li>
                        <strong>@nodo.Id:</strong> @nodo.VehiculosEnEspera vehículos – Semáforo:
                        @nodo.EstadoSemaforo – Tiempo: @nodo.TiempoPromedioCruce s
                    </li>
                }
            </ul>
        </div>
    }

<div class="mt-4 d-flex flex-wrap gap-2">
    <button class="btn btn-danger" @onclick="EliminarTodo">🗑 Eliminar Todo</button>
    <button class="btn btn-success" @onclick="GuardarReporte">💾 Guardar Reporte</button>
    <button class="btn btn-primary" @onclick="DescargarPdf">📄 Descargar PDF</button>
    <button class="btn btn-secondary" @onclick="IrAHistorial">📁 Ver Historial de Reportes</button>
</div>
}

</div>

@code {
    private ListaNodos listaNodos = new();
    private Nodo? nodoMasCongestionado;
    private ListaNodos cuellosBotella = new();

    protected override void OnInitialized()
    {
        var nodos = Db.Nodos.ToList();
        var mapa = nodos.ToDictionary(n => n.Id);

        foreach (var nodo in nodos)
        {
            if (!string.IsNullOrEmpty(nodo.IdNorte) && mapa.ContainsKey(nodo.IdNorte)) nodo.Norte =
                mapa[nodo.IdNorte];
        }
    }
}

```

```

        if (!string.IsNullOrEmpty(nodo.IdSur) && mapa.ContainsKey(nodo.IdSur)) nodo.Sur =
mapa[nodo.IdSur];
        if (!string.IsNullOrEmpty(nodo.IdEste) && mapa.ContainsKey(nodo.IdEste)) nodo.Este =
mapa[nodo.IdEste];
        if (!string.IsNullOrEmpty(nodo.IdOeste) && mapa.ContainsKey(nodo.IdOeste)) nodo.Oeste =
mapa[nodo.IdOeste];

        listaNodos.Agregar(nodo);

        if (nodo.VehiculosEnEspera >= 10 && nodo.EstadoSemaforo == "Rojo" &&
nodo.TiempoPromedioCruce >= 30)
            cuellosBotella.Agregar(nodo);
    }

    nodoMasCongestionado = nodos.OrderByDescending(n => n.VehiculosEnEspera).FirstOrDefault();
}

private void EliminarTodo()
{
    Db.Nodos.RemoveRange(Db.Nodos);
    Db.SaveChanges();
    Nav.NavigateTo(Nav.Uri, forceLoad: true);
}

private void GuardarReporte()
{
    string nodos = string.Join("; ", listaNodos.ObtenerTodos().Select(n => $"{{n.Id}} [{n.EstadoSemaforo}}\n{{n.VehiculosEnEspera}}"));
    string interseccion = nodoMasCongestionado?.Id ?? "Ninguna";
    string cuellos = string.Join("; ", cuellosBotella.ObtenerTodos().Select(n => n.Id));
    Reportes.GuardarReporte(nodos, interseccion, cuellos);
}

private void DescargarPdf()
{
    Nav.NavigateTo("/descargar-reporte", forceLoad: true);
}

private void IrAHistorial()
{
    Nav.NavigateTo("/historial-reportes");
}

```

```

        }
    }

SimularFlujo.razor

@page "/simular-flujo"
@using Proyecto1.Models
@inject Proyecto1.Services.RedVial Red

<PageTitle>Simular Flujo</PageTitle>

<div class="container mt-4 p-4 shadow rounded bg-white">
    <h3 class="text-center text-primary mb-4">Simulación de Flujo Vehicular</h3>

    <EditForm Model="datos" OnValidSubmit="EjecutarSimulacion">
        <DataAnnotationsValidator />
        <ValidationSummary />

        <div class="mb-3">
            <label class="form-label">ID de Intersección Origen</label>
            <InputText class="form-control" @bind-Value="datos.IdOrigen" />
        </div>

        <div class="mb-3">
            <label class="form-label">ID de Intersección Destino</label>
            <InputText class="form-control" @bind-Value="datos.IdDestino" />
        </div>

        <button type="submit" class="btn btn-primary">Simular Flujo</button>
    </EditForm>

    @if (!resultado.EstaVacio())
    {
        <div class="mt-4">
            <h5 class="text-success">Ruta Encontrada:</h5>

            <div class="d-flex flex-wrap align-items-center">
                @foreach (var paso in resultado.ObtenerRuta())
                {
                    var colorSemaforo = paso.EstadoSemaforo.ToLower() switch
                    {
                        "verde" => "●",

```

```

        "amarillo" => "🟡",
        "rojo" => "🔴",
        _ => "⚪"
    };

    <div class="ruta-paso d-flex align-items-center me-2">
        <span class="fs-5 me-1">🚦 @colorSemaforo</span>
        <div class="interseccion">@paso.Id</div>
        <span class="mx-2 fs-4 text-secondary">➡</span>
        <span>🚗 </span>
    </div>
}

</div>

<p class="mt-3 fw-bold">
    <strong>Tiempo Total Estimado:</strong> @resultado.TiempoTotal() segundos
</p>
</div>
}

else if (mostrarMensaje)
{
    <div class="alert alert-danger mt-4">
        No se encontró un camino desde <strong>@datos.IdOrigen</strong> hasta
<strong>@datos.IdDestino</strong>.
    </div>
}
</div>

@code {
    private CaminoNodos resultado = new CaminoNodos();
    private bool mostrarMensaje = false;

    private NodoDatos datos = new();

    private class NodoDatos
    {
        public string IdOrigen { get; set; } = "";
        public string IdDestino { get; set; } = "";
    }

    private void EjecutarSimulacion()

```

```

    {
        var resultadoTemp = Red.EncontrarCamino(datos.IdOrigen, datos.IdDestino);
        if (resultadoTemp == null || resultadoTemp.EstaVacio())
        {
            mostrarMensaje = true;
            resultado = new CaminoNodos();
        }
        else
        {
            resultado = resultadoTemp;
            mostrarMensaje = false;
        }
    }
}

```



PdfGeneratorService.cs

```

using QuestPDF.Fluent;
using QuestPDF.Helpers;
using QuestPDF.Infrastructure;
using Proyecto1.Models;

namespace Proyecto1.Services
{
    public class PdfGeneratorService
    {
        public byte[] GenerarPdf(ListaNodos nodos, Nodo? masCongestionado, ListaNodos cuellos)
        {
            var fecha = DateTime.Now;

            var documento = Document.Create(container =>
            {
                container.Page(page =>
                {
                    page.Size(PageSizes.A4);
                    page.Margin(40);
                    page.Content().Column(col =>
                    {
                        col.Item().Text($"Reporte de Red Vial - {fecha:dd/MM/yyyy HH:mm}").Bold().FontSize(16);
                    });
                });
            });
        }
    }
}

```

```

col.Item().Text(" ");

col.Item().Text("Intersección más congestionada:").Bold();
if (masCongestionado != null)
    col.Item().Text($"{masCongestionado.Id} con {masCongestionado.VehiculosEnEspera}
vehículos");

col.Item().Text(" ");

col.Item().Text("Intersecciones con Cuello de Botella:").Bold();
if (!cuellos.EstaVacia())
{
    foreach (var n in cuellos.ObtenerTodos())
    {
        col.Item().Text($"- {n.Id}: {n.VehiculosEnEspera} vehículos, {n.EstadoSemaforo},
{n.TiempoPromedioCruce}s");
    }
}
else
{
    col.Item().Text("Ninguna.");
}

col.Item().Text(" ");
col.Item().Text("Tabla de Intersecciones:").Bold();

col.Item().Table(t =>
{
    t.ColumnsDefinition(c =>
    {
        for (int i = 0; i < 8; i++) c.RelativeColumn();
    });
}

t.Header(h =>
{
    h.Cell().Element(CellStyle).Text("ID");
    h.Cell().Element(CellStyle).Text("Vehículos");
    h.Cell().Element(CellStyle).Text("Semáforo");
    h.Cell().Element(CellStyle).Text("Tiempo");
    h.Cell().Element(CellStyle).Text("Norte");
    h.Cell().Element(CellStyle).Text("Sur");
}

```

```

        h.Cell().Element(CellStyle).Text("Este");
        h.Cell().Element(CellStyle).Text("Oeste");

        static IContainer CellStyle(IContainer container) =>

container.Padding(5).Background(Colors.Grey.Lighten2).Border(1).BorderColor(Colors.Grey.Darken1);
    });

foreach (var nodo in nodos.ObtenerTodos())
{
    t.Cell().Element(CellStyle).Text(nodo.Id);
    t.Cell().Element(CellStyle).Text(nodo.VehiculosEnEspera.ToString());
    t.Cell().Element(CellStyle).Text(nodo.EstadoSemaforo);
    t.Cell().Element(CellStyle).Text(nodo.TiempoPromedioCruce.ToString());
    t.Cell().Element(CellStyle).Text(nodo.Norte?.Id ?? "-");
    t.Cell().Element(CellStyle).Text(nodo.Sur?.Id ?? "-");
    t.Cell().Element(CellStyle).Text(nodo.Este?.Id ?? "-");
    t.Cell().Element(CellStyle).Text(nodo.Oeste?.Id ?? "-");

    static IContainer CellStyle(IContainer container) =>
        container.Border(1).BorderColor(Colors.Grey.Lighten2).Padding(5);
    }

    });
};

});

return documento.GeneratePdf();
}
}
}

PdfHistorialService.cs
```

```

using Proyecto1.Models;
using QuestPDF.Fluent;
using QuestPDF.Helpers;
using QuestPDF.Infrastructure;

namespace Proyecto1.Services
{
    public class PdfHistorialService
```

```

{
    public byte[] GenerarHistorialPdf(ListaReportes reportes)
    {
        var fecha = DateTime.Now;

        var doc = Document.Create(container =>
        {
            container.Page(page =>
            {
                page.Size(PageSizes.A4.Landscape());
                page.Margin(40);
                page.DefaultTextStyle(x => x.FontSize(11));

                page.Content().Column(col =>
                {
                    col.Item().Text($"Historial de Reportes - {fecha:dd/MM/yyyy HH:mm}")
                        .Bold().FontSize(16);
                    col.Item().Text(" ");

                    col.Item().Table(t =>
                    {
                        t.ColumnsDefinition(columns =>
                        {
                            columns.RelativeColumn(1); // ID
                            columns.RelativeColumn(2); // Fecha
                            columns.RelativeColumn(2); // Intersección
                            columns.RelativeColumn(3); // Cuellos
                            columns.RelativeColumn(4); // Nodos
                        });
                    });

                    t.Header(header =>
                    {
                        header.Cell().Element(CellStyle).Text("ID");
                        header.Cell().Element(CellStyle).Text("Fecha");
                        header.Cell().Element(CellStyle).Text("Intersección Congestionada");
                        header.Cell().Element(CellStyle).Text("Cuellos de Botella");
                        header.Cell().Element(CellStyle).Text("Nodos");
                    });
                });
            });
        });

        static IContainer CellStyle(IContainer container) =>
        container.Padding(5).Background(Colors.Grey.Lighten2).Border(1).BorderColor(Colors.Grey.Darken1);
    }
}

```

```

    });

    foreach (var reporte in reportes.ObtenerTodos())
    {
        t.Cell().Element(CellStyle).Text(reporte.Id.ToString());
        t.Cell().Element(CellStyle).Text(reporte.FechaGeneracion.ToString("dd/MM/yyyy
HH:mm"));

        t.Cell().Element(CellStyle).Text(reporte.InterseccionMasCongestionada);
        t.Cell().Element(CellStyle).Text(reporte.CuellosBotellaTexto);
        t.Cell().Element(CellStyle).Text(reporte.NodosTexto);

        static IContainer CellStyle(IContainer container) =>
            container.Border(1).BorderColor(Colors.Grey.Lighten2).Padding(5);
    }
}

};

};

return doc.GeneratePdf();
}
}
}
}

RedVial.cs
```

```

using Proyecto1.Data;
using Proyecto1.Models;

namespace Proyecto1.Services
{
    public class RedVial
    {
        private readonly RedVialContext _context;

        public RedVial(RedVialContext context)
        {
            _context = context;
        }

        public string AgregarNodoCentral(Nodo nodo)
        {

```

```

        if (_context.Nodos.Any(n => n.Id == nodo.Id))
            return $"Ya existe una intersección con ID '{nodo.Id}'.";

        _context.Nodos.Add(nodo);
        _context.SaveChanges();
        return $"Nodo '{nodo.Id}' agregado correctamente.";
    }

public string Conectar(string idOrigen, string direccion, Nodo destino, bool bidireccional = false)
{
    var origen = _context.Nodos.FirstOrDefault(n => n.Id == idOrigen);
    if (origen == null) return $"No se encontró el nodo con ID '{idOrigen}'.";

    if (_context.Nodos.Any(n => n.Id == destino.Id))
        return $"Ya existe una intersección con ID '{destino.Id}'.";

    // Guardamos el nuevo nodo destino
    _context.Nodos.Add(destino);

    // Conexión del nodo origen hacia el nodo destino
    switch (direccion.ToLower())
    {
        case "norte": origen.IdNorte = destino.Id; break;
        case "sur": origen.IdSur = destino.Id; break;
        case "este": origen.IdEste = destino.Id; break;
        case "oeste": origen.IdOeste = destino.Id; break;
        default: return "Dirección inválida. Usa norte, sur, este u oeste.";
    }

    // Si es bidireccional, conectar también desde destino a origen
    if (bidireccional)
    {
        switch (direccion.ToLower())
        {
            case "norte": destino.IdSur = origen.Id; break;
            case "sur": destino.IdNorte = origen.Id; break;
            case "este": destino.IdOeste = origen.Id; break;
            case "oeste": destino.IdEste = origen.Id; break;
        }
    }
}

```

```

        _context.SaveChanges();
        return $"Nodo '{destino.Id}' conectado al {direccion} de '{origen.Id}'" +
            (bidireccional ? " (conexión bidireccional)." : ".");
    }

    public CaminoNodos? EncontrarCamino(string idOrigen, string idDestino)
    {
        var nodos = _context.Nodos.ToList();
        var mapa = nodos.ToDictionary(n => n.Id);

        // Reconstrucción en memoria
        foreach (var nodo in nodos)
        {
            if (!string.IsNullOrEmpty(nodo.IdNorte) && mapa.ContainsKey(nodo.IdNorte))
                nodo.Norte = mapa[nodo.IdNorte];
            if (!string.IsNullOrEmpty(nodo.IdSur) && mapa.ContainsKey(nodo.IdSur))
                nodo.Sur = mapa[nodo.IdSur];
            if (!string.IsNullOrEmpty(nodo.IdEste) && mapa.ContainsKey(nodo.IdEste))
                nodo.Este = mapa[nodo.IdEste];
            if (!string.IsNullOrEmpty(nodo.IdOeste) && mapa.ContainsKey(nodo.IdOeste))
                nodo.Oeste = mapa[nodo.IdOeste];
        }

        if (!mapa.ContainsKey(idOrigen) || !mapa.ContainsKey(idDestino))
            return null;

        var origen = mapa[idOrigen];
        var destino = mapa[idDestino];
        var visitados = new HashSet<string>();
        var camino = new CaminoNodos();

        if (Buscar(origen, destino, visitados, camino))
            return camino;
    }

    private bool Buscar(Nodo actual, Nodo destino, HashSet<string> visitados, CaminoNodos camino)
    {
        if (visitados.Contains(actual.Id)) return false;

```

```

visitados.Add(actual.Id);
camino.AgregarPaso(actual);

if (actual.Id == destino.Id)
    return true;

if (actual.Norte != null && Buscar(actual.Norte, destino, visitados, camino)) return true;
if (actual.Sur != null && Buscar(actual.Sur, destino, visitados, camino)) return true;
if (actual.Este != null && Buscar(actual.Este, destino, visitados, camino)) return true;
if (actual.Oeste != null && Buscar(actual.Oeste, destino, visitados, camino)) return true;

camino.EliminarUltimoPaso();
return false;
}
}
}

```

ReporteService.cs

```

using Proyecto1.Data;
using Proyecto1.Models;

namespace Proyecto1.Services
{
    public class ReporteService
    {
        private readonly RedVialContext _context;

        public ReporteService(RedVialContext context)
        {
            _context = context;
        }

        public void GuardarReporte(string nodos, string interseccion, string cuellosBotella)
        {
            var reporte = new ReporteRedVial
            {
                NodosTexto = nodos,
                InterseccionMasCongestionada = interseccion,
                CuellosBotellaTexto = cuellosBotella,
                FechaGeneracion = DateTime.Now
            };
        }
    }
}

```

```

        _context.Reportes.Add(report);
        _context.SaveChanges();
    }
    public void EliminarHistorial()
    {
        _context.Reportes.RemoveRange(_context.Reportes);
        _context.SaveChanges();
    }
}

```



NavMenu.razor

```

<nav>
    <h5 class="text-primary mb-4"> 🚗 Red Vial Urbana</h5>
    <ul class="nav flex-column">
        <li class="nav-item mb-2">
            <NavLink class="nav-link" href="/">
                 Inicio
            </NavLink>
        </li>
        <li class="nav-item mb-2">
            <NavLink class="nav-link" href="/agregar-nodo">
                 Agregar Intersección
            </NavLink>
        </li>
        <li class="nav-item mb-2">
            <NavLink class="nav-link" href="/conectar-nodos">
                 Conectar Intersecciones
            </NavLink>
        </li>
        <li class="nav-item mb-2">
            <NavLink class="nav-link" href="/simular-flujo">
                 Simular Flujo Vehicular
            </NavLink>
        </li>
    </ul>
</nav>

```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using Proyecto1.Data;
using Proyecto1.Models;
using Proyecto1.Services;
using QuestPDF.Infrastructure;

QuestPDF.Settings.License = LicenseType.Community;

var builder = WebApplication.CreateBuilder(args);

// Base de datos
builder.Services.AddDbContext<RedVialContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Servicios Blazor
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();

// Servicios personalizados
builder.Services.AddScoped<ReporteService>();
builder.Services.AddScoped<RedVial>();
builder.Services.AddScoped<PdfGeneratorService>();
builder.Services.AddScoped<PdfHistorialService>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.MapBlazorHub();
app.MapFallbackToPage("/_Host");
```

```

app.MapGet("/descargar-reporte", async (HttpContext http, PdfGeneratorService pdf, RedVialContext db) =>
{
    var nodos = new ListaNodos();
    var mapa = new Dictionary<string, Nodo>();

    foreach (var n in db.Nodos)
    {
        nodos.Agregar(n);
        mapa[n.Id] = n;
    }

    foreach (var n in nodos.ObtenerTodos())
    {
        if (!string.IsNullOrEmpty(n.IdNorte) && mapa.ContainsKey(n.IdNorte)) n.Norte = mapa[n.IdNorte];
        if (!string.IsNullOrEmpty(n.IdSur) && mapa.ContainsKey(n.IdSur)) n.Sur = mapa[n.IdSur];
        if (!string.IsNullOrEmpty(n.IdEste) && mapa.ContainsKey(n.IdEste)) n.Este = mapa[n.IdEste];
        if (!string.IsNullOrEmpty(n.IdOeste) && mapa.ContainsKey(n.IdOeste)) n.Oeste = mapa[n.IdOeste];
    }
}

Nodo? masCongestionado = null;
var cuellos = new ListaNodos();

foreach (var n in nodos.ObtenerTodos())
{
    if (masCongestionado == null || n.VehiculosEnEspera > masCongestionado.VehiculosEnEspera)
        masCongestionado = n;

    if (n.VehiculosEnEspera >= 10 && n.EstadoSemaforo == "Rojo" && n.TiempoPromedioCruce >= 30)
        cuellos.Agregar(n);
}

var contenido = pdf.GenerarPdf(nodos, masCongestionado, cuellos);

var nombreArchivo = $"reporte-redvial-{DateTime.Now:yyyy-MM-dd_HH-mm}.pdf";
http.Response.Headers.ContentDisposition = $"attachment; filename={nombreArchivo}";
http.Response.ContentType = "application/pdf";
await http.Response.Body.WriteAsync(contenido);
});

```

```

app.MapGet("/descargar-historial", async (HttpContext http, PdfHistorialService pdf, RedVialContext db) =>

```

```
{
    var reportesDb = db.Reportes.OrderBy(r => r.FechaGeneracion);
    var historial = new ListaReportes();

    foreach (var r in reportesDb)
        historial.Agregar(r);

    var contenido = pdf.GenerarHistorialPdf(historial);

    var nombreArchivo = $"historial-reportes-{DateTime.Now:yyyy-MM-dd_HH-mm}.pdf";
    http.Response.Headers.ContentDisposition = $"attachment; filename={nombreArchivo}";
    http.Response.ContentType = "application/pdf";
    await http.Response.Body.WriteAsync(contenido);

});
```

```
app.Run();
```

ID	Vehículos	Semáforo	Tiempo (s)	Norte	Sur	Este	Oeste
1	23	Rojo	23	2	-	-	-
2	34	Amarillo	5	-	1	-	-

Intersección más congestionada: 2 con 34 vehículos en espera.

■ Eliminar Todo ■ Guardar Reporte ■ Descargar PDF ■ Ver Historial de Reportes

ID de Intersección
0

Vehículos en Espera
0

Estado del Semáforo
Rojo

Tiempo Promedio de Tránsito (segundos)
0

Tipo de Vía al Norte
Ninguna

Tipo de Vía al Sur
Ninguna

Tipo de Vía al Este
Ninguna

Tipo de Vía al Oeste
Ninguna

■ Agregar Nodo Central

Conectar Intersecciones

ID de Intersección Origen

Dirección a Conectar

Seleccione una dirección

Conexión bidireccional

Datos de la Nueva Intersección Destino

ID

Vehículos en espera

0

Estado del semáforo

Rojo

Tiempo promedio de tránsito (seg)

0

Tipo de vía al Norte

Ninguna

Tipo de vía al Sur

Ninguna

Tipo de vía al Este

Ninguna

Tipo de vía al Oeste

Ninguna

Conectar

Simulación de Flujo Vehicular

ID de Intersección Origen

ID de Intersección Destino

Simular Flujo

Ruta Encontrada:



Tiempo Total Estimado: 28 segundos

Base de datos

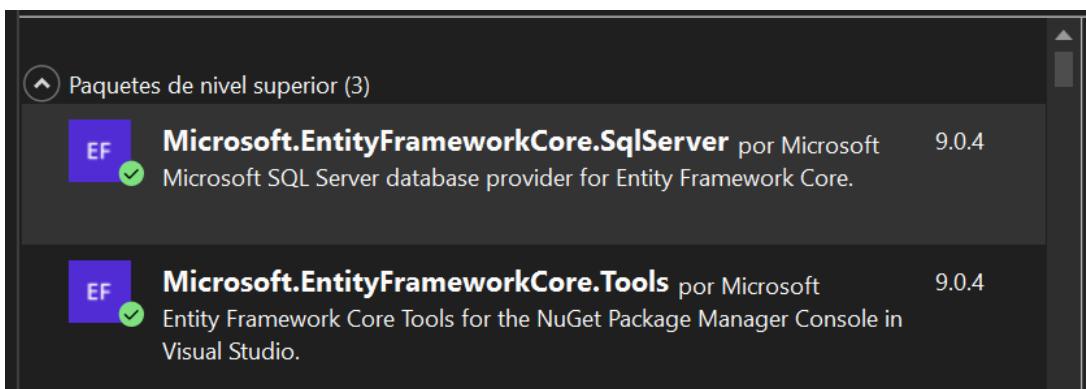
Para el desarrollo de la base de datos, se utilizó el enfoque de migraciones. Inicialmente, se diseñaron las clases de las entidades dentro del proyecto, siguiendo las buenas prácticas de modelado de datos. Posteriormente, se configuró la cadena de conexión hacia SQL Server y se ejecutaron las migraciones correspondientes. Esto permitió crear automáticamente la estructura de la base de datos directamente desde el código, facilitando la sincronización entre el modelo de datos y la base en SQL Server.

En el archivo appsettings.json, se modifica el Server con el nombre correcto de la instancia SQL Server:

Appsettings.json

```
{  
    "ConnectionStrings": {  
        "DefaultConnection":  
            "Server=RG\\SQLEXPRESS01;Database=RedVialDB;Trusted_Connection=True;TrustServerCertificate=True;"  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft.AspNetCore": "Warning"  
        }  
    },  
    "AllowedHosts": "*"  
}
```

Se verifica que Install-Package Microsoft.EntityFrameworkCore.SqlServer y Install-Package Microsoft.EntityFrameworkCore.Tools estén instalados.



Después, accedemos a la Consola del Administrador de Paquetes NuGet, para ejecutar los siguientes comandos:

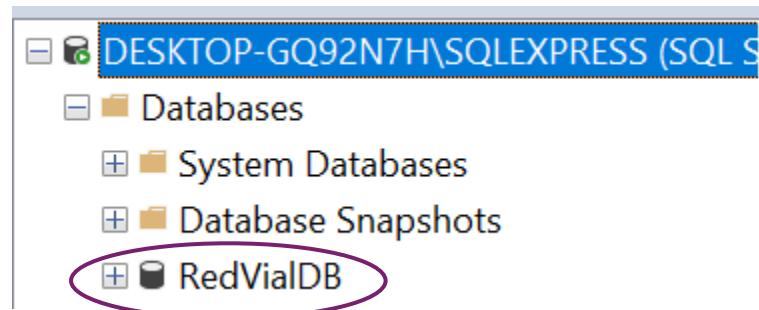
```
Add-Migration CrearTablasIniciales -Context RedVialContext
```

```
PM> Add-Migration CrearTablasIniciales -Context RedVialContext
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> |
```

```
Update-Database -Context RedVialContext
```

```
PM> Update-Database -Context RedVialContext
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (26ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
        CREATE DATABASE [RedVialDB];
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
        IF SERVERPROPERTY('EnginEdition')><5
        BEGIN
            ALTER DATABASE [RedVialDB] SET READ_COMMITTED_SNAPSHOT ON;
        END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Migrations[20401]
    Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
    Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (27ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        EXEC @result = sp_getapplock @Resource = '_EFMigrationsLock', @LockOwner = 'Session', @LockMode = 'Exclusive';
DECLARE @result int;
SELECT @result
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (12ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        IF OBJECT_ID(N'[_EFMigrationsHistory]') IS NULL
        BEGIN
            CREATE TABLE [_EFMigrationsHistory] (
                [MigrationId] nvarchar(150) NOT NULL,
                [ProductVersion] nvarchar(32) NOT NULL,
                ...
            );
        END;
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20250509011102_CrearTablaNodos2'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (601ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        EXEC sp_rename N'[Nodos].[IdNodoSur]', N'IdSur', 'COLUMN';
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        EXEC sp_rename N'[Nodos].[IdNodoOeste]', N'IdOeste', 'COLUMN';
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        EXEC sp_rename N'[Nodos].[IdNodoNorte]', N'IdNorte', 'COLUMN';
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        EXEC sp_rename N'[Nodos].[IdNodoEste]', N'IdEste', 'COLUMN';
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        INSERT INTO [_EFMigrationsHistory] ([MigrationId], [ProductVersion])
        VALUES (N'20250509011102_CrearTablaNodos2', N'9.0.4');
Applying migration '20250511013402_CrearTablasIniciales'.
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20250511013402_CrearTablasIniciales'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        INSERT INTO [_EFMigrationsHistory] ([MigrationId], [ProductVersion])
        VALUES (N'20250511013402_CrearTablasIniciales', N'9.0.4');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    DECLARE @result int;
    EXEC @result = sp_releaseapplock @Resource = '_EFMigrationsLock', @LockOwner = 'Session';
    SELECT @result
Done.
```

Luego de ejecutar los comandos para realizar la migración de la base de datos, se accede a Microsoft SQL Server, para verificar que nuestra base de datos ya está creada.



Conclusión

La realización de este proyecto ha permitido comprender y aplicar de forma práctica los principios fundamentales de las estructuras dinámicas de datos, en el contexto de un sistema real que simula el flujo vehicular dentro de una red vial urbana. A través de esta implementación, lo que fortalece la lógica de programación y estimula una mayor comprensión del manejo de memoria y enlaces entre objetos.

El desarrollo de una funcionalidad que permite simular el flujo vehicular entre dos intersecciones proporcionó una experiencia enriquecedora, tanto desde el punto de vista técnico. La implementación de algoritmos de búsqueda y recorrido permitió detectar rutas posibles, estimar tiempos de desplazamiento y visualizar gráficamente el trayecto, haciendo evidente el comportamiento del sistema frente a condiciones de tráfico variadas. Cabe destacar que uno de los aspectos más desafiantes del proyecto fue mantener la coherencia entre la lógica del backend (listas enlazadas y algoritmos) y la visualización gráfica del frontend (iconos, flechas, vehículos en movimiento). Sin embargo, esta complejidad permitió afianzar habilidades clave de integración, depuración, diseño modular y trabajo estructurado en capas.

Este proyecto no solo cumple con los requerimientos académicos planteados al inicio del curso, sino que también aporta una visión clara del potencial de las estructuras de datos en el desarrollo de soluciones reales y visuales. Representa un ejemplo concreto de cómo la teoría de programación puede ser aplicada para resolver problemas prácticos, fomentando en el estudiante una mentalidad analítica, lógica y creativa, esencial en su formación como futuro ingeniero en sistemas.