Universidad Mariano Gálvez de Guatemala

Ingeniería en Sistemas de Información y Ciencias de la Computación

Programación III

Ing. José Luis Xiloj

Manual Técnico

Jaqueline Yajaira Lorenzana Cisneros 1590-23-12507

Dayli Yadira Santos Herrarte 1590-23-6686

Kimberli Yanet Hernández Rodríguez 1590-23-12625

Robin Elias García Santos 1590-21-7722

Alexander Jossimar García de León 1590-23-18540

José Manuel Monterroso Avalos 1590-23-14680

HistorialReportes.razor

Este archivo define la página que muestra el historial de reportes guardados. Incluye funcionalidades para eliminar los reportes o descargarlos en PDF.

@page "/historial-reportes"

Define la ruta URL de esta página en la aplicación Blazor.

@using Proyecto1.Models

Importa el espacio de nombres que contiene los modelos de datos como ReporteRedVial.

@using Proyecto1.Data

Importa el espacio de nombres que contiene el contexto de base de datos RedVialContext.

@inject RedVialContext Db

Inyecta la instancia del contexto de base de datos para acceder a los datos.

@inject NavigationManager Nav

Inyecta el servicio de navegación para redirigir o recargar páginas.

<PageTitle>Historial de Reportes</PageTitle>

Define el título de la pestaña del navegador.

** **

Contenedor principal con clases Bootstrap para estilo visual.

@if (historial.EstaVacia())

Verifica si la lista de reportes está vacía para mostrar un mensaje.

** No hay reportes guardados aún. **

Mensaje informativo cuando no hay reportes.

<table class="table ...">

Tabla HTML con estilos Bootstrap para mostrar los reportes.

@foreach (var r in historial.ObtenerTodos())

Itera sobre todos los reportes almacenados en la lista.

<td>@r.Id</td>

Muestra el ID del reporte.

<td>@r.FechaGeneracion.ToString(...</td>

Muestra la fecha de generación del reporte con formato.

<td>@r.InterseccionMasCongestionada</td>

Muestra la intersección más congestionada.

<td>@r.CuellosBotellaTexto</td>

Muestra los cuellos de botella identificados en texto.

<td>@r.NodosTexto</td>

Muestra el detalle de nodos involucrados.

<button class="btn btn-danger" @onclick="EliminarHistorial">

Botón para eliminar todos los reportes guardados.

<button class="btn btn-success" @onclick="DescargarPdf">

Botón para descargar los reportes en formato PDF.

private ListaReportes historial = new();

Inicializa la lista enlazada donde se guardarán los reportes en memoria.

protected override void OnInitialized()

Método del ciclo de vida de Blazor que se ejecuta al cargar la página.

foreach (var reporte in Db.Reportes.OrderBy(...))

Ordena los reportes por fecha y los agrega a la lista enlazada.

Db.Reportes.RemoveRange(Db.Reportes);

Elimina todos los registros de reportes en la base de datos.

Db.SaveChanges();

Guarda los cambios en la base de datos luego de eliminar los reportes.

Nav.NavigateTo(Nav.Uri, forceLoad: true);

Recarga la página actual después de eliminar los reportes.

Nav.NavigateTo("/descargar-historial", forceLoad: true);

Navega a la ruta que permite descargar el PDF con el historial.

RedVial.cs

Este archivo contiene la clase RedVial, que proporciona la lógica para agregar nodos, conectarlos y encontrar caminos entre ellos en una red vial urbana.

Análisis línea por línea

using Proyecto1.Data;

Importa el espacio de nombres donde se encuentra el contexto de base de datos.

using Proyecto1.Models;

Importa las definiciones de modelos como Nodo y CaminoNodos.

namespace Proyecto1.Services

Define el espacio de nombres para los servicios del proyecto.

public class RedVial

Declara la clase RedVial, encargada de la lógica de red vial.

private readonly RedVialContext _context;

Campo privado solo lectura para acceder a la base de datos.

public RedVial(RedVialContext context)

Constructor que recibe el contexto de base de datos.

_context = context;

Asigna el contexto inyectado al campo privado para uso interno.

public string AgregarNodoCentral(Nodo nodo)

Método que agrega un nodo si no existe otro con el mismo ID.

if (_context.Nodos.Any(n => n.Id == nodo.Id))

Verifica si ya existe un nodo con el mismo ID.

return $"Ya existe una intersección..."

Retorna un mensaje si el nodo ya existe.

_context.Nodos.Add(nodo);

Agrega el nuevo nodo a la base de datos.

_context.SaveChanges();

Guarda los cambios realizados en la base de datos.

return $"Nodo '{nodo.Id}' agregado correctamente.";

Retorna un mensaje de éxito.

public string Conectar(...)

Conecta un nodo origen con uno destino en una dirección específica.

var origen = _context.Nodos.FirstOrDefault(...)

Busca el nodo origen en la base de datos.

if (origen == null) return ...

Retorna mensaje si el nodo origen no existe.

if (_context.Nodos.Any(n => n.Id == destino.Id))

Verifica si el nodo destino ya existe.

_context.Nodos.Add(destino);

Agrega el nodo destino a la base de datos.

switch (direccion.ToLower())

Determina qué propiedad del nodo origen se debe actualizar según la dirección.

case "norte": origen.IdNorte = destino.Id;

Conecta el nodo origen hacia el norte.

default: return "Dirección inválida..."

Manejo de error si la dirección no es válida.

if (bidireccional)

Si es bidireccional, también conecta el nodo destino hacia el origen.

_context.SaveChanges();

Guarda los cambios en la base de datos.

return $"Nodo '{destino.Id}' conectado..."

Retorna mensaje indicando la conexión realizada.

public CaminoNodos? EncontrarCamino(...)

Busca un camino entre dos nodos usando búsqueda en profundidad.

var nodos = _context.Nodos.ToList();

Obtiene todos los nodos de la base de datos.

var mapa = nodos.ToDictionary(n => n.Id);

Crea un diccionario para acceso rápido por ID.

foreach (var nodo in nodos)

Reconstruye las conexiones entre nodos en memoria.

if (!mapa.ContainsKey(idOrigen) || ...)

Verifica que ambos nodos existan.

var origen = mapa[idOrigen];

Obtiene el nodo origen desde el diccionario.

var camino = new CaminoNodos();

Inicializa una nueva ruta/camino.

if (Buscar(...)) return camino;

Llama al método recursivo de búsqueda.

return null;

Devuelve null si no se encontró camino.

private bool Buscar(...)

Método recursivo para encontrar ruta usando DFS.

if (visitados.Contains(actual.Id))

Evita ciclos verificando si el nodo ya fue visitado.

visitados.Add(actual.Id);

Marca el nodo actual como visitado.

camino.AgregarPaso(actual);

Agrega el nodo actual al camino.

if (actual.Id == destino.Id)

Verifica si se llegó al destino.

Buscar(actual.Norte, ...)

Llama recursivamente hacia el norte, si existe.

camino.EliminarUltimoPaso();

Elimina el último nodo si no lleva al destino.

return false;

Retorna false si no se encuentra una ruta válida.

Program.cs (Configuración de la Aplicación)

Este archivo configura la aplicación ASP.NET Core con Blazor Server, registrando servicios, base de datos y rutas HTTP para generar reportes en PDF.

Análisis línea por línea

using Microsoft.EntityFrameworkCore;

Importa el espacio de nombres para trabajar con Entity Framework Core.

using Proyecto1.Data;

Importa el contexto de base de datos personalizado RedVialContext.

using Proyecto1.Models;

Importa las clases de modelos como Nodo, ListaNodos, etc.

using Proyecto1.Services;

Importa los servicios lógicos del proyecto como RedVial, ReporteService.

using QuestPDF.Infrastructure;

Importa las herramientas de QuestPDF para la generación de documentos PDF.

QuestPDF.Settings.License = LicenseType.Community;

Configura la licencia comunitaria para QuestPDF.

var builder = WebApplication.CreateBuilder(args);

Inicializa la construcción de la aplicación ASP.NET Core.

builder.Services.AddDbContext<RedVialContext>(...)

Registra el contexto de base de datos para inyección de dependencias.

builder.Services.AddRazorPages();

Habilita el soporte para páginas Razor.

builder.Services.AddServerSideBlazor();

Habilita Blazor Server para componentes interactivos.

builder.Services.AddScoped<ReporteService>();

Registra el servicio que gestiona reportes para inyección de dependencias.

builder.Services.AddScoped<RedVial>();

Registra el servicio de lógica vial para inyección.

builder.Services.AddScoped<PdfGeneratorService>();

Registra el servicio para generación de PDFs de reporte.

builder.Services.AddScoped<PdfHistorialService>();

Registra el servicio para generación de PDFs de historial.

var app = builder.Build();

Construye la aplicación con las configuraciones definidas.

if (!app.Environment.IsDevelopment())

Verifica si el entorno no es desarrollo para aplicar manejo de errores.

app.UseExceptionHandler("/Error");

Usa una página de error personalizada en producción.

app.UseHsts();

Agrega encabezados HTTP Strict Transport Security para mejorar la seguridad.

app.UseHttpsRedirection();

Redirecciona todo tráfico HTTP a HTTPS.

app.UseStaticFiles();

Sirve archivos estáticos como CSS, JS o imágenes.

app.UseRouting();

Habilita el enrutamiento dentro de la aplicación.

app.MapBlazorHub();

Mapea la conexión del cliente Blazor con SignalR.

app.MapFallbackToPage("/_Host");

Define la página fallback para rutas no encontradas.

app.MapGet("/descargar-reporte", async (...)

Define un endpoint GET para generar y descargar un reporte de red vial en PDF.

var nodos = new ListaNodos();

Crea una lista enlazada para almacenar los nodos de forma temporal.

var mapa = new Dictionary<string, Nodo>();

Crea un diccionario para acceder rápidamente a nodos por ID.

foreach (var n in db.Nodos)

Recorre todos los nodos en la base de datos para reconstruirlos en memoria.

if (!string.IsNullOrEmpty(n.IdNorte) ...

Reconstruye las conexiones entre nodos en memoria.

Nodo? masCongestionado = null;

Inicializa la variable que almacenará el nodo con más vehículos.

if (n.VehiculosEnEspera >= 10 && ...

Detecta los nodos con cuellos de botella basados en condiciones específicas.

var contenido = pdf.GenerarPdf(...);

Genera el PDF a partir de la estructura de nodos actual.

http.Response.Headers.ContentDisposition = ...

Indica que el PDF se debe descargar como archivo adjunto.

await http.Response.Body.WriteAsync(contenido);

Escribe el contenido del PDF directamente en la respuesta HTTP.

app.MapGet("/descargar-historial", async (...

Define otro endpoint GET para descargar el historial de reportes como PDF.

var reportesDb = db.Reportes.OrderBy(...);

Obtiene los reportes ordenados por fecha de generación.

historial.Agregar(r);

Agrega los reportes a la lista enlazada de historial.

var contenido = pdf.GenerarHistorialPdf(historial);

Genera el PDF del historial de reportes.

await http.Response.Body.WriteAsync(contenido);

Devuelve el archivo PDF generado al usuario.

app.Run();

Ejecuta la aplicación web.

CrearTablaNodos2.cs (Migración)

Este archivo representa una migración de Entity Framework Core utilizada para renombrar columnas en la tabla 'Nodos'. Es parte del control de versiones del esquema de base de datos.

using Microsoft.EntityFrameworkCore.Migrations;

Importa las herramientas necesarias para crear y ejecutar migraciones en EF Core.

#nullable disable

Desactiva las advertencias de anotaciones de nulabilidad para este archivo generado automáticamente.

namespace Proyecto1.Migrations

Define el espacio de nombres donde reside la migración.

public partial class CrearTablaNodos2 : Migration

Declara la clase de migración que extiende de Migration.

protected override void Up(MigrationBuilder migrationBuilder)

Método que define los cambios que se deben aplicar a la base de datos al ejecutar la migración.

migrationBuilder.RenameColumn(...)

Renombra una columna existente en la tabla 'Nodos'. A continuación, se explican cada una.

name: "IdNodoSur", table: "Nodos", newName: "IdSur"

Cambia el nombre de 'IdNodoSur' a 'IdSur'.

name: "IdNodoOeste", table: "Nodos", newName: "IdOeste"

Cambia el nombre de 'IdNodoOeste' a 'IdOeste'.

name: "IdNodoNorte", table: "Nodos", newName: "IdNorte"

Cambia el nombre de 'IdNodoNorte' a 'IdNorte'.

name: "IdNodoEste", table: "Nodos", newName: "IdEste"

Cambia el nombre de 'IdNodoEste' a 'IdEste'.

protected override void Down(MigrationBuilder migrationBuilder)

Método que revierte los cambios aplicados en el método Up al hacer un rollback de la migración.

name: "IdSur", table: "Nodos", newName: "IdNodoSur"

Revierte el nombre de 'IdSur' a 'IdNodoSur'.

name: "IdOeste", table: "Nodos", newName: "IdNodoOeste"

Revierte el nombre de 'IdOeste' a 'IdNodoOeste'.

name: "IdNorte", table: "Nodos", newName: "IdNodoNorte"

Revierte el nombre de 'IdNorte' a 'IdNodoNorte'.

name: "IdEste", table: "Nodos", newName: "IdNodoEste"

Revierte el nombre de 'IdEste' a 'IdNodoEste'.

ListaReportes.cs

Este archivo define una estructura de lista enlazada personalizada para almacenar objetos del tipo ReporteRedVial. La clase permite agregar, recorrer y verificar si la lista está vacía.

using System.Collections.Generic;

Importa colecciones genéricas como IEnumerable, utilizadas para recorrer elementos.

using Proyecto1.Models;

Importa los modelos del proyecto, incluyendo ReporteRedVial.

namespace Proyecto1.Models

Define el espacio de nombres en el que se encuentra la clase ListaReportes.

public class ListaReportes

Declara una clase pública llamada ListaReportes que implementa una lista enlazada.

public class Elemento

Clase interna que representa cada nodo de la lista enlazada.

public ReporteRedVial Valor { get; set; }

Almacena el objeto ReporteRedVial en el nodo.

public Elemento? Siguiente { get; set; }

Referencia al siguiente nodo de la lista.

public Elemento(ReporteRedVial valor)

Constructor que asigna el valor recibido y establece el siguiente nodo como null.

Valor = valor;

Asigna el reporte recibido a la propiedad Valor.

Siguiente = null;

Inicializa la referencia al siguiente nodo como nula.

private Elemento? cabeza;

Campo privado que representa el primer elemento (cabeza) de la lista enlazada.

public void Agregar(ReporteRedVial reporte)

Método público para agregar un nuevo reporte al final de la lista.

var nuevo = new Elemento(reporte);

Crea un nuevo nodo con el reporte proporcionado.

if (cabeza == null)

Verifica si la lista está vacía.

cabeza = nuevo;

Establece el nuevo nodo como cabeza si la lista estaba vacía.

Elemento actual = cabeza;

Variable auxiliar para recorrer la lista.

while (actual.Siguiente != null)

Recorre la lista hasta llegar al último nodo.

actual = actual.Siguiente;

Avanza al siguiente nodo.

actual.Siguiente = nuevo;

Agrega el nuevo nodo al final de la lista.

public IEnumerable<ReporteRedVial> ObtenerTodos()

Permite recorrer todos los elementos de la lista usando yield return.

var actual = cabeza;

Comienza el recorrido desde la cabeza de la lista.

while (actual != null)

Itera mientras existan elementos en la lista.

yield return actual.Valor;

Devuelve el valor del nodo actual.

actual = actual.Siguiente;

Pasa al siguiente nodo.

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

Devuelve true si la lista está vacía (sin elementos).

Configuración, PdfHistorialService y RedVial.razor

Este documento analiza tres archivos: el archivo de configuración appsettings.json, el servicio PdfHistorialService para generar reportes en PDF del historial, y la página principal de Blazor RedVial.razor que muestra el estado de la red vial.

Archivo: appsettings.json

Define la configuración principal de la aplicación, como la conexión a base de datos y los niveles de log de ASP.NET Core.

Análisis

"ConnectionStrings"

Contenedor de las cadenas de conexión para la base de datos.

"DefaultConnection"

Cadena que conecta a SQL Server local con base de datos RedVialDB.

"Logging"

Configuraciones para los niveles de log del sistema.

"Default": "Information"

El nivel de detalle por defecto es 'Information'.

"Microsoft.AspNetCore": "Warning"

Se reduce la verbosidad del logging interno de ASP.NET.

"AllowedHosts": "*"

Permite que cualquier host acceda a la aplicación.

Archivo: PdfHistorialService.cs

Este servicio genera un archivo PDF que contiene el historial completo de reportes guardados en el sistema.

Análisis

using QuestPDF.Fluent;

Importa la API fluida para definir el contenido del documento PDF.

public class PdfHistorialService

Clase encargada de construir el PDF del historial.

public byte[] GenerarHistorialPdf(...)

Método que genera el PDF a partir de una ListaReportes.

var fecha = DateTime.Now;

Captura la fecha actual para incluirla en el encabezado del documento.

page.Size(PageSizes.A4.Landscape());

Configura la página en orientación horizontal tamaño A4.

page.Margin(40);

Establece márgenes alrededor de la página.

t.ColumnsDefinition(...)

Define cinco columnas relativas en la tabla para ID, fecha, intersección, cuellos y nodos.

t.Header(...)

Define los encabezados de la tabla con estilo de fondo y borde.

foreach (var reporte in reportes.ObtenerTodos())

Itera sobre cada reporte para generar una fila en la tabla PDF.

return doc.GeneratePdf();

Genera el PDF final y lo devuelve como arreglo de bytes.

Archivo: RedVial.razor

Esta es la página principal del sistema. Muestra en tabla todos los nodos registrados en la red vial, identifica los cuellos de botella, permite guardar un reporte o descargarlo en PDF.

Análisis

@page "/"

Define la ruta de navegación de la página principal.

@inject RedVialContext Db

Inyección del contexto de base de datos.

@inject ReporteService Reportes

Inyección del servicio que guarda reportes.

@if (listaNodos.EstaVacia())

Muestra mensaje si no hay nodos registrados.

foreach (var nodo in listaNodos.ObtenerTodos())

Itera sobre los nodos para mostrarlos en tabla.

@(nodo.Norte?.Id ?? "-")

Muestra ID del nodo al norte o '-' si no existe conexión.

if (nodoMasCongestionado != null)

Muestra una alerta si existe un nodo con mayor cantidad de vehículos.

if (!cuellosBotella.EstaVacia())

Muestra lista de intersecciones identificadas como cuello de botella.

@onclick="EliminarTodo"

Botón para eliminar todos los nodos en la base de datos.

@onclick="GuardarReporte"

Guarda un reporte con el estado actual de la red vial.

@onclick="DescargarPdf"

Redirige al endpoint que descarga el PDF generado.

OnInitialized()

Carga todos los nodos desde la base de datos, reconstruye el grafo y detecta problemas.

Generación de Reportes, Servicios y Formulario

Cada uno cumple una función específica en la lógica de generación de reportes, almacenamiento y gestión de nodos.

Archivo: PdfGeneratorService.cs

public class PdfGeneratorService

Clase que se encarga de construir el PDF del estado actual de la red vial.

public byte[] GenerarPdf(...)

Método principal que genera el documento a partir de los datos recibidos.

page.Size(PageSizes.A4);

Define el tamaño de la página como A4.

col.Item().Text(...).Bold().FontSize(16);

Agrega un encabezado al documento con fecha de generación.

if (masCongestionado != null)

Verifica si hay una intersección más congestionada y la muestra.

if (!cuellos.EstaVacia())

Muestra la lista de intersecciones con cuello de botella.

col.Item().Table(...)

Crea una tabla con todos los nodos registrados.

foreach (var nodo in nodos.ObtenerTodos())

Itera sobre todos los nodos para agregarlos a la tabla.

return documento.GeneratePdf();

Genera y devuelve el PDF como arreglo de bytes.

Archivo: ReporteService.cs

public class ReporteService

Servicio para guardar y eliminar reportes de la base de datos.

GuardarReporte(...)

Método que crea y guarda un nuevo reporte con los datos actuales.

_context.Reportes.Add(reporte);

Agrega el reporte a la base de datos.

_context.SaveChanges();

Confirma los cambios en la base de datos.

EliminarHistorial()

Elimina todos los registros de reportes existentes.

Archivo: Utilidades.cs

public static class Utilidades

Clase estática que contiene funciones auxiliares.

ConvertirAListaNodos(List<Nodo> lista)

Convierte una lista estándar de nodos a una estructura personalizada ListaNodos.

resultado.Agregar(nodo);

Agrega cada nodo a la lista enlazada personalizada.

Archivo: AgregarNodo.razor

@page "/agregar-nodo"

Define la URL para esta página del formulario.

@inject RedVialContext Db

Permite acceder al contexto de base de datos desde esta página.

<EditForm Model="@nuevoNodo" OnValidSubmit="AgregarNodoCentral">

Formulario que se ejecuta al enviar datos válidos.

@bind-Value="nuevoNodo.Id"

Enlaza el campo de texto con la propiedad Id del nuevo nodo.

<InputSelect ...>

Componente para seleccionar el estado del semáforo.

<button type="submit" ...>

Botón para guardar el nodo en la base de datos.

AgregarNodoCentral()

Método que guarda el nodo ingresado.

mensaje = ...

Muestra un mensaje de confirmación al usuario.

nuevoNodo = new Nodo();

Reinicia el formulario después de guardar.