public class Usuario

{

    public Guid id { get; set; }

    public string nombre { get; set; } = null!;

    public string apellido { get; set; } = null!;

    public string email { get; set; } = null!;

    public string hashpassword { get; set; } = null!;

    public string permisosSerializados { get; set; } = "";

    public List<Permiso> permisos { get; set; } = new();

}

public class IniciarSesion

{

    private readonly IRepositorioUsuario \_repo;

    public IniciarSesion(IRepositorioUsuario repo)

    {

        \_repo = repo;

    }

    public Usuario Ejecutar(string email, string contraseña, ValidadorUsuario validador)

    {

        var usuario = \_repo.ObtenerPorEmail(email)

            ?? throw new EntidadNotFoundException("Usuario no encontrado");

        validador.ValidacionHash(usuario,contraseña);

        return usuario;

    }

}

public class UsuarioAltaUseCase

{

    private readonly IRepositorioUsuario \_repo;

    private readonly IServicioAutorizacion \_autorizacion;

    public UsuarioAltaUseCase(IRepositorioUsuario repo, IServicioAutorizacion autorizacion)

    {

        \_repo = repo;

        \_autorizacion = autorizacion;

    }

    public void Ejecutar(Usuario usuario, string contraseña, Guid idUsuarioLogueado,ValidadorUsuario validador)

    {

        if (\_repo.ListarTodos().Count > 0 && !\_autorizacion.PoseeElPermiso(idUsuarioLogueado, Permiso.UsuarioAlta))

            throw new FalloAutorizacionException();

        validador.Validar(usuario, esNuevo: true);

        validador.ValidarContraseña(contraseña);

        usuario.hashpassword = ServicioDeHash.HashSHA256(contraseña);

        if (\_repo.ListarTodos().Count == 0)

            usuario.permisos = Enum.GetValues<Permiso>().ToList();

        \_repo.Agregar(usuario);

    }

}

public class CrearEventoUseCase {

    private readonly IServicioAutorizacion \_servicioAutorizacion;

    private readonly IRepositorioEventoDeportivo \_repositorioEvento;

    public CrearEventoUseCase(IServicioAutorizacion servicioAutorizacion, IRepositorioEventoDeportivo repositorioEvento)

    {

        \_servicioAutorizacion = servicioAutorizacion;

        \_repositorioEvento = repositorioEvento;

    }

    public void Ejecutar(Guid idUsuario, EventoDeportivo evento,ValidadorEventoDeportivo validador) {

        if (!\_servicioAutorizacion.PoseeElPermiso(idUsuario, Permiso.EventoAlta))

            throw new UnauthorizedAccessException("El usuario no tiene permiso para crear eventos.");

         validador.Validar(evento);

        \_repositorioEvento.Guardar(evento);

    }

}

public class UsuarioBajaUseCase

{

    private readonly IRepositorioUsuario \_repo;

    private readonly IServicioAutorizacion \_autorizacion;

    public UsuarioBajaUseCase(IRepositorioUsuario repo, IServicioAutorizacion autorizacion)

    {

        \_repo = repo;

        \_autorizacion = autorizacion;

    }

    public void Ejecutar(Guid idUsuarioAEliminar, Guid idUsuarioLogueado)

    {

        if (!\_autorizacion.PoseeElPermiso(idUsuarioLogueado, Permiso.UsuarioBaja))

            throw new FalloAutorizacionException();

        var usuario = \_repo.ObtenerPorId(idUsuarioAEliminar)

            ?? throw new EntidadNotFoundException("Usuario no encontrado.");

        \_repo.Eliminar(idUsuarioAEliminar);

    }

}

public class UsuarioModificarUseCase

{

    private readonly IRepositorioUsuario \_repo;

    private readonly IServicioAutorizacion \_autorizacion;

    public UsuarioModificarUseCase(IRepositorioUsuario repo, IServicioAutorizacion autorizacion)

    {

        \_repo = repo;

        \_autorizacion = autorizacion;

    }

    public void Ejecutar(Usuario usuario, Guid idUsuarioLogueado, ValidadorUsuario validador)

    {

        if (!\_autorizacion.PoseeElPermiso(idUsuarioLogueado, Permiso.UsuarioModificacion))

            throw new FalloAutorizacionException();

        var actual = \_repo.ObtenerPorId(usuario.id)

            ?? throw new EntidadNotFoundException("Usuario no encontrado.");

        validador.Validar(usuario, esNuevo: false);

        usuario.hashpassword = actual.hashpassword;

        \_repo.Modificar(usuario);

    }

}

public class ListarUsuariosUseCase

{

    private readonly IRepositorioUsuario \_repo;

    public ListarUsuariosUseCase(IRepositorioUsuario repo)

    {

        \_repo = repo;

    }

    public List<Usuario> Ejecutar()

    {

        return \_repo.ListarTodos();

    }

}

public interface IRepositorioUsuario

{

    void Agregar(Usuario usuario);

    void Modificar(Usuario usuario);

    void Eliminar(Guid Id);

    Usuario? ObtenerPorEmail(string email);

    Usuario? ObtenerPorId(Guid id);

    List<Usuario> ListarTodos();

}

public interface IServicioAutorizacion {

    bool PoseeElPermiso(Guid idUsuario, Permiso permiso);

}

public interface IServicioSesionUsuario

{

    Usuario? UsuarioActual { get; }

    void IniciarSesion(Usuario usuario);

    void CerrarSesion();

}

public enum EstadoAsistencia{ Pendiente,Presente,Ausente }

public enum Permiso {

    EventoAlta, EventoModificacion,EventoBaja,

    ReservaBaja, ReservaAlta, ReservaModificacion,

    UsuarioBaja, UsuarioAlta, UsuarioModificacion

}

public class ServicioAutorizacion : IServicioAutorizacion

{

    private readonly IRepositorioUsuario \_repo;

    public ServicioAutorizacion(IRepositorioUsuario repo)

    {

        \_repo = repo;

    }

    public bool PoseeElPermiso(Guid idUsuario, Permiso permiso)

    {

        var usuario = \_repo.ObtenerPorId(idUsuario);

        if (usuario == null )

            return false;

        return usuario.permisos.Contains(permiso);

    }

}

public static class ServicioDeHash

{

    public static string HashSHA256(string contraseña)

    {

        using var sha = SHA256.Create();

        byte[] bytes = Encoding.UTF8.GetBytes(contraseña);

        byte[] hash = sha.ComputeHash(bytes);

        return Convert.ToHexString(hash);

    }

}

public class ServicioSesionUsuario : IServicioSesionUsuario

{

    public Usuario? UsuarioActual { get; private set; }

    public void IniciarSesion(Usuario? usuario) => UsuarioActual = usuario;

    public void CerrarSesion() => UsuarioActual = null;

    public bool EstaLogeado => UsuarioActual != null;

}

public class ValidadorUsuario

{

    private readonly IRepositorioUsuario \_repo;

    public ValidadorUsuario(IRepositorioUsuario repo)

    {

        \_repo = repo;

    }

    public void Validar(Usuario usuario, bool esNuevo)

    {

        if (string.IsNullOrWhiteSpace(usuario.nombre))

            throw new ValidacionException("El nombre no puede estar vacío.");

        if (string.IsNullOrWhiteSpace(usuario.apellido))

            throw new ValidacionException("El apellido no puede estar vacío.");

        if (string.IsNullOrWhiteSpace(usuario.email))

            throw new ValidacionException("El email no puede estar vacío.");

        var existente = \_repo.ObtenerPorEmail(usuario.email);

        if (existente != null && (esNuevo || existente.id != usuario.id))

            throw new DuplicadoException("Ya existe un usuario con ese email.");

    }

    public void ValidarContraseña(string contraseña)

    {

        if (string.IsNullOrWhiteSpace(contraseña))

            throw new ValidacionException("La contraseña no puede estar vacía.");

        if (contraseña.Length < 8)

            throw new ValidacionException("La contraseña debe tener al menos 8 caracteres.");

    }

    public void ValidacionHash(Usuario usuario, string contraseña)

    {

        string hash =ServicioDeHash.HashSHA256(contraseña);

        if (usuario.hashpassword != hash)

            throw new OperacionInvalidaException("Contraseña incorrecta.");

    }

}

public class RepositorioUsuarioEF : IRepositorioUsuario

{

    private readonly CentroEventosContext \_context;

    public RepositorioUsuarioEF(CentroEventosContext context)

    {

        \_context = context;

    }

    public void Agregar(Usuario usuario)

    {

        \_context.Usuarios.Add(usuario);

        \_context.SaveChanges();

    }

    public void Modificar(Usuario usuario)

    {

        \_context.Usuarios.Update(usuario);

        \_context.SaveChanges();

    }

    public void Eliminar(Guid id)

    {

        var usuario = ObtenerPorId(id);

        if (usuario != null)

        {

            \_context.Remove(usuario);

            \_context.SaveChanges();

        }

    }

    public Usuario? ObtenerPorId(Guid id) => \_context.Usuarios.Find(id);

    public Usuario? ObtenerPorEmail(string email) => \_context.Usuarios.FirstOrDefault(u => u.email == email);

    public List<Usuario> ListarTodos() => \_context.Usuarios.ToList();

}

public class CentroEventosContext : DbContext

{

    public CentroEventosContext() : base()

    {

    }

    public DbSet<Usuario> Usuarios => Set<Usuario>();

    public DbSet<EventoDeportivo> Eventos => Set<EventoDeportivo>();

    public DbSet<Persona> Persona =>Set<Persona>();

    public DbSet<Reserva> Reservas => Set<Reserva>();

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

    protected override void OnModelCreating(ModelBuilder modelBuilder)

    {

        modelBuilder.ApplyConfigurationsFromAssembly(typeof(CentroEventosContext).Assembly);

    }

}

namespace CentroEventos.Repositorios.Contexto

{

    public class CentroEventosContextFactory : IDesignTimeDbContextFactory<CentroEventosContext>

    {

        public CentroEventosContext CreateDbContext(string[] args)

        {

            var optionsBuilder = new DbContextOptionsBuilder<CentroEventosContext>();

            optionsBuilder.UseSqlite("Data Source=centroeventos.db");

            return new CentroEventosContext(optionsBuilder.Options);

        }

    }

}