Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

OBLIGATORIO PROGRAMACIÓN 3

DOCUMENTO

Nicolás Giménez – 291950

Cristian García – 317010

Grupo N3D

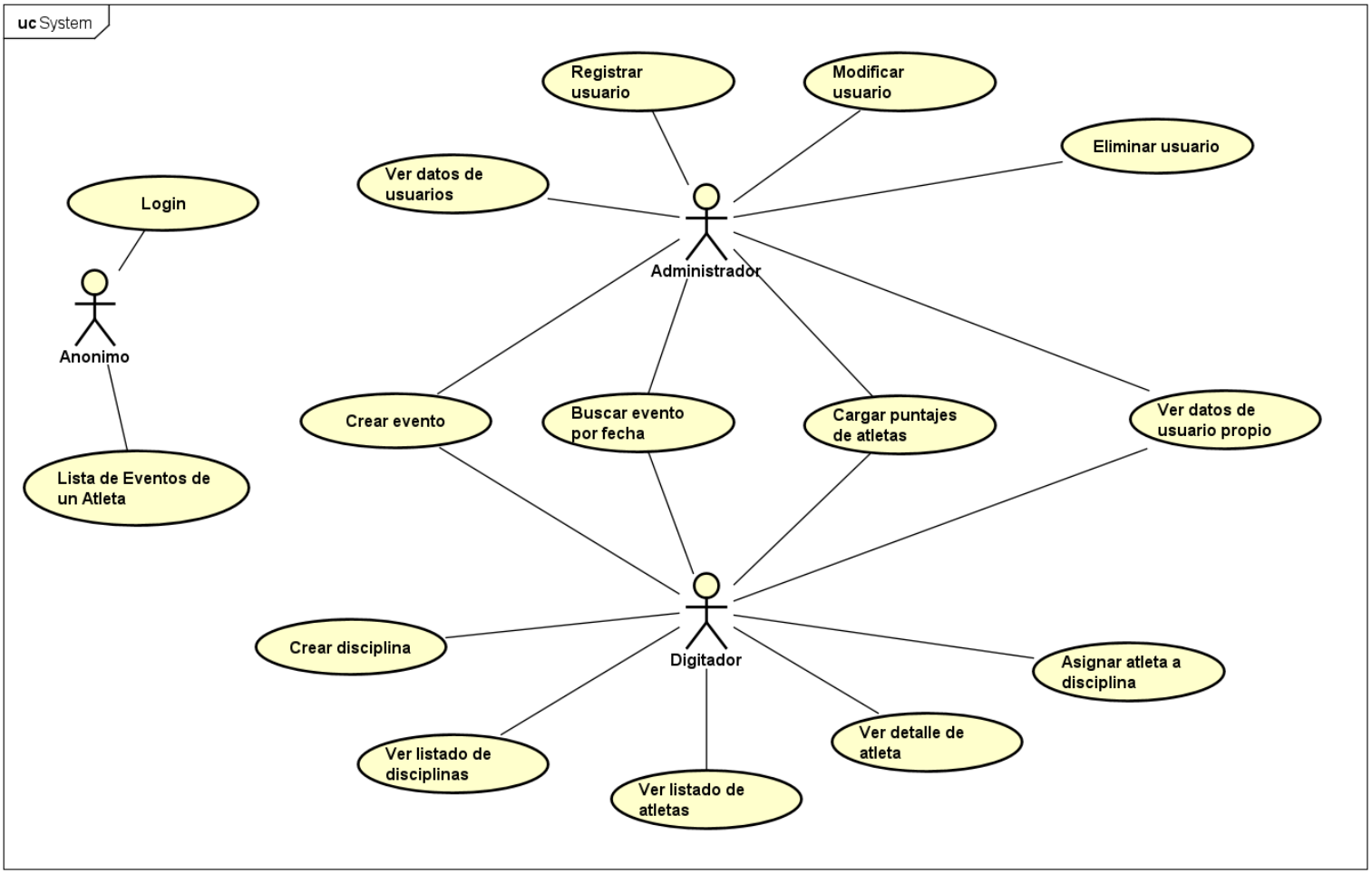
Docente: Liliana Pino

Analista en tecnologías de la información

21/10/2024

Índice

1. Diagrama de casos de uso.



2. Descripción narrativa de casos de uso.

2.1 Alta de atleta.

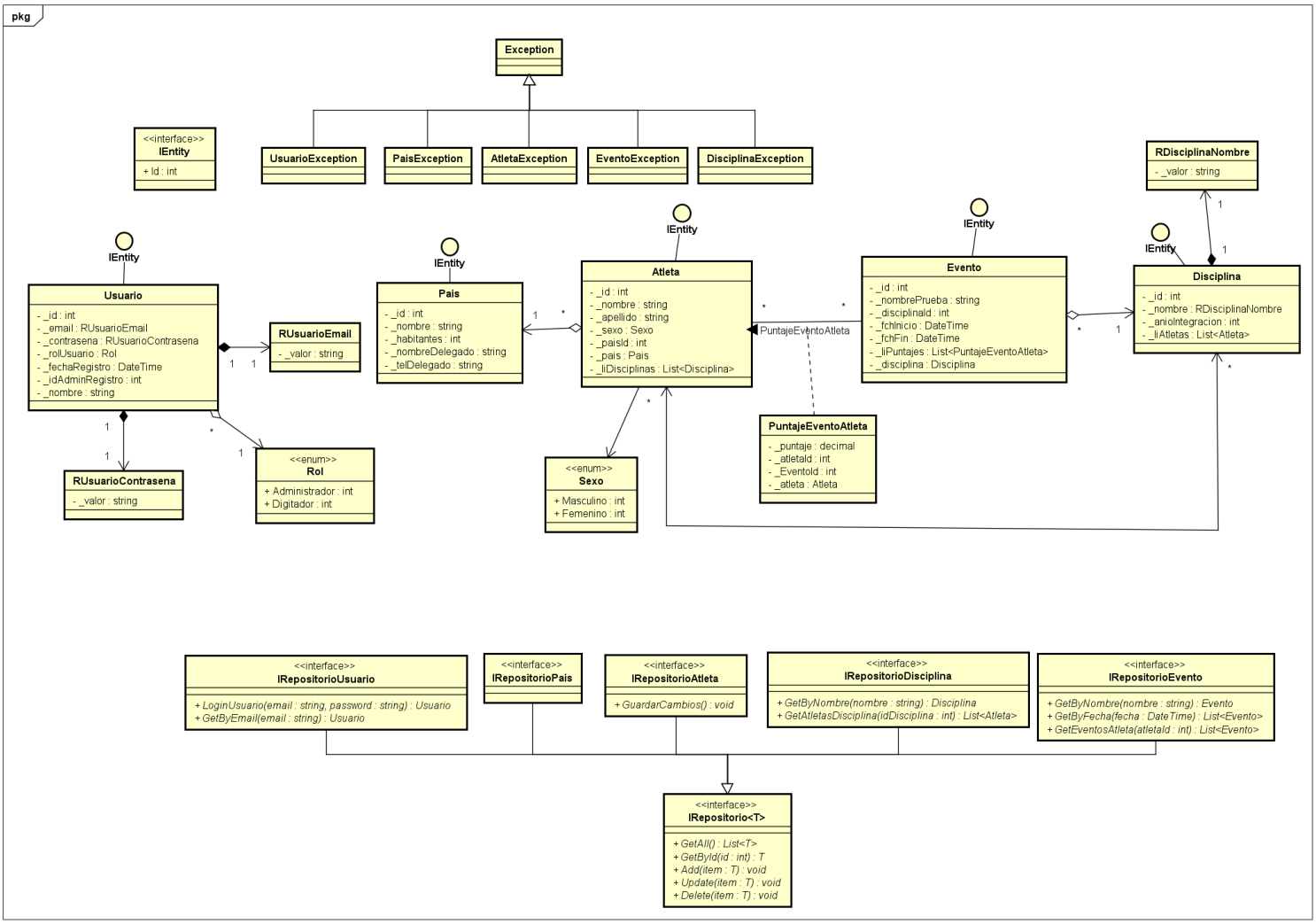
Identificador: CU-2	**Nombre**: Ingreso de puntaje de atleta
Autor:	Nicolás Giménez, Cristian García
Fecha:	21/10/2024
Descripción: Permite cargar el puntaje obtenido en el evento al atleta	
Actor/es: Administrador y Digitador	
Precondiciones: El actor debe estar logueado en el sistema. El atleta debe estar registrado en el evento.	
Flujo Normal: 1-El actor busca los eventos por fecha. 2-El actor selecciona el evento. 3-El actor ingresa el puntaje obtenido por el atleta y envía los datos 4-El sistema valida los datos ingresados, genera un update del puntaje y guarda el cambio en el sistema.	
Flujo/s Alternativo/s: 3ª. El actor carga datos vacíos o menor a 1. 3ª 1. El sistema descarta los datos ingresados, muestra un mensaje de error y permite volver a ingresar los datos correctamente.	
Flujo/s Excepcionales/s: Se interrumpe la comunicación con el servidor. Los datos no son guardados	
Pos condiciones: Se actualizan los datos en el sistema.	

2.2 Ingreso de puntaje de atleta.

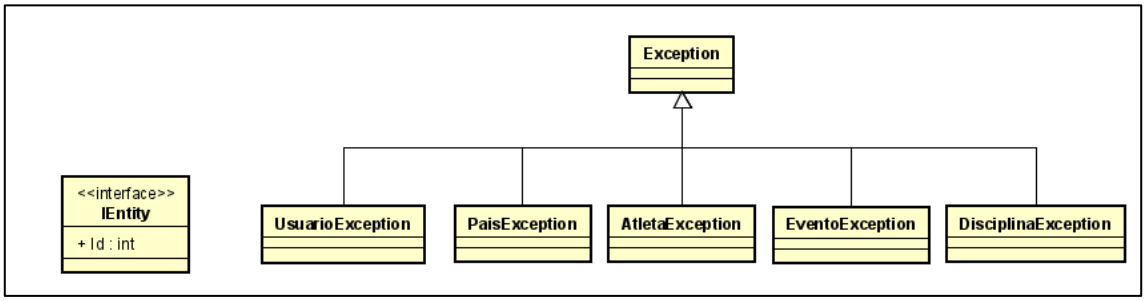
Identificador: CU-1	**Nombre**: Alta de atleta
Autor:	Nicolás Giménez, Cristian García
Fecha:	21/10/2024
Descripción: Permite dar de alta un nuevo atleta al sistema.	
Actor/es: Digitador	
Precondiciones: El actor debe estar logueado en el sistema.	
Flujo Normal: 1-El actor ingresa a la vista con el formulario para cargar los datos del nuevo atleta. 2-El actor completa los datos necesarios para crear un nuevo atleta y envía los datos. 3-El sistema valida los datos ingresados, genera un insert con los datos ingresados y guarda el cambio en el sistema.	
Flujo/s Alternativo/s: 3ª. El actor carga datos vacíos que son requeridos o carga datos inválidos. 3ª 1. El sistema no genera el insert del atleta, muestra un mensaje de error y permite corregir datos.	
Flujo/s Excepcionales/s: Se interrumpe la comunicación con el servidor. Los datos no son guardados	
Pos condiciones: Se da de alta el nuevo atleta en el sistema.	

3. Diagrama de clases (UML).

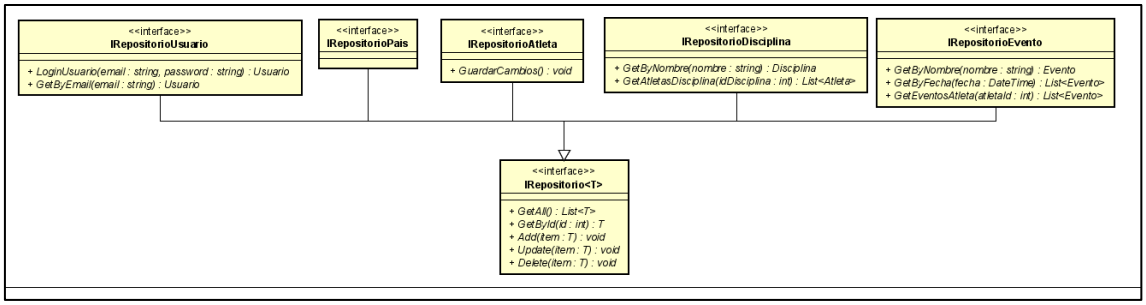
3.1 Lógica de negocio.



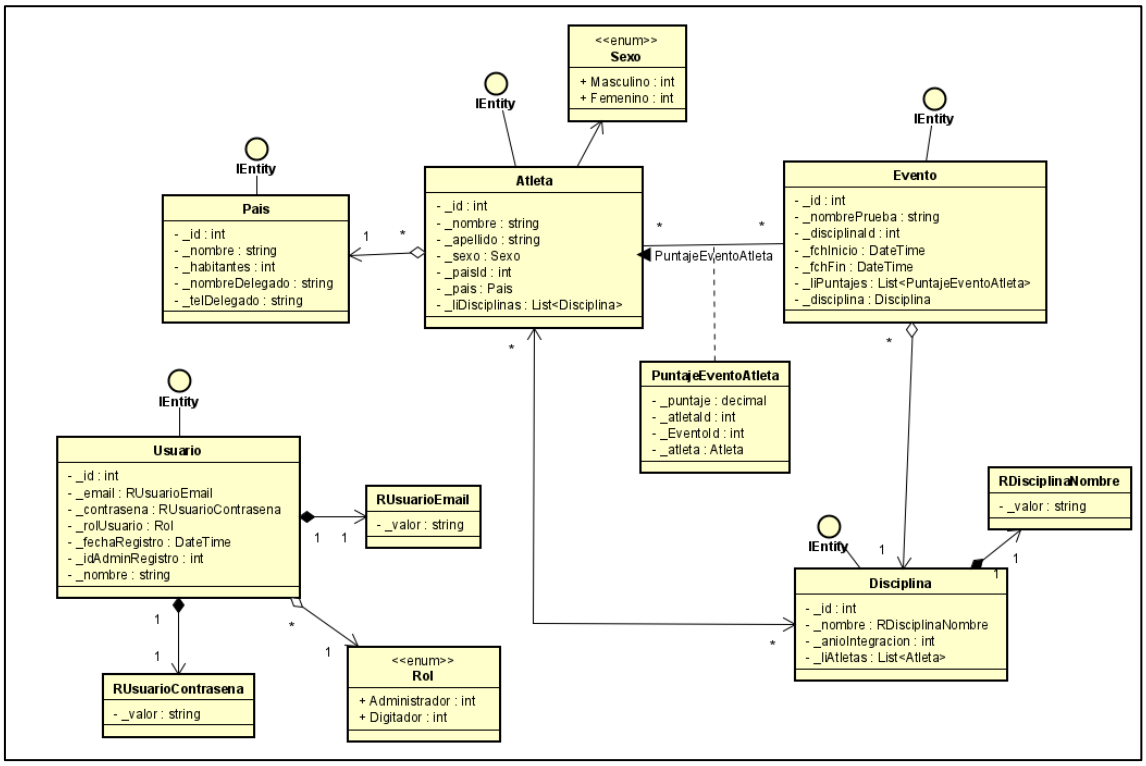
Excepciones e interface de entity.



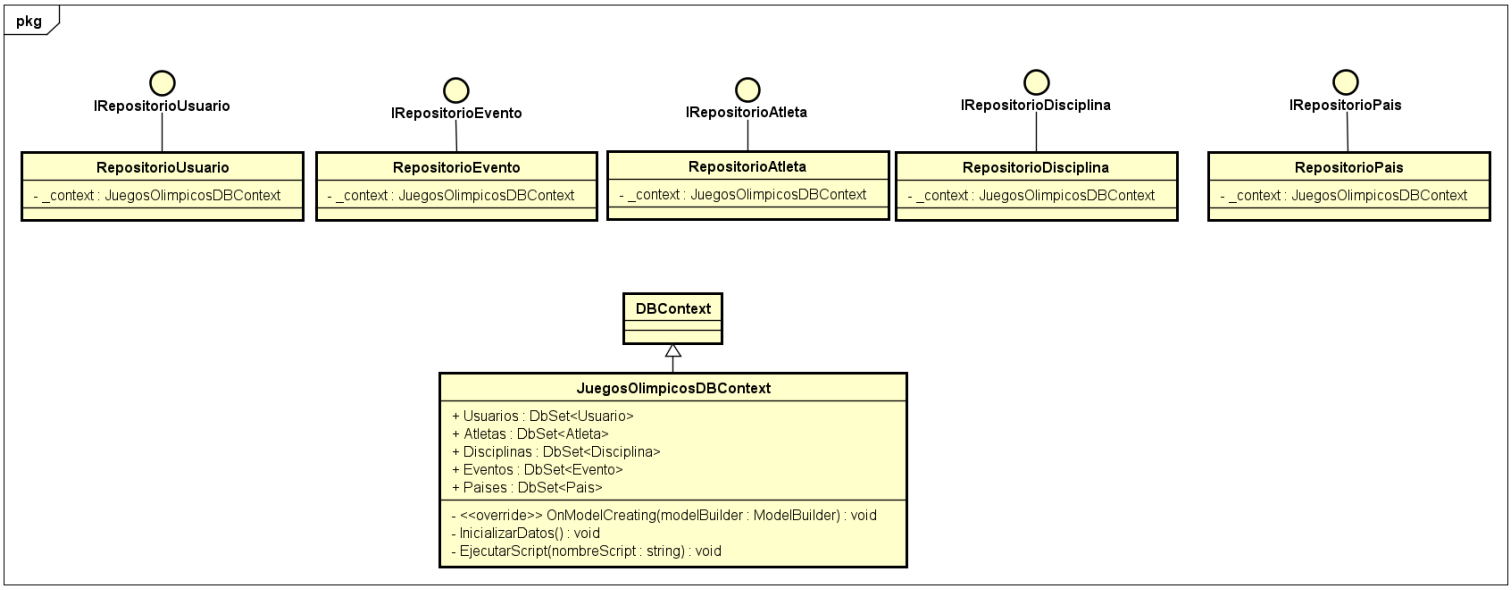
Interfaces de repositorios.



Entidades y value objects.

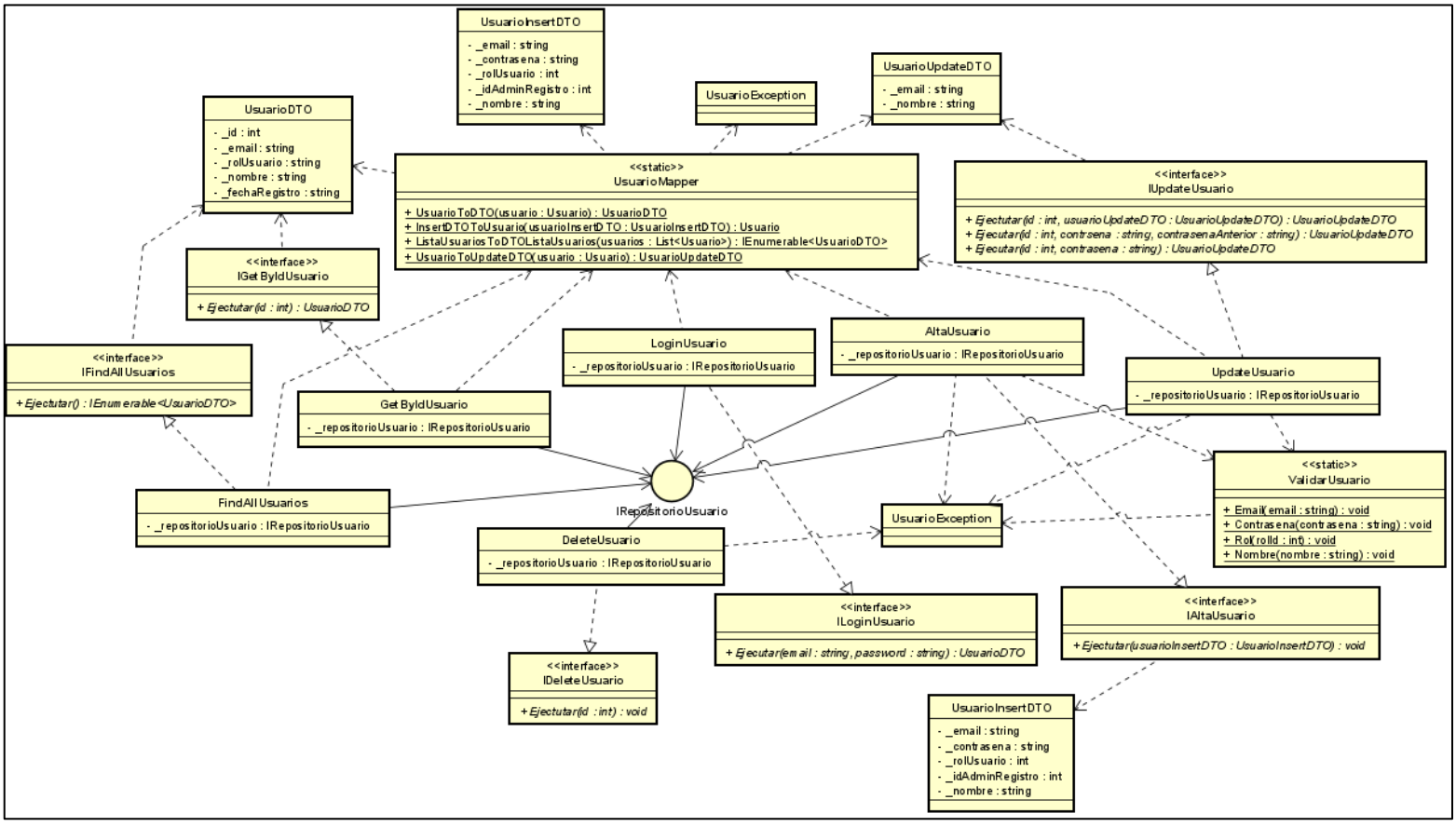


3.2 Lógica de acceso a datos.

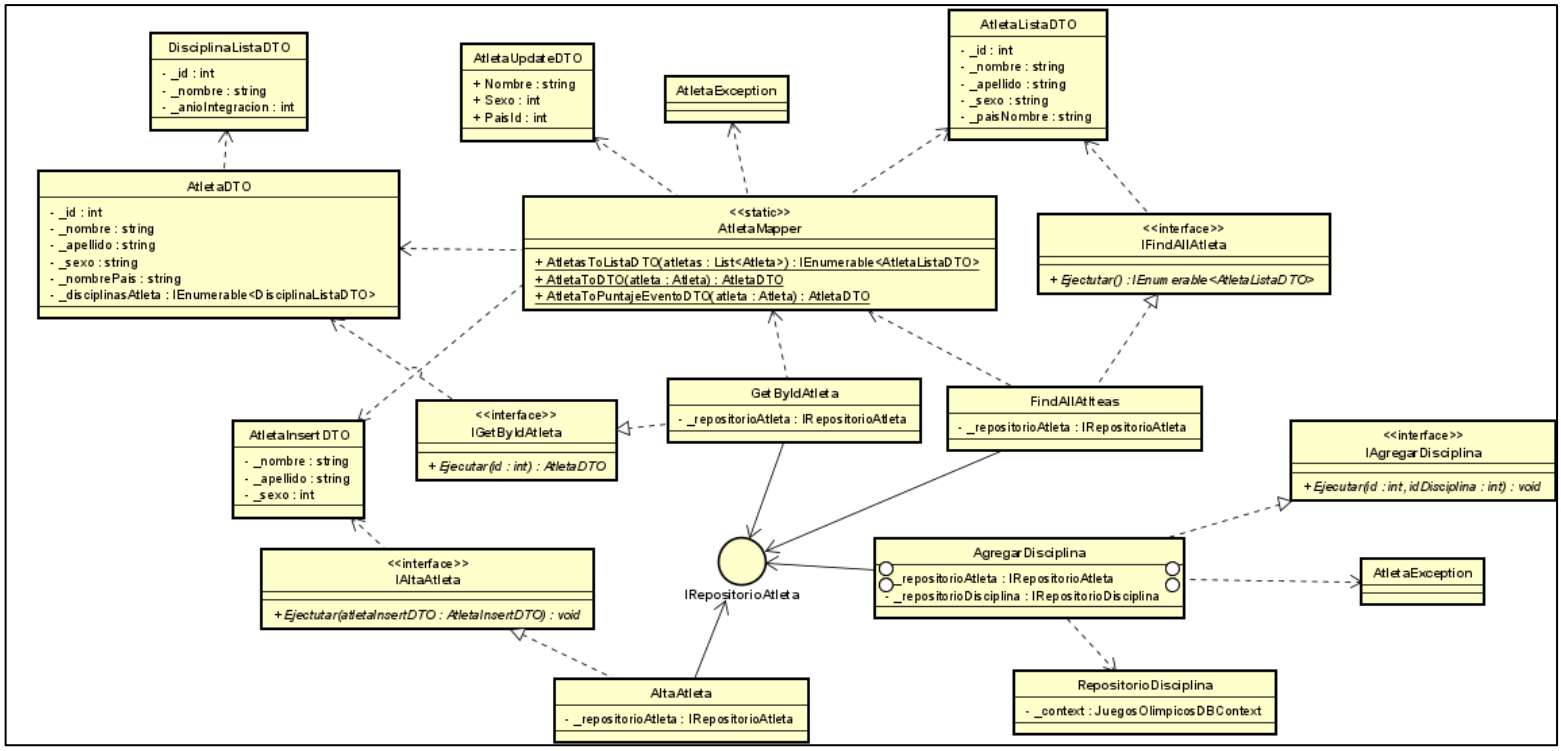


3.3 Lógica de aplicación.

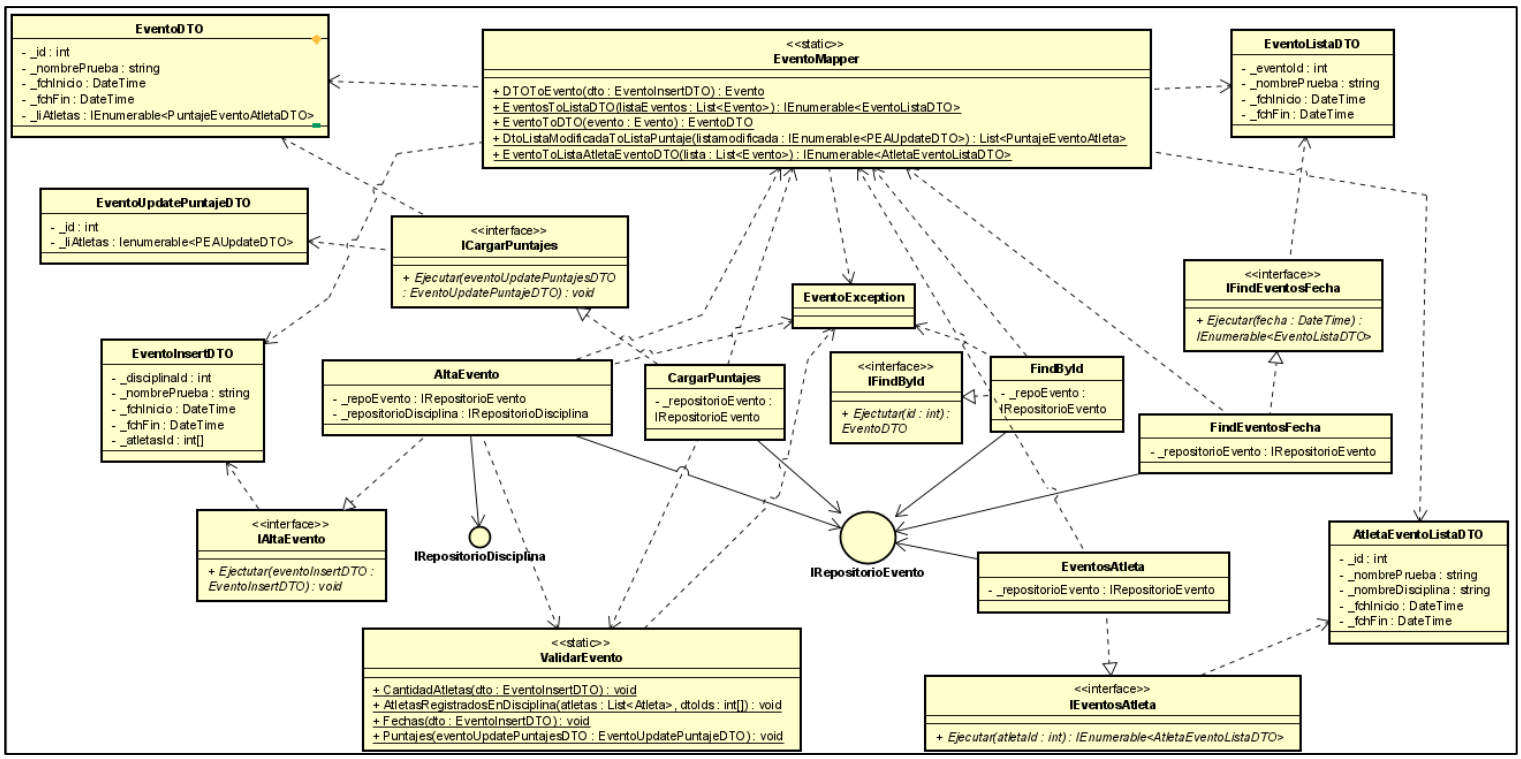
Usuario.



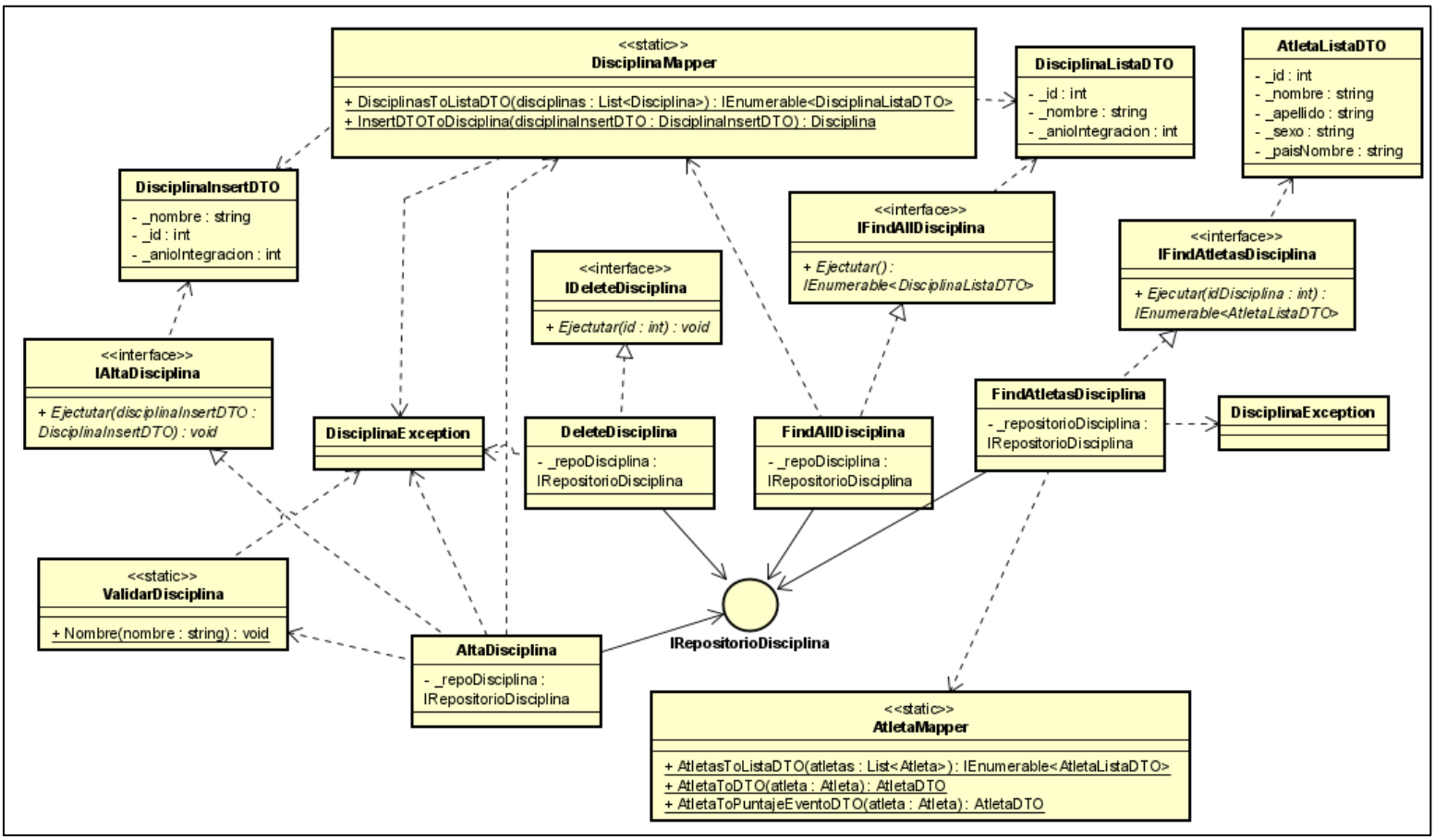
Atleta.



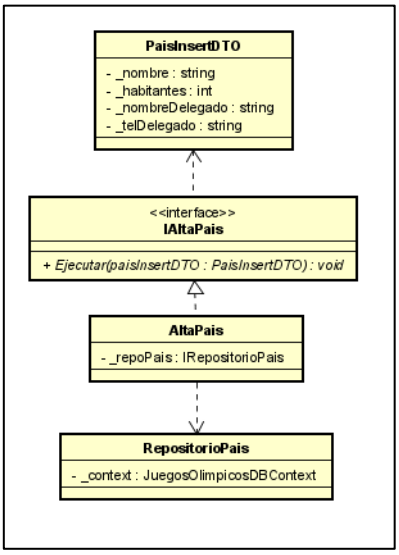
Evento.



Disciplina.

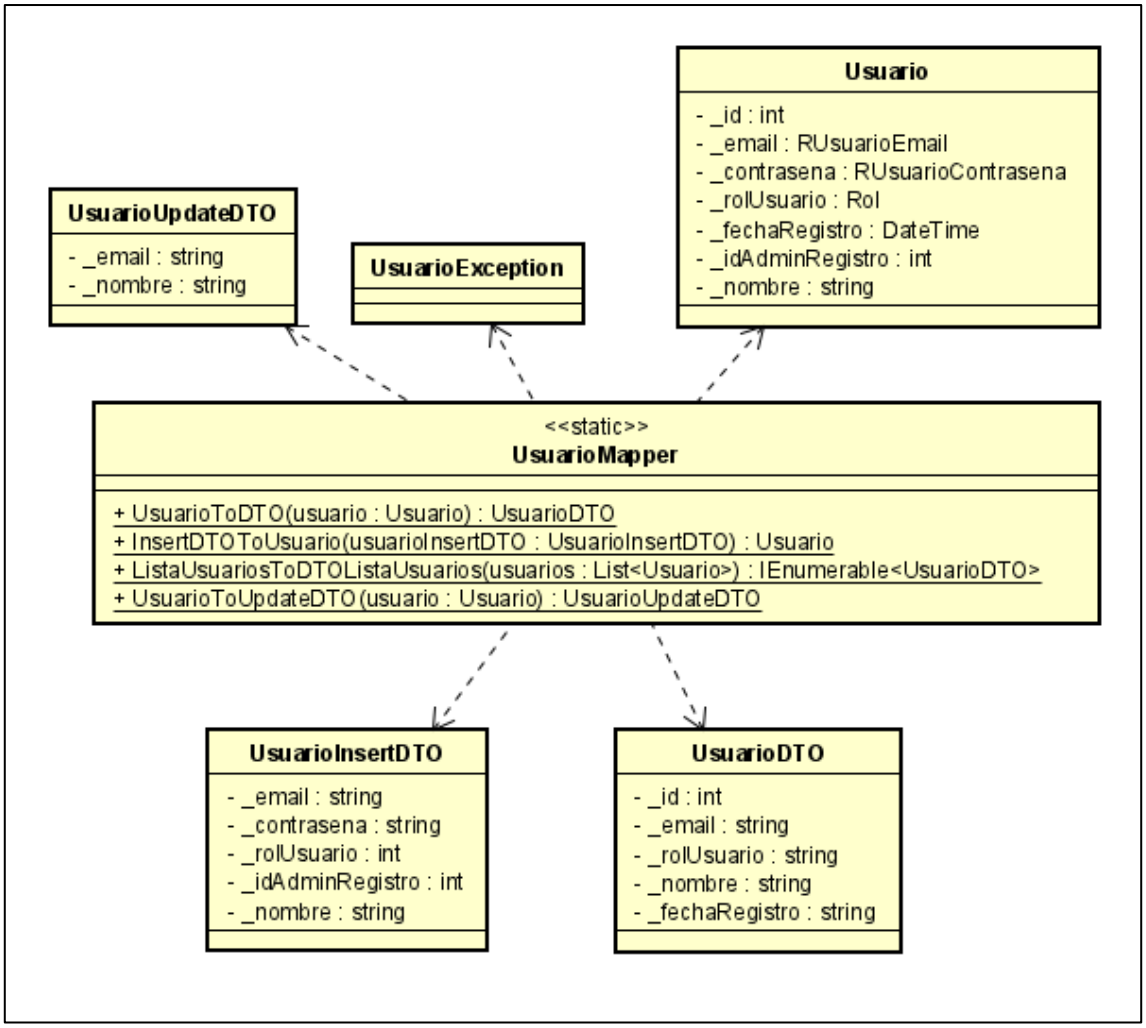


País.

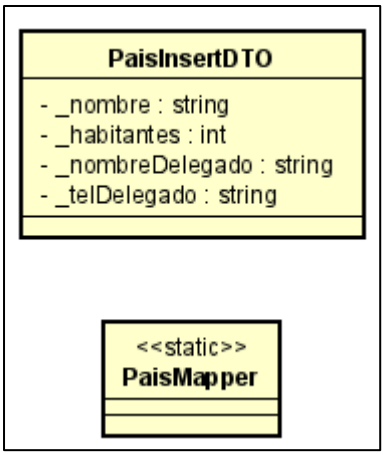


3.4 Compartido.

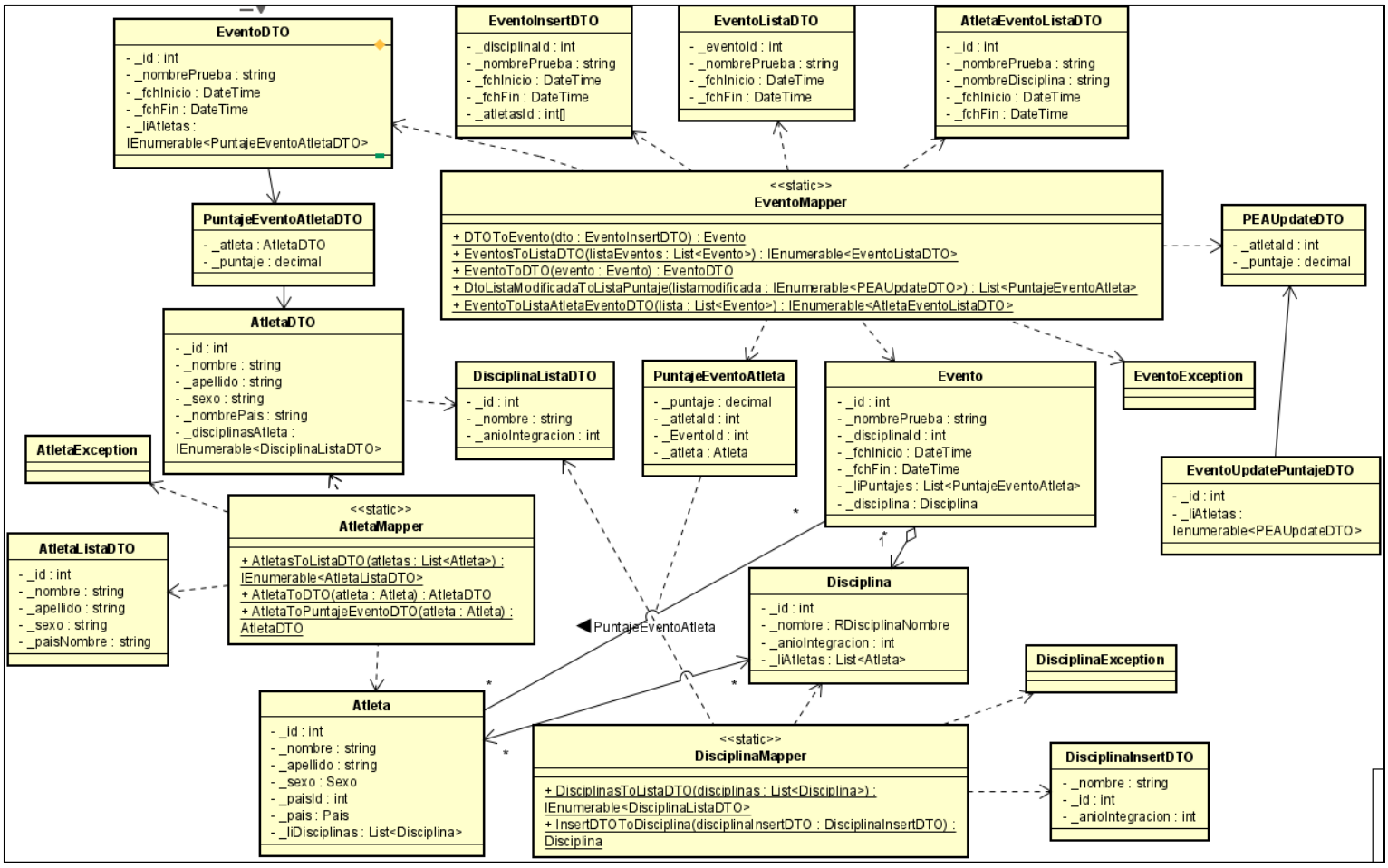
Usuario.



País.

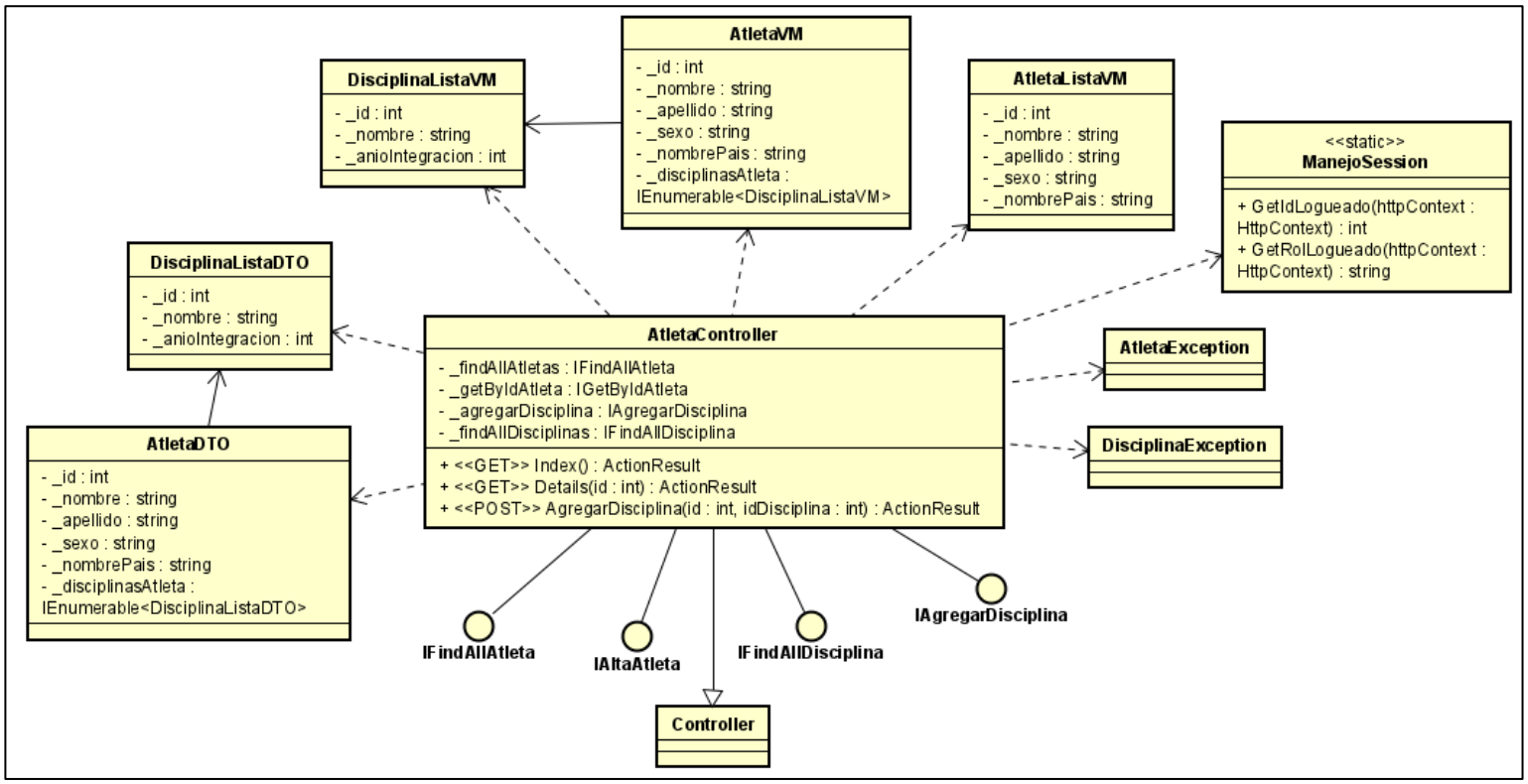


Atleta, evento y disciplina.

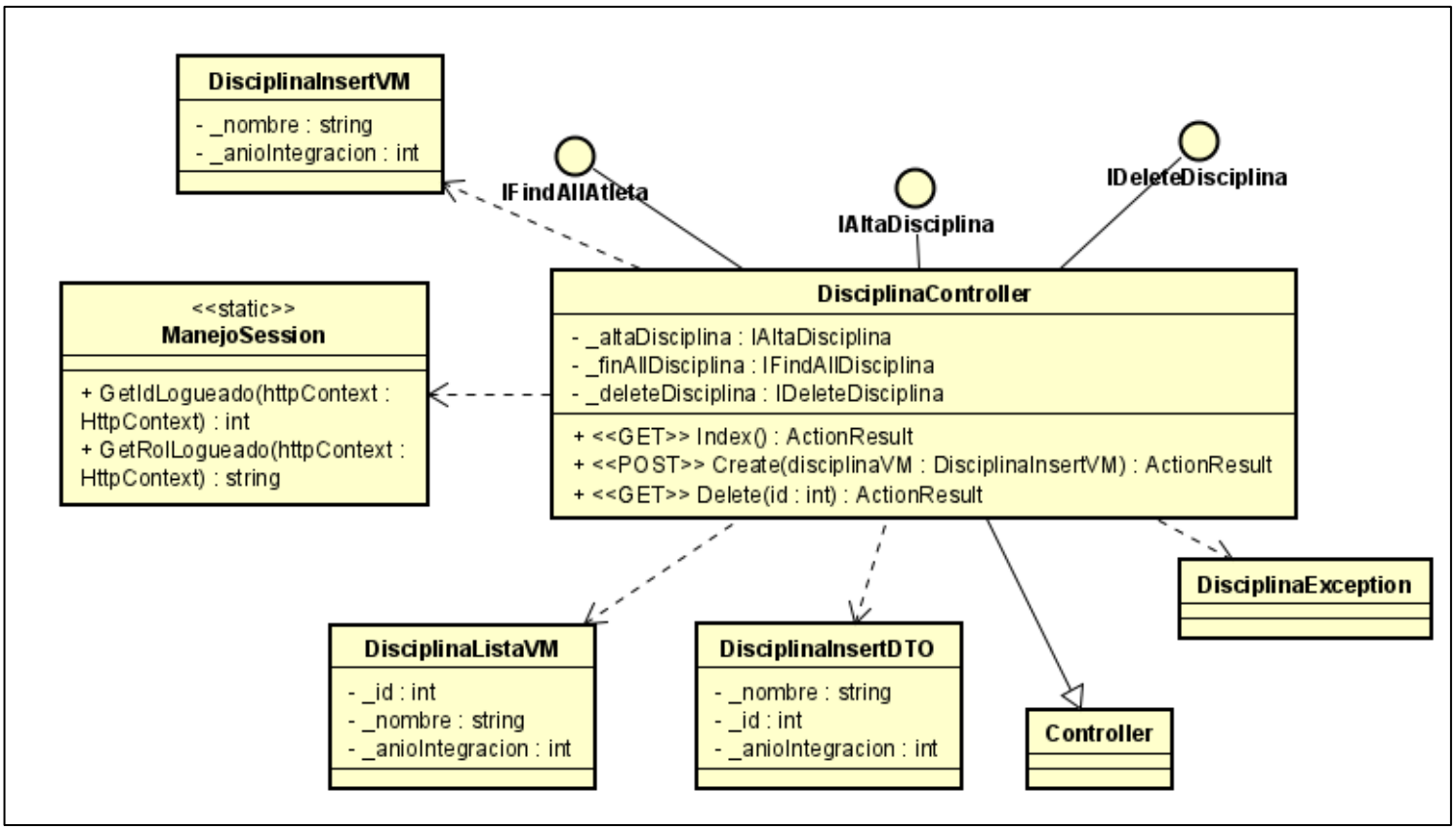


3.5 MVC.

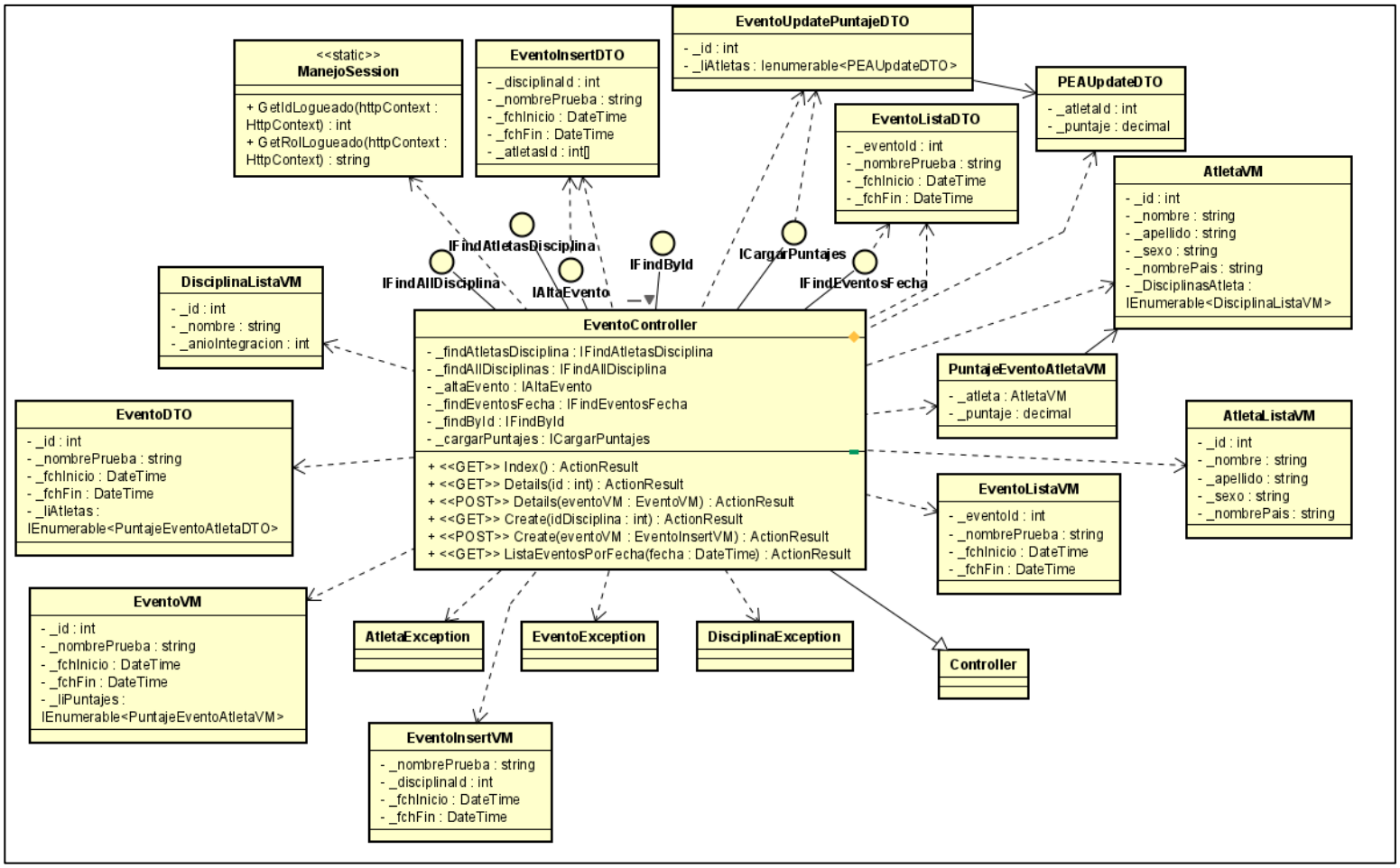
AtletaController.



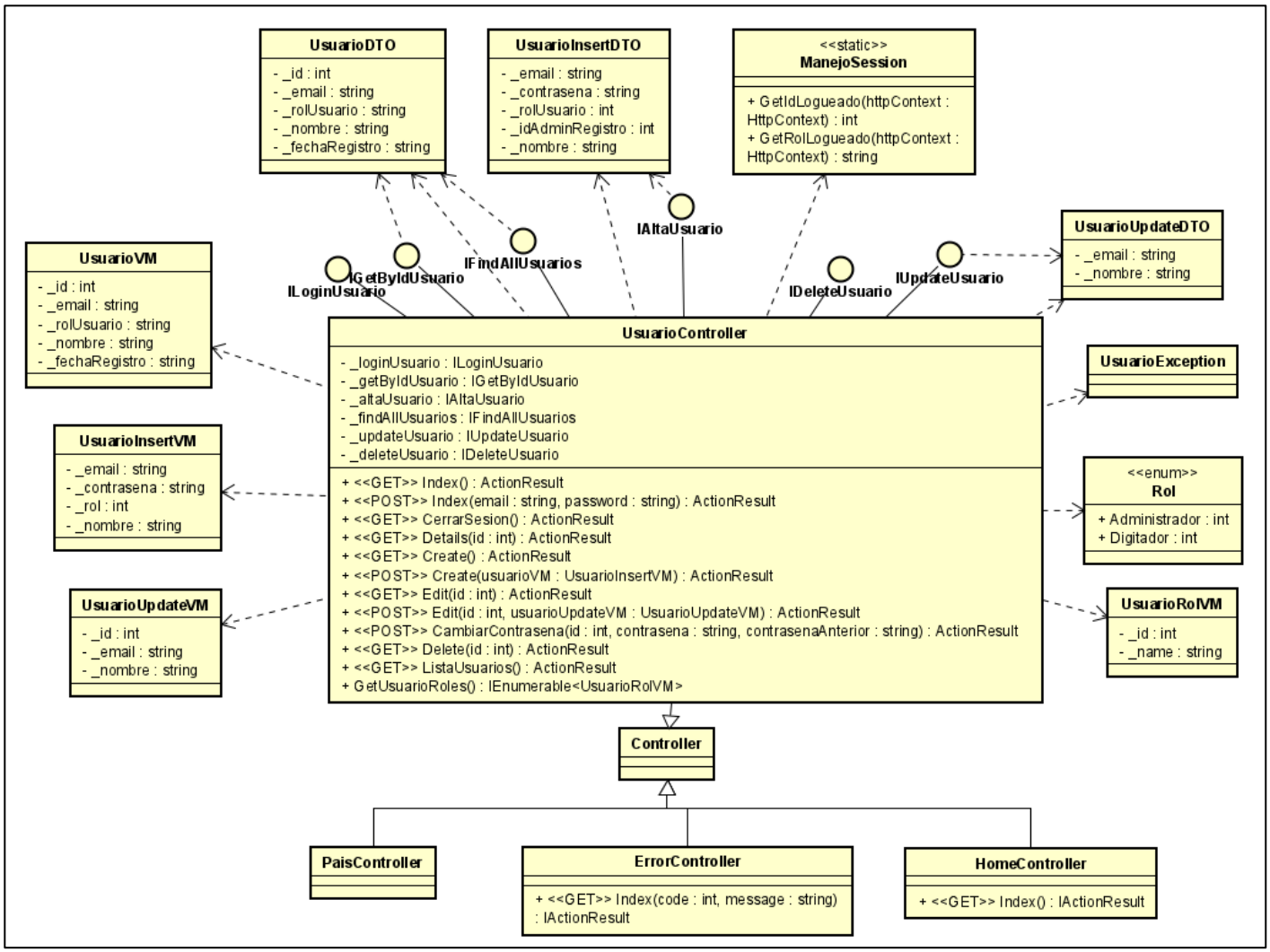
DisciplinaController.



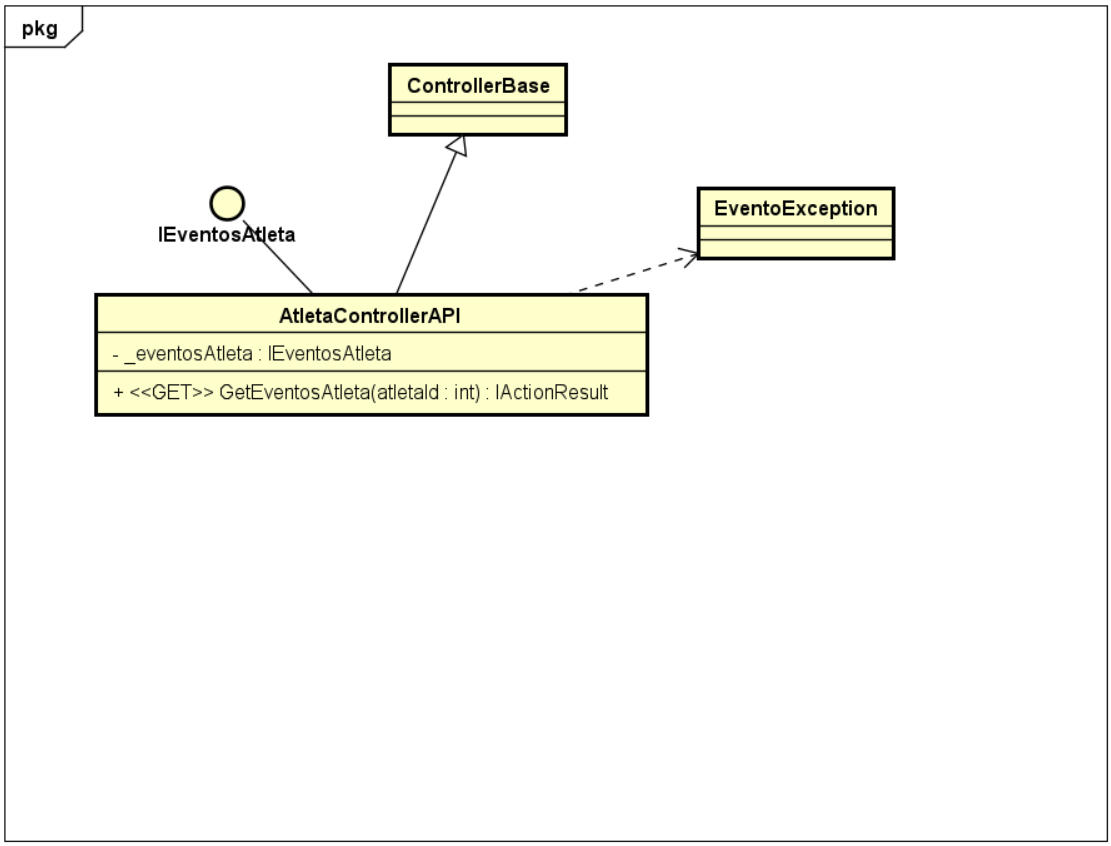
EventoController.



UsuarioController, HomeController, ErrorController y PaisController.



3.6 Web Api.


pkg
ControllerBase
IEventosAtleta
EventoException
AtletaControllerAPI
- _eventosAtleta : IEventosAtleta
+ <<GET>> GetEventosAtleta(atletaId : int) : IActionResult

4. Código fuente.

4.1 Lógica de negocio.

4.1.1 Entidades.

Usuario.

```
public class Usuario : IEntity
{
    public int Id { get; set; }
    public RUsuarioEmail Email { get; set; }
    public RUsuarioContrasena Contrasena { get; set; }
    public Rol RolUsuario { get; set; }
    public DateTime FechaRegistro { get; private set; } = DateTime.Now;
    public int IdAdminRegistro { get; set; }
    [MaxLength(50)]
    public string? Nombre { get; set; }
    public Usuario() { }
    public Usuario(string email, string contrasena, Rol rolUsuario, int idAdminRegistro)
    {
        Email = new RUsuarioEmail(email);
        Contrasena = new RUsuarioContrasena(contrasena);
        RolUsuario = rolUsuario;
        IdAdminRegistro = idAdminRegistro;
    }
    public Usuario(string email, string contrasena, Rol rolUsuario, int idAdminRegistro,
string nombre)
    {
        Email = new RUsuarioEmail(email);
        Contrasena = new RUsuarioContrasena(contrasena);
        RolUsuario = rolUsuario;
```

```
        IdAdminRegistro = idAdminRegistro;
        Nombre = nombre;
    }
}
```

País.

```
public class Pais : IEntity
 {
    public int Id { get; set; }
    public string Nombre { get; set; }
    public int Habitantes { get; set; }
    public string NombreDelegado { get; set; }
    public string TelDelegado { get; set; }
    public Pais() { }
    Pais(int id, string nombre, int habitantes, string nombreDelegado, string telDelegado)
    {
        Id = id;
        Nombre = nombre;
        Habitantes = habitantes;
        NombreDelegado = nombreDelegado;
        TelDelegado = telDelegado;
    }
 }
```

Disciplina.

```
public class Disciplina : IEntity
 {
    public int Id { get; set; }
    public RDisciplinaNombre Nombre { get; set; }
```

```
    public int AnioIntegracion { get; set; }
    public List<Atleta> LiAtletas { get; set; }
    public Disciplina() { }
    public Disciplina(int id, string nombre, int anioIntegracion)
    {
        Id = id;
        Nombre = new RDisciplinaNombre(nombre);
        AnioIntegracion = anioIntegracion;
    }
}
```

Atleta.

```
public class Atleta : IEntity
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public Sexo Sexo { get; set; }
    [ForeignKey("Pais")]
    public int PaisId { get; set; }
    public Pais Pais { get; set; }
    public List<Disciplina> LiDisciplinas { get; set; }
    public Atleta() { }
    public Atleta(int id, string nombre, string apellido, Sexo sexo, Pais pais)
    {
        Id = id;
        Nombre = nombre;
        Apellido = apellido;
        Sexo = sexo;
```

```
        Pais = pais;
    }
}
```

Evento.

```
public class Evento : IEntity
{
    public int Id { get; set; }
    public string NombrePrueba { get; set; }
    public int DisciplinaId { get; set; }
    public DateTime FchInicio { get; set; }
    public DateTime FchFin { get; set; }
    public List<PuntajeEventoAtleta> LiPuntajes { get; set; } = new
List<PuntajeEventoAtleta>();
    [ForeignKey("DisciplinaId")]
    public Disciplina Disciplina { get; set; }
    public Evento() { }
    public Evento(int id, Disciplina disciplina, string nombrePrueba, DateTime fchInicio,
DateTime fchFin)
    {
        Id = id;
        Disciplina = disciplina;
        NombrePrueba = nombrePrueba;
        FchInicio = fchInicio;
        FchFin = fchFin;
    }
}
```

PuntajeEventoAtleta.

```
[PrimaryKey(nameof(AtletaId), nameof(EventoId))]
public class PuntajeEventoAtleta
{
    public decimal Puntaje { get; set; }
    public int AtletaId { get; set; }
    public int EventoId { get; set; }
    [ForeignKey("AtletaId")]
    public Atleta Atleta { get; set; }
    public PuntajeEventoAtleta() { }
    public PuntajeEventoAtleta(Atleta atleta, decimal puntaje)
    {
        Atleta = atleta;
        Puntaje = puntaje;
    }
}
```

4.1.2 Enums.

Rol.

```
public enum Rol
{
    Administrador = 0,
    Digitador = 1
}
```

Sexo.

```
public enum Sexo
{
    Masculino = 0,
```

```
    Femenino = 1
}
```

4.1.3 Excepciones Entidades.

AtletaException.

```
public class AtletaException : Exception
{
    public AtletaException() { }
    public AtletaException(string message) : base(message) { }
    public AtletaException(string message, Exception innerException) : base(message,
innerException) { }
}
```

DisciplinaException.

```
public class DisciplinaException : Exception
{
    public DisciplinaException() { }
    public DisciplinaException(string message) : base(message) { }
    public DisciplinaException(string message, Exception innerException) :
base(message, innerException) { }
}
```

EventoException.

```
public class EventoException : Exception
{
    public EventoException() { }
    public EventoException(string message) : base(message) { }
    public EventoException(string message, Exception innerException) : base(message,
innerException) { }
}
```

PaisException.

```
public class PaisException : Exception
{
   public PaisException() { }
   public PaisException(string message) : base(message) { }
   public PaisException(string message, Exception innerException) : base(message,
innerException) { }
}
```

UsuarioException.

```
public class UsuarioException : Exception
{
   public UsuarioException() { }
   public UsuarioException(string message) : base(message) { }
   public UsuarioException(string message,  Exception innerException) : base(message,
innerException) { }
}
```

4.1.4 Interfaces Entidades.

IEntity.

```
public interface IEntity
{
  int Id { get; set; }
}
```

4.1.5 Interfaces Repositorios.

IRepositorio.

```
public interface IRepositorio<T>
{
   List<T> GetAll();
```

```
    T? GetById(int id);
    void Add(T item);
    void Update(T item);
    void Delete(T item);
}
```

IRepositorioUsuario.

```
public interface IRepositorioUsuario : IRepositorio<Usuario>
{
    Usuario? LoginUsuario(string email, string password);
    Usuario? GetByEmail(string email);
}
```

IRepositorioPais.

```
public interface IRepositorioPais : IRepositorio<Pais>
{}
```

IRepositorioDisciplina.

```
public interface IRepositorioDisciplina : IRepositorio<Disciplina>
{
    Disciplina? GetByNombre(string nombre);
    List<Atleta> GetAtletasDisciplina(int idDisciplina);
}
```

IRepositorioAtleta.

```
public interface IRepositorioAtleta : IRepositorio<Atleta>
{
    public void GuardarCambios();
}
```

IRepositorioEvento.

```
public interface IRepositorioEvento : IRepositorio<Evento>
{
    Evento GetByNombre(string nombre);
    List<Evento> GetByFecha (DateTime fecha);
    List<Evento> GetEventosAtleta(int atletaId);
}
```

4.1.6 Value Objects

RDisciplinaNombre.

```
[ComplexType]
public record RDisciplinaNombre
{
    [Required]
    [StringLength(50, MinimumLength = 10)]
    public string Valor { get; init; }
    public RDisciplinaNombre(string valor)
    {
        Valor = valor;
    }
}
```

RUsuarioContrasena.

```
[ComplexType]
public record RUsuarioContrasena
{
    [Required]
    public string Valor { get; init; }
    public RUsuarioContrasena(string valor)
```

```
    {
        Valor = valor;
    }
}
```

RUsuarioEmail.

```
[ComplexType]
public record RUsuarioEmail
{
    [Required]
    [MaxLength(255)]
    public string Valor { get; init; }
    public RUsuarioEmail(string valor)
    {
        Valor = valor;
    }
}
```

4.2 Lógica de acceso a datos.

DBContext.

```
public class JuegosOlimpicosDBContext : DbContext
{
    public DbSet<Usuario> Usuarios { get; set; }
    public DbSet<Atleta> Atletas { get; set; }
    public DbSet<Disciplina> Disciplinas { get; set; }
    public DbSet<Evento> Eventos { get; set; }
    public DbSet<Pais> Paises { get; set; }
    public JuegosOlimpicosDBContext(DbContextOptions opt) : base(opt)
    {
```

```
    if (Database.EnsureCreated())
    {
        InicializarDatos();
    }
}
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Usuario>()
        .OwnsOne(u => u.Email)
        .HasIndex(e => e.Valor)
        .IsUnique();
    modelBuilder.Entity<Disciplina>()
        .OwnsOne(d => d.Nombre)
        .HasIndex(n => n.Valor)
        .IsUnique();
}
private void InicializarDatos()
{
    if (!Usuarios.Any())
    {
        EjecutarScript("Usuarios_Admin.sql");
    }
    if (!Paises.Any())
    {
        EjecutarScript("Paises.sql");
    }
    if (!Atletas.Any())
    {
        EjecutarScript("Atletas.sql");
```

```
            EjecutarScript("Disciplinas.sql");
            EjecutarScript("AtletasDisciplina.sql");
        }
    }
    private void EjecutarScript(string nombreScript)
    {
        string rutaCompleta = Path.Combine(Directory.GetCurrentDirectory(), "..",
"ScriptsDatos", nombreScript);
        string sql = File.ReadAllText(rutaCompleta);
        Database.ExecuteSqlRaw(sql);
    }
}
```

4.2.1 Repositorios.

RepositorioUsuario.

```
public class RepositorioUsuario : IRepositorioUsuario
{
    private JuegosOlimpicosDBContext _context;
    public RepositorioUsuario(JuegosOlimpicosDBContext context)
    {
        _context = context;
    }
    public void Add(Usuario item)
    {
        if (item == null)
        {
            throw new UsuarioException("El usuario no puede ser vacío");
        }
        _context.Add(item);
        _context.SaveChanges();
```

```
    }
    public void Delete(Usuario item)
    {
        _context.Remove(item);
        _context.SaveChanges();
    }
    public List<Usuario> GetAll()
    {
        return _context.Usuarios.ToList();
    }
    public Usuario? GetById(int id) =>
        _context.Usuarios.SingleOrDefault(usuario => usuario.Id == id);

    public Usuario? GetByEmail(string email) =>
        _context.Usuarios.SingleOrDefault(usuario => usuario.Email.Valor == email);
    public void Update(Usuario item)
    {
        _context.Usuarios.Update(item);
        _context.SaveChanges();
    }
    public Usuario? LoginUsuario(string email, string password) =>
        _context.Usuarios.SingleOrDefault(usuario => usuario.Email.Valor == email &&
usuario.Contrasena.Valor == password);
}
```

RepositorioPais.

```
public class RepositorioPais : IRepositorioPais
{
    private readonly JuegosOlimpicosDBContext _context;
```

```
    public RepositorioPais(JuegosOlimpicosDBContext context)
    {
        _context = context;
    }
    public void Add(Pais item)
    {
        throw new NotImplementedException();
    }
    public void Delete(Pais item)
    {
        throw new NotImplementedException();
    }
    public List<Pais> GetAll()
    {
        throw new NotImplementedException();
    }
    public Pais GetById(int id)
    {
        throw new NotImplementedException();
    }
    public void Update(Pais item)
    {
        throw new NotImplementedException();
    }
}
```

RepositorioDisciplina.

```
public class RepositorioDisciplina : IRepositorioDisciplina
{
```

```
private readonly JuegosOlimpicosDBContext _context;


public RepositorioDisciplina(JuegosOlimpicosDBContext context)
{
    _context = context;
}
public void Add(Disciplina item)
{
    _context.Disciplinas.Add(item);
    _context.SaveChanges();
}
public void Delete(Disciplina item)
{
    _context.Disciplinas.Remove(item);
    _context.SaveChanges();
}
public List<Disciplina> GetAll() =>
    _context.Disciplinas
        .AsEnumerable()
        .OrderBy(d => d.Nombre.Valor)
        .ToList();
// NOTA: AsEnumerable porque Nombre es un ValueObject
public List<Atleta> GetAtletasDisciplina(int idDisciplina)
{
    Disciplina buscar = _context.Disciplinas
        .Include(d => d.LiAtletas)
        .ThenInclude(a => a.Pais)
        .SingleOrDefault(d => d.Id == idDisciplina);
```

```
        if (buscar == null) throw new DisciplinaException("Disciplina no encontrada con
ese id");
        return buscar.LiAtletas;
    }
    public Disciplina GetById(int id)
    {
        Disciplina disciplina = _context.Disciplinas.SingleOrDefault(d => d.Id == id);
        if (disciplina == null)
        {
            throw new DisciplinaException("Disciplina no encontrada por id");
        }
        return disciplina;
    }
    public void Update(Disciplina item)
    {
        throw new NotImplementedException();
    }
    public Disciplina? GetByNombre(string nombre) =>
        _context.Disciplinas.SingleOrDefault(disiplina => disiplina.Nombre.Valor ==
nombre);
}
```

RepositorioAtleta.

```
public class RepositorioAtleta : IRepositorioAtleta
{
    private readonly JuegosOlimpicosDBContext _context;
    public RepositorioAtleta(JuegosOlimpicosDBContext context)
    {
        _context = context;
    }
```

```
public void Add(Atleta item)
{
    throw new NotImplementedException();
}
public void Delete(Atleta item)
{
    throw new NotImplementedException();
}
public List<Atleta> GetAll() =>
    _context.Atletas
        .Include(a => a.Pais)
        .OrderBy(a => a.Pais.Nombre)
        .ThenBy(a => a.Apellido)
        .ThenBy(a => a.Nombre)
        .ToList();
public Atleta GetById(int id)
{
    Atleta atleta = _context.Atletas
                    .Include(a => a.Pais)
                    .Include(a => a.LiDisciplinas)
                    .SingleOrDefault(a => a.Id == id);
    if (atleta == null)
    {
        throw new AtletaException("Atleta no encontrado por id");
    }
    return atleta;
}
public void Update(Atleta item)
{
```

```
        throw new NotImplementedException();
    }
    public void GuardarCambios()
    {
        _context.SaveChanges();
    }
}
```

RepositorioEvento.

```
public class RepositorioEvento : IRepositorioEvento
{
    private readonly JuegosOlimpicosDBContext _dbContext;
    public RepositorioEvento(JuegosOlimpicosDBContext dbContext)
    {
        _dbContext = dbContext;
    }
    public void Add(Evento item)
    {
        _dbContext.Eventos.Add(item);
        _dbContext.SaveChanges();
    }
    public void Delete(Evento item)
    {
        throw new NotImplementedException();
    }
    public List<Evento> GetAll()
    {
        throw new NotImplementedException();
    }
```

```
    public Evento GetById(int id) =>
        _dbContext.Eventos
            .Include(e => e.LiPuntajes)
            .ThenInclude(p => p.Atleta)
            .ThenInclude(a => a.Pais)
            .SingleOrDefault(e => e.Id == id);


    public Evento GetByNombre(string nombre) =>
_dbContext.Eventos.SingleOrDefault(e => e.NombrePrueba == nombre);
    public List<Evento> GetByFecha(DateTime fecha)
    {
        return _dbContext.Eventos.Where(e => e.FchInicio <= fecha && e.FchFin >=
fecha).ToList();
    }
    public void Update(Evento item)
    {
        _dbContext.Eventos.Update(item);
        _dbContext.SaveChanges();
    }
    public List<Evento> GetEventosAtleta(int atletaId) =>
        _dbContext.Eventos
            .Include(e => e.Disciplina)
            .Where(e => e.LiPuntajes.Any(p => p.AtletaId == atletaId))
            .OrderBy(e => e.Disciplina.Nombre.Valor)
            .ToList();
}
```

4.3 Lógica de aplicación.

4.3.1 Interfaces de casos de usos.

4.3.1.1 Usuarios.

IAltaUsuario.

```
public interface IAltaUsuario
{
   void Ejecutar(UsuarioInsertDTO usuarioInsertDTO);
}
```

IDeleteUsuario.

```
public interface IDeleteUsuario
{
   void Ejecutar(int id);
}
```

IFindAllUsuarios.

```
public interface IFindAllUsuarios
{
   IEnumerable<UsuarioDTO> Ejecutar();
}
```

IGetByIdUsuario.

```
public interface IGetByIdUsuario
{
   UsuarioDTO Ejecutar(int id);
}
```

ILoginUsuario.

```
public interface ILoginUsuario
```

```
{
   UsuarioDTO Ejecutar(string email, string password);
}
```

IUpdateUsuario.

```
public interface IUpdateUsuario
{
   UsuarioUpdateDTO Ejecutar(int id, UsuarioUpdateDTO usuarioUpdateDTO);
   UsuarioUpdateDTO Ejecutar(int id, string contrasena, string contrasenaAnterior);
   UsuarioUpdateDTO Ejecutar(int id, string contrasena);
}
```

4.3.1.2 Países.

IAltaPais.

```
public interface IAltaPais
{
   void Ejecutar(PaisInsertDTO paisInsertDTO);
}
```

4.3.1.3 Disciplinas.

IAltaDisciplina.

```
public interface IAltaDisciplina
{
   void Ejecutar(DisciplinaInsertDTO disciplinaInsertDTO);
}
```

IDeleteDisciplina.

```
public interface IDeleteDisciplina
{
   void Ejecutar(int id);
```

```
}
```

IFindAllDisciplinas.

```
public interface IFindAllDisciplinas
{
    IEnumerable<DisciplinaListaDTO> Ejecutar();
}
```

IFindAtletasDisciplina.

```
public interface IFindAtletasDisciplina
{
    IEnumerable<AtletaListaDTO> Ejecutar(int idDisciplina);
}
```

4.3.1.4 Atletas.

IAltaAtleta.

```
public interface IAltaAtleta
{
    void Ejecutar(AtletaInsertDTO atletaInsertDTO);
}
```

IGetByIdAtleta.

```
public interface IGetByIdAtleta
{
    AtletaDTO Ejecutar(int id);
}
```

IFindAllAtletas.

```
public interface IFindAllAtletas
```

```
{
    IEnumerable<AtletaListaDTO> Ejectuar();
}
```

IAgregarDisciplina.

```
public interface IAgregarDisciplina
{
    void Ejecutar(int id, int? idDisciplina);
}
```

IEventosAtleta.

```
public interface IEventosAtleta
{
    IEnumerable<AtletaEventoListaDTO> Ejecutar(int atletaId);
}
```

4.3.1.5 Eventos.

IAltaEvento.

```
public interface IAltaEvento
{
    void Ejecutar(EventoInsertDTO eventoInsertDTO);
}
```

ICargarPuntajes.

```
public interface ICargarPuntajes
{
    EventoDTO Ejecutar(EventoUpdatePuntajesDTO eventoUpdatePuntajesDTO);
}
```

IFindById.

```
public interface IFindById
{
   EventoDTO Ejecutar(int id);
}
```

IFindEventosFecha.

```
public interface IFindEventosFecha
{
   IEnumerable<EventoListaDTO> Ejecutar(DateTime fecha);
}
```

4.3.2 Casos de usos.

4.3.2.1 Usuarios.

AltaUsuario.

```
public partial class AltaUsuario : IAltaUsuario
{
   private IRepositorioUsuario _repoUsuario;
   public AltaUsuario(IRepositorioUsuario repoUsuario)
   {
      _repoUsuario = repoUsuario;
   }
   public void Ejecutar(UsuarioInsertDTO usuarioInsertDTO)
   {
      if (usuarioInsertDTO == null)
      {
         throw new UsuarioException("El usuario no puede ser vacío");
      }
```

```
        Usuario? buscarSiExiste = _repoUsuario.GetByEmail(usuarioInsertDTO.Email);
        if (buscarSiExiste != null)
        {
            throw new UsuarioException("Ya se registró un usuario con ese email");
        }
        // Validaciones
        var (email, contrasena, rol, nombre) =
            (usuarioInsertDTO.Email, usuarioInsertDTO.Contrasena,
usuarioInsertDTO.RolUsuario, usuarioInsertDTO.Nombre);
        ValidarUsuario.Email(email);
        ValidarUsuario.Contrasena(contrasena);
        ValidarUsuario.Rol(rol);
        ValidarUsuario.Nombre(nombre);
        _repoUsuario.Add(UsuarioMapper.InsertDTOToUsuario(usuarioInsertDTO));
    }
}
```

DeleteUsuario.

```
public class DeleteUsuario : IDeleteUsuario
{
    private readonly IRepositorioUsuario _repositorioUsuario;
    public DeleteUsuario(IRepositorioUsuario repositorioUsuario)
    {
        _repositorioUsuario = repositorioUsuario;
    }
    public void Ejecutar(int id)
    {
        Usuario resDB = _repositorioUsuario.GetById(id);
        if (resDB == null)
```

```
        {
            throw new UsuarioException("Usuario no encontrado por ese ID");
        }
        _repositorioUsuario.Delete(resDB);
    }
}
```

FindAllUsuarios.

```
public class FindAllUsuarios : IFindAllUsuarios
{
    private IRepositorioUsuario _repoUsuario;
    public FindAllUsuarios(IRepositorioUsuario repoUsuario)
    {
        _repoUsuario = repoUsuario;
    }
    public IEnumerable<UsuarioDTO> Ejecutar()
    {
        List<Usuario> usuarios = _repoUsuario.GetAll();
        return UsuarioMapper.ListaUsuariosToDTOListaUsuarios(usuarios);
    }
}
```

GetByIdUsuario.

```
public class GetByIdUsuario : IGetByIdUsuario
{
    private IRepositorioUsuario _repositorioUsuario;
    public GetByIdUsuario(IRepositorioUsuario repositorioUsuario)
    {
        _repositorioUsuario = repositorioUsuario;
```

```
    }
    public UsuarioDTO Ejecutar(int id)
    {
        LogicaNegocio.Entidades.Usuario? res = _repositorioUsuario.GetById(id);

        if (res == null)
        {
            throw new UsuarioException("Usuario no encontrado por ID");
        }
        return UsuarioMapper.UsuarioToDTO(res);
    }
}
```

LoginUsuario.

```
public class LoginUsuario : ILoginUsuario
{
    private IRepositorioUsuario _repositorioUsuario;
    public LoginUsuario(IRepositorioUsuario repositorioUsuario)
    {
        _repositorioUsuario = repositorioUsuario;
    }
    public UsuarioDTO? Ejecutar(string email, string password)
    {
        Usuario resBD = _repositorioUsuario.LoginUsuario(email, password);
        if (resBD == null)
        {
            throw new UsuarioException("Usuario o contraseña incorrectos");
        }
        return UsuarioMapper.UsuarioToDTO(resBD);
```

```
    }
}
```

UpdateUsuario.

```
public class UpdateUsuario : IUpdateUsuario
{
    private readonly IRepositorioUsuario _repositorioUsuario;
    public UpdateUsuario(IRepositorioUsuario repositorioUsuario)
    {
        _repositorioUsuario = repositorioUsuario;
    }
    public UsuarioUpdateDTO Ejecutar(int id, UsuarioUpdateDTO usuarioUpdateDTO)
    {
        var (email, nombre) =
            (usuarioUpdateDTO.Email, usuarioUpdateDTO.Nombre);
        ValidarUsuario.Email(email);
        ValidarUsuario.Nombre(nombre);
        Usuario actualizarUsuario = _repositorioUsuario.GetById(id);
        if (actualizarUsuario == null)
        {
            throw new UsuarioException("No se encontró el usuario que intenta
actualizar");
        }
        if (_repositorioUsuario.GetByEmail(email) != null)
            throw new UsuarioException("Este email ya pertenece a un usuario");
        actualizarUsuario.Email = new RUsuarioEmail(email);
        actualizarUsuario.Nombre = nombre;
        _repositorioUsuario.Update(actualizarUsuario);
        return UsuarioMapper.UsuarioToUpdateDTO(actualizarUsuario);
```

```
    }
    public UsuarioUpdateDTO Ejecutar(int id, string contrasena)
    {
        ValidarUsuario.Contrasena(contrasena);
        Usuario actualizarUsuario = _repositorioUsuario.GetById(id);
        if (actualizarUsuario == null)
        {
            throw new UsuarioException("No se encontró el usuario que intenta
actualizar");
        }
        actualizarUsuario.Contrasena = new RUsuarioContrasena(contrasena);
        _repositorioUsuario.Update(actualizarUsuario);
        return UsuarioMapper.UsuarioToUpdateDTO(actualizarUsuario);
    }
    public UsuarioUpdateDTO Ejecutar(int id, string contrasena, string
contrasenaAnterior)
    {
        ValidarUsuario.Contrasena(contrasena);
        Usuario actualizarUsuario = _repositorioUsuario.GetById(id);
        if (actualizarUsuario == null)
        {
            throw new UsuarioException("No se encontró el usuario que intenta
actualizar");
        }
        if (contrasenaAnterior != actualizarUsuario.Contrasena.Valor)
        {
            throw new UsuarioException("La contraseña actual que ingresó es incorrecta");
        }
        if (contrasenaAnterior == contrasena)
        {
```

```
            throw new UsuarioException("La nueva contraseña que intenta ingresar es igual
a la actual");
        }
        actualizarUsuario.Contrasena = new RUsuarioContrasena(contrasena);
        _repositorioUsuario.Update(actualizarUsuario);
        return UsuarioMapper.UsuarioToUpdateDTO(actualizarUsuario);
    }
}
```

4.3.2.2 Países.

AltaPais.

```
public class AltaPais : IAltaPais
{
    private readonly IRepositorioPais _repositorioPais;
    public AltaPais(IRepositorioPais repositorioPais)
    {
        _repositorioPais = repositorioPais;
    }
    public void Ejecutar(PaisInsertDTO paisInsertDTO)
    {
        throw new NotImplementedException();
    }
}
```

4.3.2.3 Disciplinas.

AltaDisciplina.

```
public class AltaDisciplina : IAltaDisciplina
{
    private IRepositorioDisciplina _repositorioDisciplina;
    public AltaDisciplina(IRepositorioDisciplina repositorioDisciplina)
```

```
    {
        _repositorioDisciplina = repositorioDisciplina;
    }
    public void Ejecutar(DisciplinaInsertDTO disciplinaInsertDTO)
    {
        Disciplina? BuscarSiExiste =
_repositorioDisciplina.GetByNombre(disciplinaInsertDTO.Nombre);
        if (BuscarSiExiste != null)
        {
            throw new DisciplinaException("Ya existe una disciplina con ese nombre");
        }
        ValidarDisciplina.Nombre(disciplinaInsertDTO.Nombre);

_repositorioDisciplina.Add(DisciplinaMapper.InsertDTOToDisciplina(disciplinaInsertD
TO));
    }
}
```

DeleteDisciplina.

```
public class DeleteDisciplina : IDeleteDisciplina
{
    private readonly IRepositorioDisciplina _repositorioDisciplina;
    public DeleteDisciplina(IRepositorioDisciplina repositorioDisciplina)
    {
        _repositorioDisciplina = repositorioDisciplina;
    }
    public void Ejecutar(int id)
    {
        Disciplina buscarDisciplina = _repositorioDisciplina.GetById(id);
```

```
        if (buscarDisciplina == null) throw new DisciplinaException($"No se encontró una
Disciplina con id: {id}");
        _repositorioDisciplina.Delete(buscarDisciplina);
    }
}
```

FindAllDisciplinas.

```
public class FindAllDisciplinas : IFindAllDisciplinas
{
    private readonly IRepositorioDisciplina _repositorioDisciplina;
    public FindAllDisciplinas(IRepositorioDisciplina repositorioDisciplina)
    {
        _repositorioDisciplina = repositorioDisciplina;
    }
    public IEnumerable<DisciplinaListaDTO> Ejecutar() =>
DisciplinaMapper.DisciplinasToListaDTO(_repositorioDisciplina.GetAll());
}
```

FindAtletasDisciplina.

```
public class FindAtletasDisciplina : IFindAtletasDisciplina
{
    private IRepositorioDisciplina _repositorioDisciplina;
    public FindAtletasDisciplina(IRepositorioDisciplina repositorioDisciplina)
    {
        _repositorioDisciplina = repositorioDisciplina;
    }
    public IEnumerable<AtletaListaDTO> Ejecutar(int idDisciplina)
    {
        if (idDisciplina == 0)
        {
```

```
            throw new DisciplinaException("El id de la disciplina no es correcto");
        }
        return
AtletaMapper.AtletasToListaDTO(_repositorioDisciplina.GetAtletasDisciplina(idDiscip
lina));
    }
}
```

4.3.2.4 Atletas.

AltaAtleta.

```
public class AltaAtleta : IAltaAtleta
{
    private readonly IRepositorioAtleta _repositorioAtleta;
    public AltaAtleta(IRepositorioAtleta repositorioAtleta)
    {
        _repositorioAtleta = repositorioAtleta;
    }
    public void Ejecutar(AtletaInsertDTO atletaInsertDTO)
    {
        throw new NotImplementedException();
    }
}
```

GetByIdAtleta.

```
public class GetByIdAtleta : IGetByIdAtleta
{
    private readonly IRepositorioAtleta _repositorioAtleta;
    public GetByIdAtleta(IRepositorioAtleta repositorioAtleta)
    {
        _repositorioAtleta = repositorioAtleta;
```

```
    }
    public AtletaDTO Ejecutar(int id) =>
AtletaMapper.AtletaToDTO(_repositorioAtleta.GetById(id));
}
```

FindAllAtletas.

```
public class FindAllAtletas : IFindAllAtletas
{
    private readonly IRepositorioAtleta _repositorioAtleta;
    public FindAllAtletas(IRepositorioAtleta repositorioAtleta)
    {
        _repositorioAtleta = repositorioAtleta;
    }
    public IEnumerable<AtletaListaDTO> Ejectuar()
    {
        return AtletaMapper.AtletasToListaDTO(_repositorioAtleta.GetAll());
    }
}
```

AgregarDisciplina.

```
public class AgregarDisciplina : IAgregarDisciplina
{
    private readonly IRepositorioAtleta _repositorioAtleta;
    private readonly IRepositorioDisciplina _repositorioDisciplina;
    public AgregarDisciplina(IRepositorioAtleta repositorioAtleta,
IRepositorioDisciplina repositorioDisciplina)
    {
        _repositorioAtleta = repositorioAtleta;
        _repositorioDisciplina = repositorioDisciplina;
    }
```

```
    public void Ejecutar(int id, int? idDisciplina)
    {
        if (id == 0)
        {
            throw new AtletaException("Id de atleta incorrecto");
        }
        if (idDisciplina == null || idDisciplina == 0)
        {
            throw new AtletaException("Id de disciplina incorrecto");
        }
        Atleta atleta = _repositorioAtleta.GetById(id);
        Disciplina tieneDisciplina = atleta.LiDisciplinas.FirstOrDefault(d => d.Id ==
idDisciplina);

        if (tieneDisciplina != null)
        {
            throw new AtletaException("El atleta ya está registrado en esta disciplina");
        };
        Disciplina disciplinaAgregar =
_repositorioDisciplina.GetById(idDisciplina.Value);
        atleta.LiDisciplinas.Add(disciplinaAgregar);
        _repositorioAtleta.GuardarCambios();
    }
}
```

EventosAtleta.

```
public class EventosAtleta : IEventosAtleta
{
    private readonly IRepositorioEvento _repositorioEvento;
    public EventosAtleta(IRepositorioEvento repositorioEvento)
```

```
    {
        _repositorioEvento = repositorioEvento;
    }
    public IEnumerable<AtletaEventoListaDTO> Ejecutar(int atletaId)
        =>
EventoMapper.EventoToListaAtletaEventoDTO(_repositorioEvento.GetEventosAtleta(
atletaId));
}
```

4.3.2.5 Eventos.

AltaEvento.

```
public class AltaEvento : IAltaEvento
{
    private readonly IRepositorioEvento _repositorioEvento;
    private readonly IRepositorioDisciplina _repositorioDisciplina;
    public AltaEvento(IRepositorioEvento repositorioEvento, IRepositorioDisciplina
repositorioDisciplina)
    {
        _repositorioEvento = repositorioEvento;
        _repositorioDisciplina = repositorioDisciplina;
    }
    public void Ejecutar(EventoInsertDTO eventoInsertDTO)
    {
        if (eventoInsertDTO == null)
            throw new EventoException("El Evento DTO se encuentra vacío");
        if (_repositorioEvento.GetByNombre(eventoInsertDTO.NombrePrueba) != null)
            throw new EventoException("Ya existe un evento con ese nombre");
        ValidarEvento.CantidadAtletas(eventoInsertDTO);
        ValidarEvento.AtletasRegistradosEnDisciplina
```

```
        (_repositorioDisciplina.GetAtletasDisciplina(eventoInsertDTO.DisciplinaId),
eventoInsertDTO.AtletasId);
        ValidarEvento.Fechas(eventoInsertDTO);
        _repositorioEvento.Add(EventoMapper.DtoToEvento(eventoInsertDTO));
    }
}
```

CargarPuntajes.

```
public class CargarPuntajes : ICargarPuntajes
{
    private readonly IRepositorioEvento _repositorioEvento;
    public CargarPuntajes(IRepositorioEvento repositorioEvento)
    {
        _repositorioEvento = repositorioEvento;
    }
    public EventoDTO Ejecutar(EventoUpdatePuntajesDTO eventoUpdatePuntajesDTO)
    {
        // Validar puntajes >= 0
        ValidarEvento.Puntajes(eventoUpdatePuntajesDTO);
        // Validar que existe el Evento
        Evento eventoAModificar =
_repositorioEvento.GetById(eventoUpdatePuntajesDTO.Id);
        if (eventoAModificar == null) throw new EventoException("No se encontró un
evento con ese Id");
        // Update
        eventoAModificar.LiPuntajes =
EventoMapper.DtoListaModificadaToListaPuntaje(eventoUpdatePuntajesDTO.LiAtleta
s);
        _repositorioEvento.Update(eventoAModificar);
        return EventoMapper.EventoToDTO(eventoAModificar);
    }
```

```
}
```

FindById.

```
public class FindById : IFindById
{
    private readonly IRepositorioEvento _repositorioEvento;
    public FindById(IRepositorioEvento repositorioEvento)
    {
        _repositorioEvento = repositorioEvento;
    }
    public EventoDTO Ejecutar(int id)
    {
        if (id < 1) throw new EventoException("No existe atleta con ese id");
        return EventoMapper.EventoToDTO(_repositorioEvento.GetById(id));
    }
}
```

FindEventosFecha.

```
public class FindEventosFecha : IFindEventosFecha
{
    private readonly IRepositorioEvento _repositorioEvento;
    public FindEventosFecha(IRepositorioEvento repositorioEvento)
    {
        _repositorioEvento = repositorioEvento;
    }
    public IEnumerable<EventoListaDTO> Ejecutar(DateTime fecha) =>
EventoMapper.EventosToListaDTO(_repositorioEvento.GetByFecha(fecha));
}
```

4.3.3 Validadores.

ValidarDisciplina.

```
public static class ValidarDisciplina
{
   public static void Nombre(string nombre)
   {
      if (nombre.Length < 10)
      {
         throw new DisciplinaException("El nombre de la disciplina debe contener al menos 10 caracteres");
      }
      if (nombre.Length > 50)
      {
         throw new DisciplinaException("El nombre de la disciplina puede contener hasta 50 caracteres");
      }
   }
}
```

ValidarEvento.

```
public static class ValidarEvento
{
   public static void CantidadAtletas(EventoInsertDTO dto)
   {
      if (dto.AtletasId != null ? dto.AtletasId.Count() < 3 : true)
         throw new EventoException("Se requieren al menos 3 Atletas");
   }
   public static void AtletasRegistradosEnDisciplina(List<Atleta> atletas, int[] dtoIds)
   {
      int verificador = atletas.Where(a => dtoIds.Any(id => id == a.Id)).Count();
```

```
        if (verificador != dtoIds.Length) throw new EventoException("Hay atletas no registrados en esta Disciplina");
    }
    public static void Fechas(EventoInsertDTO dto)
    {
        if (dto.FchInicio > dto.FchFin)
            throw new EventoException("Fechas incorrectas");
    }
    public static void Puntajes(EventoUpdatePuntajesDTO eventoUpdatePuntajesDTO)
    {
        if (eventoUpdatePuntajesDTO.LiAtletas.Any(p => p.Puntaje < 1))
            throw new EventoException("Los puntajes no pueden ser menores a 1");
    }
}
```

ValidarUsuario.

```
public static class ValidarUsuario
{
    public static void Email(string email)
    {
        if (!Regex.IsMatch(email, @"^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+$"))
        {
            throw new UsuarioException("El formato del mail no es correcto");
        }
        if (email.Length > 255)
        {
            throw new UsuarioException("El email puede contener hasta 255 caracteres");
        }
    }
```

```
    public static void Contrasena(string contrasena)
    {
        if (!Regex.IsMatch(contrasena, @"^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-
9])(?=.*[\.;,!]).{6,}$"))
        {
            throw new UsuarioException("La contraseña debe contener al menos 6
caracteres y al menos una mayúscula, minúscula, dígito y [. ; , !]");
        }
    }
    public static void Rol(int rolId)
    {
        if (rolId < 0 || rolId > 1)
        {
            throw new UsuarioException("El rol solamente puede ser como Administrador o
Digitador");
        }
    }
    public static void Nombre(string nombre)
    {
        if (nombre != null && nombre.Length > 50)
        {
            throw new UsuarioException("El Nombre puede contener hasta 50 caracteres");
        }
    }
}
```

4.4 Compartido.

4.4.1 DTOs.

4.4.1.1 Usuarios.

UsuarioDTO.

```
public class UsuarioDTO
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string RolUsuario { get; set; }
    public string? Nombre { get; set; }
    public string FechaRegistro { get; set; }
}
```

UsuarioInsertDTO.

```
public class UsuarioInsertDTO
{
    public string Email { get; set; }
    public string Contrasena { get; set; }
    public int RolUsuario { get; set; }
    public int IdAdminRegistro { get; set; }
    public string? Nombre { get; set; }
}
```

UsuarioUpdateDTO.

```
public class UsuarioUpdateDTO
{
    public string Email { get; set; }
    public string Nombre { get; set; }
}
```

4.4.1.2 Países.

PaisInsertDTO.

```
public class PaisInsertDTO
```

```
{
    public string Nombre { get; set; }
    public int Habitantes { get; set; }
    public string NombreDelegado { get; set; }
    public string TelDelegado { get; set; }
}
```

4.4.1.3 Disciplinas.

DisciplinaInsertDTO.

```
public class DisciplinaInsertDTO
{
    public string Nombre { get; set; }
    public int AnioIntegracion { get; set; }
}
```

DisciplinaListaDTO.

```
public class DisciplinaListaDTO
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public int AnioIntegracion { get; set; }
}
```

4.4.1.4 Atletas.

AtletaDTO.

```
public class AtletaDTO
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
```

```
    public string Sexo { get; set; }
    public string NombrePais { get; set; }
    public IEnumerable<DisciplinaListaDTO> DisciplinasAtleta { get; set; }
}
```

AtletaInsertDTO.

```
public class AtletaInsertDTO
{
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public int Sexo { get; set; }
}
```

AtletaListaDTO.

```
public class AtletaListaDTO
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public string Sexo { get; set; }
    public string NombrePais { get; set; }
}
```

4.4.1.5 Eventos.

EventoDTO.

```
public class EventoDTO
{
    public int Id { get; set; }
    public string NombrePrueba { get; set; }
```

```
    public DateTime FchInicio { get; set; }
    public DateTime FchFin { get; set; }
    public IEnumerable<PuntajeEventoAtletaDTO> LiAtletas { get; set; }
}
```

EventoInsertDTO.

```
public class EventoInsertDTO
{
    public int DisciplinaId { get; set; }
    public string NombrePrueba { get; set; }
    public DateTime FchInicio { get; set; }
    public DateTime FchFin { get; set; }
    public int[]? AtletasId { get; set; }
}
```

EventoListaDTO.

```
public class EventoListaDTO
{
    public int EventoId { get; set; }
    public string NombrePrueba { get; set; }
    public DateTime FchInicio { get; set; }
    public DateTime FchFin { get; set; }
}
```

EventoUpdatePuntajesDTO.

```
public class EventoUpdatePuntajesDTO
{
    public int Id { get; set; }
    public IEnumerable<PEAUpdateDTO> LiAtletas { get; set; }
}
```

PEAUpdateDTO.

```
public class PEAUpdateDTO
{
    public int AtletaId { get; set; }
    public decimal Puntaje { get; set; }
}
```

PuntajeEventoAtletaDTO.

```
public class PuntajeEventoAtletaDTO
{
    public AtletaDTO Atleta { get; set; }
    public decimal Puntaje { get; set; }
}
```

AtletaEventoListaDTO.

```
public class AtletaEventoListaDTO
{
    public int Id { get; set; }
    public string NombrePrueba { get; set; }
    public string NombreDisciplina { get; set; }
    public DateTime FchInicio { get; set; }
    public DateTime FchFin { get; set; }
}
```

4.4.2 Mappers.

UsuarioMapper.

```
public static class UsuarioMapper
{
```

```
    public static UsuarioDTO UsuarioToDTO(Usuario usuario)
    {
        if (usuario == null)
        {
            throw new UsuarioException("Error en mapper: el Usuario ingresado está
vacío");
        }
        return new UsuarioDTO
        {
            Id = usuario.Id,
            Email = usuario.Email.Valor,
            RolUsuario = usuario.RolUsuario.ToString(),
            Nombre = usuario.Nombre,
            FechaRegistro = usuario.FechaRegistro.ToString("dd/MM/yyyy")
        };
    }
    public static Usuario InsertDTOToUsuario(UsuarioInsertDTO usuarioInsertDto)
    {
        if (usuarioInsertDto == null)
        {
            throw new UsuarioException("Error en mapper: el Usuario insert está vacío");
        }
        return new Usuario
        {
            Email = new RUsuarioEmail(usuarioInsertDto.Email),
            Nombre = usuarioInsertDto.Nombre,
            Contrasena = new RUsuarioContrasena(usuarioInsertDto.Contrasena),
            RolUsuario = usuarioInsertDto.RolUsuario == 0 ? Rol.Administrador :
Rol.Digitador,
            IdAdminRegistro = usuarioInsertDto.IdAdminRegistro
```

```
        };
    }
    public static IEnumerable<UsuarioDTO>
ListaUsuariosToDTOListaUsuarios(List<Usuario> usuarios)
    {
        IEnumerable<UsuarioDTO> usuariosDTOs = usuarios.Select(u => new
UsuarioDTO()
        {
            Email = u.Email.Valor,
            Nombre = u.Nombre,
            Id = u.Id,
            RolUsuario = u.RolUsuario.ToString(),
            FechaRegistro = u.FechaRegistro.ToString("dd/MM/yyyy")
        });
        return usuariosDTOs;
    }
    public static UsuarioUpdateDTO UsuarioToUpdateDTO(Usuario usuario)
    {
        if (usuario == null)
        {
            throw new UsuarioException("Error en mapper: el Usuario está vacío");
        }
        return new UsuarioUpdateDTO
        {
            Email = usuario.Email.Valor,
            Nombre = usuario.Nombre
        };
    }
}
```

PaisMapper.

```
public static class PaisMapper
{}
```

DisciplinaMapper.

```
public static class DisciplinaMapper
{
   public static IEnumerable<DisciplinaListaDTO>
DisciplinasToListaDTO(List<Disciplina> disciplinas)
   {
      IEnumerable<DisciplinaListaDTO> disciplinasListaDTO = disciplinas.Select(d =>
new DisciplinaListaDTO
      {
         Id = d.Id,
         Nombre = d.Nombre.Valor,
         AnioIntegracion = d.AnioIntegracion
      });
      return disciplinasListaDTO;
   }
   public static Disciplina InsertDTOToDisciplina(DisciplinaInsertDTO
disciplinaInsertDTO)
   {
      if (disciplinaInsertDTO == null)
      {
         throw new DisciplinaException("Discplina insert vacia en mapper");
      }
      Disciplina res = new Disciplina
      {
         Nombre = new RDisciplinaNombre(disciplinaInsertDTO.Nombre),
         AnioIntegracion = disciplinaInsertDTO.AnioIntegracion,
```

```
        };
        return res;
    }
}
```

AtletaMapper.

```
public static class AtletaMapper
{
    public static IEnumerable<AtletaListaDTO> AtletasToListaDTO(List<Atleta>
atletas)
    {
        return atletas.Select(a => new AtletaListaDTO
        {
            Id = a.Id,
            Nombre = a.Nombre,
            Apellido = a.Apellido,
            NombrePais = a.Pais.Nombre,
            Sexo = a.Sexo.ToString(),
        });
    }
    public static AtletaDTO AtletaToDTO(Atleta atleta)
    {
        if (atleta == null)
        {
            throw new AtletaException("Atleta vacío en mapper");
        }
        AtletaDTO res = new AtletaDTO
        {
            Id = atleta.Id,
```

```
            Nombre = atleta.Nombre,
            Apellido = atleta.Apellido,
            NombrePais = atleta.Pais.Nombre,
            Sexo = atleta.Sexo.ToString(),
            DisciplinasAtleta =
DisciplinaMapper.DisciplinasToListaDTO(atleta.LiDisciplinas)
        };
        return res;
    }
    public static AtletaDTO AtletaToPuntajeEventoDTO(Atleta atleta)
    {
        if (atleta == null)
        {
            throw new AtletaException("Atleta vacío en mapper");
        }
        AtletaDTO res = new AtletaDTO
        {
            Id = atleta.Id,
            Nombre = atleta.Nombre,
            Apellido = atleta.Apellido,
            NombrePais = atleta.Pais.Nombre,
            Sexo = atleta.Sexo.ToString()
        };
        return res;
    }
}
```

EventoMapper.

```
public static class EventoMapper
```

```
{
    public static Evento DtoToEvento(EventoInsertDTO dto)
    {
        if (dto == null) throw new EventoException("Evento insert DTO vacío en
mapper");
        Evento res = new Evento
        {
            DisciplinaId = dto.DisciplinaId,
            FchInicio = dto.FchInicio,
            FchFin = dto.FchFin,
            NombrePrueba = dto.NombrePrueba,
            LiPuntajes = dto.AtletasId.Select(id => new PuntajeEventoAtleta
            {
                AtletaId = id,
                Puntaje = 0
            }).ToList()
        };
        return res;
    }
    public static IEnumerable<EventoListaDTO> EventosToListaDTO(List<Evento>
listaEventos)
    {
        IEnumerable<EventoListaDTO> eventos = listaEventos.Select(e => new
EventoListaDTO
        {
            EventoId = e.Id,
            NombrePrueba = e.NombrePrueba,
            FchInicio = e.FchInicio,
            FchFin = e.FchFin
        });
```

```
        return eventos;
    }
    public static EventoDTO EventoToDTO(Evento evento)
    {
        if (evento == null) throw new EventoException("Evento vacío en mapper");
        EventoDTO eventoDto = new EventoDTO
        {
            Id = evento.Id,
            FchInicio = evento.FchInicio,
            FchFin = evento.FchFin,
            NombrePrueba = evento.NombrePrueba,
            LiAtletas = evento.LiPuntajes.Select(p => new PuntajeEventoAtletaDTO
            {
                Atleta = AtletaMapper.AtletaToPuntajeEventoDTO(p.Atleta),
                Puntaje = p.Puntaje
            })
        };
        return eventoDto;
    }
    public static List<PuntajeEventoAtleta>
DtoListaModificadaToListaPuntaje(IEnumerable<PEAUpdateDTO> listaModificada)
    {
        if (listaModificada == null) throw new EventoException("Lista de puntajes DTO
vacía en mapper");
        return listaModificada.Select(p => new PuntajeEventoAtleta
        {
            AtletaId = p.AtletaId,
            Puntaje = p.Puntaje,
        }).ToList();
    }
```

```
    public static IEnumerable<AtletaEventoListaDTO>
EventoToListaAtletaEventoDTO(List<Evento> lista)
    {
        if (lista == null) throw new EventoException("Lista de eventos vacía en mapper");
        return lista.Select(e => new AtletaEventoListaDTO
        {
            Id = e.Id,
            NombrePrueba = e.NombrePrueba,
            NombreDisciplina = e.Disciplina.Nombre.Valor,
            FchFin = e.FchFin,
            FchInicio = e.FchInicio
        });
    }
}
```

4.5 MVC.

4.5.1 Controllers.

HomeController.

```
public IActionResult Index()
{
    return View();
}
```

UsuarioController.

```
public class UsuarioController : Controller
{
    private ILoginUsuario _loginUsuario;
    private IGetByIdUsuario _getByIdUsuario;
    private IAltaUsuario _altaUsuario;
```

```
private IFindAllUsuarios _findAllUsuarios;
private IUpdateUsuario _updateUsuario;
private IDeleteUsuario _deleteUsuario;
public UsuarioController(
    ILoginUsuario loginUsuario,
    IGetByIdUsuario getByIdUsuario,
    IAltaUsuario altaUsuario,
    IFindAllUsuarios findAllUsuarios,
    IUpdateUsuario updateUsuario,
    IDeleteUsuario deleteUsuario
)
{
    _loginUsuario = loginUsuario;
    _getByIdUsuario = getByIdUsuario;
    _altaUsuario = altaUsuario;
    _findAllUsuarios = findAllUsuarios;
    _updateUsuario = updateUsuario;
    _deleteUsuario = deleteUsuario;
}
// GET: UsuarioController
public ActionResult Index()
{
    int? usuarioId = ManejoSession.GetIdLogueado(HttpContext);
    if (usuarioId != null)
    {
        return RedirectToAction("Details", new { id = usuarioId });
    }
    return View();
}
```

```
[HttpPost]
public ActionResult Index(string email, string password)
{
    try
    {
        UsuarioDTO res = _loginUsuario.Ejecutar(email, password);
        HttpContext.Session.SetInt32("idLogueado", res.Id);
        HttpContext.Session.SetString("rolLogueado", res.RolUsuario);
        return RedirectToAction("Details", new { id = res.Id });
    }
    catch (UsuarioException uex)
    {
        ViewBag.ErrorMessage = uex.Message;
    }
    catch (Exception ex)
    {
        ViewBag.ErrorMessage = ex.Message;
    }
    return View();
}
[HttpGet]
public ActionResult CerrarSesion()
{
    HttpContext.Session.Clear();
    return RedirectToAction("Index", "Home");
}
// GET: UsuarioController/Details/5
public ActionResult Details(int id)
{
```

```
        try
        {
            if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" &&
ManejoSession.GetIdLogueado(HttpContext) != null)
            {
                UsuarioDTO usuarioDTO = _getByIdUsuario.Ejecutar(id);
                UsuarioVM usuarioVM = new UsuarioVM
                {
                    Email = usuarioDTO.Email,
                    Nombre = usuarioDTO.Nombre,
                    Id = id,
                    RolUsuario = usuarioDTO.RolUsuario,
                    FechaRegistro = usuarioDTO.FechaRegistro
                };
                return View(usuarioVM);
            }
            // Caso de que no sea administrador
            if (id != ManejoSession.GetIdLogueado(HttpContext))
            {
                return RedirectToAction("Index", "Error", new { code = 401, message = "No
tiene permisos para ver esta información" });
            }
            else
            {
                UsuarioDTO usuarioDTO = _getByIdUsuario.Ejecutar(id);
                UsuarioVM usuarioVM = new UsuarioVM
                {
                    Email = usuarioDTO.Email,
                    Nombre = usuarioDTO.Nombre,
                    Id = id,
```

```
                RolUsuario = usuarioDTO.RolUsuario,
                FechaRegistro = usuarioDTO.FechaRegistro
            };
            return View(usuarioVM);
        }
    }
    catch (UsuarioException uex)
    {
        ViewBag.ErrorMessage = uex.Message;
    }
    catch (Exception ex)
    {
        ViewBag.ErrorMessage = ex.Message;
    }
    return View();
}
// GET: UsuarioController/Create
public ActionResult Create()
{
    if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" &&
ManejoSession.GetIdLogueado(HttpContext) != null)
    {
        ViewBag.Roles = GetUsuarioRoles();
        return View();
    }
    return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene
permisos para ver esta información" });
}
// POST: UsuarioController/Create
[HttpPost]
```

```
    [ValidateAntiForgeryToken]
    public ActionResult Create(UsuarioInsertVM usuarioInsertVM)
    {
        if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" &&
ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            ViewBag.Roles = GetUsuarioRoles();
            try
            {
                if (usuarioInsertVM == null)
                {
                    throw new UsuarioException("El usuario no puede ser vacío");
                }
                UsuarioInsertDTO usuarioNuevo = new UsuarioInsertDTO()
                {
                    Email = usuarioInsertVM.Email,
                    Contrasena = usuarioInsertVM.Contrasena,
                    Nombre = usuarioInsertVM.Nombre,
                    RolUsuario = usuarioInsertVM.RolUsuario,
                    IdAdminRegistro = (int)ManejoSession.GetIdLogueado(HttpContext)
                };
                _altaUsuario.Ejecutar(usuarioNuevo);
                TempData["Message"] = "Alta realizada con éxito";
                return RedirectToAction("Create");
            }
            catch (UsuarioException uex)
            {
                ViewBag.ErrorMessage = uex.Message;
                return View();
```

```
            }
            catch (Exception ex)
            {
                ViewBag.ErrorMessage = ex.Message;
                return View();
            }
        }
        else
        {
            return RedirectToAction("Index", "Error", new { code = 401, message = "No
tiene permisos para ver esta información" });
        }
    }
    // GET: UsuarioController/Edit/5
    public ActionResult Edit(int id)
    {
        // Lo primero chequear si hay sesion
        int? idLogueado = ManejoSession.GetIdLogueado(HttpContext);
        if (idLogueado != null)
        {
            // Si es administrador o digitador con mismo id al que intenta modificar
            if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" ||
idLogueado == id)
            {
                try
                {
                    // Obtengo los datos
                    UsuarioDTO usuarioDTO = _getByIdUsuario.Ejecutar(id);
                    UsuarioUpdateVM vm = new UsuarioUpdateVM
                    {
```

```
                Email = usuarioDTO.Email,
                Id = usuarioDTO.Id,
                Nombre = usuarioDTO.Nombre
            };
            ViewBag.Roles = GetUsuarioRoles();
            return View(vm);
        }
        catch (UsuarioException uex)
        {
            ViewBag.ErrorMessage = uex.Message;
        }
        catch (Exception ex)
        {
            ViewBag.ErrorMessage = ex.Message;
        }
        return View();
    }
}
// Si no cumplio ninguno de esos escenarios
return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
}
// POST: UsuarioController/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, UsuarioUpdateVM usuarioUpdateVM)
{
int? idLogueado = ManejoSession.GetIdLogueado(HttpContext);
if (idLogueado != null)
```

```
        {
            ViewBag.Roles = GetUsuarioRoles();
            UsuarioUpdateDTO usuarioUpdateDTO = new UsuarioUpdateDTO
            {
                Email = usuarioUpdateVM.Email,
                Nombre = usuarioUpdateVM.Nombre
            };
            // En caso de ser o Administrador un digitador con mismo id
            if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" ||
idLogueado == id)
            {
                try
                {
                    // Luego de actualizar, devuelvo los nuevos datos para mantenerlos en la
vista
                    UsuarioUpdateDTO nuevosDatosUsuario = _updateUsuario.Ejecutar(id,
usuarioUpdateDTO);
                    UsuarioUpdateVM vm = new UsuarioUpdateVM
                    {
                        Email = nuevosDatosUsuario.Email,
                        Nombre = nuevosDatosUsuario.Nombre,
                        Id = id,
                    };
                    ViewBag.Message = "Actualizado correctamente";
                    return View(vm);
                }
                catch (UsuarioException uex)
                {
                    ViewBag.ErrorMessage = uex.Message;
                }
```

```
            catch (Exception ex)
            {
                ViewBag.ErrorMessage = ex.Message;
            }
            return View(usuarioUpdateVM); // VM que llega por parametro al Edit
        }
    }
    return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult CambiarContrasena(int id, string contrasena, string contrasenaAnterior)
{
    int? idLogueado = ManejoSession.GetIdLogueado(HttpContext);
    string? rolLogueado = ManejoSession.GetRolLogueado(HttpContext);
    if (idLogueado != null)
    {
        // Obtengo los datos
        // No se tira excepcion aca porque el metodo getById ya lo hace si no lo encuentra
        UsuarioDTO usuarioDTO = _getByIdUsuario.Ejecutar(id);
        UsuarioUpdateVM vm = new UsuarioUpdateVM
        {
            Email = usuarioDTO.Email,
            Id = usuarioDTO.Id,
            Nombre = usuarioDTO.Nombre
        };
        ViewBag.Roles = GetUsuarioRoles();
```

```
        if (rolLogueado == "Administrador")
        {
            try
            {
                // Intento cambiar la contraseña
                // El admin puede cambiar la constraseña sin necesidad de escribir la actual, sobreescribe
                UsuarioUpdateDTO usuarioActualizado = _updateUsuario.Ejecutar(id, contrasena);
                TempData["MessageContrasena"] = "Contraseña actualizada correctamente";
                return RedirectToAction("Edit", new { Id = id });
            }
            catch (UsuarioException uex)
            {
                TempData["ErrorContrasena"] = uex.Message;
            }
            catch (Exception ex)
            {
                TempData["ErrorContrasena"] = ex.Message;
            }
            return RedirectToAction("Edit", new { Id = id });
        }
        if (id == idLogueado)
        {
            try
            {
                // Intento cambiar la contraseña
                UsuarioUpdateDTO usuarioActualizado = _updateUsuario.Ejecutar(id, contrasena, contrasenaAnterior);
```

```
                TempData["MessageContrasena"] = "Contraseña actualizada
correctamente";
                return RedirectToAction("Edit", new { Id = id });
            }
            catch (UsuarioException uex)
            {
                TempData["ErrorContrasena"] = uex.Message;
            }
            catch (Exception ex)
            {
                TempData["ErrorContrasena"] = ex.Message;
            }
            return RedirectToAction("Edit", new { Id = id });
        }
    }
    return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene
permisos para ver esta información" });
}
// GET: UsuarioController/Delete/5
public ActionResult Delete(int id)
{
    if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" &&
ManejoSession.GetIdLogueado(HttpContext) != null)
    {
        try
        {
            _deleteUsuario.Ejecutar(id);
            TempData["MessageDelete"] = "Usuario eliminado con éxito";
        }
        catch (UsuarioException uex)
```

```
            {
                TempData["MessageDelete"] = uex.Message;
            }
            catch (Exception ex)
            {
                TempData["MessageDelete"] = ex.Message;
            }
            return RedirectToAction("ListaUsuarios");
        }
        return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
    }
    [HttpGet("Usuario/Lista-Usuarios")]
    public ActionResult ListaUsuarios()
    {
        if (ManejoSession.GetRolLogueado(HttpContext) == "Administrador" && ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            IEnumerable<UsuarioVM> listaUsuarios =
                _findAllUsuarios.Ejecutar().Select(u => new UsuarioVM()
                {
                    Email = u.Email,
                    Nombre = u.Nombre,
                    Id = u.Id,
                    RolUsuario = u.RolUsuario,
                }).ToList();
            return View(listaUsuarios);
        }
        return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
```

```
    }
    public IEnumerable<UsuarioRolVM> GetUsuarioRoles()
    {
        // Primero se obtienen los valores del enum Rol
        var roles = Enum.GetValues(typeof(Rol)).Cast<Rol>().ToList();
        // Convertir a lista con el valor int y string de cada Rol
        return roles.Select(r => new UsuarioRolVM
        {
            Id = (int)r,
            Name = r.ToString()
        });
    }
}
```

DisciplinaController.

```
public class DisciplinaController : Controller
{
    private readonly IAltaDisciplina _altaDisciplina;
    private readonly IFindAllDisciplinas _findAllDisciplinas;
    private readonly IDeleteDisciplina _deleteDisciplina;
    public DisciplinaController(
        IAltaDisciplina altaDisciplina,
        IFindAllDisciplinas findAllDisciplinas,
        IDeleteDisciplina deleteDisciplina
    )
    {
        _altaDisciplina = altaDisciplina;
        _findAllDisciplinas = findAllDisciplinas;
        _deleteDisciplina = deleteDisciplina;
```

```
    }
    // GET: DisciplinaController
    public ActionResult Index()
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null &&
ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
        {
            try
            {
                return View(_findAllDisciplinas.Ejecutar().Select(d => new
DisciplinaListaVM
                {
                    Id = d.Id,
                    Nombre = d.Nombre,
                    AnioIntegracion = d.AnioIntegracion
                }));
            }
            catch (DisciplinaException dex)
            {
                return RedirectToAction("Index", "Error", new { code = 400, message =
dex.Message });
            }
            catch (Exception ex)
            {
                return RedirectToAction("Index", "Error", new { code = 400, message =
ex.Message });
            }
        }
        else
        {
```

```
            return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
        }
    }
    // GET: DisciplinaController/Details/5
    public ActionResult Details(int id)
    {
        return View();
    }
    // GET: DisciplinaController/Create
    public ActionResult Create()
    {
        return View();
    }
    // POST: DisciplinaController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(DisciplinaInsertVM disciplinaInsertVM)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null && ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
        {
            try
            {
                DisciplinaInsertDTO disciplina = new DisciplinaInsertDTO
                {
                    Nombre = disciplinaInsertVM.Nombre,
                    AnioIntegracion = disciplinaInsertVM.AnioIntegracion
                };
                _altaDisciplina.Ejecutar(disciplina);
```

```
            TempData["Message"] = "Disciplina creada correctamente";
            return RedirectToAction("Create");
        }
        catch (DisciplinaException dex)
        {
            ViewBag.ErrorMessage = dex.Message;
            return View();
        }
        catch (Exception ex)
        {
            ViewBag.ErrorMessage = ex.Message;
            return View();
        }
    }
    else
    {
        return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
    }
}
// GET: DisciplinaController/Edit/5
public ActionResult Edit(int id)
{
    return View();
}
// POST: DisciplinaController/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, IFormCollection collection)
```

```
    {
        try
        {
            return RedirectToAction(nameof(Index));
        }
        catch
        {
            return View();
        }
    }
    // GET: DisciplinaController/Delete/5
    public ActionResult Delete(int id)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null &&
ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
        {
            try
            {
                _deleteDisciplina.Ejecutar(id);
                TempData["Message"] = "Disciplina eliminada correctamente";
                return RedirectToAction("Index");
            }
            catch (DisciplinaException dex)
            {
                TempData["ErrorMessage"] = dex.Message;
            }
            catch (Exception ex)
            {
                TempData["ErrorMessage"] = "Algo no salió correctamente. Es posible que
existan referencias de esta Disciplina en Atletas.";
```

```
            }
            return RedirectToAction("Index");
        }
        else
        {
            return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
        }
    }
    // POST: DisciplinaController/Delete/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id, IFormCollection collection)
    {
        try
        {
            return RedirectToAction(nameof(Index));
        }
        catch
        {
            return View();
        }
    }
}
```

AtletaController.

```
public class AtletaController : Controller
{
    private readonly IFindAllAtletas _findAllAtletas;
    private readonly IGetByIdAtleta _getByIdAtleta;
    private readonly IAgregarDisciplina _agregarDisciplina;
    private readonly IFindAllDisciplinas _findAllDisciplinas;
    public AtletaController(IFindAllAtletas findAllAtletas, IGetByIdAtleta
getByIdAtleta, IAgregarDisciplina agregarDisciplina, IFindAllDisciplinas
findAllDisciplinas)
    {
        _findAllAtletas = findAllAtletas;
        _getByIdAtleta = getByIdAtleta;
        _agregarDisciplina = agregarDisciplina;
        _findAllDisciplinas = findAllDisciplinas;
    }
    // GET: AtletaController
    public ActionResult Index()
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null &&
ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
        {
            IEnumerable<AtletaListaVM> res = null;
            try
            {
                res = _findAllAtletas.Ejectuar().Select(a => new AtletaListaVM
                {
                    Id = a.Id,
                    Nombre = a.Nombre,
                    Apellido = a.Apellido,
```

```
                NombrePais = a.NombrePais,
                Sexo = a.Sexo,
            });
        }
        catch (Exception ex)
        {
            return RedirectToAction("Index", "Error");
        }
        return View(res);
    }
    else
    {
        return RedirectToAction("Index", "Error", new { code = 401, message = "No
tiene permisos para ver esta información" });
    }
}
// GET: AtletaController/Details/5
public ActionResult Details(int id)
{
    if (ManejoSession.GetIdLogueado(HttpContext) != null &&
ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
    {
        try
        {
            IEnumerable<DisciplinaListaDTO> liD = _findAllDisciplinas.Ejecutar();
            ViewBag.Disciplinas = liD.Select(l => new DisciplinaListaVM
            {
                Id = l.Id,
                Nombre = l.Nombre,
            });
```

```
            AtletaDTO atletaDTO = _getByIdAtleta.Ejecutar(id);
            AtletaVM atletaVM = new AtletaVM
            {
                Id = atletaDTO.Id,
                Nombre = atletaDTO.Nombre,
                Apellido = atletaDTO.Apellido,
                Sexo = atletaDTO.Sexo,
                NombrePais = atletaDTO.NombrePais,
                DisciplinasAtleta = atletaDTO.DisciplinasAtleta.Select(d => new
DisciplinaListaVM
                {
                    Id = d.Id,
                    Nombre = d.Nombre,
                })
            };
            return View(atletaVM);
        }
        catch (AtletaException aex)
        {
            return RedirectToAction("Index", "Error", new { code = 404, message =
aex.Message });
        }
        catch (Exception ex)
        {
            return View();
        }
    }
    else
    {
```

```
            return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
        }
    }
    // GET: AtletaController/Create
    public ActionResult Create()
    {
        return View();
    }
    // POST: AtletaController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(IFormCollection collection)
    {
        try
        {
            return RedirectToAction(nameof(Index));
        }
        catch
        {
            return View();
        }
    }
    // GET: AtletaController/Edit/5
    public ActionResult Edit(int id)
    {
        return View();
    }
    // POST: AtletaController/Edit/5
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, IFormCollection collection)
{
    try
    {
        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}
// GET: AtletaController/Delete/5
public ActionResult Delete(int id)
{
    return View();
}
// POST: AtletaController/Delete/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Delete(int id, IFormCollection collection)
{
    try
    {
        return RedirectToAction(nameof(Index));
    }
    catch
    {
```

```
            return View();
        }
    }
    [HttpPost]
    public ActionResult AgregarDisciplina(int? id, int? idDisciplina)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null &&
ManejoSession.GetRolLogueado(HttpContext) == "Digitador")
        {
            if (id == null)
            {
                return RedirectToAction("Index", "Error", new { code = 404, message = "El
Atleta no existe" });
            }
            IEnumerable<DisciplinaListaDTO> liD = _findAllDisciplinas.Ejecutar();
            ViewBag.Disciplinas = liD.Select(l => new DisciplinaListaVM
            {
                Id = l.Id,
                Nombre = l.Nombre,
            });
            try
            {
                _agregarDisciplina.Ejecutar(id.Value, idDisciplina);
                TempData["Message"] = "Atleta registrado correctamente en la disciplina";
            }
            catch (AtletaException aex)
            {
                TempData["ErrorMessage"] = aex.Message;
            }
            catch (DisciplinaException dex)
```

```
            {
                TempData["ErrorMessage"] = dex.Message;
            }
            catch (Exception ex)
            {
                TempData["ErrorMessage"] = "Algo no sucedió correctamente";
            }
            return RedirectToAction("Details", new { id = id.Value });
        }
        else
        {
            return RedirectToAction("Index", "Error", new { code = 401, message = "No tiene permisos para ver esta información" });
        }
    }
}
```

EventoController.

```
public class EventoController : Controller
{
    private readonly IFindAtletasDisciplina _findAtletasDisciplina;
    private readonly IFindAllDisciplinas _findAllDisciplinas;
    private readonly IAltaEvento _altaEvento;
    private readonly IFindEventosFecha _findEventosFecha;
    private readonly IFindById _findById;
    private readonly ICargarPuntajes _cargarPuntajes;
    public EventoController(IFindAtletasDisciplina findAtletasDisciplina, IFindAllDisciplinas findAllDisciplinas, IAltaEvento altaEvento, IFindEventosFecha findEventosFecha, IFindById findById, ICargarPuntajes cargarPuntajes)
    {
```

```
        _findAtletasDisciplina = findAtletasDisciplina;
        _findAllDisciplinas = findAllDisciplinas;
        _altaEvento = altaEvento;
        _findEventosFecha = findEventosFecha;
        _findById = findById;
        _cargarPuntajes = cargarPuntajes;
    }
    // GET: EventoController
    public ActionResult Index()
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            try
            {
                ViewBag.Disciplinas = _findAllDisciplinas.Ejecutar().Select(d => new
DisciplinaListaVM
                {
                    Id = d.Id,
                    Nombre = d.Nombre,
                });
            }
            catch (DisciplinaException dex)
            {
                ViewBag.ErrorMessage = dex.Message;
            }
            catch (Exception ex)
            {
                ViewBag.ErrorMessage = ex.Message;
            }
```

```
            return View();
        }
        return RedirectToAction("Index", "Error");
    }
    // GET: EventoController/Details/5
    public ActionResult Details(int id)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            try
            {
                EventoDTO eventoDTO = _findById.Ejecutar(id);
                EventoVM eventoVM = new EventoVM()
                {
                    Id = eventoDTO.Id,
                    FchInicio = eventoDTO.FchInicio,
                    FchFin = eventoDTO.FchFin,
                    NombrePrueba = eventoDTO.NombrePrueba,
                    LiPuntajes = eventoDTO.LiAtletas.Select(p => new
PuntajeEventoAtletaVM
                    {
                        Atleta = new AtletaVM
                        {
                            Id = p.Atleta.Id,
                            Nombre = p.Atleta.Nombre,
                            Apellido = p.Atleta.Apellido,
                            NombrePais = p.Atleta.NombrePais,
                            Sexo = p.Atleta.Sexo
                        },
```

```
                Puntaje = p.Puntaje
            }).ToList()
        };
        return View(eventoVM);
    }
    catch (EventoException eex)
    {
        TempData["ErrorMessage"] = eex.Message;
    }
    catch (AtletaException aex)
    {
        TempData["ErrorMessage"] = aex.Message;
    }
    catch (Exception ex)
    {
        TempData["ErrorMessage"] = "Algo no salió correctamente";
    }
    return View();
}
    return RedirectToAction("Index", "Error", new { code = 404, message = "No tiene
permisos para ver este recurso" });
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Details(EventoVM eventoVM)
{
    if (ManejoSession.GetIdLogueado(HttpContext) != null)
    {
        try
```

```
        {
            EventoUpdatePuntajesDTO eventoModificado = new
EventoUpdatePuntajesDTO()
            {
               Id = eventoVM.Id,
               LiAtletas = eventoVM.LiPuntajes.Select(p => new PEAUpdateDTO
               {
                  Puntaje = p.Puntaje,
                  AtletaId = p.Atleta.Id
               })
            };
            EventoDTO eventoDTO = _cargarPuntajes.Ejecutar(eventoModificado);
            EventoVM eventoVMModificado = new EventoVM()
            {
               Id = eventoDTO.Id,
               FchInicio = eventoDTO.FchInicio,
               FchFin = eventoDTO.FchFin,
               NombrePrueba = eventoDTO.NombrePrueba,
               LiPuntajes = eventoDTO.LiAtletas.Select(p => new
PuntajeEventoAtletaVM
               {
                  Atleta = new AtletaVM
                  {
                     Id = p.Atleta.Id,
                     Nombre = p.Atleta.Nombre,
                     Apellido = p.Atleta.Apellido,
                     NombrePais = p.Atleta.NombrePais,
                     Sexo = p.Atleta.Sexo
                  },
                  Puntaje = p.Puntaje
```

```
                }).ToList()
            };
            TempData["Message"] = "Puntajes actualizados";
            return View(eventoVMModificado);
        }
        catch (EventoException eex)
        {
            TempData["ErrorMessage"] = eex.Message;
        }
        catch (AtletaException aex)
        {
            TempData["ErrorMessage"] = aex.Message;
        }
        catch (Exception ex)
        {
            TempData["ErrorMessage"] = "Algo no salió correctamente";
        }
        return RedirectToAction("Details", new { id = eventoVM.Id });
    }
    return RedirectToAction("Index", "Error", new { code = 404, message = "No tiene permisos para ver este recurso" });
}
// GET: EventoController/Create
public ActionResult Create(int idDisciplina)
{
    if (ManejoSession.GetIdLogueado(HttpContext) != null)
    {
        EventoInsertVM EventoVM = new EventoInsertVM();
        try
```

```
        {
            IEnumerable<AtletaListaVM> atletas =
_findAtletasDisciplina.Ejecutar(idDisciplina).Select(a => new AtletaListaVM()
            {
                Id = a.Id,
                Nombre = a.Nombre,
                Apellido = a.Apellido,
                NombrePais = a.NombrePais,
                Sexo = a.Sexo
            });
            EventoVM.DisciplinaId = idDisciplina;
            EventoVM.Atletas = atletas;
            return View(EventoVM);
        }
        catch (EventoException eex)
        {
            TempData["ErrorMessage"] = eex.Message;
        }
        catch (AtletaException aex)
        {
            TempData["ErrorMessage"] = aex.Message;
        }
        catch (Exception ex)
        {
            TempData["ErrorMessage"] = "Algo no salió correctamente, por favor intente
nuevamente";
        }
        return View(EventoVM);
    }
    return RedirectToAction("Index", "Error");
```

```
    }
    // POST: EventoController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(EventoInsertVM eventoInsertVM)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            try
            {
                EventoInsertDTO evento = new EventoInsertDTO()
                {
                    DisciplinaId = eventoInsertVM.DisciplinaId,
                    FchInicio = eventoInsertVM.FchInicio,
                    FchFin = eventoInsertVM.FchFin,
                    NombrePrueba = eventoInsertVM.NombrePrueba,
                    AtletasId = eventoInsertVM.AtletasId
                };
                _altaEvento.Ejecutar(evento);
                TempData["Message"] = "Evento agregado con éxito";
                return RedirectToAction("Create", new { idDisciplina =
eventoInsertVM.DisciplinaId });
            }
            catch (EventoException eex)
            {
                TempData["ErrorMessage"] = eex.Message;
            }
            catch (Exception ex)
            {
```

```
            TempData["ErrorMessage"] = ex.Message;
        }
        EventoInsertVM EventoVM = new EventoInsertVM();
        IEnumerable<AtletaListaVM> atletas;
        try
        {
            atletas =
_findAtletasDisciplina.Ejecutar(eventoInsertVM.DisciplinaId).Select(a => new
AtletaListaVM()
            {
                Id = a.Id,
                Nombre = a.Nombre,
                Apellido = a.Apellido,
                NombrePais = a.NombrePais,
                Sexo = a.Sexo
            });
        }
        catch (Exception ex)
        {
            return RedirectToAction("Index", "Error");
        }
        EventoVM.DisciplinaId = eventoInsertVM.DisciplinaId;
        EventoVM.Atletas = atletas;
        return View(EventoVM);
    }
    return RedirectToAction("Index", "Error", new { code = 404, message = "No tiene
permisos para ver este recurso" });
}
// GET: EventoController/Edit/5
public ActionResult Edit(int id)
```

```
{
    return View();
}
// POST: EventoController/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, IFormCollection collection)
{
    try
    {
        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}
// GET: EventoController/Delete/5
public ActionResult Delete(int id)
{
    return View();
}
// POST: EventoController/Delete/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Delete(int id, IFormCollection collection)
{
    try
    {
```

```
            return RedirectToAction(nameof(Index));
        }
        catch
        {
            return View();
        }
    }
    public ActionResult ListEventosPorFecha(DateTime fecha)
    {
        if (ManejoSession.GetIdLogueado(HttpContext) != null)
        {
            try
            {
                IEnumerable<EventoListaDTO> eventosFechaDTO =
_findEventosFecha.Ejecutar(fecha);
                IEnumerable<EventoListaVM> eventosVM = eventosFechaDTO.Select(e =>
new EventoListaVM
                {
                    EventoId = e.EventoId,
                    NombrePrueba = e.NombrePrueba,
                    FchInicio = e.FchInicio,
                    FchFin = e.FchFin
                });
                return View(eventosVM);
            }
            catch (EventoException eex)
            {
                TempData["ErrorMessage"] = eex.Message;
            }
            catch (Exception ex)
```

```
            {
                TempData["ErrorMessage"] = ex.Message;
            }
            return View();
        }
        return RedirectToAction("Index", "Error", new { code = 404, message = "No tiene permisos para ver este recurso" });
    }
}
```

ErrorController.

```
public class ErrorController : Controller
{
    public IActionResult Index(int? code, string? message)
    {
        if (code != null)
        {
            ViewBag.Message = new { code, message };
        }
        return View();
    }
}
```

4.5.2 Models.

4.5.2.1 Usuarios.

UsuarioVM.

```
public class UsuarioVM
{
    [DisplayName("ID")]
    public int Id { get; set; }
```

```
    public string Email { get; set; }
    [DisplayName("Rol")]
    public string RolUsuario { get; set; }
    public string? Nombre { get; set; }
    [DisplayName("Fecha de registro")]
    public string FechaRegistro { get; set; }
}
```

UsuarioInsertVM.

```
public class UsuarioInsertVM
{
    [Required(ErrorMessage = "El Email es requerido")]
    [RegularExpression(@"^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+$",
        ErrorMessage = "El formato del email no es correcto")]
    [MaxLength(255, ErrorMessage = "El email puede contener hasta 255 caracteres")]
    public string? Email { get; set; }
    [DisplayName("Contraseña")]
    [Required(ErrorMessage = "La contraseña es requerida")]
    [RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[\.;,!]).{6,}$",
        ErrorMessage = "La contraseña debe contener al menos 6 caracteres y al menos
una mayúscula, minúscula, dígito y [. ; , !]")]
    [DataType(DataType.Password)]
    public string? Contrasena { get; set; }
    [DisplayName("Rol de usuario")]
    [Required(ErrorMessage = "El Rol es requerido")]
    public int RolUsuario { get; set; }
    [MaxLength(50, ErrorMessage = "El Nombre puede contener hasta 50 caracteres")]
    public string? Nombre { get; set; }
}
```

UsuarioRolVM.

```
public class UsuarioRolVM
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

UsuarioUpdateVM.

```
public class UsuarioUpdateVM
{
    public int Id { get; set; }
    [Required(ErrorMessage = "El Email es requerido")]
    [RegularExpression(@"^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+$",
        ErrorMessage = "El formato del email no es correcto")]
    [MaxLength(255, ErrorMessage = "El email puede contener hasta 255 caracteres")]
    public string Email { get; set; }
    [MaxLength(50, ErrorMessage = "El Nombre puede contener hasta 50 caracteres")]
    public string? Nombre { get; set; }
}
```

4.5.2.2 Disciplina.

DisciplinaInsertVM.

```
public class DisciplinaInsertVM
{
    [DisplayName("Nombre de disciplina")]
    [Length(10,50,ErrorMessage = "El nombre de la disciplina debe tener entre 10 y 50 caracteres")]
    [Required(ErrorMessage = "El nombre es requerido")]
    public string Nombre { get; set; }
```

```
    [DisplayName("Año de integración")]
    [Required(ErrorMessage = "El año es requerido")]
    public int AnioIntegracion { get; set; }
}
```

DisciplinaListaVM.

```
public class DisciplinaListaVM
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    [DisplayName("Año integración")]
    public int AnioIntegracion { get; set; }
}
```

4.5.2.3 Atletas.

AtletaVM.

```
public class AtletaVM
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public string Sexo { get; set; }
    [DisplayName("Nombre del país")]
    public string NombrePais { get; set; }
    [DisplayName("Lista de disciplinas")]
    public IEnumerable<DisciplinaListaVM> DisciplinasAtleta { get; set; }
}
```

AtletaInsertVM.

```
public class AtletaInsertVM
{
    public string Nombre { get; set; }
    public int Sexo { get; set; }
}
```

AtletaListaVM.

```
public class AtletaListaVM
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public string Sexo { get; set; }
    [DisplayName("País")]
    public string NombrePais { get; set; }
}
```

4.5.2.4 Eventos.

EventoVM.

```
public class EventoVM
{
    public int Id { get; set; }
    [DisplayName("Nombre de la prueba")]
    public string NombrePrueba { get; set; }
    [DisplayName("Fecha de inicio")]
    [DataType(DataType.Date)]
    public DateTime FchInicio { get; set; }
    [DisplayName("Fecha de finalización")]
    [DataType(DataType.Date)]
```

```
    public DateTime FchFin { get; set; }
    [Required(ErrorMessage = "Los puntajes son requeridos")]
    public List<PuntajeEventoAtletaVM> LiPuntajes { get; set; }
}
```

EventoInsertVM

```
public class EventoInsertVM
{
    public int DisciplinaId { get; set; }
    [Required(ErrorMessage = "El Nombre no puede estar vacío")]
    [DisplayName("Nombre de la Prueba")]
    public string NombrePrueba { get; set; }
    [Required(ErrorMessage = "La fecha de inicio es requerida")]
    [DisplayName("Fecha Inicio")]
    [DataType(DataType.Date)]
    public DateTime FchInicio { get; set; } = DateTime.Now;
    [DataType(DataType.Date)]
    [Required(ErrorMessage = "La fecha de finalización es requerida")]
    [DisplayName("Fecha Finalización")]
    public DateTime FchFin { get; set; } = DateTime.Now;
    public int[] ? AtletasId { get; set; }
    public IEnumerable<AtletaListaVM> Atletas { get; set; }
}
```

EventoListaVM.

```
public class EventoListaVM
{
    public int EventoId { get; set; }
    [DisplayName("Nombre de la Prueba")]
```

```
    public string NombrePrueba { get; set; }
    [DisplayName("Fecha Inicio")]
    [DataType(DataType.Date)]
    public DateTime FchInicio { get; set; }
    [DataType(DataType.Date)]
    [DisplayName("Fecha Finalización")]
    public DateTime FchFin { get; set; }
}
```

PuntajeEventoAtletaVM.

```
public class PuntajeEventoAtletaVM
{
    public AtletaVM Atleta { get; set; }
    public decimal Puntaje { get; set; }
}
```

4.5.3 Utils.

ManejoSession.

```
public static class ManejoSession
{
    public static int? GetIdLogueado(HttpContext httpContext)
    {
        return httpContext.Session.GetInt32("idLogueado");
    }
    public static string? GetRolLogueado(HttpContext httpContext)
    {
        return httpContext.Session.GetString("rolLogueado");
    }
}
```

4.6 WebApi.

4.6.1 Controllers.

AtletaController.

```
[Route("api/[controller]")]
[ApiController]
public class AtletaController : ControllerBase
{
    private readonly IEventosAtleta _eventosAtleta;
    public AtletaController(IEventosAtleta eventosAtleta)
    {
        _eventosAtleta = eventosAtleta;
    }
    [HttpGet("{atletaId}")]
    public IActionResult GetEventosAtleta(int atletaId)
    {
        try
        {
            if (atletaId <= 0) throw new EventoException("El Id del Atleta no es válido");
            return Ok(_eventosAtleta.Ejecutar(atletaId));
        }
        catch (EventoException eex)
        {
            return BadRequest(eex.Message);
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }}
```