Estoy haciendo el back para una aplicación que gestione una competencia de clavados. la estoy haciendo en asp.net c# y la idea es exponer los endpoints para las ntareas q se realizaran:

\* Poder crear una competencia nueva (generandole un codigo de 6 digitos unico)

\* poder registrar jueces y agregar jueces a una competencia (muchos a muchos)

\* poder registrar divers y agregar divers a una competencia (muchos a muchos)

\* poder registrar dives y agregar dives a un diver (muchos a muchos)

\* poder registrar los scores que los jueces asignen a cada dive en una competencia

\* luego miraremos mas funcionalidades

aqui te voy a compartir los modelos que tengo:

namespace DivingCompetitionAPI.Models

{

public class Competition

{

public int CompetitionId { get; set; }

public string? Code { get; set; }

public string? Name { get; set; }

public DateTime Date { get; set; }

// Navegación hacia los CompetitionDiver y CompetitionJudge

public ICollection<CompetitionDiver> CompetitionDivers { get; set; }

public ICollection<CompetitionJudge> CompetitionJudges { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class Judge

{

public int JudgeId { get; set; }

public string Name { get; set; }

public ICollection<CompetitionJudge> CompetitionJudges { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class Diver

{

public int DiverId { get; set; }

public string Name { get; set; }

public ICollection<CompetitionDiver> CompetitionDivers { get; set; }

public ICollection<DiverDive> DiverDives { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class Dive

{

public int DiveId { get; set; }

public string DiveCode { get; set; }

public double Difficulty { get; set; }

public ICollection<DiverDive> DiverDives { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class Score

{

public int ScoreId { get; set; }

public double Points { get; set; }

public int DiverDiveDiverId { get; set; }

public int DiverDiveDiveId { get; set; }

public int DiverDiveCompetitionId { get; set; }

public DiverDive DiverDive { get; set; }

public int JudgeId { get; set; }

public Judge Judge { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class CompetitionDiver

{

public int CompetitionId { get; set; }

public Competition Competition { get; set; }

public int DiverId { get; set; }

public Diver Diver { get; set; }

}

}

namespace DivingCompetitionAPI.Models

{

public class CompetitionJudge

{

public int CompetitionId { get; set; }

public Competition Competition { get; set; }

public int JudgeId { get; set; }

public Judge Judge { get; set; }

}

}

aqui te dejo los controladores q tengo hasta ahora

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using DivingCompetitionAPI.Data;

using DivingCompetitionAPI.Models;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace DivingCompetitionAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class CompetitionController : ControllerBase

{

private readonly ApplicationDbContext \_context;

public CompetitionController(ApplicationDbContext context)

{

\_context = context;

}

[HttpPost("create")]

public async Task<ActionResult<Competition>> CreateCompetition()

{

var competition = new Competition

{

Code = GenerateUniqueCode(),

Name = "New Competition",

Date = DateTime.UtcNow

};

\_context.Competitions.Add(competition);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetCompetition", new { id = competition.CompetitionId }, competition);

}

private string GenerateUniqueCode()

{

var random = new Random();

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

return new string(Enumerable.Repeat(chars, 6)

.Select(s => s[random.Next(s.Length)]).ToArray());

}

[HttpGet("{id}")]

public async Task<ActionResult<Competition>> GetCompetition(int id)

{

var competition = await \_context.Competitions

.Include(c => c.Divers)

.Include(c => c.Judges)

.FirstOrDefaultAsync(c => c.CompetitionId == id);

if (competition == null)

{

return NotFound(new { message = "Competition not found" });

}

return competition;

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using DivingCompetitionAPI.Data;

using DivingCompetitionAPI.Models;

using System.Threading.Tasks;

namespace DivingCompetitionAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class DiverController : ControllerBase

{

private readonly ApplicationDbContext \_context;

public DiverController(ApplicationDbContext context)

{

\_context = context;

}

[HttpPost("add")]

public async Task<ActionResult<Diver>> AddDiver([FromBody] Diver diver)

{

var competition = await \_context.Competitions.FindAsync(diver.CompetitionId);

if (competition == null)

{

return NotFound(new { message = "Competition not found" });

}

\_context.Divers.Add(diver);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetDiver", new { id = diver.DiverId }, diver);

}

[HttpGet("{id}")]

public async Task<ActionResult<Diver>> GetDiver(int id)

{

var diver = await \_context.Divers.FindAsync(id);

if (diver == null)

{

return NotFound(new { message = "Diver not found" });

}

return diver;

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using DivingCompetitionAPI.Data;

using DivingCompetitionAPI.Models;

using System.Threading.Tasks;

namespace DivingCompetitionAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class JudgeController : ControllerBase

{

private readonly ApplicationDbContext \_context;

public JudgeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpPost("add")]

public async Task<ActionResult<Judge>> AddJudge([FromBody] Judge judge)

{

var competition = await \_context.Competitions.FindAsync(judge.CompetitionId);

if (competition == null)

{

return NotFound(new { message = "Competition not found" });

}

\_context.Judges.Add(judge);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetJudge", new { id = judge.JudgeId }, judge);

}

[HttpGet("{id}")]

public async Task<ActionResult<Judge>> GetJudge(int id)

{

var judge = await \_context.Judges.FindAsync(id);

if (judge == null)

{

return NotFound(new { message = "Judge not found" });

}

return judge;

}

}

}

 **Controlador CompetitionController**:

* Crea una competición y genera un código único.
* Obtiene la competición por ID, incluyendo los clavadistas y jueces asociados.

 **Controlador DiverController**:

* Agrega clavadistas a una competición existente.
* Obtiene un clavadista por ID.

 **Controlador JudgeController**:

* Agrega jueces a una competición existente.
* Obtiene un juez por ID.