**Project structure (suggested)**

dotnetapp/

├─ Controllers/

│ └─ PlayerController.cs

├─ Data/

│ └─ AppDbContext.cs

├─ Interfaces/

│ └─ IPlayerRepository.cs

├─ Models/

│ └─ Player.cs

├─ Repositories/

│ ├─ PlayerRepository.cs // EF Core implementation

│ └─ InMemoryPlayerRepository.cs// Simple in-memory store

├─ Program.cs

├─ dotnetapp.csproj

tests/

└─ MyTestNUnitProject/

├─ MyTestNUnitProject.csproj

├─ PlayerControllerTests.cs

└─ PlayerRepositoryTests.cs

**1) Models/Player.cs**

namespace dotnetapp.Models

{

public class Player

{

public int PlayerId { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Team { get; set; }

public string Position { get; set; }

public int Goals { get; set; }

}

}

**2) Interfaces/IPlayerRepository.cs**

using System.Collections.Generic;

using dotnetapp.Models;

namespace dotnetapp.Interfaces

{

public interface IPlayerRepository

{

IEnumerable<Player> GetPlayers();

Player GetPlayer(int playerId);

Player CreatePlayer(Player player);

Player UpdatePlayer(Player player);

bool DeletePlayer(int playerId);

}

}

**3) Data/AppDbContext.cs (EF Core)**

using Microsoft.EntityFrameworkCore;

using dotnetapp.Models;

namespace dotnetapp.Data

{

public class AppDbContext : DbContext

{

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

public DbSet<Player> Players { get; set; }

}

}

**4) Repositories/PlayerRepository.cs (EF Core implementation)**

using System.Collections.Generic;

using System.Linq;

using dotnetapp.Data;

using dotnetapp.Interfaces;

using dotnetapp.Models;

namespace dotnetapp.Repositories

{

public class PlayerRepository : IPlayerRepository

{

private readonly AppDbContext \_db;

public PlayerRepository(AppDbContext db)

{

\_db = db;

}

public IEnumerable<Player> GetPlayers()

{

return \_db.Players.ToList();

}

public Player GetPlayer(int playerId)

{

return \_db.Players.Find(playerId);

}

public Player CreatePlayer(Player player)

{

\_db.Players.Add(player);

\_db.SaveChanges();

return player;

}

public Player UpdatePlayer(Player player)

{

var existing = \_db.Players.Find(player.PlayerId);

if (existing == null) return null;

existing.Name = player.Name;

existing.Age = player.Age;

existing.Team = player.Team;

existing.Position = player.Position;

existing.Goals = player.Goals;

\_db.SaveChanges();

return existing;

}

public bool DeletePlayer(int playerId)

{

var existing = \_db.Players.Find(playerId);

if (existing == null) return false;

\_db.Players.Remove(existing);

\_db.SaveChanges();

return true;

}

}

}

**5) Repositories/InMemoryPlayerRepository.cs (useful for quick dev/tests)**

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Linq;

using dotnetapp.Interfaces;

using dotnetapp.Models;

namespace dotnetapp.Repositories

{

public class InMemoryPlayerRepository : IPlayerRepository

{

private readonly ConcurrentDictionary<int, Player> \_store = new();

private int \_idCounter = 0;

public IEnumerable<Player> GetPlayers()

{

return \_store.Values.OrderBy(p => p.PlayerId);

}

public Player GetPlayer(int playerId)

{

\_store.TryGetValue(playerId, out var p);

return p;

}

public Player CreatePlayer(Player player)

{

var id = System.Threading.Interlocked.Increment(ref \_idCounter);

player.PlayerId = id;

\_store[id] = player;

return player;

}

public Player UpdatePlayer(Player player)

{

if (!\_store.ContainsKey(player.PlayerId)) return null;

\_store[player.PlayerId] = player;

return player;

}

public bool DeletePlayer(int playerId)

{

return \_store.TryRemove(playerId, out \_);

}

}

}

**6) Controllers/PlayerController.cs**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using dotnetapp.Interfaces;

using dotnetapp.Models;

namespace dotnetapp.Controllers

{

[ApiController]

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

public class PlayersController : ControllerBase

{

private readonly IPlayerRepository \_repo;

public PlayersController(IPlayerRepository repo)

{

\_repo = repo;

}

// GET /api/players

[HttpGet]

public ActionResult<IEnumerable<Player>> Get()

{

var players = \_repo.GetPlayers();

return Ok(players);

}

// GET /api/players/{id}

[HttpGet("{id:int}")]

public ActionResult<Player> Get(int id)

{

var player = \_repo.GetPlayer(id);

if (player == null) return NotFound();

return Ok(player);

}

// POST /api/players

[HttpPost]

public ActionResult<Player> Post([FromBody] Player player)

{

if (player == null) return BadRequest();

var created = \_repo.CreatePlayer(player);

return CreatedAtAction(nameof(Get), new { id = created.PlayerId }, created);

}

// PUT /api/players/{id}

[HttpPut("{id:int}")]

public ActionResult<Player> Put(int id, [FromBody] Player player)

{

if (player == null || id != player.PlayerId) return BadRequest();

var updated = \_repo.UpdatePlayer(player);

if (updated == null) return NotFound();

return Ok(updated);

}

// DELETE /api/players/{id}

[HttpDelete("{id:int}")]

public ActionResult Delete(int id)

{

var ok = \_repo.DeletePlayer(id);

if (!ok) return NotFound();

return NoContent();

}

}

}

**7) Program.cs (.NET 6 minimal hosting)**

using Microsoft.EntityFrameworkCore;

using dotnetapp.Data;

using dotnetapp.Interfaces;

using dotnetapp.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Register DbContext (replace connection string as needed)

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")

?? "User ID=sa;password=examlyMssql@123; server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False";

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(connectionString));

// Register repository (choose PlayerRepository or InMemoryPlayerRepository)

builder.Services.AddScoped<IPlayerRepository, PlayerRepository>();

// For local development replace above with:

// builder.Services.AddSingleton<IPlayerRepository, InMemoryPlayerRepository>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline (dev)

app.UseSwagger();

app.UseSwaggerUI();

app.UseRouting();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Note:** To use the in-memory repository for fast dev, swap AddScoped<IPlayerRepository, PlayerRepository>() with AddSingleton<IPlayerRepository, InMemoryPlayerRepository>().

**8) Tests — NUnit + Moq**

**tests/MyTestNUnitProject/MyTestNUnitProject.csproj**

Make sure this project references the API project. Example .csproj dependencies:

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>net6.0</TargetFramework>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="NUnit" Version="3.13.3" />

<PackageReference Include="Moq" Version="4.18.4" />

<PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.6.3" />

<PackageReference Include="NUnit3TestAdapter" Version="4.3.1" />

<PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="6.0.0" />

</ItemGroup>

<ItemGroup>

<ProjectReference Include="..\..\dotnetapp\dotnetapp.csproj" />

</ItemGroup>

</Project>

**tests/MyTestNUnitProject/PlayerControllerTests.cs**

using NUnit.Framework;

using Moq;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using dotnetapp.Controllers;

using dotnetapp.Interfaces;

using dotnetapp.Models;

using System.Linq;

namespace MyTestNUnitProject

{

public class PlayerControllerTests

{

private Mock<IPlayerRepository> \_repoMock;

private PlayersController \_controller;

[SetUp]

public void Setup()

{

\_repoMock = new Mock<IPlayerRepository>();

\_controller = new PlayersController(\_repoMock.Object);

}

[Test]

public void Get\_ReturnsAllPlayers()

{

var sample = new List<Player>

{

new Player { PlayerId = 1, Name = "Alice" },

new Player { PlayerId = 2, Name = "Bob" }

};

\_repoMock.Setup(r => r.GetPlayers()).Returns(sample);

var result = \_controller.Get();

Assert.IsInstanceOf<OkObjectResult>(result.Result);

var ok = result.Result as OkObjectResult;

var returned = ok.Value as IEnumerable<Player>;

Assert.AreEqual(2, returned.Count());

}

[Test]

public void Get\_ById\_ReturnsNotFound\_WhenMissing()

{

\_repoMock.Setup(r => r.GetPlayer(5)).Returns((Player)null);

var res = \_controller.Get(5);

Assert.IsInstanceOf<NotFoundResult>(res.Result);

}

[Test]

public void Post\_CreatesPlayer\_ReturnsCreated()

{

var player = new Player { Name = "New", Age = 20 };

var created = new Player { PlayerId = 10, Name = "New", Age = 20 };

\_repoMock.Setup(r => r.CreatePlayer(player)).Returns(created);

var res = \_controller.Post(player);

Assert.IsInstanceOf<CreatedAtActionResult>(res.Result);

var createdRes = res.Result as CreatedAtActionResult;

Assert.AreEqual(10, ((Player)createdRes.Value).PlayerId);

}

[Test]

public void Put\_ReturnsBadRequest\_WhenIdMismatch()

{

var player = new Player { PlayerId = 2, Name = "X" };

var res = \_controller.Put(1, player);

Assert.IsInstanceOf<BadRequestResult>(res.Result);

}

[Test]

public void Put\_ReturnsNotFound\_WhenPlayerMissing()

{

var player = new Player { PlayerId = 5, Name = "X" };

\_repoMock.Setup(r => r.UpdatePlayer(player)).Returns((Player)null);

var res = \_controller.Put(5, player);

Assert.IsInstanceOf<NotFoundResult>(res.Result);

}

[Test]

public void Delete\_ReturnsNoContent\_WhenDeleted()

{

\_repoMock.Setup(r => r.DeletePlayer(3)).Returns(true);

var res = \_controller.Delete(3);

Assert.IsInstanceOf<NoContentResult>(res);

}

[Test]

public void Delete\_ReturnsNotFound\_WhenMissing()

{

\_repoMock.Setup(r => r.DeletePlayer(4)).Returns(false);

var res = \_controller.Delete(4);

Assert.IsInstanceOf<NotFoundResult>(res);

}

}

}

**tests/MyTestNUnitProject/PlayerRepositoryTests.cs**

Tests repository behaviours using UseInMemoryDatabase.

using NUnit.Framework;

using Microsoft.EntityFrameworkCore;

using dotnetapp.Data;

using dotnetapp.Repositories;

using dotnetapp.Models;

using System.Linq;

namespace MyTestNUnitProject

{

public class PlayerRepositoryTests

{

private AppDbContext \_db;

private PlayerRepository \_repo;

[SetUp]

public void Setup()

{

var options = new DbContextOptionsBuilder<AppDbContext>()

.UseInMemoryDatabase(databaseName: $"testdb\_{System.Guid.NewGuid()}")

.Options;

\_db = new AppDbContext(options);

\_repo = new PlayerRepository(\_db);

}

[Test]

public void CreatePlayer\_AddsPlayer()

{

var p = new Player { Name = "RepoTest", Age = 21 };

var created = \_repo.CreatePlayer(p);

Assert.IsTrue(created.PlayerId > 0);

Assert.AreEqual(1, \_db.Players.Count());

}

[Test]

public void UpdatePlayer\_ReturnsNull\_WhenNotFound()

{

var p = new Player { PlayerId = 999, Name = "X" };

var res = \_repo.UpdatePlayer(p);

Assert.IsNull(res);

}

[Test]

public void DeletePlayer\_RemovesPlayer()

{

var p = new Player { Name = "ToDelete" };

var created = \_repo.CreatePlayer(p);

var ok = \_repo.DeletePlayer(created.PlayerId);

Assert.IsTrue(ok);

Assert.AreEqual(0, \_db.Players.Count());

}

}

}

**9) Commands (create solution, add test project, link references)**

Run these steps from your workspace root:

# create main project

dotnet new webapi -n dotnetapp

cd dotnetapp

dotnet build

# go back, create solution and test project

cd ..

dotnet new sln -n DotnetAppSolution

dotnet new nunit -n MyTestNUnitProject

# Add projects to the solution

dotnet sln DotnetAppSolution.sln add dotnetapp/dotnetapp.csproj

dotnet sln DotnetAppSolution.sln add MyTestNUnitProject/MyTestNUnitProject.csproj

# add project reference in test project

dotnet add MyTestNUnitProject/MyTestNUnitProject.csproj reference dotnetapp/dotnetapp.csproj

# add packages to main project and test project (if not already)

cd dotnetapp

dotnet add package Microsoft.EntityFrameworkCore --version 6.0.0

dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 6.0.0

dotnet add package Microsoft.EntityFrameworkCore.Tools --version 6.0.0

dotnet add package Moq --version 4.18.4

cd ../MyTestNUnitProject

dotnet add package Moq --version 4.18.4

dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 6.0.0

dotnet restore

# run the API

cd ../dotnetapp

dotnet run

# run tests (from solution folder)

cd ..

dotnet test

**10) Port 8080 configuration**

Edit Properties/launchSettings.json in the dotnetapp project. Under profiles -> your profile -> set applicationUrl to "http://0.0.0.0:8080" as you described. Or run via dotnet run --urls "http://0.0.0.0:8080".

**Notes & Tips**

* I used RESTful conventions and appropriate status codes (200, 201, 204, 400, 404).
* For production use, validate model fields (e.g., [Required], [Range]) and add DTOs + AutoMapper if needed.
* Use PlayerRepository with SQL Server for persistent storage (remember to run EF migrations: dotnet ef migrations add InitialSetup and dotnet ef database update).
* For fast iteration or CI tests you can use InMemoryPlayerRepository.
* Tests separate concerns:
  + Controller tests mock the repository (Moq) to ensure controller logic is correct.
  + Repository tests use EF Core UseInMemoryDatabase to test DB interactions.

If you want, I can:

* Paste the full dotnetapp.csproj and launchSettings.json contents,
* Create ready-to-run migration commands and sample appsettings.Development.json with connection string,
* Or generate a sample set of seed players and a Postman collection for the API.

Which of those would you like next?

Top of Form

Bottom of Form