**AI Prompt:**

I am a .Net Developer with over 15 years of experience in web application development. I am currently developing an HR Saas application using Blazor Web Apps using C# programming language and Entity Framework Core for backend coding. Write a sample Blazor Web App using clean architecture approach and explain the code.

**ChatGPT said:**

Updated saved memory

Great! Since you're experienced in .NET development and working on a Blazor Web App using Clean Architecture with Entity Framework Core, I’ll provide you with a **minimal working sample** structure that reflects the Clean Architecture pattern, and walk you through each layer.

**✅ What We’ll Build (Conceptual Sample)**

A basic **Employee Management** feature in an HR SaaS Blazor Web App using:

* **Blazor Web App (Server-side for now)**
* **EF Core with SQLite or InMemory for demo**
* **Clean Architecture** layers:
  + Domain (Entities, Interfaces)
  + Application (Use Cases, DTOs, Interfaces)
  + Infrastructure (EF Core DB Context, Repositories)
  + Presentation (Blazor UI)
  + Web (Composition Root / DI + Blazor entry point)

**🧱 Project Structure**

arduino

CopyEdit

HRApp/

├── HRApp.Domain/

│ └── Entities/

│ └── Employee.cs

├── HRApp.Application/

│ └── Interfaces/

│ └── DTOs/

│ └── Services/

│ └── EmployeeService.cs

├── HRApp.Infrastructure/

│ └── Data/

│ └── AppDbContext.cs

│ └── Repositories/

│ └── EmployeeRepository.cs

├── HRApp.Web/ (Blazor Web App)

│ └── Pages/

│ └── Employees.razor

│ └── Program.cs

│ └── DI setup

**🔹 1. Domain Layer (HRApp.Domain)**

**Entities/Employee.cs**

csharp

CopyEdit

namespace HRApp.Domain.Entities;

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; } = string.Empty;

public string Position { get; set; } = string.Empty;

public DateTime HireDate { get; set; }

}

**🔹 2. Application Layer (HRApp.Application)**

**DTOs/EmployeeDto.cs**

csharp

CopyEdit

namespace HRApp.Application.DTOs;

public class EmployeeDto

{

public int Id { get; set; }

public string FullName { get; set; } = string.Empty;

public string Position { get; set; } = string.Empty;

}

**Interfaces/IEmployeeService.cs**

csharp

CopyEdit

using HRApp.Application.DTOs;

namespace HRApp.Application.Interfaces;

public interface IEmployeeService

{

Task<List<EmployeeDto>> GetAllAsync();

Task AddAsync(EmployeeDto employee);

}

**Services/EmployeeService.cs**

csharp

CopyEdit

using HRApp.Application.DTOs;

using HRApp.Application.Interfaces;

using HRApp.Domain.Entities;

using HRApp.Infrastructure.Repositories;

namespace HRApp.Application.Services;

public class EmployeeService : IEmployeeService

{

private readonly IEmployeeRepository \_repository;

public EmployeeService(IEmployeeRepository repository)

{

\_repository = repository;

}

public async Task<List<EmployeeDto>> GetAllAsync()

{

var employees = await \_repository.GetAllAsync();

return employees.Select(e => new EmployeeDto

{

Id = e.Id,

FullName = e.FullName,

Position = e.Position

}).ToList();

}

public async Task AddAsync(EmployeeDto dto)

{

var employee = new Employee

{

FullName = dto.FullName,

Position = dto.Position,

HireDate = DateTime.Now

};

await \_repository.AddAsync(employee);

}

}

**🔹 3. Infrastructure Layer (HRApp.Infrastructure)**

**Data/AppDbContext.cs**

csharp

CopyEdit

using Microsoft.EntityFrameworkCore;

using HRApp.Domain.Entities;

namespace HRApp.Infrastructure.Data;

public class AppDbContext : DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options)

: base(options) { }

public DbSet<Employee> Employees => Set<Employee>();

}

**Repositories/IEmployeeRepository.cs**

csharp

CopyEdit

using HRApp.Domain.Entities;

namespace HRApp.Infrastructure.Repositories;

public interface IEmployeeRepository

{

Task<List<Employee>> GetAllAsync();

Task AddAsync(Employee employee);

}

**Repositories/EmployeeRepository.cs**

csharp

CopyEdit

using HRApp.Domain.Entities;

using HRApp.Infrastructure.Data;

using Microsoft.EntityFrameworkCore;

namespace HRApp.Infrastructure.Repositories;

public class EmployeeRepository : IEmployeeRepository

{

private readonly AppDbContext \_context;

public EmployeeRepository(AppDbContext context)

{

\_context = context;

}

public async Task<List<Employee>> GetAllAsync()

{

return await \_context.Employees.ToListAsync();

}

public async Task AddAsync(Employee employee)

{

\_context.Employees.Add(employee);

await \_context.SaveChangesAsync();

}

}

**🔹 4. Presentation (Blazor UI - HRApp.Web)**

**Pages/Employees.razor**

razor

CopyEdit

@page "/employees"

@inject HRApp.Application.Interfaces.IEmployeeService EmployeeService

<h3>Employees</h3>

<input @bind="newEmployeeName" placeholder="Name" />

<input @bind="newEmployeePosition" placeholder="Position" />

<button @onclick="AddEmployee">Add</button>

<ul>

@foreach (var emp in employees)

{

<li>@emp.FullName - @emp.Position</li>

}

</ul>

@code {

private List<HRApp.Application.DTOs.EmployeeDto> employees = new();

private string newEmployeeName = "";

private string newEmployeePosition = "";

protected override async Task OnInitializedAsync()

{

employees = await EmployeeService.GetAllAsync();

}

private async Task AddEmployee()

{

await EmployeeService.AddAsync(new()

{

FullName = newEmployeeName,

Position = newEmployeePosition

});

employees = await EmployeeService.GetAllAsync();

newEmployeeName = newEmployeePosition = string.Empty;

}

}

**🔹 5. Program.cs (DI Setup)**

csharp

CopyEdit

using HRApp.Application.Interfaces;

using HRApp.Application.Services;

using HRApp.Infrastructure.Data;

using HRApp.Infrastructure.Repositories;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// EF Core with InMemory for simplicity

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

options.UseInMemoryDatabase("HRAppDb"));

// Dependency Injection

builder.Services.AddScoped<IEmployeeRepository, EmployeeRepository>();

builder.Services.AddScoped<IEmployeeService, EmployeeService>();

builder.Services.AddRazorComponents();

builder.Services.AddServerSideBlazor();

var app = builder.Build();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

**🧠 Summary of Clean Architecture Roles**

| **Layer** | **Responsibility** |
| --- | --- |
| **Domain** | Core entities, no dependencies |
| **Application** | Business logic, interfaces, and DTOs |
| **Infrastructure** | Implementation details (e.g., EF Core, DB access) |
| **Web/UI** | UI rendering with Blazor, user interaction |

Would you like this as a downloadable solution template or would you like to expand it to include features like Edit, Delete, Pagination, or Authentication?

**You said:**

Revise the above code to connect Entity Framework Core to SQL Server database

**ChatGPT said:**

To revise the sample Blazor Web App code to use **SQL Server** instead of **InMemory**, we just need to update a few key areas related to **Entity Framework Core** configuration:

**✅ Changes Required:**

1. **Update EF Core package** (if not already)
2. **Update Program.cs to use SQL Server**
3. **Ensure a valid connection string**
4. **Apply EF Core migrations**

**🔧 STEP-BY-STEP REVISIONS**

**📦 1. Install Required NuGet Package**

In the HRApp.Infrastructure project:

bash

CopyEdit

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

If using **Package Manager Console in Visual Studio**:

powershell

CopyEdit

Install-Package Microsoft.EntityFrameworkCore.SqlServer

**🛠️ 2. Update Program.cs in HRApp.Web**

csharp

CopyEdit

using HRApp.Application.Interfaces;

using HRApp.Application.Services;

using HRApp.Infrastructure.Data;

using HRApp.Infrastructure.Repositories;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// SQL Server connection string from appsettings.json

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

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<IEmployeeRepository, EmployeeRepository>();

builder.Services.AddScoped<IEmployeeService, EmployeeService>();

builder.Services.AddRazorComponents();

builder.Services.AddServerSideBlazor();

var app = builder.Build();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

app.Run();

**⚙️ 3. Add appsettings.json Configuration**

In the HRApp.Web project, edit or add appsettings.json:

json

CopyEdit

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=HRAppDb;Trusted\_Connection=True;TrustServerCertificate=True;"

}

}

✅ Replace localhost with your actual SQL Server instance name if needed.  
You can also use SQL Authentication like:  
"Server=.;Database=HRAppDb;User Id=sa;Password=YourPassword;"

**🧱 4. Enable Migrations**

In the terminal or Visual Studio Package Manager Console, navigate to the **HRApp.Infrastructure** project (or the project where AppDbContext is defined), and run:

bash

CopyEdit

dotnet ef migrations add InitialCreate --startup-project ../HRApp.Web

dotnet ef database update --startup-project ../HRApp.Web

Or from Package Manager Console:

powershell

CopyEdit

PM> Add-Migration InitialCreate -StartupProject HRApp.Web

PM> Update-Database -StartupProject HRApp.Web

⚠️ Ensure that HRApp.Infrastructure is the **default project** and HRApp.Web is the **startup project**.

**✅ That’s it! Now your app is connected to SQL Server!**

When you run the Blazor Web App:

* It will connect to SQL Server
* Create the HRAppDb database (if not exists)
* Run migrations and create the Employees table

Would you like a script to seed the database with sample data, or help with adding EF Core logging/debugging for development?