

### Scenario:

The problem is to create an online interrogation panel application using the Model-View-Controller (MVC) architecture that allows interrogators to create problems with four choices and solution through a form. When the program is launched, the predefined problems should need to load immediately into the database.

### Exercise Steps:




1. Create a new ASP.NET Core MVC project in Visual Studio.
2. Add a new folder called "**Models**" in the project and create a class called "**Problem.cs**" with the following properties and include the data annotations.

```
using System.ComponentModel.DataAnnotations;

namespace ASPNET6_Template.Models
{
    8 references
    public class Problem
    {
        [Key]
        0 references
        public int Id { get; set; }
        [Required(ErrorMessage = "Please provide Problem")]
        5 references
        public string InterrogatorsProblem { get; set; }
        [Required(ErrorMessage = "Please provide Choice1")]
        5 references
        public string Choice1 { get; set; }
        [Required(ErrorMessage = "Please provide Choice2")]
        5 references
        public string Choice2 { get; set; }
        [Required(ErrorMessage = "Please provide Choice3")]
        5 references
        public string Choice3 { get; set; }
        [Required(ErrorMessage = "Please provide Choice4")]
        5 references
        public string Choice4 { get; set; }
        [Required(ErrorMessage = "Please provide Solution")]
        5 references
        public string Solution { get; set; }
    }
}
```

3. Add a new folder called "**Data**" in the project and create a class called "**InterrogationDbContext.cs**" which inherits DbContext class.

4. Right click on the Solution Explorer and Open the Manage NuGet Packages.. and install the below-mentioned mandatory Nuget packages.

	<b>Microsoft.EntityFrameworkCore</b> by Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	6.0.14 7.0.3
	<b>Microsoft.EntityFrameworkCore.SqlServer</b> by Microsoft Microsoft SQL Server database provider for Entity Framework Core.	6.0.14 7.0.3
	<b>Microsoft.EntityFrameworkCore.Tools</b> by Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	6.0.14 7.0.3

5. Let us create DbSet property in the **InterrogationDbContext** class.

```
using ASPNET6_Template.Models;
using Microsoft.EntityFrameworkCore;

namespace ASPNET6_Template.Data
{
    7 references
    public class InterrogationDbContext : DbContext
    {
        0 references
        public InterrogationDbContext(DbContextOptions<InterrogationDbContext> options) : base(options)
        {
        }

        2 references
        public DbSet<Problem> Problems { set; get; }
    }
}
```

6. Let us create a interface named **IDbInitializer.cs** under the Data folder and add a method in it.

```
namespace ASPNET6_Template.Data
{
    3 references
    public interface IDbInitializer
    {
        2 references
        void Initialize();
    }
}
```

7. Open the appsettings.json file and add the below-mentioned connection string to connect the database.

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DBConnection": "server=(LocalDB)\\MSSQLLocalDB;database=DataDB;Trusted_Connection=true;"
  }
}

```

8. In class **Program.cs** initialize the sql server and mention the database connection, which exists in the appsettings.json file. Also include the Dependency injection through **AddScoped** request.

```

using ASPNET6_Template.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<InterrogationDbContext>(
    options => options.UseSqlServer(
        (builder.Configuration.GetConnectionString("DBConnection")));
builder.Services.AddScoped<IDbInitializer, DbInitializer>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days.
    // You may want to change this for production scenarios,
    // see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

```

```

app.UseAuthorization();

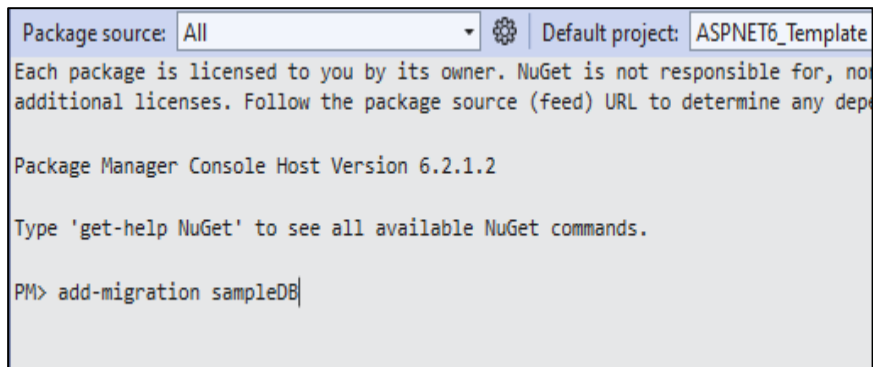
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

void SeedDatabase()
{
    using (var scope = app.Services.CreateScope())
    {
        var dbinitializer = scope.ServiceProvider.GetRequiredService<IDbInitializer>();
        dbinitializer.Initialize();
    }
}

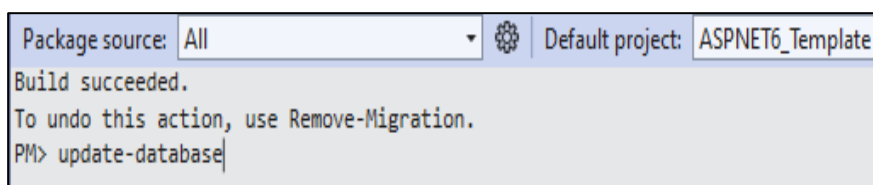
```

9. Open Package manager console to create migration in order to create database.



The screenshot shows the Package Manager Console interface. At the top, there are two dropdown menus: 'Package source' set to 'All' and 'Default project' set to 'ASPNET6\_Template'. Below these, a disclaimer states: 'Each package is licensed to you by its owner. NuGet is not responsible for, nor does it endorse, any additional licenses. Follow the package source (feed) URL to determine any dependencies.' The console version is 'Package Manager Console Host Version 6.2.1.2'. A prompt says 'Type \'get-help NuGet\' to see all available NuGet commands.' The command prompt shows 'PM> add-migration sampleDB'.

When the build is succeeded then update database by this below comment,



The screenshot shows the Package Manager Console interface. At the top, there are two dropdown menus: 'Package source' set to 'All' and 'Default project' set to 'ASPNET6\_Template'. Below these, a message says 'Build succeeded. To undo this action, use Remove-Migration.' The command prompt shows 'PM> update-database'.

You have successfully created the database. There is a new folder named Migrations available in your project when the migration is successful.

10. Now create a class named **DbInitializer.cs** under the **Data** folder, that should inherit the **IDbInitializer.cs** and add these below given methods in it. Also use dependency injection of the class **InterrogationDbContext** in the **DbInitializer**.

11. Include these two records in the database in **DbInitializer.cs**.
12. Whenever you run the application, the predefined data should need to load immediately into the database.

```
using ASPNET6_Template.Models;
using Microsoft.EntityFrameworkCore;

namespace ASPNET6_Template.Data
{
    2 references
    public class DbInitializer : IDbInitializer
    {
        private readonly InterrogationDbContext _context;

        0 references
        public DbInitializer(InterrogationDbContext context)
        {
            _context = context;
        }

        2 references
        public void Initialize()
        {
            try
            {
                if (_context.Database.GetPendingMigrations().Count() > 0)
                {
                    _context.Database.Migrate();
                }
            }
            catch (Exception ex) { }

            if (!_context.Problems.Any())
            {
                var questions = new List<Problem>()
                {
```

```
                    new Problem()
                    {
                        InterrogatorsProblem = "The first metal used by the man was",
                        Choice1 = "Iron",
                        Choice2 = "Copper",
                        Choice3 = "Aluminium",
                        Choice4 = "Gold",
                        Solution = "Copper"
                    },
                    new Problem()
                    {
                        InterrogatorsProblem = "Which of the following is a balanced fertiliser for plants ?",
                        Choice1 = "Urea",
                        Choice2 = "Ammonia sulphate",
                        Choice3 = "Nitrates",
                        Choice4 = "Compost",
                        Solution = "Compost"
                    }
                };

            _context.Problems.AddRange(questions);
            _context.SaveChanges();
        }
    }
}
```

**To summarize,**

This exercise demonstrates how to use Entity Framework, Data seed, Migrations, Dependency injection. We also learnt about model binding, setting those values to the properties we declared. We also declared Data annotations with custom error messages