ASP.NET CORE MVC Web Application Development Guide

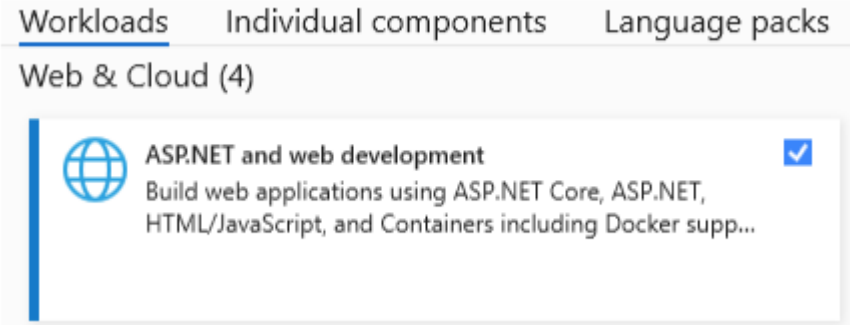
I) Environment and Tools

A) Visual Studio 2022 Community:

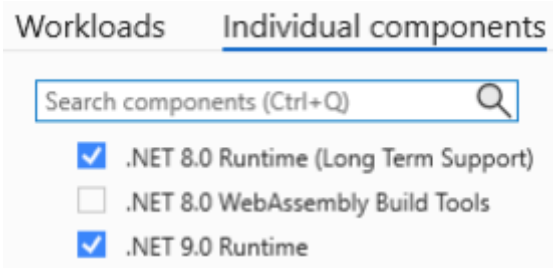
1. Download Visual Studio 2022 Community from: https://need4code.com/DotNet/Home/Index?path=.NET%5C00_Files%5CVisual%20Studio%20Community

2. After running the Visual Studio Installer:

   a) Select only **ASP.NET Web Development** under **Workloads**.



   b) Do not uncheck any component, only select **.NET 8 Runtime (Long Term Support)** under **Individual components**. **.NET 9 Runtime** should have already been selected.



   c) English is recommended for **Language packs**.

   d) **Visual Studio IDE** installation is recommended on your SSD drive if you have any. **Download cache** and **Shared components, tools and SDKs** can either be installed on your SSD drive or on your other hard drive if you have any.

B) Rider for MAC:

https://www.jetbrains.com/rider

C) SQLite Database:

https://www.sqlite.org

II) Solution Setup

1. Create the ASP.NET Core Web App (Model-View-Controller) project.

2. Give the Project name MVC. You may change the solution folder in Location. Give the Solution name your project name. Place solution and project in the same directory sould not be checked.

3. Select ".NET 8.0" as the "Framework", choose "None" for "Authentication type", check "Configure for HTTPS", do not check "Enable container support", do not check "Do not use top-levet statements" and do not check "Enlist in .NET Aspire orchestration".

Note: When you run your application in Rider on Mac and if you get the exception below:

Failed to read NuGet.Config due to unauthorized access. Path: '/Users/YourUserFolder/.nuget/NuGet/NuGet.Config'

you should run the following commands in the terminal of Rider:

sudo chown $USER ~/.nuget/NuGet/NuGet.Config

chmod 644 ~/.nuget/NuGet/NuGet.Config

III) Projects Setup for CRUD Operations

Note: A solution is a set of one or more projects.

1. Right-click the solution in Solution Explorer, then Add -> New Project to create a project named CORE as a Class Library with .NET 8.0. You can delete the Class1.cs file after.

2. Right-click the solution in Solution Explorer, then Add -> New Project to create a project named APP as a Class Library with .NET 8.0. You can delete the Class1.cs file after.

3. Set Nullable to Disable for CORE and APP Projects (via project properties or XML).

4. Right-click APP Project then click Add -> Project Reference then select the CORE Project to use the classes of the CORE Project in the APP Project.

5. Right-click MVC Project then click Add -> Project Reference then select the APP Project to use both the classes of the CORE Project and APP Project in the MVC Project.

6. If MVC project name doesn't appear bold in Solution Explorer, right-click MVC Project then click Set as Startup Project.

7. Right-click CORE Project then click Manage NuGet Packages then in Browse tab search for Microsoft.EntityFrameworkCore latest version starting with 8 and install.

8. To use SQLite Database, right-click APP Project then click Manage NuGet Packages then in Browse tab search for System.Data.SQLite.Core latest version and install then search for Microsoft.EntityFrameworkCore.Sqlite latest version starting with 8 and install.

9. For using Code-First Approach (create entity classes and DbContext class first, database later), right-click MVC Project then click Manage NuGet Packages then in Browse tab search for Microsoft.EntityFrameworkCore.Tools latest version starting with 8 and install.

A) CORE Project for CRUD Operations

1. Right-click CORE Project then click Add -> New Folder to create a folder named Domain.

2. Right-click Domain folder then click Add -> Class to create a class file named Record.cs as below: Don't forget to change the **internal** to **public** class visibility modifier for all classes and interfaces!

```
namespace CORE.Domain
{
    // Base abstract class for all entity, request and response classes.
    public abstract class Record
    {
        // Property for primary key of entities, unique identifier of
        // requests and responses.
        public int Id { get; set; }

        // Constructor with id parameter to set to Id property value
        // from a derived class constructor with id parameter.
        protected Record(int id)
        {
            Id = id;
        }
```

```
        // Default constructor to set the Id property default value 0.
        protected Record()
        {
        }
    }
}
```

3. Right-click CORE Project then click Add -> New Folder to create a folder named Models for base Data Transfer Object (DTO) classes.

4. Right-Click Models folder then create a class file called CommandResponse.cs as below:

```
using CORE.Domain;

namespace CORE.Models
{
    // Concrete class for returning create, update and delete
    // operation results.
    public class CommandResponse : Record
    {
        // Readonly property to return the success status as true or false.
        // Since no setter, the property value can only be set in the below
        // line or in the constructor.
        public bool IsSuccessful { get; }

        // Readonly property to return additional information such as a
        // success message or an error message containing details.
        public string Message { get; }

        // Constructor with parameters to set the IsSuccessful and Message
        // property values with id default parameter value passed to the
        // base abstract Record class constructor with id parameter.
        // If no value is passed for the id parameter, the default value 0
        // will be used.
        public CommandResponse(bool isSuccessful, string message, int id = 0)
            : base(id)
        {
            IsSuccessful = isSuccessful;
            Message = message;
        }
    }
}
```

5. Right-click CORE Project then click Add -> New Folder to create a folder called Services for base business logic classes.

6. Right-Click Services folder then create a class file called Service.cs as below:

```
using CORE.Models;
using System.Globalization;

namespace CORE.Services
```

```
{
    // Abstract base class for culture configuration and returning
    // successful or failure operation result CommandResponse objects.
    public abstract class Service
    {
        // Field to store and retrieve the backing culture value
        // using Encapsulation with the Culture property.
        private string _culture;

        // Property to retrieve the culture value and set the culture value
        // with assigning the service's current thread's culture information
        // to "en-US" for United States English or "tr-TR" for Turkish to
        // ensure consistent formatting and localization.
        protected string Culture
        {
            get
            {
                return _culture;
            }
            set
            {
                _culture = value;
                Thread.CurrentThread.CurrentCulture = new CultureInfo(_culture);
                Thread.CurrentThread.CurrentUICulture =
                    new CultureInfo(_culture);
            }
        }

        // Default constructor to set the default culture to "en-US" for
        // United States English. "tr-TR" can be assigned to the Culture property
        // in the derived class constructor or methods for Turkish culture.
        protected Service()
        {
            Culture = "en-US";
        }

        // Behavior (method, function) to return a successful CommandResponse
        // object for the operation with the entity's ID and an optional message
        // default parameter.
        // If no message is provided, a default success message is used
        // by using the Null Coalescing Operator meaning that if message is null,
        // return "Operation successful." message, otherwise return
        // the provided message.
        protected CommandResponse Success(int id, string message = default)
            => new CommandResponse(true, message ?? "Operation successful.", id);

        // Behavior to return a failure CommandResponse object for the operation
        // with an optional message default parameter. If no message is provided,
        // a default error message is used by using the Null Coalescing Operator
        // meaning that if message is null, return "Operation failed!" message,
        // otherwise return the provided message.
        protected CommandResponse Error(string message = default)
            => new CommandResponse(false, message ?? "Operation failed!");
    }
}
```

7. Right-click Services folder then create DbService class file as below:

```
using CORE.Domain;
using Microsoft.EntityFrameworkCore;
```

```
namespace CORE.Services
{
    // Abstract base class to perform CRUD (Create, Read, Update and Delete)
    // database operations for a specific entity type inherited from the
    // Record class and has a parameterless constructor.
    // Inherits from the Service class to manage culture and return
    // successful or failure operation result CommandResponse objects
    // after operations.
    // Implements the IDisposable interface to release unmanaged resources.
    public abstract class DbService<TEntity> : Service, IDisposable
        where TEntity : Record, new()
    {
        // Field for Dependency Injection for the DbContext instance to
        // perform database operations in the below methods.
        private readonly DbContext _db;

        // Constructor with an injected DbContext instance parameter
        // for performing database operations in the below methods by
        // using the injected instance assigned _db field.
        protected DbService(DbContext db)
        {
            _db = db;
        }

        // Method to return the entity query such as "select * from table"
        // where table is the related entity DbSet.
        // Defined as virtual to allow derived classes to override the
        // query if needed such as for ordering, filtering, including
        // the related entities to the query, etc.
        // AsNoTracking is used for not tracking the returned entities to
        // improve performance. If not written, the default
        // behavior is tracking.
        protected virtual IQueryable<TEntity> DbQuery()
            => _db.Set<TEntity>().AsNoTracking();

        // Method to return a single tracked (AsTracking) entity for update
        // or delete operations by its ID using the base or derived overridden
        // DbQuery method. If the entity is not found, returns null since
        // OrDefault version of the Single method is used.
        protected TEntity DbSingle(int id)
        {
            return DbQuery().AsTracking()
                .SingleOrDefault(entity => entity.Id == id);
        }

       // Method to return a single tracked (AsTracking) entity
       // according to a lambda expression filter parameter (predicate)
       // which may contain one or more conditions using the base or
       // derived overridden DbQuery method. If the entity is not found,
       // returns null since OrDefault version of the Single method is used.
       protected TEntity DbSingle(Expression<Func<TEntity, bool>> predicate)
            => DbQuery().AsTracking().SingleOrDefault(predicate);

        // Method to persist changes to the database and return the number
        // of effected rows of the table.
        // Defined as virtual to allow derived classes to override the behavior
        // if needed such as for extra logging, error handling, etc.
        protected virtual int DbSave() => _db.SaveChanges();

        // Method to insert a new entity to the entity DbSet and persist changes
        // to the database if the save default parameter value is true.
        // If the save parameter value is false, the changes will not be
        // persisted to the database, which can be used for multiple insert
```

```
        // operations. Then DbSave method can be invoked to persist all changes
        // to the database once (Unit of Work) to increase performance.
        protected void DbAdd(TEntity entity, bool save = true)
        {
            _db.Set<TEntity>().Add(entity);
            if (save)
                DbSave();
        }


        // Method to update an existing entity retrieved by DbSingle method
        // and persist changes to the database if the save default parameter
        // value is true. If the save parameter value is false, the changes will
        // not be persisted to the database, which can be used for multiple
        // update operations. Then DbSave method can be invoked to persist all
        // changes to the database once (Unit of Work) to increase performance.
        protected void DbUpdate(TEntity entity, bool save = true)
        {
            _db.Set<TEntity>().Update(entity);
            if (save)
                DbSave();
        }

        // Method to delete an existing entity retrieved by DbSingle method
        // and persist changes to the database if the save default parameter
        // value is true. If the save parameter value is false, the changes will
        // not be persisted to the database, which can be used for multiple
        // delete operations. Then DbSave method can be invoked to persist all
        // changes to the database once (Unit of Work) to increase performance.
        protected void DbRemove(TEntity entity, bool save = true)
        {
            _db.Set<TEntity>().Remove(entity);
            if (save)
                DbSave();
        }

        // Method to delete the related navigation entities of an entity.
        // The changes will not be persisted to the database, therefore DbAdd,
        // DbUpdate, DbRemove or DbSave methods should be invoked after
        // to persist all changes to the database.
        protected void DbRemove<TNavigationEntity>(List<TNavigationEntity>
            navigationEntities) where TNavigationEntity : Record, new()
        {
            _db.Set<TNavigationEntity>().RemoveRange(navigationEntities);
        }

        // Method to dispose the DbContext instance and suppress finalization
        // (don't call the finalizer) of the instance of this class to
        // improve performance and avoid memory leaks.
        public void Dispose()
        {
            _db.Dispose();
            GC.SuppressFinalize(this);
        }
    }
}
```

8. Right-click Services folder then create IService interface file as below:

```
using CORE.Domain;
using CORE.Models;


namespace CORE.Services
{
    // Interface to perform CRUD (Create, Read, Update, Delete)
    // operations for specific request and response types
    // inherited from the Record class and have a parameterless
    // constructor. These methods will be used within the controller
    // actions after the implemented service class injection and
    // will implement business logic such as querying and projecting
    // (mapping) entity data to response data, validation in database
    // table, mapping request data to entity data then inserting and
    // updating the entity data in the database table, deleting entity
    // data by ID from the database table, etc.
    public interface IService<TRequest, TResponse>
        where TRequest : Record, new()
        where TResponse : Record, new()
    {
        // Returns a projected query of type TResponse from an entity.
        // public may not be written.
        public IQueryable<TResponse> Query();

        // Returns a list of TResponse from the Query method.
        public List<TResponse> List() => Query().ToList();

        // Returns an item of TResponse by ID from the Query method.
        public TResponse Single(int id)
            => Query().SingleOrDefault(response => response.Id == id);

        // Returns an item of TRequest by ID.
        public TRequest Edit(int id);

        // Maps the request to the entity and inserts it to the database table.
        public CommandResponse Add(TRequest request);

        // Maps the request to the entity and updates it in the database table.
        public CommandResponse Update(TRequest request);

        // Deletes the entity from the database table by ID.
        public CommandResponse Remove(int id);
    }
}
```

B) APP Project for CRUD Operations

1. Right-click APP Project then click Add -> New Folder to create a folder called Domain for entity classes and DbContext class (Entity Framework base database operations and configurations class).

2. Right-click Domain folder then click Add -> Class to create a class file called Role.cs as an entity for the Roles database table as below:

```
using CORE.Domain;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace APP.Domain
{
    // Role is a Record.
    public class Role : Record
    {
        // Reference Type: string, array, class and interface
        // are reference types (can be null).
        // Required and StringLength are called attributes
        // (or data annotations when used in an entity or a request).
        // Name can't be null and must contain maximum 25 characters.
        [Required, StringLength(25)]
        public string Name { get; set; } // = "Admin";
                                         // Initial assignments may also
                                         // be done to the properties.
                                         // No need to assign "Admin" here.

        // Reference Type and Navigation Property:
        // Navigation properties are used to navigate from one entity
        // to another related entity to get or set related data.
        // Will be an empty list of UserRole if no assignment
        // or no include to a query.
        // For roles-users many to many relationship.
        public List<UserRole> UserRoles { get; set; }
            = new(); // = new List<UserRole>(); can also be written

        // This property helps to easily manage the UserRoles relational entities
        // by Role entity's User Id values.
        // NotMapped attribute means no column in the Roles table will be created
        // for this property.
        //[NotMapped]
        //public List<int> UserIds
        //{
        //    // Returns the User Id values of the Role entity.
        //    get => UserRoles.Select(userRoleEntity
        //        => userRoleEntity.UserId).ToList();
        //
        //    // Sets the UserRoles relational entities of the Role entity
        //    // by the assigned User Id values.
        //    set => UserRoles = value.Select(userIdValue => new UserRole
        //    {
        //        UserId = userIdValue
        //    }).ToList();
        //}
        // Since we won't update the relational user data (UserRoles)
        // through Role entity, we don't need the UserIds property here.
    }
}
```

3. Right-click Domain folder then click Add -> Class to create a class file called UserRole.cs as an entity for the UserRoles many to many relation database table as below:

```
using CORE.Domain;


namespace APP.Domain
{
    // UserRole is a Record.
    // For users-roles many to many relationship.
    public class UserRole : Record
    {
        // Value Type: Types other than string, array, class and interface
        // are value types (can't be null).
        // UserId can't be null and will have the default value 0
        // if no value is assigned.
        public int UserId { get; set; } // foreign key to the User entity

        // Reference Type and Navigation Property: string, array, class
        // and interface are reference types (can be null).
        // Navigation properties are used to navigate from one entity
        // to another related entity to get or set related data.
        // Will be null if no assignment or no include to a query.
        public User User { get; set; }

        // Value Type:
        // RoleId can't be null and will have the default value 0
        // if no value is assigned.
        public int RoleId { get; set; } // foreign key to the Role entity

        // Reference Type and Navigation Property:
        // Will be null if no assignment or no include to a query.
        public Role Role { get; set; }
    }
}
```

4. Right-click Domain folder then click Add -> Class to create a class file called User.cs as an entity for the Users database table as below:

```
using CORE.Domain;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace APP.Domain
{
    // User is a Record.
    public class User : Record
    {
        // Reference Type: string, array, class and interface
        // are reference types (can be null).
        // UserName can't be null and must contain maximum 30 characters.
        [Required, StringLength(30)]
        public string UserName { get; set; }

        // Reference Type:
        // Password can't be null and must contain maximum 15 characters.
```

```
[Required, StringLength(15)]
public string Password { get; set; }

// Reference Type:
// FirstName can be null and must contain maximum 50 characters.
// Will be null if no assignment.
[StringLength(50)]
public string FirstName { get; set; }


// Reference Type:
// LastName can be null and must contain maximum 50 characters.
// Will be null if no assignment.
[StringLength(50)]
public string LastName { get; set; }

// Value Type: Types other than string, array, class and interface
// are value types (can't be null).
// Gender can't be null and will have the first value of the Genders
// enum if no value is assigned.
public Genders Gender { get; set; }

// Value Type: ? after the type is used to make the value type act
// like reference type.
// BirthDate can be null since ? is used after the type and
// will be null if no value is assigned.
public DateTime? BirthDate { get; set; }

// Value Type:
// RegistrationDate can't be null and will have the default value
// 01/01/0001 00:00:00 (DateTime.MinValue) if no value is assigned.
public DateTime RegistrationDate { get; set; }

// Value Type:
// Score can't be null and will have the default value 0
// if no value is assigned.
public double Score { get; set; }

// Value Type:
// IsOnline can't be null and will have the default value false
// if no value is assigned.
public bool IsOnline { get; set; }

// Value Type:
// StatusId can't be null and will have the default value 0
// if no value is assigned.
// For users-status one to many relationship.
public int StatusId { get; set; } // foreign key to the Status entity

// Reference Type and Navigation Property:
// Navigation properties are used to navigate from one entity
// to another related entity to get or set related data.
// Will be null if no assignment or no include to a query.
// For users-status one to many relationship.
public Status Status { get; set; }

// Reference Type and Navigation Property:
// Will be an empty list of UserRole if no assignment
// or no include to a query.
// For users-roles many to many relationship.
public List<UserRole> UserRoles { get; set; }
    = new(); // = new List<UserRole>(); can also be written
```

```
        // This property helps to easily manage the UserRoles relational entities
        // by User entity's Role Id values.
        // NotMapped attribute means no column in the Users table will be created
        // for this property.
        [NotMapped]
        public List<int> RoleIds
        {
            // Returns the Role Id values of the User entity.
            get => UserRoles.Select(userRoleEntity
                => userRoleEntity.RoleId).ToList();

            // Sets the UserRoles relational entities of the User entity
            // by the assigned Role Id values.
            set => UserRoles = value.Select(roleIdValue => new UserRole
            {
                RoleId = roleIdValue
            }).ToList();
        }
    }
}
```

5. Right-click Domain folder then click Add -> Class to create a class file called Status.cs as an entity for the Statuses database table as below:

```
using CORE.Domain;
using System.ComponentModel.DataAnnotations;

namespace APP.Domain
{
    // Status is a Record.
    public class Status : Record
    {
        // Reference Type: string, array, class and interface
        // are reference types (can be null).
        // Title can't be null and must contain maximum 5 characters.
        [Required, StringLength(5)]
        public string Title { get; set; }

        // Reference Type and Navigation Property:
        // Navigation properties are used to navigate from one entity
        // to another related entity to get or set related data.
        // Will be an empty list of User if no assignment
        // or no include to a query.
        // For status-users one to many relationship.
        public List<User> Users { get; set; }
            = new(); // = new List<User>(); can also be written
    }
}
```

6. Right-click Domain folder then click Add -> Class to create an enum file named Genders.cs as below:

```
namespace APP.Domain
{
```

```
    // Enums are used for predefined values and help not to remember or
    // check each element's value when being used.
    // Getting the value of an enum element:
    // int womanValue = (int)Genders.Woman;
    // Getting the text of an enum element:
    // string manText = Genders.Man.ToString();
    public enum Genders
    {
        Woman, // if no assignment, will start from 0
        Man // will automatically get the next value 1 and other elements
            // will continue to get the next values after 1
    }
}
```

7. Right-click Domain folder then click Add -> Class to create a class file called Db for Entity Framework database configurations and operations as below:

```
using Microsoft.EntityFrameworkCore;

namespace APP.Domain
{
    // Inherits from the Entity Framework's DbContext class for
    // database configurations and operations.
    public class Db : DbContext
    {
        // Represents the Users table in the database.
        public DbSet<User> Users { get; set; }

        // Represents the Roles table in the database.
        public DbSet<Role> Roles { get; set; }

        // Represents the UserRoles table in the database.
        public DbSet<UserRole> UserRoles { get; set; }

        // Represents the Statuses table in the database.
        public DbSet<Status> Statuses { get; set; }

        // Constructor that takes options of type DbContextOptions
        // parameter and passes it to the base DbContext class constructor.
        // The options parameter is used to configure the database
        // connection with such as the connection string.
        public Db(DbContextOptions options) : base(options)
        {
        }
    }
}
```

Database Creation:

8. In the MVC Project, open appsettings.json, which will contain the application configuration such as the connection string, static information, etc., then add the ConnectionStrings section as below:

```
"ConnectionStrings": {
  "Db": "data source=DB.db"
},
```

Db is the connection string name, DB.db will be the SQLite database file that will be created.

9. In the MVC Project, open Program.cs and add below in the IoC (Inversion of Control) Container:

```
// Register the application's DbContext dependency injection by sending Db instances
// to the injected service class constructors' DbContext or Db parameter, using SQLite
// with the connection string from appsettings.json.
builder.Services.AddDbContext<DbContext, Db>(
    options => options.UseSqlite(
       builder.Configuration.GetConnectionString(nameof(Db))));
// nameof(Db) = "Db" which is the class name.
```

10. Create your DB.db database using migrations:
    - Open Package Manager Console from Visual Studio menu -> Tools -> NuGet Package Manager -> Package Manager Console
    - Set APP as Default Project in Package Manager Console
    - Run:
        add-migration v1
        update-database
    - For Rider, you can use the UI as described in JetBrains documentation, or from the terminal
      Install:
        dotnet tool install -g dotnet-ef
      Run:
        dotnet ef migrations add v1
        dotnet ef database update -p APP -s MVC
    - You can see the created DB.db database file in MVC Project.
    - Optionally in Visual Studio, you can install the SQLite and SQL Server Compact Toolbox extension from Visual Studio menu -> Extensions -> Manage Extensions to connect to the created DB.db SQLite database.

11. In the APP Project under Domain folder, optionally a DbFactory class can be created to be used for Scaffolding (creation of MVC Controllers and Views automatically using templates) and seeding the database tables with initial data as below:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using System.Globalization;

namespace APP.Domain
{
    // Provides a factory for creating Db instances at design time.
    // This is used by Entity Framework (EF) tools such as
    // migrations to construct the database context when the
    // application is not running.
    // This class may need to be created if there are any exceptions
    // during Scaffolding.
    public class DbFactory : IDesignTimeDbContextFactory<Db>
    {
        // The connection string in the configuration file
        // (appsettings.json).
```

```
const string CONNECTIONSTRING = "data source=DB.db";


// Creates a new instance of the Db type using the connection
// string. This method is called by EF tooling at design time.
public Db CreateDbContext(string[] args)
{
    var optionsBuilder = new DbContextOptionsBuilder<Db>();
    optionsBuilder.UseSqlite(CONNECTIONSTRING);
    return new Db(optionsBuilder.Options);
}



// Seeds the database tables with initial data (Optional).
public void SeedDb()
{
    // Create a new instance of the Db type:
    var db = CreateDbContext(null);

    // Delete existing data:
    var userRoles = db.UserRoles.ToList();
    db.UserRoles.RemoveRange(userRoles);
    var roles = db.Roles.ToList();
    db.Roles.RemoveRange(roles);
    var users = db.Users.ToList();
    db.Users.RemoveRange(users);
    var statuses = db.Statuses.ToList();
    db.Statuses.RemoveRange(statuses);

    // Reset identity columns (for SQLite):
    // IDs will start from 1
    db.Database.ExecuteSqlRaw("UPDATE SQLITE_SEQUENCE " +
        "SET SEQ=0 WHERE NAME='UserRoles'");
    db.Database.ExecuteSqlRaw("UPDATE SQLITE_SEQUENCE " +
        "SET SEQ=0 WHERE NAME='Roles'");
    db.Database.ExecuteSqlRaw("UPDATE SQLITE_SEQUENCE " +
        "SET SEQ=0 WHERE NAME='Users'");
    db.Database.ExecuteSqlRaw("UPDATE SQLITE_SEQUENCE " +
        "SET SEQ=0 WHERE NAME='Statuses'");

    // Insert new data:
    // Roles:
    var role = new Role() { Name = "Admin" };
    db.Roles.Add(role);
    // Role may also be written instead of Role()
    role = new Role { Name = "User" };
    db.Roles.Add(role);
    db.SaveChanges(); // commit changes to the database
    // Statuses with Users:
    db.Statuses.Add(new Status
    {
        Title = "Active",
        Users = new List<User>
        {
            new User
            {
                UserName = "admin",
                Password = "admin",
                RegistrationDate = new DateTime(
                    2026, 7, 16, 20, 57, 45),
                UserRoles = new List<UserRole>
                {
```

```
                        new UserRole { RoleId = db.Roles.Single(
                            r => r.Name == "Admin").Id }
                    }
                },
                new User
                {
                    UserName = "user",
                    Password = "user",
                    RegistrationDate = DateTime.Parse(
                        "07/16/2026 20:58:13", new CultureInfo("en-US")),
                    UserRoles = new List<UserRole>
                    {
                        new UserRole { RoleId = db.Roles.Single(
                            r => r.Name == "User").Id }
                    }
                }
            }
        });
        db.Statuses.Add(new Status { Title = "Inactive" });
        db.SaveChanges();
    }
  }
}
```

12. In the MVC Project, right-click Controllers folder then Add -> Controller -> Common -> MVC -> MVC Controller - Empty to create the DbController for seeding intial data to the database tables as below:

```
using APP.Domain;
using Microsoft.AspNetCore.Mvc;
using System.Text;

namespace MVC.Controllers
{
    // Database Controller for seeding the database with initial data
    // through the Seed action.
    public class DbController : Controller
    {
        // Action method to seed the database with initial data.
        [Route("SeedDb")] // will change the route of the action to
                          // /SeedDb instead of /Db/Seed
        public IActionResult Seed()
        {
            // Initialize the DbFactory instance.
            var dbFactory = new DbFactory();

            // Seed the database using the SeedDb method.
            dbFactory.SeedDb();

            // Return a success message as HTML content using
            // UTF-8 character set.
            return Content("<h1>Database seed successful.</h1>",
                "text/html", Encoding.UTF8);
        }
    }
}
```

13. Run the solution by hitting F5 or play icon with https on the top toolbar. If a warning dialog appears about SSL, click Yes for all dialogs. If you have any problems running your solution, change the play icon with https to http then run. Finally, enter /SeedDb after the server name (localhost) and port number in the address bar of the browser e.g. “https://localhost:7021/seeddb” to seed the database tables with initial data.

CRUD Operations Business Logic Implementation:

14. Right-click the APP Project then click Add -> New Folder to create a folder named Models.

15. Right-click the Models folder then click Add -> Class to create a model (DTO: Data Transfer Object) class file called RoleResponse.cs as below:

```
using CORE.Domain;

namespace APP.Models
{
    // Response properties are created according to the data to be
    // presented in API responses or UIs (Views).
    public class RoleResponse : Record
    {
        // Copy all the non navigation properties from Role entity.
        public string Name { get; set; }
    }
}
```

16. Right-click the Models folder then click Add -> Class to create a model class file called UserResponse.cs as below:

```
using APP.Domain;
using CORE.Domain;
using System.ComponentModel;

namespace APP.Models
{
    // Response properties are created according to the data to be
    // presented in API responses or UIs (Views).
    public class UserResponse : Record
    {
        // Copy all the non navigation properties from User entity.
        // DisplayName attribute is used to write the title
        // for the property in the view instead of the property
        // name, so "User Name" will be written in the view instead
        // of "UserName". If no DisplayName attribute is defined,
        // the property name will be written in the view.
        // Therefore, DisplayName attribute can be used to show user
        // friendly names in both validation error messages and views
        // using DisplayNameFor HTML Helper.
```

```
[DisplayName("User Name")]
public string UserName { get; set; }

public string Password { get; set; }

[DisplayName("First Name")]
public string FirstName { get; set; }

[DisplayName("Last Name")]
public string LastName { get; set; }

public Genders Gender { get; set; }

public DateTime? BirthDate { get; set; }

public DateTime RegistrationDate { get; set; }

public double Score { get; set; }

public bool IsOnline { get; set; }

public int StatusId { get; set; }



// Add the new properties ending with R declaring Response
// for custom properties or formatted string value properties.
[DisplayName("Online")]
public string IsOnlineR { get; set; }

[DisplayName("Score")]
public string ScoreR { get; set; }

[DisplayName("Registration Date")]
public string RegistrationDateR { get; set; }

[DisplayName("Birth Date")]
public string BirthDateR { get; set; }

[DisplayName("Gender")]
public string GenderR { get; set; }

// FirstName + " " + LastName concatenated value.
[DisplayName("Full Name")]
public string FullNameR { get; set; }

// Password value hidden with * characters.
[DisplayName("Password")]
public string PasswordR { get; set; }



// Way 1:
// Property for the Title value of the
// related Status entity.
[DisplayName("Status")]
public string StatusTitleR { get; set; }

// Way 2:
// Property for the mapped StatusResponse object
// from the related Status entity.
// Either Way 1, Way 2 or both can be used.
```

```
        [DisplayName("Status")]
        public StatusResponse StatusR { get; set; }



        // Way 1:
        // Property for the concatenated role names
        // of related Role entities.
        [DisplayName("Roles")]
        public string RoleNamesR { get; set; }

        [DisplayName("Roles")]
        // Way 2:
        // Property for the RoleResponse objects projected
        // from the related Role entities.
        // Either Way 1, Way 2 or both can be used.
        public List<RoleResponse> RolesR { get; set; }
    }
}
```

17. Right-click the Models folder then click Add -> Class to create a model class file called StatusResponse.cs as below:

```
using CORE.Domain;
using System.ComponentModel;

namespace APP.Models
{
    // Response properties are created according to the data to be
    // presented in API responses or UIs (Views).
    public class StatusResponse : Record
    {
        // Copy all the non navigation properties from Status entity.
        public string Title { get; set; }



        // Add the new properties ending with R declaring Response
        // for custom properties or formatted string value properties.
        // Way 1: Map primitive type properties for basic information.
        [DisplayName("Users")]
        public string UserNamesR { get; set; }

        // Property value for the count of the users of the status.
        [DisplayName("User Count")]
        public int UserCountR { get; set; }

        // Way 2: Map complex type object for more information.
        // Either Way 1, Way 2 or both can be used.
        [DisplayName("Users")]
        public List<UserResponse> UsersR { get; set; }
    }
}
```

18. Right-click the Models folder then click Add -> Class to create a model class file called RoleRequest.cs as below:

```
using CORE.Domain;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace APP.Models
{
    // Request properties are created according to the data that
    // will be retrieved from APIs or UIs (views).
    public class RoleRequest : Record
    {
        // Copy all the non navigation properties from Role entity.
        // Name can't be null and can be maximum 25 characters.
        [Required, StringLength(25)]
        public string Name { get; set; }



        // Modify (1 to many) or add (many to many) the relationship properties.
        // Since we won't update the relational user data (UserRoles)
        // through Role request, we don't need the UserIds property here.
        // Required may not be defined if a role may not have a user.
        // For roles-users many to many relationship.
        //[Required]
        //[DisplayName("Users")]
        //public List<int> UserIds { get; set; } = new();
    }
}
```

19. Right-click the Models folder then click Add -> Class to create a model class file called UserRequest.cs as below:

```
using APP.Domain;
using CORE.Domain;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace APP.Models
{
    // Request properties are created according to the data that
    // will be retrieved from APIs or UIs (views).
    public class UserRequest : Record
    {
        // Copy all the non navigation properties from User entity.
        /*
        ErrorMessage parameter can be set in all data annotations
        to show custom validation error messages:
        Example 1: [Required(ErrorMessage = "{0} is required!")]
        where {0} is the DisplayName if defined otherwise property name
        which is "User Name".
        Example 2: [StringLength(30, 3,
        ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        where {0} is the DisplayName if defined otherwise property name
        which is "User Name", {1} is the first parameter which is 30 and
        {2} is the second parameter which is 3.
        */
```

```
        // User Name is required and can be minimum 3 maximum 30 characters.
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(30, MinimumLength = 3,
        ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        [DisplayName("User Name")]
        public string UserName { get; set; }

        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(15, MinimumLength = 3,
        ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        public string Password { get; set; }

        [StringLength(50, ErrorMessage = "{0} must be maximum {1} characters!")]
        [DisplayName("First Name")]
        public string FirstName { get; set; }

        [StringLength(50, ErrorMessage = "{0} must be maximum {1} characters!")]
        [DisplayName("Last Name")]
        public string LastName { get; set; }

        public Genders Gender { get; set; }

        [DisplayName("Birth Date")]
        public DateTime? BirthDate { get; set; }

        // We don't need to get the RegistrationDate value from the client
        // since it will be assigned automatically in the service.

        // Minimum value can be 0, maximum value can be 5.
        /*
        The type changed from double to double? to be able to use [Required]
        attribute. If the type is double, the default value will be 0 and
        the validation will always be successful even if no value is assigned.
        */
        [Required(ErrorMessage = "{0} is required!")]
        [Range(0, 5, ErrorMessage = "{0} must be between {1} and {2}!")]
        public double? Score { get; set; }

        // We don't need to get the IsOnline value from the client
        // since it will be assigned automatically during login and logout.



        // Modify (1 to many) or add (many to many) the relationship properties.
        /*
        The type changed from int to int? to be able to use [Required] attribute.
        If the type is int, the default value will be 0 and the validation will
        always be successful even if no value is assigned. Also we wan't to get a
        null value from the drop down list (select HTML tag) when the
        "-- Select --" option is selected, therefore the application user must
        select a status option other than "-- Select --".
        */
        // For users-status one to many relationship.
        [Required(ErrorMessage = "{0} is required!")]
        [DisplayName("Status")]
        public int? StatusId { get; set; }

        // Required may not be defined if a user may not have a role.
        // For users-roles many to many relationship.
        [Required(ErrorMessage = "At least one role is required!")]
        [DisplayName("Roles")]
        public List<int> RoleIds { get; set; }
    }
}
```

20. Right-click the Models folder then click Add -> Class to create a model class file called StatusRequest.cs as below:

```
using CORE.Domain;
using System.ComponentModel.DataAnnotations;

namespace APP.Models
{
    // Request properties are created according to the data that
    // will be retrieved from APIs or UIs (views).
    public class StatusRequest : Record
    {
        // Copy all the non navigation properties from Status entity.
        [Required, StringLength(5)]
        public string Title { get; set; }
    }
}
```

21. Right-click the APP Project then click Add -> New Folder to create a folder named Services.

22. Right-click the Services folder then click Add -> Class to create a service class file called RoleService.cs as below:

```
using APP.Domain;
using APP.Models;
using CORE.Models;
using CORE.Services;
using Microsoft.EntityFrameworkCore;

namespace APP.Services
{
    // RoleService inherits the DbService class for Role entity type,
    // therefore Role entity database operations of the DbService
    // class can be used in this class.
    // RoleService also implements the IService interface for
    // RoleRequest and RoleResponse types, therefore IService method
    // definitions can be implemented for types RoleRequest and
    // RoleResponse in this class, which will be used by
    // the controller actions.
    public class RoleService : DbService<Role>,
        IService<RoleRequest, RoleResponse>
    {
        // Constructor that passes the injected DbContext instance
        // to the DbService base class constructor.
        public RoleService(DbContext db) : base(db)
        {
        }

        // Gets the Role entity query then projects the Role
        // entity query to the RoleResponse query and returns it.
        // Example query: "select Name from Roles".
        // where Name is the RoleResponse property.
        // The returned query can be executed by invoking
        // ToList, SingleOrDefault, FirstOrDefault, Any,
        // etc. LINQ (Language Integrated Query) methods
        // and then the returned result can be used.
```

```
// This projected query by the Select LINQ method
// will be used by the List and Single methods of the
// IService interface.
public IQueryable<RoleResponse> Query()
{
    return DbQuery() // "select * from Roles" entity query.
        .Select(roleEntity => new RoleResponse()
        // () after the class name may not be written.
        {
            Id = roleEntity.Id,
            Name = roleEntity.Name
        }); // Map each Role entity property
            // to RoleResponse property.
}


// Gets the Role entity by ID then returns a mapped
// RoleRequest instance from the found Role entity.
public RoleRequest Edit(int id)
{
    var roleEntity = DbSingle(id);
    // Example SQL: "select * from Roles where Id = 3".

    // roleEntity may be null since the OrDefault version
    // of the Single method is used. If Single method
    // was used, an exception would be thrown.
    if (roleEntity is null) // if (roleEntity == null) can
                            // also be written.
        return null;

    // Map each Role entity property to RoleRequest property
    // and return the RoleRequest instance.
    return new RoleRequest
    {
        Id = roleEntity.Id,
        Name = roleEntity.Name
    };
}

// Checks whether the role with the same trimmed and
// case sensitive name exists in the Roles database table
// first. If it doesn't exist, inserts the Role entity
// mapped from the RoleRequest instance to the Roles
// database table.
public CommandResponse Add(RoleRequest request)
{
    if (DbQuery()
        .Any(roleEntity =>
            roleEntity.Name == request.Name.Trim()))
        return Error("\"" + request.Name + "\"" + " role exists!");
    // Trim method removes white space characters in the
    // beginning and at the end.

    // Create a new Role entity instance from the
    // RoleRequest instance then insert the entity in the Roles
    // database table (second save default parameter of the
    // DbAdd method is true therefore changes of the DbSets
    // are commited to the related database tables).
    var roleEntity = new Role
    {
        Name = request.Name.Trim()
    };
    DbAdd(roleEntity);

    // Entity's Id value will be updated by the database
    // after the insert operation.
```

```
            return Success(roleEntity.Id, "\"" + roleEntity.Name + "\"" +
                " role created successfully.");
        }

        // Checks whether the role with the same trimmed and
        // case sensitive name other than the role request record
        // exists in the Roles database table first.
        // If the entity doesn't exist, gets the Role entity by ID from the
        // table, then updates the Role entity properties
        // from role request properties and finally updates the
        // Role entity in the Roles database table.
        public CommandResponse Update(RoleRequest request)
        {
            if (DbQuery()
                .Any(roleEntity => roleEntity.Id != request.Id &&
                    roleEntity.Name == request.Name.Trim()))
                return Error($"\"{request.Name}\" role exists!");
            // $ with strings can be used for concatenation with { and }.

            // Get the Role entity by ID from the Roles database table
            // and check whether it is null or not.
            var roleEntity = DbSingle(request.Id);
            if (roleEntity is null)
                return Error("Role not found!");

            // Update Role entity properties from role request properties.
            // Then update the entity in the Roles database table (second save
            // default parameter of the DbUpdate method is true
            // therefore changes of the DbSets are commited to the related
            // database tables).
            roleEntity.Name = request.Name.Trim();
            DbUpdate(roleEntity);

            return Success(roleEntity.Id,
                $"\"{roleEntity.Name}\" role updated successfully.");
        }

        // Gets the Role entity by ID from the Roles database table.
        // If the entity exists, deletes the entity from the Roles
        // database table.
        public CommandResponse Remove(int id)
        {
            // Get the Role entity by ID from the Roles database table
            // and check whether it is null or not.
            var roleEntity = DbSingle(id);
            if (roleEntity is null)
                return Error("Role not found!");

            // Delete the Role entity from the Roles database table
            // (second save default parameter of the DbRemove method is
            // true therefore changes of the DbSets are commited to
            // the related database tables).
            // Since the delete rule of the Roles and UserRoles relation
            // in the database is Cascade, the relational UserRole entities
            // will be deleted automatically (not recommended).
            DbRemove(roleEntity);

            return Success(roleEntity.Id,
                $"\"{roleEntity.Name}\" role deleted successfully.");
        }
    }
}
```

23. Right-click the Services folder then click Add -> Class to create a service class file called StatusService.cs as below:

```
using APP.Domain;
using APP.Models;
using CORE.Models;
using CORE.Services;
using Microsoft.EntityFrameworkCore;

namespace APP.Services
{
    // StatusService inherits the DbService class for Status entity type,
    // therefore Status entity database operations of the DbService
    // class can be used in this class.
    // StatusService also implements the IService interface for
    // StatusRequest and StatusResponse types, therefore IService method
    // definitions can be implemented for types StatusRequest and
    // StatusResponse in this class, which will be used by
    // the controller actions.
    public class StatusService : DbService<Status>,
        IService<StatusRequest, StatusResponse>
    {
        // Constructor that passes the injected DbContext instance
        // to the DbService base class constructor.
        public StatusService(DbContext db) : base(db)
        {
        }

        // Overrides the base virtual DbQuery method to include
        // User entities data to the query to be used in the
        // Query and Remove methods and orders the query by Status
        // entity Title. The overridden DbQuery method
        // can be used in any method below by invoking
        // the DbQuery method.
        protected override IQueryable<Status> DbQuery()
        {
            return base.DbQuery() // "select * from Statuses"
                                  // entity query.
                .Include(s => s.Users) // Includes the relational
                                       // User entities data to the query
                                       // by using left outer join.
                .OrderBy(s => s.Title); // Orders the query by Title
                                        // property of the Status entity
                                        // in ascending order.
                                        // OrderByDescending can be used
                                        // for descending order.
            // s: Status entity delegate.
        }

        // Gets the updated query from the DbQuery method then projects
        // the Status entities to status responses and returns the
        // status response query.
        // Example query: "select Title, UserNamesR, UserCountR from Statuses
        // left outer join Users on Statuses.Id = Users.StatusId order by
        // Title". where Title, UserNamesR and UserCountR are the
        // StatusResponse properties.
        // The returned query can be executed by invoking
        // ToList, SingleOrDefault, FirstOrDefault, Any, etc. LINQ
        // (Language Integrated Query) methods and then the returned result
        // can be used.
        // This projected query by the Select LINQ method will be used by
        // the List and Single methods of the IService interface.
```

```
public IQueryable<StatusResponse> Query()
{
    return DbQuery()
        .Select(s => new StatusResponse
        {
            // Map each Status entity property to
            // StatusResponse property.
            Id = s.Id,
            Title = s.Title,
            // Way 1: Map primitive type data for basic information.
            UserNamesR = string.Join(", ", s.Users.Select(u => u.UserName)),
            // string type's Join method gets a seperator as the first
            // parameter and gets a collection of strings as the second
            // parameter then concatenates each string item in collection
            // with the seperator and returns a string.
            UserCountR = s.Users.Count,
            // Count property of the List type returns the item count
            // of the collection.
            // Way 2: Map complex type object for more information.
            // Either Way 1, Way 2 or both can be used.
            UsersR = s.Users.Select(u => new UserResponse
            {
                // Map each User entity property to
                // StatusResponse's UserResponse property.
                Id = u.Id,
                UserName = u.UserName,
                PasswordR = new string('*', u.Password.Length),
                // Let's hide the password with *.
                FirstName = u.FirstName,
                LastName = u.LastName,
                FullNameR = u.FirstName + " " + u.LastName,
                // Concatenate the first name and last name
                // with a white space for the full name.
                RegistrationDate = u.RegistrationDate,
                BirthDate = u.BirthDate,
                Gender = u.Gender,
                IsOnline = u.IsOnline,
                Score = u.Score,
                StatusId = u.StatusId
            }).ToList()
        });
    // s: Status entity delegate, u: User entity delegate.
}

// Gets the Status entity by ID then returns a mapped
// StatusRequest instance from the found Status entity.
public StatusRequest Edit(int id)
{
    var statusEntity = DbSingle(id);
    // Example SQL: "select * from Statuses where Id = 5".

    // statusEntity may be null since the OrDefault version
    // of the Single method is used. If Single method
    // was used, an exception would be thrown.
    if (statusEntity is null) // if (statusEntity == null) can
                              // also be written.
        return null;

    // Map each Status entity property to StatusRequest property
    // and return the StatusRequest instance.
    return new StatusRequest
    {
        Id = statusEntity.Id,
        Title = statusEntity.Title
    };
}
```

```
// Checks whether the status with the same trimmed and
// case sensitive title exists in the Statuses database table
// first. If it doesn't exist, inserts the Status entity
// mapped from the StatusRequest instance to the Statuses
// database table.
public CommandResponse Add(StatusRequest request)
{
    if (DbQuery()
        .Any(s => s.Title == request.Title.Trim()))
        return Error("Status with the same title exists!");
    // s: Status entity delegate.
    // Trim method removes white space characters in the
    // beginning and at the end.

    // Create a new Status entity instance from the
    // StatusRequest instance then insert the entity in the
    // Statuses database table (second save default parameter
    // of the DbAdd method is true therefore changes of the
    // DbSets are commited to the related database tables).
    var statusEntity = new Status
    {
        Title = request.Title.Trim()
    };
    DbAdd(statusEntity);

    // Entity's Id value will be updated by the database
    // after the insert operation.
    return Success(statusEntity.Id, "Status created successfully.");
}

// Checks whether the status with the same trimmed and
// case sensitive title other than the status request record
// exists in the Statuses database table first.
// If the entity doesn't exist, gets the Status entity by ID
// from the table, then updates the Status entity properties
// from status request properties and finally updates the
// Status entity in the Statuses database table.
public CommandResponse Update(StatusRequest request)
{
    if (DbQuery()
        .Any(s => s.Id != request.Id &&
            s.Title == request.Title.Trim()))
        return Error("Status with the same title exists!");

    // Get the Status entity by ID from the Statuses database
    // table and check whether it is null or not.
    var statusEntity = DbSingle(request.Id);
    if (statusEntity is null)
        return Error("Status not found!");

    // Update Status entity properties from status request properties.
    // Then update the entity in the Statuses database table (second save
    // default parameter of the DbUpdate method is true
    // therefore changes of the DbSets are commited to the related
    // database tables).
    statusEntity.Title = request.Title.Trim();
    DbUpdate(statusEntity);

    return Success(statusEntity.Id, "Status updated successfully.");
}

// Gets the Status entity by ID from the Statuses database table.
// If the entity exists, checks if the Status entity has
// relational User entities. If not, deletes the entity from the
// Statuses database table.
```

```
        public CommandResponse Remove(int id)
        {
            // Get the Status entity by ID from the Statuses database table
            // and check whether it is null or not.
            var statusEntity = DbSingle(id);
            if (statusEntity is null)
                return Error("Status not found!");

            // Check if the Status entity has relational User entities.
            if (statusEntity.Users.Any()) // if (statusEntity.Users.Count > 0)
                                          // can also be written.
                return Error("Status has relational users!");

            // Delete the Status entity from the Statuses database table
            // (second save default parameter of the DbRemove method is
            // true therefore changes of the DbSets are commited to
            // the related database tables).
            DbRemove(statusEntity);

            return Success(statusEntity.Id, "Status deleted successfully.");
        }
    }
}
```

24. Right-click the Services folder then click Add -> Class to create a service class file called UserService.cs as below:

```
using APP.Domain;
using APP.Models;
using CORE.Models;
using CORE.Services;
using Microsoft.EntityFrameworkCore;

namespace APP.Services
{
    // UserService inherits the DbService class for User entity type,
    // therefore User entity database operations of the DbService
    // class can be used in this class.
    // UserService also implements the IService interface for
    // UserRequest and UserResponse types, therefore IService method
    // definitions can be implemented for types UserRequest and
    // UserResponse in this class, which will be used by
    // the controller actions.
    public class UserService : DbService<User>,
        IService<UserRequest, UserResponse>
    {
        // Constructor that passes the injected DbContext instance
        // to the DbService base class constructor.
        public UserService(DbContext db) : base(db)
        {
        }

        // Overrides the base virtual DbQuery method to include
        // Status, UserRole and Role entities data to the query
        // to be used in the Query, Edit, Update and Remove
        // methods and orders the query by User entity Score
        // property descending first, then the ordered records
        // are ordered descending by User entity RegistrationDate
        // property. Finally the ordered records are ordered
        // ascending by User entity UserName property.
        // The overridden DbQuery method can be used in any method
        // below by invoking the DbQuery method.
        protected override IQueryable<User> DbQuery()
```

```
{
    // Relationships:
    // User <-> Status.
    // User <-> UserRoles <-> Roles.
    return base.DbQuery()
        .Include(u => u.Status) // Include Status entities
                               // from User entities.
        .Include(u => u.UserRoles) // Include UserRole entities
                                   // from User entities.
        .ThenInclude(ur => ur.Role) // Then include Role entities
                                    // from UserRole entities.
        .OrderByDescending(u => u.Score)
        .ThenByDescending(u => u.RegistrationDate)
        .ThenBy(u => u.UserName);
        // Only one OrderBy method must be called.
        // The other called ordering methods must be ThenBy.
    // u: User entity delegate, ur: UserRole entity delegate.
}

// Gets the updated query from the DbQuery method then projects
// the User entities to user responses and returns the
// user response query.
// The returned query can be executed by invoking
// ToList, SingleOrDefault, FirstOrDefault, Any, etc. LINQ
// (Language Integrated Query) methods and then the returned result
// can be used.
// This projected query by the Select LINQ method will be used by
// the List and Single methods of the IService interface.
public IQueryable<UserResponse> Query()
{
    return DbQuery().Select(u => new UserResponse
    {
        // Map each User entity property to UserResponse property.
        Id = u.Id,
        UserName = u.UserName,
        Password = u.Password,
        FirstName = u.FirstName,
        LastName = u.LastName,
        RegistrationDate = u.RegistrationDate,
        BirthDate = u.BirthDate,
        IsOnline = u.IsOnline,
        Gender = u.Gender,
        Score = u.Score,
        StatusId = u.StatusId,
        // Way 1: Map primitive type data for basic information.
        StatusTitleR = u.Status.Title,
        // Way 2: Map complex type object for more information.
        // Either Way 1, Way 2 or both can be used.
        StatusR = new StatusResponse
        {
            // Map each Status entity property to
            // UserResponse's StatusResponse property.
            Id = u.Status.Id,
            Title = u.Status.Title
        },
        // Way 1: Map primitive type data for basic information.
        RoleNamesR = string.Join("<br>", u.UserRoles
            .OrderBy(ur => ur.Role.Name)
            .Select(ur => ur.Role.Name)),
        // Way 2: Map complex type object for more information.
        // Either Way 1, Way 2 or both can be used.
        RolesR = u.UserRoles.OrderBy(ur => ur.Role.Name)
            .Select(ur => new RoleResponse
        {
            // Map each Role entity property to
            // UserResponse's RoleResponse property through UserRoles.
```

```
                Id = ur.Role.Id,
                Name = ur.Role.Name
            }).ToList(),
            // Hidden password with * character.
            PasswordR = new string('*', u.Password.Length),
            // Concatenated first name and list name with white space character.
            FullNameR = u.FirstName + " " + u.LastName,
            // Formatted date and time in month/day/year hour:minute:second
            // format. No need to use the CultureInfo instance for the second
            // parameter of the ToString method since CultureInfo has been
            // assigned in the base abstract Service class by the Culture
            // property assignment. If Culture property of the base abstract
            // Service class is changed within this class constructor,
            // the changed value will be used by the CultureInfo instance and
            // then ToString method.
            RegistrationDateR = u.RegistrationDate
                .ToString("MM/dd/yyyy HH:mm.ss"),
            // Ternary operator:
            BirthDateR = u.BirthDate.HasValue ? // if (u.BirthDate != null).
                u.BirthDate.Value.ToShortDateString() : // Assign u.BirthDate's
                                                        // value since u.BirthDate
                                                        // is nullable.
                string.Empty, // "" string.
            // ToShortDateString returns the date in month/day/year format.
            IsOnlineR = u.IsOnline ? "Yes" : "No",
            GenderR = u.Gender.ToString(), // Assigns "Man" or "Woman".
            ScoreR = u.Score.ToString("N1") // N: Number format.
                                            // 1: 1 decimal.
                                            // C can also be used for currency.
                                            // No need to define the second
                                            // CultureInfo parameter since Culture
                                            // property has been assigned in the
                                            // base abstract Service class.
        }); // u: User entity delegate, ur: UserRole entity delegate.
    }

    // Gets the User entity by ID then returns a mapped
    // UserRequest instance from the found User entity.
    public UserRequest Edit(int id)
    {
        var userEntity = DbSingle(id);
        // Example SQL: "select * from Users where Id = 7".

        // userEntity may be null since the OrDefault version
        // of the Single method is used. If Single method
        // was used, an exception would be thrown.
        if (userEntity is null) // if (userEntity == null) can
                                // also be written.
            return null;

        // Map each User entity property to UserRequest property
        // and return the UserRequest instance.
        return new UserRequest
        {
            Id = userEntity.Id,
            UserName = userEntity.UserName,
            Password = userEntity.Password,
            FirstName = userEntity.FirstName,
            LastName = userEntity.LastName,
            BirthDate = userEntity.BirthDate,
            Gender = userEntity.Gender,
            Score = userEntity.Score,
            StatusId = userEntity.StatusId,
            RoleIds = userEntity.RoleIds
        };
    }
```

```
// Checks whether the user with the same trimmed and
// case sensitive user name exists in the Users database table
// first. If it doesn't exist, inserts the User entity
// mapped from the UserRequest instance to the Users
// database table.
public CommandResponse Add(UserRequest request)
{
    if (DbQuery().Any(u =>
        u.UserName == request.UserName.Trim()
        //&& u.BirthDate == request.BirthDate
        // One or many conditions can be used by && (and), || (or)
        // and ! (not) operators.
    ))
        return Error("User with the same user name exists!");
    // u: User entity delegate.
    // Trim method removes white space characters in the
    // beginning and at the end.

    // Create a new User entity instance from the
    // UserRequest instance then insert the entity in the
    // Users database table (second save default parameter
    // of the DbAdd method is true therefore changes of the
    // DbSets are commited to the related database tables).
    var userEntity = new User
    {
        UserName = request.UserName.Trim(),
        Password = request.Password.Trim(),
        FirstName = request.FirstName?.Trim(),
        // Since request.FirstName may be null, ? is used after
        // meaning that if request.FirstName is null, assign
        // null to the User entity's FirstName property else
        // assign the trimmed value of request.FirstName.
        LastName = request.LastName?.Trim(),
        // Since request.LastName may be null, ? is used after
        // meaning that if request.LastName is null, assign
        // null to the User entity's LastName property else
        // assign the trimmed value of request.LastName.
        RegistrationDate = DateTime.Now,
        BirthDate = request.BirthDate,
        Gender = request.Gender,
        Score = request.Score ?? 0,
        StatusId = request.StatusId ?? 0,
        // ??: Null coalescing operator: If request.StatusId is
        // null assign 0 else assign request.StatusId value to the
        // User entity StatusId property. request.StatusId.Value
        // may also be used since request.StatusId is required.
        // Same for request.Score.
        RoleIds = request.RoleIds
    };
    DbAdd(userEntity);

    // Entity's Id value will be updated by the database after insert.
    return Success(userEntity.Id, "User created successfully.");
}

// Checks whether the user with the same trimmed and
// case sensitive user name other than the user request record
// exists in the Users database table first.
// If the entity doesn't exist, gets the User entity by ID
// from the table, deletes the relational UserRole entities,
// then updates the User entity properties from user request
// properties and finally updates the User entity in the
// Users database table.
public CommandResponse Update(UserRequest request)
{
```

```
        if (DbQuery().Any(u => u.Id != request.Id
            && u.UserName == request.UserName.Trim()))
            return Error("User with the same user name exists!");

        // Get the User entity by ID from the Users database
        // table and check whether it is null or not.
        var userEntity = DbSingle(request.Id);
        if (userEntity is null)
            return Error("User not found!");

        // Delete the relational UserRole entities first.
        if (request.RoleIds is not null)
            DbRemove(userEntity.UserRoles);

        // Then update User entity properties from user request properties.
        // Then update the entity in the Users database table (second save
        // default parameter of the DbUpdate method is true
        // therefore changes of the DbSets are commited to the related
        // database tables).
        userEntity.UserName = request.UserName.Trim();
        userEntity.Password = request.Password.Trim();
        userEntity.FirstName = request.FirstName?.Trim();
        // Since request.FirstName may be null, ? is used after
        // meaning that if request.FirstName is null, assign
        // null to the User entity's FirstName property else
        // assign the trimmed value of request.FirstName.
        userEntity.LastName = request.LastName?.Trim();
        // Since request.LastName may be null, ? is used after
        // meaning that if request.LastName is null, assign
        // null to the User entity's LastName property else
        // assign the trimmed value of request.LastName.
        userEntity.BirthDate = request.BirthDate;
        userEntity.Gender = request.Gender;
        if (request.Score.HasValue)
            userEntity.Score = request.Score.Value;
        if (request.StatusId.HasValue)
            userEntity.StatusId = request.StatusId.Value;
        if (request.RoleIds is not null)
            userEntity.RoleIds = request.RoleIds;
        DbUpdate(userEntity);

        return Success(userEntity.Id, "User updated successfully.");
    }

    // Gets the User entity by ID from the Users database table.
    // If the entity exists, deletes the relational UserRole entities.
    // Then deletes the entity from the Users database table.
    public CommandResponse Remove(int id)
    {
        // Get the User entity by ID from the Users database table
        // and check whether it is null or not.
        var userEntity = DbSingle(id);
        if (userEntity is null)
            return Error("User not found!");

        // Delete the relational UserRole entities first even Cascade
        // delete rule is defined in the database (recommended).
        // This way would also be suitable if the delete rule was
        // defined as No Action in the database.
        DbRemove(userEntity.UserRoles);

        // Then delete the User entity from the Users database table
        // (second save default parameter of the DbRemove method is
        // true therefore changes of the DbSets are commited to
        // the related database tables).
        DbRemove(userEntity);
```

```
            return Success(userEntity.Id, "User deleted successfully.");
        }
    }
}
```

C) MVC (Model-View-Controller) Project for CRUD Operations

1. In Program.cs IoC (Inversion of Control) Container section, add dependency registrations under DbContext dependency registration as below (The complete implementation of the Program.cs can be found at Additional Information section of this document (1)):

```
// Register the UserService dependency injection by sending
// IService<UserRequest, UserResponse> instance reference to
// the injected controller class constructor's
// IService<UserRequest, UserResponse> parameter.
// Injection through abstract class or interface is suitable
// for SOLID Principles.
builder.Services
    .AddScoped<IService<UserRequest, UserResponse>, UserService>();

// Register the RoleService dependency injection by sending
// IService<RoleRequest, RoleResponse> instance reference to
// the injected controller class constructor's
// IService<RoleRequest, RoleResponse> parameter.
builder.Services
    .AddScoped<IService<RoleRequest, RoleResponse>, RoleService>();

// Register the StatusService dependency injection by sending
// IService<StatusRequest, StatusResponse> instance reference to
// the injected controller class constructor's
// IService<StatusRequest, StatusResponse> parameter.
builder.Services
    .AddScoped<IService<StatusRequest, StatusResponse>, StatusService>();
```

2. A Controller class has one or more Actions (methods) which HTTP requests can be sent. Generally each Action returns an IActionResult instance for HTTP responses after executed.
   In MVC, generally View results are returned.
   The starting Action of the MVC Application is the Index Action of the Home Controller since it is defined at the bottom of the Program.cs.
   You don't need to modify the HomeController.cs in the Controllers folder. However, if you want to see different Action results, you can modify the HomeController.cs as below:

```
using Microsoft.AspNetCore.Mvc;
using MVC.Models;
using System.Diagnostics;
```

```
namespace MVC.Controllers
{
    public class HomeController : Controller
    {
        // Declares an optional logger for this controller and assigns
        // it via dependency injection in the constructor.
        // Provides logging information, warnings, errors, and other
        // diagnostic messages within this controller.
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        // This action can be executed in the browser's address bar
        // to return the Index view as entering the URLs
        // "https://localhost:port/Home/Index" or
        // "http://localhost:port/Home/Index".
        // The route of the action is /Home/Index according to the default
        // MVC routing pattern defined in Program.cs file as
        // "{controller=Home}/{action=Index}/{id?}".
        // The default action is defined as Index therefore the Index action
        // of a controller will be executed when action name is not provided
        // such as the route: "/Home" ("https://localhost:port/Home").
        // The default controller is Home therefore the Index action of the
        // Home controller will be executed when both action and controller
        // names are not provided such as the route:
        // "/" ("https://localhost:port").
        // ? means id value is optional. So if provided as the action's
        // parameter as Index(int id), the route will be: "/Home/Index/7"
        // ("https://localhost:port/Home/Index/7") where id parameter value
        // will be 7.
        public IActionResult Index() // Index action without returning a model
                                     // to the Index view.
        {
            return View(); // Right-click and then Go to View to see the
                           // Index.cshtml view under Views/Home folder.

            // The returned view (Views/Home/Index.cshtml) is first searched
            // under Views/Home folder, then Views/Shared folder, and the
            // view is returned if found.
            // Views in the Shared folder can be returned from any controller's
            // any action.
        }



        // Some action result examples:

        /*
        ActionResult inheritance:
        IActionResult : general return type of actions in a controller
        |
        ActionResult: base class that implements IActionResult
        |
        ViewResult (returned by View method) -
        ContentResult (returned by Content method) -
        NotFoundResult (returned by NotFound method) -
        RedirectResults -
        HttpStatusCodeResults -
        JsonResult -
        etc.
        */
```

```
// The route of the action is /Home/View1.

// Way 1:
//public ViewResult View1()
// Way 2: Since ViewResult inherits ActionResult and ActionResult
// implements IActionResult, IActionResult can be used as return type.
public IActionResult View1() // Right-click the action name then click
                             // Add View to create an Empty view named
                             // View1.cshtml under Views/Home folder.        {
    // Way 1:
    //string model = "View1 view returned with this string Model " +
    //    "object from View1 action.";
    // Way 2:
    object model = "View1 view returned with this string Model " +
        "object from View1 action."; // All classes in C# inherit from
                                     // the Object class.


    // Carrying extra data other than the Model to the view:
    // Way 1:
    //ViewData["Text"] = "Extra string text other than the Model " +
    //    "object is carried via ViewData dictionary from View1 action.";
    // Way 2:
    ViewBag.Text = "Extra string text other than the Model " +
        "object is carried via ViewData dictionary from View1 action.";
    // Objects can also be assigned.

    return View(model); // Model object types can be string, int, etc.
                        // also generally complex types such as class.
}

// The route of the action is /Home/View2.
public IActionResult View2()
{
    // Way 1:
    //string model = "View1 view returned with this string Model " +
    //    "object from View2 action.";
    // Way 2: var can be used instead of types if there is an assignment.
    var model = "View1 view returned with this string Model " +
        "object from View2 action.";
    return View("View1", model);
}

// The route of the action is /Home/Content1.
public IActionResult Content1()
{
    return Content("Plain string text content returned from " +
        "Content1 action."); // Optionally text/plain and Encoding.UTF8
                             // parameters (for Turkish character problems)
                             // can be provided.
}

// The route of the action is /Home/Content2.
public IActionResult Content2()
{
    return Content("<b>String HTML content returned from " +
        "Content2 action.</b>", "text/html"); // Optionally Encoding.UTF8
                                              // parameter (for Turkish
                                              // character problems) can be
                                              // provided.
}

// The route of the action is /Home/Redirect1.
public IActionResult Redirect1()
{
    // If a view is returned from an action, extra data may be sent
```

```
            // to the view via ViewData (ViewBag) dictionary.
            // If a redirection to another action occurs in the action,
            // data may be sent to the redirected action then the
            // redirected action's view via TempData dictionary.
            TempData["Message"] = "Message string is carried from Redirect1 action " +
                "to the View1 action, therefore View1 action's returned view " +
                "via TempData dictionary.";

            // Way 1:
            //return RedirectToAction("View1", "Home");
            // Way 2:
            //return RedirectToAction("View1");
            // Way 3:
            return RedirectToAction(nameof(View1)); // nameof returns the string
                                                    // representation of the View1
                                                    // method and can also be used for
                                                    // classes, class properties, etc.
        }

        // The route of the action is /Home/NotFound1.
        public IActionResult NotFound1()
        {
            // Logs a warning message to the Kestrel Console or Visual Studio Output
            // window. LogError, LogInformation, LogDebug, LogCritical and LogTrace
            // methods can also be used. Logging is optional, therefore ILogger
            // instance doesn't need to be injected in every controller.
            _logger.LogWarning("Home controller NotFound1 action is executed.");

            return NotFound("Message string returned from the NotFound1 action.");
            // Returns HTTP 404 Not Found status code, parameter is optional.
        }



        // The route of the action is /Home/Privacy.
        public IActionResult Privacy()
        {
            return View(); // Will return the Privacy.cshtml view under
                           // Views/Home folder.
        }

        // This action is used for displaying error information of the operations.
        // ResponseCache is an attribute used for disabling response caching for this
        // action so that every request gets a fresh result.
        // Attributes gain new features to the methods, properties, classes, etc.
        // they are applied to.
        // The route of the action is /Home/Error.
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
            NoStore = true)]
        public IActionResult Error() // Will return the Error.cshtml view under
                                     // Views/Shared folder sending the Model object
                                     // as type ErrorViewModel class.
        {
            return View(new ErrorViewModel
            {
                // Set RequestId to the current activity's ID if available, otherwise
                // uses the HTTP request's unique trace identifier.
                RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier
            });
        }
    }
}
```

3. You may also add the View1.cshtml view returned by the View1 action of the Home controller by right-clicking the action name then clicking add View to create an Empty view named View1.cshtml under Views/Home folder as below:

```
@*
    Model object type declaration according to the type
    returned by the action (View1).
*@
@model string

@*
    This is a Razor comment.
    Razor syntax provides the usage of HTML and C# together
    in views.

    In order to switch to C# in HTML, @ is used.

    C# code blocks can be written in @{ ... }, for example
    @{
        var name = "Cagil Alsac";
    }
    and the value of the name variable can be printed in
    HTML like
    <h1>@name</h1>

    Other C# code blocks such as if, foreach, etc. can be
    written like
    @if (true)
    {
        <label>Success.</label>
    }
    else
    {
        <label>Error!</label>
    }

    @foreach (string item in list)
    {
        <p>@item</p>
    }
*@

<p>
    @*
        Model object of type string to print the assigned and
        returned value in the action (View1).
    *@
    @Model
</p>

@*
    ViewData and ViewBag are the same collection (dictionary).
    They carry extra data other than the model object from a
    controller action to its view, or between views.
    ViewData["Text"] can also be used instead of ViewBag.Text.
*@
@if (ViewBag.Text is not null) // If Text key of the ViewData
                               // dictionary has a value, print
                               // the value in HTML.
{
```

```
    <hr />
    <p>@ViewBag.Text</p>
}

@*
    TempData carries data from an action to the redirected action,
    therefore to the redirected action's view.
*@
@if (TempData["Message"] is not null) // If Message key of the
                                      // TempData dictionary has
                                      // a value, print the value
                                      // in HTML.
{
    <hr />
    <p>@TempData["Message"]</p>
}
```

4. You need to modify the Index.cshtml view under Views/Home folder according to your project as below:

```
@{
    /*
    ViewData and ViewBag are the same collection (dictionary).
    They carry extra data other than the model object from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views/Shared/_Layout.cshtml view's title tag
    of the head tag.
    */
    // Way 1:
    // ViewData["Title"] = "Home Page";
    // Way 2:
    ViewBag.Title = "Home Page";
}

<div class="text-center">
    @* Change Users MVC to your project name. *@
    <h1 class="display-4">Welcome to Users MVC</h1>
</div>
```

5. You also need to modify the Privacy.cshtml view under Views/Home folder according to your project as below:

```
@{
    /*
    ViewData and ViewBag are the same collection (dictionary).
    They carry extra data other than the model object from
    a controller action to its view, or between views.
    The value assigned will be shown in
    Views/Shared/_Layout.cshtml view's title tag of the head tag.
    */
    ViewData["Title"] = "Privacy Policy";
}
```

```
@*
    Will print the "Privacy Policy" value assigned to the Title
    key of the ViewData dictionary above.
*@
<h1>@ViewData["Title"]</h1>

@*
    The date and time value will be retrieved from the server
    when DateTime.Now is executed and the Year part of the DateTime
    struct will be printed in the paragraph tag.
*@
<p>&copy; @DateTime.Now.Year - Cagil Alsac</p>
```

6. Download the Scaffolding Templates that will generate the code for the controllers and views automatically from:
   https://need4code.com/DotNet/Home/Index?path=.NET%5C00_Files%5CScaffolding%20Templates%5CTemplates.7z

   Extract the Templates folder in the compressed file to your MVC Project.

   Right-click the Templates folder then click Exclude From Project for the template codes not being compiled and published.

   If you want to see the excluded folders or files of your project, you can click the fifth icon from left in top of the Solution Explorer with description Show All Files.

   The excluded files or folders are seen as dashed points in Solution Explorer.

   If you want to include a folder or file in your project, right-click the file or folder with dashed points then click Include In Project, therefore the codes will be compiled and published.

   You don't have to use these templates, however if you choose not to, you need to write or modify the dependency injections and actions with views.

   Note: If you get any exceptions during scaffolding, create the DbFactory class in the APP/Domain folder.

   Note: For Rider run:
   "dotnet aspnet-codegenerator controller -name {ModelName}Controller -m {Namespace}.{ModelName} -dc {Namespace}.{DbContextName} --relativeFolderPath Controllers"
   in the terminal for scaffolding with the templates under the Templates folder.

7. Right-click the Controllers folder then Add -> Controller -> Common -> MVC -> MVC Controller with views, using Entity Framework and select Role entity as Model class, select Db as DbContext class, check Generate views, do not check Reference script libraries, check Use a layout page (leave the text box below empty), and give the name RolesController as Controller name. Then modify the controller and views if necessary.

   Note:

   `@using APP.Models`

   must be added in the _ViewImports.cshtml file in the Views folder as below:

Views_ViewImports.cshtml:

```
@using MVC
@using MVC.Models

@using APP.Models

@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Controllers\RolesController.cs:

```
#nullable disable
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using CORE.Services;
using APP.Models;

// Generated from Custom MVC Template.

namespace MVC.Controllers
{
    public class RolesController : Controller
    {
        // Service injections:
        private readonly IService<RoleRequest, RoleResponse> _roleService;

        /*
            Can be uncommented and used for many to many relationships,
            "entity" may be replaced with the related entity name in
            the controller and views.
        */
        //private readonly IService<EntityRequest, EntityResponse> _EntityService;

        public RolesController(
                    IService<RoleRequest, RoleResponse> roleService

            /*
                Can be uncommented and used for many to many relationships,
                "entity" may be replaced with the related entity name in
                the controller and views.
            */
```

```
    //, IService<EntityRequest, EntityResponse> EntityService
)
{
    _roleService = roleService;

    /*
        Can be uncommented and used for many to many relationships,
        "entity" may be replaced with the related entity name in
        the controller and views.
    */
    //_EntityService = EntityService;
}

private void SetViewData()
{
    /*
        ViewBag and ViewData are the same collection (dictionary).
        They carry extra data other than the model from a controller
        action to its view, or between views.
    */

    // Related items service logic to set ViewData (Id and
    // Name parameters may need to be changed in the SelectList
    // constructor according to the model):

    /*
        Can be uncommented and used for many to many relationships,
        "entity" may be replaced with the related entity name in
        the controller and views.
    */
    //ViewBag.EntityIds = new MultiSelectList(_EntityService.List(),
    //  "Id", "Name");
}

private void SetTempData(string message, string key = "Message")
{
    /*
        TempData is used to carry extra data to the redirected
        controller action's view.
    */

    TempData[key] = message;
}

// GET: Roles
public IActionResult Index()
{
    // Get collection service logic:
    var list = _roleService.List();
    return View(list);
    // return response collection as model to the Index view
}

// GET: Roles/Details/5
public IActionResult Details(int id)
{
    // Get item service logic:
    var item = _roleService.Single(id);
    return View(item);
    // return response item as model to the Details view
}

// GET: Roles/Create
public IActionResult Create()
{
    SetViewData();
```

```
    // set ViewData dictionary to carry extra data other
    // than the model to the view
    return View();
    // return Create view with no model
}

// POST: Roles/Create
[HttpPost, ValidateAntiForgeryToken]
public IActionResult Create(RoleRequest role)
{
    if (ModelState.IsValid)
    // check data annotation validation errors in the request
    {
        // Insert item service logic:
        var response = _roleService.Add(role);
        if (response.IsSuccessful)
        {
            SetTempData(response.Message);
            // set TempData dictionary to carry the message
            // to the redirected action's view
            return RedirectToAction(nameof(Details),
                new { id = response.Id });
            // redirect to Details action with id parameter
            // as response.Id route value
        }
        ModelState.AddModelError("", response.Message);
        // to display service error message in the validation
        // summary of the view
    }
    SetViewData();
    // set ViewData dictionary to carry extra data other
    // than the model to the view
    return View(role);
    // return request as model to the Create view
}

// GET: Roles/Edit/5
public IActionResult Edit(int id)
{
    // Get item to edit service logic:
    var item = _roleService.Edit(id);
    SetViewData();
    // set ViewData dictionary to carry extra data other
    // than the model to the view
    return View(item);
    // return request as model to the Edit view
}

// POST: Roles/Edit
[HttpPost, ValidateAntiForgeryToken]
public IActionResult Edit(RoleRequest role)
{
    if (ModelState.IsValid)
    // check data annotation validation errors in the request
    {
        // Update item service logic:
        var response = _roleService.Update(role);
        if (response.IsSuccessful)
        {
            SetTempData(response.Message);
            // set TempData dictionary to carry the message
            // to the redirected action's view
            return RedirectToAction(nameof(Details),
                new { id = response.Id });
            // redirect to Details action with id parameter
            // as response.Id route value
```

```
                }
                ModelState.AddModelError("", response.Message);
                // to display service error message in the validation
                // summary of the view
            }
            SetViewData();
            // set ViewData dictionary to carry extra data other
            // than the model to the view
            return View(role);
            // return request as model to the Edit view
        }

        // GET: Roles/Delete/5
        public IActionResult Delete(int id)
        {
            // Get item to delete service logic:
            var item = _roleService.Single(id);
            return View(item);
            // return response item as model to the Delete view
        }

        // POST: Roles/Delete
        [HttpPost, ValidateAntiForgeryToken, ActionName("Delete")]
        public IActionResult DeleteConfirmed(int id)
        {
            // Delete item service logic:
            var response = _roleService.Remove(id);
            SetTempData(response.Message);
            // set TempData dictionary to carry the message
            // to the redirected action's view
            return RedirectToAction(nameof(Index));
            // redirect to the Index action
        }
    }
}
```

Views\Roles\Index.cshtml:

```
@model IEnumerable<RoleResponse>

@* Generated from Custom MVC Template. *@
@*
    Model namespace using directive should be added to
    _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "Role List";
}
```

```
<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>

@if (Model is not null)
{
    <form asp-action="Index" autocomplete="off">
        <div class="@containerDivClass">
            <div class="row">
                <div class="col-12 text-danger">
                    @*
                    TempData is used to carry extra data to the
                    redirected controller action's view.
                    *@
                    @TempData["Message"]
                </div>
            </div>
            <div class="row mb-3">
                <div class="col-10 text-success">
                    @Model.Count() record(s) found.
                </div>
                <div class="col-2 text-end">
                    <a asp-action="Create">Create</a>
                </div>
            </div>
            <table class="table table-striped table-hover">
                <thead class="table-secondary">
                    <tr>
                        <th>
                            @Html.DisplayNameFor(model => model.Name)
                        </th>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var item in Model) {
                            <tr>
                                <td>
                                    @Html.DisplayFor(modelItem => item.Name)
                                </td>
                                <td class="text-end w-25">
                                    <a asp-action="Details"
                                        asp-route-id="@item.Id">
                                            Details
                                    </a> | 
                                    <a asp-action="Edit"
                                        asp-route-id="@item.Id">
                                            Edit
                                    </a> | 
                                    <a asp-action="Delete"
                                        asp-route-id="@item.Id">
                                            Delete
                                    </a>
                                </td>
                            </tr>
                    }
                </tbody>
            </table>
        </div>
    </form>
}
```

Views\Roles\Details.cshtml:

```
@model RoleResponse

@* Generated from Custom MVC Template. *@
@*
    Model namespace using directive should be added to
    _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "Role Details";
}

<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>

@if (Model is not null)
{
<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
    /*
    TempData is used to carry extra data to the redirected
    controller action's view.
    */
    <p class="text-danger">
        @TempData["Message"]
    </p>
    }
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            @Html.DisplayNameFor(model => model.Name)
        </div>
        <div class="col-10">
            @Html.DisplayFor(model => model.Name)
        </div>
    </div>


@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in the
    controller and views.
    Raw HTML Helper is used in case string parameter may have
    HTML tags.
*@
```

```
    @*
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            <b>@Html.DisplayNameFor(model => model.Entitys)</b>
        </div>
        <div class="col-10">
            @Html.Raw(Model.Entitys)
        </div>
    </div>
    *@

    <hr />
    <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> | 
    <a asp-action="Delete" asp-route-id="@Model.Id">Delete</a> | 
    <a asp-action="Index">Back to List</a>
</div>
}
```

Views\Roles\Create.cshtml:

```
@model RoleRequest

@* Generated from Custom MVC Template. *@
@*
    Model namespace using directive should be added to
    _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
    var dateTimePickerClass = "datetimepicker";
    // "jquery-datetimepicker" must be added as client side library
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "Role Create";
}

<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>

<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
    /*
    TempData is used to carry extra data to the redirected
    controller action's view.
    */
    <p class="text-danger">
        @TempData["Message"]
    </p>
    }
```

```
    <form asp-action="Create" autocomplete="off" enctype="multipart/form-data">
        @Html.AntiForgeryToken()
        <div asp-validation-summary="All" class="text-danger"></div>
        @*
        Use ModelOnly instead of All to see only service response messages
        in validation summary
        *@
        <div class="row mb-3">
             <label asp-for="Name" class="col-2 col-form-label fw-bold"></label>
             <div class="col-10">
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name" class="text-danger"></span>
             </div>
        </div>
@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in the
    controller and views.
*@
        @*
        <div class="row mb-3">
             <label asp-for="EntityIds" class="col-2 col-form-label fw-bold"></label>
             <div class="col-10">
                    <select multiple asp-for="EntityIds" class="form-select"
                          asp-items="ViewBag.EntityIds"></select>
                    <span asp-validation-for="EntityIds" class="text-danger"></span>
             </div>
        </div>
        *@

        <hr />
        <div class="row mb-3">
            <div class="offset-2 col-10">
                <button type="submit" class="btn btn-primary">Save</button>
                  
                <button type="reset" class="btn btn-outline-primary">Reset</button>
                  
                <a asp-action="Index">Back to List</a>
            </div>
        </div>
    </form>
</div>

@section Scripts {
    @*
        Can be uncommented to use client-side validation using jQuery
        instead of server-side validation.
    *@
    <partial name="_ValidationScriptsPartial" />
}
```

Views\Roles\Edit.cshtml:

```
@model RoleRequest

@* Generated from Custom MVC Template. *@
@*
    Model namespace using directive should be added to
    _ViewImports.cshtml.
*@
```

```
@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
    var dateTimePickerClass = "datetimepicker";
    // "jquery-datetimepicker" must be added as client side library
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "Role Edit";
}
<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>
@if (Model is not null)
{
<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
    /*
    TempData is used to carry extra data to the redirected
    controller action's view.
    */
    <p class="text-danger">
        @TempData["Message"]
    </p>
    }
    <form asp-action="Edit" autocomplete="off" enctype="multipart/form-data">
        @Html.AntiForgeryToken()
        <div asp-validation-summary="All" class="text-danger"></div>
        @*
        Use ModelOnly instead of All to see only service response messages
        in validation summary
        *@
        <div class="row mb-3">
             <label asp-for="Name" class="col-2 col-form-label fw-bold"></label>
             <div class="col-10">
                  <input asp-for="Name" class="form-control" />
                 <span asp-validation-for="Name" class="text-danger"></span>
              </div>
        </div>
        <input type="hidden" asp-for="Id" />

@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in the
    controller and views.
*@
        @*
        <div class="row mb-3">
              <label asp-for="EntityIds" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <select multiple asp-for="EntityIds" class="form-select"
                          asp-items="ViewBag.EntityIds"></select>
                    <span asp-validation-for="EntityIds" class="text-danger"></span>
              </div>
        </div>
        *@
```

```
        <hr />
        <div class="row mb-3">
            <div class="offset-2 col-10">
                <button type="submit" class="btn btn-primary">Save</button>
                  
                <button type="reset" class="btn btn-outline-primary">Reset</button>
                  
                <a asp-action="Delete" asp-route-id="@Model.Id">Delete</a>
                  
                <a asp-action="Index">Back to List</a>
            </div>
        </div>
    </form>
</div>
}


@section Scripts {
    @*
        Can be uncommented to use client-side validation using jQuery
        instead of server-side validation.
    *@
    <partial name="_ValidationScriptsPartial" />
}
```

Note: Client-side validation through jQuery instead of Server-side validation is enabled at the bottom in Create and Edit views.

Views\Roles\Delete.cshtml:

```
@model RoleResponse

@* Generated from Custom MVC Template. *@
@*
    Model namespace using directive should be added to
    _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "Role Delete";
}

<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>
```

```
@if (Model is not null)
{
<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
    /*
    TempData is used to carry extra data to the redirected
    controller action's view.
    */
    <p class="text-danger">
        @TempData["Message"]
    </p>
    }
    <h3 class="text-danger">Are you sure you want to delete?</h3>
    <hr />
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            @Html.DisplayNameFor(model => model.Name)
        </div>
        <div class="col-10">
            @Html.DisplayFor(model => model.Name)
        </div>
    </div>

@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in the
    controller and views.
    Raw HTML Helper is used in case string parameter may have
    HTML tags.
*@
    @*
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            <b>@Html.DisplayNameFor(model => model.Entitys)</b>
        </div>
        <div class="col-10">
            @Html.Raw(Model.Entitys)
        </div>
    </div>
    *@

    <hr />
    <form asp-action="Delete">
        @Html.AntiForgeryToken()
        <input type="hidden" asp-for="Id" />
        <input type="submit" value="Yes" class="btn btn-danger" />  
        <a asp-action="Index" class="btn btn-outline-danger">No</a>
    </form>
</div>
}
```

8. In Views/Shared/_Layout.cshtml, change the title HTML tag in the head HTML tag from

   ```
   <title>@ViewData["Title"] - MVC</title>
   ```
   to

   ```
   @* Change Users MVC to your project name. *@
   <title>@ViewData["Title"] - Users MVC</title>
   ```

   Then in the Application Links section, change

```
@* Application Links: *@
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
        asp-controller="Home" asp-action="Index">Home</a>
</li>
```

to

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
        asp-controller="Roles" asp-action="Index">Roles</a>
</li>
```

You can remove the Privacy link in the Application Links section.

Also add Statuses and Users links under the Roles link as below:

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
       asp-controller="Statuses" asp-action="Index">Statuses</a>
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
       asp-controller="Users" asp-action="Index">Users</a>
</li>
```

Your _Layout.cshtml should be like below:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    @* @ViewData["Title"] is set in views to show the title string values. *@
    @* Change Users MVC to your project name. *@
    <title>@ViewData["Title"] - Users MVC</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/MVC.styles.css" asp-append-version="true" />
</head>
<body>
    <header>

        @* Bootstrap Navbar: top menu *@
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light
            bg-white border-bottom box-shadow mb-3">

            @*
            Bootstrap container-fluid class is used for a responsive full-width
            (without any padding from left and right) div.
            Bootstrap container class is used for a responsive fixed-width
            (with some padding from left and right) div.
            *@
```

```
<div class="container-fluid">
    @*
    Tag Helper: Tag Helpers enable server-side code to participate
    in creating and rendering HTML elements in Razor views.
    They look like standard HTML tags but add special attributes
    such as asp-controller, asp-action, asp-area, asp-route, asp-for,
    etc. that are processed on the server to generate
    dynamic content.
    Will create the HTML link:
    <a href="/Home/Index" class="navbar-brand">Users MVC</a>
    *@
    @* Change Users MVC to your project name. *@
    <a class="navbar-brand" asp-area="" asp-controller="Home"
        asp-action="Index">Users MVC</a>
    <button class="navbar-toggler" type="button"
        data-bs-toggle="collapse"
        data-bs-target=".navbar-collapse"
        aria-controls="navbarSupportedContent" aria-expanded="false"
        aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="navbar-collapse collapse d-sm-inline-flex
        justify-content-between">
        <ul class="navbar-nav flex-grow-1">

            @*
            Links directing to Home controller example actions:
            Must not exist in your projects.
            *@
            <li class="nav-item">
                <a class="nav-link text-dark"
                    asp-area="" asp-controller="Home"
                    asp-action="View1">View1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark"
                    asp-area="" asp-controller="Home"
                    asp-action="View2">View2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" target="_blank"
                    asp-area="" asp-controller="Home"
                    asp-action="Content1">Content1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" target="_blank"
                    asp-area="" asp-controller="Home"
                    asp-action="Content2">Content2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark"
                    asp-area="" asp-controller="Home"
                    asp-action="Redirect1">Redirect1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" target="_blank"
                    asp-area="" asp-controller="Home"
                    asp-action="NotFound1">NotFound1</a>
            </li>

            @* Application Links: *@
            <li class="nav-item">
```

```
                        <a class="nav-link text-dark"
                            asp-area="" asp-controller="Roles"
                            asp-action="Index">Roles</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark"
                            asp-area=""  asp-controller="Statuses"
                            asp-action="Index">Statuses</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark"
                            asp-area="" asp-controller="Users"
                            asp-action="Index">Users</a>
                    </li>

                </ul>
            </div>
        </div>
    </nav>
</header>

@*
    class changed from container to container-fluid to use
    full-width for the views.
*@
<div class="container-fluid">
    <main role="main" class="pb-3">

        @*
            Renders the views' content returned by
            controllers' actions.
        *@
        @RenderBody()

    </main>
</div>

<footer class="border-top footer text-muted">

    @*
        class changed from container to container-fluid to use
        full-width for the footer.
    *@
    <div class="container-fluid">
        <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@*
    Renders the optional Scripts section in the views which contains
    extra Javascript or jQuery codes, generally for using third-party
    Javascript-CSS client side libraries.
    Since the required parmeter is false, there must not be a Scripts
    secion in every view.
*@
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

9. Right-click the Controllers folder then Add -> Controller -> Common -> MVC -> MVC Controller with views, using Entity Framework and select Status entity as Model class, select Db as DbContext class, check Generate views, do not check Reference script libraries, check Use a layout page (leave the text box below empty), and give the name StatusesController as Controller name. Then modify the controller and views if necessary.

Views\Statuses\Index.cshtml:

Add Response Property section for the table header HTML tag:

```
@* Response Property: *@
<th>
    @Html.Label("UserCount", "User Count")
</th>
@* DisplayNameFor may also be used. *@
```

Add Response Property section for the table data HTML tag:

```
@* Response Property: *@
@*
    Way 1: Displaying mapped primitive type
    (UserCountR) data for basic information.
*@
@* <td>
    @Html.DisplayFor(modelItem => item.UserCountR)
</td> *@
@*
    Way 2: Displaying mapped complex type object
    (UsersR) data for more information.
*@
<td>
    @Html.DisplayFor(modelItem => item.UsersR.Count)
</td>
```

Note: Only primitive type mapped response properties usage (Way 1) in all views is enough for your projects.

Views\Statuses\Details.cshtml:

Add Response Property section for the div HTML tag with class row:

```
@* Response Property: *@
@*
    Way 1: Displaying mapped primitive type
    (UserCountR and UserNamesR) data for basic information.
*@
@* <div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.UserCountR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.UserCountR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.UserNamesR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.UserNamesR)
    </div>
</div> *@
@*
    Way 2: Displaying mapped complex type object
    (UsersR) data for more information.
*@
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.Label("UserCount", "User Count")
        @* DisplayNameFor may also be used. *@
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.UsersR.Count)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.Label("Users", "Users")
        @* DisplayNameFor may also be used. *@
    </div>
    <div class="col-10">
        @{
            var userNames = "";
            if (Model.UsersR.Count > 0)
            {
                foreach (var user in Model.UsersR)
                {
                    userNames += user.UserName + ", ";
                }
                userNames = userNames.Substring(0, userNames.Length - 2);
            }
        }
        @userNames
    </div>
</div>
```

Views\Statuses\Create.cshtml and Views\Statuses\Edit.cshtml:

Client side validation can be activated by uncommenting

```
@*<partial name="_ValidationScriptsPartial" />*@
```

in the Scripts section at the bottom of Create and Edit views.

Views\Statuses\Delete.cshtml:

Response property (properties ending with R) may be added like in the Details view, however only a unique identifier property such as Name, Title, etc. is enough to show the record information to be deleted.

10. Right-click the Controllers folder then Add -> Controller -> Common -> MVC -> MVC Controller with views, using Entity Framework and select User entity as Model class, select Db as DbContext class, check Generate views, do not check Reference script libraries, check Use a layout page (leave the text box below empty), and give the name UsersController as Controller name. Then modify the controller and views if necessary.

    Uncomment the lines marked with “Can be uncommented and used for many to many relationships...” and replace EntityRequest with RoleRequest, EntityResponse with RoleResponse, EntityService with roleService and EntityIds with RoleIds as below:

```
/*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in
    the controller and views.
*/
private readonly IService<RoleRequest, RoleResponse> _roleService;

public UsersController(
      IService<UserRequest, UserResponse> userService
    , IService<StatusRequest, StatusResponse> statusService

    /*
        Can be uncommented and used for many to many relationships,
        "entity" may be replaced with the related entity name
        in the controller and views.
    */
    , IService<RoleRequest, RoleResponse> roleService
)
```

```
{
    _userService = userService;
    _statusService = statusService;

    /*
        Can be uncommented and used for many to many relationships,
        "entity" may be replaced with the related entity name
        in the controller and views.
    */
    _roleService = roleService;
}

private void SetViewData()
{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a controller
    action to its view, or between views.
    */

    // Related items service logic to set ViewData (Id and Name
    // parameters may need to be changed in the SelectList constructor
    // according to the model):
    ViewData["StatusId"] = new SelectList(_statusService.List(),
        "Id", "Title"); // Second parameter changed from Name to Title.

    /*
        Can be uncommented and used for many to many relationships,
        "entity" may be replaced with the related entity name
        in the controller and views.
    */
    ViewBag.RoleIds = new MultiSelectList(_roleService.List(),
        "Id", "Name");
}
```

Views\Users\Index.cshtml:

Assign containerDivClass to “container-fluid” for full width:

```
var containerDivClass = "container-fluid";
// changed from "container" to "container-fluid" for full width
```

Add or modify Response Properties sections for the table head and table data HTML tags:

```
<table class="table table-striped table-hover">
    <thead class="table-secondary">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.UserName)
            </th>
```

```
        @* Response Properties: *@
        <th>
            @Html.DisplayNameFor(model => model.PasswordR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.FullNameR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.GenderR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.BirthDateR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.RegistrationDateR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.ScoreR)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.IsOnlineR)
        </th>

        <th>
            @Html.Label("Status", "Status")
            @* DisplayNameFor may also be used. *@
        </th>

        <th>
            @Html.Label("Roles", "Roles")
            @* DisplayNameFor may also be used. *@
        </th>

        <th></th>
    </tr>
</thead>

<tbody>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(model => item.UserName)
            </td>

            @* Response Properties: *@
            <td>
                @Html.DisplayFor(model => item.PasswordR)
            </td>
            <td>
                @Html.DisplayFor(model => item.FullNameR)
            </td>
            <td>
                @Html.DisplayFor(model => item.GenderR)
            </td>
            <td>
                @Html.DisplayFor(model => item.BirthDateR)
            </td>
            <td>
                @Html.DisplayFor(model => item.RegistrationDateR)
            </td>
            <td>
                @Html.DisplayFor(model => item.ScoreR)
```

```
                    </td>
                    <td>
                        @Html.DisplayFor(model => item.IsOnlineR)
                    </td>



                    @*
                        Way 1: Displaying mapped primitive type
                        (StatusTitleR) data for basic information.
                    *@
                    <td>
                        @Html.DisplayFor(model => item.StatusTitleR)
                    </td>
                    @*
                        Way 2: Displaying mapped complex type object
                        (StatusR) data for more information.
                    *@
                    @* <td>
                        @Html.DisplayFor(model => item.StatusR.Title)
                    </td> *@

                    @*
                        Way 1: Displaying mapped primitive type
                        (RoleNamesR) data for basic information.
                    *@
                    <td>
                        @Html.Raw(item.RoleNamesR)
                        <!--
                            Raw HTML Helper is used if there are any
                            HTML containing value assigned to the property.
                            RoleNamesR value contains the "<br>" tag.
                        -->
                    </td>
                    @*
                        Way 2: Displaying mapped complex type object
                        (RolesR) data for more information.
                    *@
                    @* <td>
                        @string.Join("<br>", item.RolesR
                            .OrderBy(role => role.Name)
                            .Select(role => role.Name))
                    </td> *@

                    <td class="text-end w-25">
                             <a asp-action="Details"
                                   asp-route-id="@item.Id">Details</a>
                              | 
                             <a asp-action="Edit"
                                   asp-route-id="@item.Id">Edit</a>
                              | 
                             <a asp-action="Delete"
                                   asp-route-id="@item.Id">Delete</a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
```

Views\Users\Details.cshtml:

Add or modify Response Properties section for the div HTML tags with class row. You can also uncomment and modify the commented code block starting with “Can be uncommented and used for many to many relationships...” for role names seperated with the
 HTML tag to be displayed.

```
@* Response Properties: *@
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.PasswordR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.PasswordR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.FullNameR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.FullNameR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.GenderR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.GenderR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.BirthDateR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.BirthDateR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.RegistrationDateR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.RegistrationDateR)
    </div>
</div>
<div class="row mb-3">
    <div class="col-2 fw-bold">
        @Html.DisplayNameFor(model => model.ScoreR)
    </div>
    <div class="col-10">
        @Html.DisplayFor(model => model.ScoreR)
    </div>
</div>
```

```
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            @Html.DisplayNameFor(model => model.IsOnlineR)
        </div>
        <div class="col-10">
            @Html.DisplayFor(model => model.IsOnlineR)
        </div>
    </div>
    <div class="row mb-3">
        <div class="col-2 fw-bold">
            @Html.DisplayNameFor(model => model.StatusTitleR)
        </div>
        <div class="col-10">
            @Html.DisplayFor(model => model.StatusTitleR)
        </div>
    </div>

@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name in
    the controller and views.
    Raw HTML Helper is used in case string parameter may have HTML tags.
*@

    <div class="row mb-3">
        <div class="col-2 fw-bold">
            <b>@Html.DisplayNameFor(model => model.RoleNamesR)</b>
        </div>
        <div class="col-10">
            @Html.Raw(Model.RoleNamesR)
        </div>
    </div>
```

Views\Users\Create.cshtml:

Add or modify the request properties of the form as below. You can also uncomment and modify the commented code block starting with “Can be uncommented and used for many to many relationships...” for role ID list to be selected:

```
<form asp-action="Create" autocomplete="off" enctype="multipart/form-data">
       @Html.AntiForgeryToken()
       <div asp-validation-summary="All" class="text-danger"></div>
       @*
            Use ModelOnly instead of All to see only service
            response messages in validation summary
       *@
       <div class="row mb-3">
             <label asp-for="UserName" class="col-2 col-form-label fw-bold"></label>
             <div class="col-10">
                   <input asp-for="UserName" class="form-control" />
                   <span asp-validation-for="UserName" class="text-danger"></span>
             </div>
        </div>
        <div class="row mb-3">
             <label asp-for="Password" class="col-2 col-form-label fw-bold"></label>
             <div class="col-10">
                   <input asp-for="Password" class="form-control" type="password" />
                   @* type HTML attribute added to hide the password. *@
```

```
            <span asp-validation-for="Password" class="text-danger"></span>
      </div>
</div>
<div class="row mb-3">
      <label asp-for="FirstName" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <input asp-for="FirstName" class="form-control" />
            <span asp-validation-for="FirstName" class="text-danger"></span>
      </div>
</div>
<div class="row mb-3">
      <label asp-for="LastName" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <input asp-for="LastName" class="form-control" />
            <span asp-validation-for="LastName" class="text-danger"></span>
      </div>
</div>
<div class="row mb-3">
      <label asp-for="Gender" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <select asp-for="Gender" class="form-select">
                  <option value="@((int)Genders.Woman)">
                        @Genders.Woman
                  </option>
                  <option value="@((int)Genders.Man)">
                        @Genders.Man
                  </option>
                  @*
                  Updated by APP.Domain.Genders enum elements.
                  APP.Domain namespace added on top.
                  *@
            </select>
            <span asp-validation-for="Gender" class="text-danger"></span>
      </div>
</div>
<div class="row mb-3">
      <label asp-for="BirthDate" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <input asp-for="BirthDate"
                  class="form-control @dateTimePickerClass" type="text" />
            <span asp-validation-for="BirthDate" class="text-danger"></span>
      </div>
</div>
@*
    RegistrationDate request property removed since
    it is assigned automatically in the UserService
    Add method.
*@
<div class="row mb-3">
      <label asp-for="Score" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <input asp-for="Score" class="form-control" />
            <span asp-validation-for="Score" class="text-danger"></span>
      </div>
</div>
@*
    IsOnline request property removed since
    it will be assigned automatically in the
    log in and log out operations.
*@
<div class="row mb-3">
      <label asp-for="StatusId" class="col-2 col-form-label fw-bold"></label>
      <div class="col-10">
            <select asp-for="StatusId" class="form-select"
                  asp-items="ViewBag.StatusId">
                        <option value="">-- Select --</option>
```

```
                    </select>
                    <span asp-validation-for="StatusId" class="text-danger"></span>
                </div>
            </div>


@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name
    in the controller and views.
*@

            <div class="row mb-3">
                <label asp-for="RoleIds" class="col-2 col-form-label fw-bold"></label>
                <div class="col-10">
                    <select multiple asp-for="RoleIds" class="form-select"
                            asp-items="ViewBag.RoleIds"></select>
                    <span asp-validation-for="RoleIds" class="text-danger"></span>
                </div>
            </div>


            <hr />
            <div class="row mb-3">
                <div class="offset-2 col-10">
                    <button type="submit" class="btn btn-primary">Save</button>
                      
                    <button type="reset" class="btn btn-outline-primary">
                        Reset
                    </button>
                      
                    <a asp-action="Index">Back to List</a>
                </div>
            </div>
</form>
```

Note: In order to use the calendar in Create and Edit views, right-click the lib folder in the wwwroot folder, then click
Add -> Client-Side Library, select "cdnjs" as the Provider, type "jquery-datetimepicker" in the Library text box,
select the latest version, check Include all library files, check "wwwroot/lib/" as Target location and click Install.

Note: Optionally you can also install the "select2" library to provide better usage for drop down lists and list boxes (select tags): https://select2.org

You need to add the following code which will convert all select tags into select2 in the Scripts section:

```
<link href="~/lib/select2/css/select2.min.css" rel="stylesheet" />
<script src="~/lib/select2/js/select2.min.js"></script>
<script>
    $(function() {
        $("select").select2();
    });
</script>
```

Views\Users\Edit.cshtml:

Add or modify the request properties of the form as below. You can also uncomment and modify the commented code block starting with “Can be uncommented and used for many to many relationships...” for role ID list to be selected:

```
<form asp-action="Create" autocomplete="off" enctype="multipart/form-data">
        @Html.AntiForgeryToken()
        <div asp-validation-summary="All" class="text-danger"></div>
        @*
             Use ModelOnly instead of All to see only service
             response messages in validation summary
        *@
        <div class="row mb-3">
              <label asp-for="UserName" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <input asp-for="UserName" class="form-control" />
                    <span asp-validation-for="UserName" class="text-danger"></span>
              </div>
         </div>
         <div class="row mb-3">
              <label asp-for="Password" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <input asp-for="Password" class="form-control" />
                    <span asp-validation-for="Password" class="text-danger"></span>
              </div>
         </div>
         <div class="row mb-3">
              <label asp-for="FirstName" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <input asp-for="FirstName" class="form-control" />
                    <span asp-validation-for="FirstName" class="text-danger"></span>
              </div>
         </div>
         <div class="row mb-3">
              <label asp-for="LastName" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <input asp-for="LastName" class="form-control" />
                    <span asp-validation-for="LastName" class="text-danger"></span>
              </div>
         </div>
         <div class="row mb-3">
              <label asp-for="Gender" class="col-2 col-form-label fw-bold"></label>
              <div class="col-10">
                    <select asp-for="Gender" class="form-select">
                          <option value="@((int)Genders.Woman)">
                                @Genders.Woman
                          </option>
                          <option value="@((int)Genders.Man)">
                                @Genders.Man
                          </option>
                          @*
                          Updated by APP.Domain.Genders enum elements.
                          APP.Domain namespace added on top.
                          *@
                    </select>
                    <span asp-validation-for="Gender" class="text-danger"></span>
              </div>
         </div>
         <div class="row mb-3">
              <label asp-for="BirthDate" class="col-2 col-form-label fw-bold"></label>
```

```
            <div class="col-10">
                <input asp-for="BirthDate"
                       class="form-control @dateTimePickerClass" type="text" />
                <span asp-validation-for="BirthDate" class="text-danger"></span>
            </div>
        </div>
        @*
            RegistrationDate request property removed since
            it is assigned automatically in the UserService
            Add method.
        *@
        <div class="row mb-3">
            <label asp-for="Score" class="col-2 col-form-label fw-bold"></label>
            <div class="col-10">
                <input asp-for="Score" class="form-control" />
                <span asp-validation-for="Score" class="text-danger"></span>
            </div>
        </div>
        @*
            IsOnline request property removed since
            it will be assigned automatically in the
            log in and log out operations.
        *@
        <div class="row mb-3">
            <label asp-for="StatusId" class="col-2 col-form-label fw-bold"></label>
            <div class="col-10">
                <select asp-for="StatusId" class="form-select"
                        asp-items="ViewBag.StatusId">
                            <option value="">-- Select --</option>
                </select>
                <span asp-validation-for="StatusId" class="text-danger"></span>
            </div>
        </div>


@*
    Can be uncommented and used for many to many relationships,
    "entity" may be replaced with the related entity name
    in the controller and views.
*@

        <div class="row mb-3">
            <label asp-for="RoleIds" class="col-2 col-form-label fw-bold"></label>
            <div class="col-10">
                <select multiple asp-for="RoleIds" class="form-select"
                        asp-items="ViewBag.RoleIds"></select>
                <span asp-validation-for="RoleIds" class="text-danger"></span>
            </div>
        </div>


        <hr />
        <div class="row mb-3">
            <div class="offset-2 col-10">
                <button type="submit" class="btn btn-primary">Save</button>
                  
                <button type="reset" class="btn btn-outline-primary">
                        Reset
                </button>
                  
                <a asp-action="Index">Back to List</a>
            </div>
        </div>
</form>
```

Views\Users\Delete.cshtml:

Displaying only the user name response property is enough to give information about the record to be deleted.

IV) Projects Setup for Authentication and Authorization

Note: Authentication: Who are you? Authroization: What can you do?

A) CORE Project for Authentication and Authorization

1. Install Microsoft.AspNetCore.Authentication.Cookies latest version package from NuGet.

2. Add a new folder named Authentication in the CORE Project.

3. Add a new folder called Services in the Authentication folder.

4. Add an interface named ICookieAuthService in the Services folder of the Authentication folder as below:

```
namespace CORE.Authentication.Services
{
    /// <summary>
    /// Defines methods for handling cookie-based
    /// authentication operations.
    /// </summary>
    public interface ICookieAuthService
    {
        // Properties can also be defined in interfaces with C#.
        // User ID readonly property to get the authenticated
        // user's ID.
        public int UserId { get; }

        /// <summary>
        /// Signs in a user by creating an authentication cookie.
        /// </summary>
        /// <param name="userId">The unique identifier of
        /// the user.</param>
        /// <param name="userName">The user name of the user.
        /// </param>
        /// <param name="userRoleNames">A collection of role
        /// names assigned to the user.</param>
        /// <param name="expiration">Optional expiration date
        /// and time for the authentication cookie.
        /// If not specified, a default value (null) is used.
        /// </param>
```

```
        /// <param name="isPersistent">Indicates whether the
        /// authentication cookie should persist across browser
        /// sessions.</param>
        /// <returns>A task representing the asynchronous
        /// sign-in operation.</returns>
        public Task SignIn(int userId, string userName,
            IEnumerable<string> userRoleNames,
            DateTime? expiration = default,
            bool isPersistent = true);

        /// <summary>
        /// Signs out the current user by removing the
        /// authentication cookie.
        /// </summary>
        /// <returns>A task representing the asynchronous
        /// sign-out operation.</returns>
        public Task SignOut();
    }
}
```

5. Add CookieAuthService.cs class file in the Services folder of the Authentication folder as below:

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Http;
using System.Security.Claims;

namespace CORE.Authentication.Services
{
    /// <summary>
    /// Provides cookie-based authentication services for
    /// signing in and signing out users.
    /// </summary>
    public class CookieAuthService : ICookieAuthService
    {
        // Field injected by the constructor to be used in
        // SignIn and SignOut methods below.
        private readonly IHttpContextAccessor _httpContextAccessor;

        /// <summary>
        /// Initializes a new instance of the CookieAuthService class.
        /// </summary>
        /// <param name="httpContextAccessor">Provides access
        /// to the current HTTP context. Injection must be
        /// managed in the IoC Container of Program.cs.</param>
        public CookieAuthService(IHttpContextAccessor httpContextAccessor)
        {
            _httpContextAccessor = httpContextAccessor;
        }

        // Readonly property to get the authenticated user's ID value.
        public int UserId => Convert.ToInt32(
            _httpContextAccessor.HttpContext.User.Claims.SingleOrDefault(
                claim => claim.Type == "Id").Value);
```

```
/// <summary>
/// Signs in a user by creating an authentication cookie
/// with the specified claims and properties.
/// </summary>
/// <param name="userId">The unique identifier of the
/// user.</param>
/// <param name="userName">The user name of the user.
/// </param>
/// <param name="userRoleNames">A collection of role
/// names assigned to the user.</param>
/// <param name="expiration">Optional expiration date
/// and time for the authentication cookie.
/// If not specified, a default value (null) is used.
/// </param>
/// <param name="isPersistent">Indicates whether the
/// authentication cookie should persist across browser
/// sessions.</param>
/// <returns>A task representing the asynchronous
/// sign-in operation.</returns>
public async Task SignIn(int userId, string userName,
    IEnumerable<string> userRoleNames,
    DateTime? expiration = default,
    bool isPersistent = true)
{
    // Create claims for user ID and username,
    // then add claims for each user role.
    var claims = new List<Claim>()
    {
        new Claim("Id", userId.ToString()),
        // custom claim with key Id and value user ID
        new Claim(ClaimTypes.Name, userName)
        // claim with key Name and value user name
    };
    foreach (var userRoleName in userRoleNames)
    {
        claims.Add(new Claim(ClaimTypes.Role, userRoleName));
        // claim with key Role and value user's role name
    }

    // Create a ClaimsIdentity with the generated claims
    // and specify the authentication scheme ("Cookies").
    var identity = new ClaimsIdentity(claims,
        CookieAuthenticationDefaults.AuthenticationScheme);

    // Create a ClaimsPrincipal from the identity.
    var principal = new ClaimsPrincipal(identity);

    // Set authentication properties,
    // including persistence and expiration.
    var authenticationProperties = new AuthenticationProperties
    {
        IsPersistent = isPersistent,
        ExpiresUtc = expiration.HasValue
            ? DateTime.SpecifyKind(expiration.Value, DateTimeKind.Utc)
            : null
    };

    // Sign in the user by issuing the authentication cookie.
    await _httpContextAccessor.HttpContext.SignInAsync(
        CookieAuthenticationDefaults.AuthenticationScheme, principal,
        authenticationProperties);
}
```

```
        /// <summary>
        /// Signs out the current user by removing the authentication cookie.
        /// </summary>
        /// <returns>A task representing the asynchronous sign-out operation.
        /// </returns>
        public async Task SignOut()
        {
            // Sign out the user by removing the authentication cookie
            // from the current HTTP context.
            await _httpContextAccessor.HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
        }
    }
}
```

B) APP Project for Authentication and Authorization

1. Create a folder named Authentication in the APP Project.
2. Create a folder called Models in the Authentication folder.
3. Under the Models folder of the Authentication folder, create LoginRequest model class as below:

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace APP.Authentication.Models
{
    /// <summary>
    /// Represents the data required for a user login request.
    /// Used for binding and validating login form input in
    /// Razor Views.
    /// </summary>
    public class LoginRequest
    {
        /// <summary>
        /// Gets or sets the user name for login.
        /// Must be between 3 and 30 characters.
        /// </summary>
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(30, MinimumLength = 3,
            ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        [DisplayName("User Name")]
        public string UserName { get; set; }

        /// <summary>
        /// Gets or sets the password for login.
        /// Must be between 3 and 15 characters.
        /// </summary>
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(15, MinimumLength = 3,
            ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        public string Password { get; set; }
```

```
    }
}
```

4. Under the Models folder of the Authentication folder, create RegisterRequest model class as below:

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace APP.Authentication.Models
{
    /// <summary>
    /// Represents the data required for a user register request.
    /// Used for binding and validating register form input
    /// in Razor Views.
    /// </summary>
    public class RegisterRequest
    {
        /// <summary>
        /// Gets or sets the user name for register.
        /// Must be between 3 and 30 characters.
        /// </summary>
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(30, MinimumLength = 3,
            ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        [DisplayName("User Name")]
        public string UserName { get; set; }

        /// <summary>
        /// Gets or sets the password for register.
        /// Must be between 3 and 15 characters.
        /// </summary>
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(15, MinimumLength = 3,
            ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        public string Password { get; set; }

        /// <summary>
        /// Gets or sets the confirm password for register.
        /// Must be between 3 and 15 characters.
        /// Must have the equal value with Password.
        /// </summary>
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(15, MinimumLength = 3,
            ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]
        [Compare("Password",
            ErrorMessage = "Password and Confirm Password must be the same!")]
        [DisplayName("Confirm Password")]
        public string ConfirmPassword { get; set; }
    }
}
```

5. Create a folder called Services in the Authentication folder.

6. Create an interface called IAuthService in the Services folder of the Authentication folder as below:

```
using APP.Authentication.Models;
using CORE.Models;

namespace APP.Authentication.Services
{
    /// <summary>
    /// Cookie authentication service interface containing
    /// Login, Logout and Register method definitions.
    /// </summary>
    public interface IAuthService
    {
        /// <summary>
        /// Authenticates a user using the provided login
        /// credentials and initiates a cookie-based sign-in.
        /// </summary>
        /// <param name="request">
        /// The LoginRequest containing the user's login
        /// information (user name and password).
        /// </param>
        /// <returns>
        /// A CommandResponse indicating the result of the
        /// login attempt.
        /// Returns an error response if credentials are invalid;
        /// otherwise, returns a success response with the user's ID.
        /// Also a cookie named ".AspNetCore.Cookies" is created
        /// in the browser to maintain the authenticated session.
        /// </returns>
        public Task<CommandResponse> Login(LoginRequest request);

        /// <summary>
        /// Signs out the currently authenticated user by removing
        /// the ".AspNetCore.Cookies" authentication cookie.
        /// </summary>
        /// <returns>
        /// A Task representing the asynchronous sign-out operation.
        /// </returns>
        public Task Logout();

        /// <summary>
        /// Registers a new user with the default "User" role and
        /// "Active" status.
        /// </summary>
        /// <param name="request">
        /// The RegisterRequest containing the registration data
        /// including user name, password and confirm password.
        /// </param>
        /// <returns>
        /// A CommandResponse indicating the result of the registration.
        /// </returns>
        public CommandResponse Register(RegisterRequest request);
    }
}
```

7. Create a class called AuthService in the Services folder of the Authentication folder as below:

```
using APP.Authentication.Models;
using APP.Domain;
using CORE.Authentication.Services;
using CORE.Models;
using CORE.Services;
using Microsoft.EntityFrameworkCore;

namespace APP.Authentication.Services
{
    /// <summary>
    /// Cookie authentication service concrete class
    /// inheriting from the base abstract DbService class
    /// for User entity query, insert and update database operations
    /// and implementing the IAuthService method definitions
    /// for log in, log out and register operations.
    /// </summary>
    public class AuthService : DbService<User>, IAuthService
    {
        /// <summary>
        /// Service interface for cookie authentication
        /// including sign in and sign out methods.
        /// The injected instance in the constructor is assigned
        /// to this field to be used in Login and Logout
        /// methods below.
        /// </summary>
        private readonly ICookieAuthService _cookieAuthService;

        /// <summary>
        /// Initializes a new instance of the AuthService class.
        /// Uses dependency injection to receive the database context
        /// and cookie authentication service.
        /// </summary>
        /// <param name="db">
        /// The DbContext instance used for database operations,
        /// which the injection is managed in the IoC Container
        /// of Program.cs.
        /// </param>
        /// <param name="cookieAuthService">
        /// The ICookieAuthService instance used for cookie authentication,
        /// which the injection is managed in the IoC Container
        /// of Program.cs.
        /// </param>
        public AuthService(DbContext db,
            ICookieAuthService cookieAuthService) : base(db)
        {
            _cookieAuthService = cookieAuthService;
        }

        // Overridden base DbQuery method to include Status, UserRole
        // and Role entities to the query.
        protected override IQueryable<User> DbQuery()
        {
            return base.DbQuery()
                .Include(u => u.Status)
                .Include(u => u.UserRoles)
                .ThenInclude(ur => ur.Role);
            // u: User entity delegate, ur: UserRole entity delegate.
```

```
}

/// <summary>
/// Authenticates a user using the provided login
/// credentials and initiates a cookie-based sign-in.
/// </summary>
/// <param name="request">
/// The LoginRequest containing the user's login
/// information (user name and password).
/// </param>
/// <returns>
/// A CommandResponse indicating the result of the login attempt.
/// Returns an error response if credentials are invalid;
/// otherwise, returns a success response with the user's ID.
/// Also a cookie named ".AspNetCore.Cookies" is created
/// in the browser to maintain the authenticated session.
/// </returns>
public async Task<CommandResponse> Login(LoginRequest request)
{
    /*
    Synchronous methods execute tasks one after another.
    Each operation must complete before the next one starts.
    The calling thread waits (or "blocks") until the method finishes.

    Asynchronous methods allow tasks to run in the background.
    The calling thread does not wait for the operation to finish and
    can continue executing other code. In C#, asynchronous methods
    often use the async and await keywords, enabling non-blocking
    operations (such as I/O or database calls) and improving
    application responsiveness.
    */
    // Attempt to find an active user matching the provided
    // user name and password.
    var userEntity = DbSingle(u => u.UserName == request.UserName
            && u.Password == request.Password
            && u.StatusId == 1);
            // "Active" status ID value is 1 in the Statuses table.
            // Instead of u.StatusId == 1, u.Status.Title == "Active"
            // may also be written.
    // u: User entity delegate, ur: UserRole entity delegate.

    // If no matching user entity is found, return an error response.
    if (userEntity is null)
        return Error("Invalid user name or password!");

    // Update the User entity's online status
    // (Optional if the User entity has an online status property).
    userEntity.IsOnline = true;
    DbUpdate(userEntity);

    // Sign in the user using cookie authentication,
    // passing user ID, user name, and role names
    // as parameters.
    await _cookieAuthService.SignIn(
        userEntity.Id,
        userEntity.UserName,
        userEntity.UserRoles.Select(ur => ur.Role.Name));

    // Return a success response with the user's ID.
    return Success(userEntity.Id, "User logged in successfully.");
}

/// <summary>
```

```
        /// Signs out the currently authenticated user by removing
        /// the ".AspNetCore.Cookies" authentication cookie.
        /// </summary>
        /// <returns>
        /// A Task representing the asynchronous sign-out operation.
        /// </returns>
        public async Task Logout()
        {
            // Find the User entity by authenticated user's ID
            // to update the online status.
            // (Optional if the User entity has an online status property).
            var userEntity = DbSingle(_cookieAuthService.UserId);
            // If User entity is found, update the online status.
            if (userEntity is not null)
            {
                userEntity.IsOnline = false;
                DbUpdate(userEntity);
            }

            // Perform sign out using the cookie authentication service.
            await _cookieAuthService.SignOut();
        }

        /// <summary>
        /// Registers a new user with the default "User" role and
        /// "Active" status.
        /// </summary>
        /// <param name="request">
        /// The RegisterRequest containing the registration data
        /// including user name, password and confirm password.
        /// </param>
        /// <returns>
        /// A CommandResponse indicating the result of the registration.
        /// </returns>
        public CommandResponse Register(RegisterRequest request)
        {
            // Check if a user with the same user name exists.
            if (DbQuery().Any(u => u.UserName == request.UserName.Trim()))
                return Error("User with the same user name exists!");

            // Insert a new User entity with request's user name and password,
            // status "Active" and role "User".
            var userEntity = new User
            {
                UserName = request.UserName.Trim(),
                Password = request.Password.Trim(),
                StatusId = 1,
                // "Active" status ID value is 1 in the Statuses table.

                // Way 1:
                //RoleIds = new List<int> { 2 }
                // Way 2:
                RoleIds = [2]
                // "User" role ID value is 2 in the Roles table.
            };
            DbAdd(userEntity);
            return Success(userEntity.Id, "User registered successfully.");
        }
    }
}
```

C) MVC Project for Authentication and Authorization

Authentication:

1. In the IoC Container of Program.cs add the sections marked as Authentication as below:

```
// --------------------------
// Authentication and Session
// --------------------------
// Register IHttpContextAccessor as a singleton service.
// Enables access to the current HttpContext from classes other than
// controller classes such as services.
// Required for services like CookieAuthService that need to read
// or modify HTTP context (e.g., authentication, user info).
// Allows constructor injection of IHttpContextAccessor throughout
// the services of the application.
builder.Services.AddHttpContextAccessor();



// --------------
// Authentication
// --------------
// Register CookieAuthService as a scoped dependency for ICookieAuthService.
// Scoped lifetime ensures a new instance per HTTP request, which is
// required for services accessing HttpContext.
// CookieAuthService handles user authentication using cookies, supporting
// sign in and sign out operations.
// This service registration enables constructor injection of
// ICookieAuthService to the controllers or services throughout the application.
builder.Services.AddScoped<ICookieAuthService, CookieAuthService>();



// --------------
// Authentication
// --------------
// Register AuthService as a scoped dependency for IAuthService.
// Scoped lifetime ensures a new instance per HTTP request, which is
// required for services accessing HttpContext.
// AuthService handles user authentication using injected ICookieAuthService
// instance, supporting log in, log out and register operations.
// This service registration enables constructor injection of
// IAuthService to the controllers or services throughout the application.
builder.Services.AddScoped<IAuthService, AuthService>();



// --------------
// Authentication
// --------------
// Configure authentication services using cookie-based authentication.
// - Sets the default authentication scheme to Cookies.
// - Specifies options for cookie authentication:
//   - LoginPath: Path to redirect unauthenticated users
//     (when authentication is required).
```

```
//   - AccessDeniedPath: Path to redirect users who are authenticated
//     but lack required permissions.
//   - ExpireTimeSpan: Duration before the authentication cookie expires
//     (here, 1 hour).
//   - SlidingExpiration: Resets the expiration time on each request
//     to keep active sessions alive.
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Login";
        // changed from /Auth/Login to /Login since route will be changed for action
        options.AccessDeniedPath = "/Login";
        // changed from /Auth/Login to /Login since route will be changed for action
        options.ExpireTimeSpan = TimeSpan.FromHours(1);
        options.SlidingExpiration = true;
    });


// ---------------
// Authentication
// ---------------
// Enable authentication middleware so that [Authorize] works.
app.UseAuthentication();
```

2. Create an empty MVC Controller called AuthController in the Controllers folder and implement the code as below:

   Note: In a controller, the HTTP Context can be directly accessed by the HttpContext property inherited from Controller base class.

```
using APP.Authentication.Models;
using APP.Authentication.Services;
using Microsoft.AspNetCore.Mvc;

namespace MVC.Controllers
{
    /// <summary>
    /// Controller for authentication operations such as
    /// Login, Logout and Register.
    /// </summary>
    public class AuthController : Controller
    {
        // AuthService constructor injection.
        private readonly IAuthService _authService;

        public AuthController(IAuthService authService)
        {
            _authService = authService;
        }



        /// <summary>
        /// Displays the login view for user authentication.
        /// Route changed from GET: /Auth/Login to
        /// GET: /Login for easier access by the Route
        /// attribute below.
        /// </summary>
```

```
[Route("~/[action]")]
public IActionResult Login()
{
    return View();
}


/// <summary>
/// Handles user login form submission.
/// Route changed from POST: /Auth/Login to
/// POST: /Login for easier access by the Route
/// attribute below.
/// </summary>
/// <param name="request">The login request containing user
/// credentials (UserName and Password).</param>
/// <returns>
/// Redirects to Home/Index on successful login; otherwise,
/// redisplays the login view with validation errors or
/// authentication failure message.
/// </returns>
[HttpPost, ValidateAntiForgeryToken]
[Route("~/[action]")]
public async Task<IActionResult> Login(LoginRequest request)
{
    if (ModelState.IsValid)
    // Validate input model (checks required fields and
    // string lengths through data annotations).
    {
        var response = await _authService.Login(request);
        // Attempt to authenticate user.
        if (response.IsSuccessful)
            return RedirectToAction("Index", "Home");
            // On success, redirect to home page.
        ModelState.AddModelError("", response.Message);
        // On failure, show error message in the validation summary
        // of the view.
    }
    return View();
    // Redisplay login view with validation errors or
    // authentication failure message.
}


/// <summary>
/// Logs out the current user and redirects to Login.
/// Route changed from GET: /Auth/Logout to
/// GET: /Logout for easier access by the Route
/// attribute below.
/// </summary>
/// <remarks>This method terminates the user's
/// authentication cookie.</remarks>
/// <returns>Redirects to Login.</returns>
[Route("~/[action]")]
public async Task<IActionResult> Logout()
{
    await _authService.Logout();
    // Sign out the user.
    return RedirectToAction(nameof(Login));
    // Redirect to the Login view through the Login action.
}
```

```
        /// <summary>
        /// Displays the register view for user registration.
        /// Route changed from GET: /Auth/Register to
        /// GET: /Register for easier access by the Route
        /// attribute below.
        /// </summary>
        [Route("~/[action]")]
        public IActionResult Register()
        {
            return View();
        }


        /// <summary>
        /// Handles user register form submission.
        /// Route changed from POST: /Auth/Register to
        /// POST: /Register for easier access by the Route
        /// attribute below.
        /// </summary>
        /// <param name="request">The register request containing
        /// UserName, Password and ConfirmPassword.</param>
        /// <returns>
        /// Redirects to Login on successful registration; otherwise,
        /// redisplays the register view with validation errors
        /// or registration failure message.
        /// </returns>
        [HttpPost, ValidateAntiForgeryToken, Route("~/[action]")]
        public IActionResult Register(RegisterRequest request)
        {
            if (ModelState.IsValid)
            // Validate input model through data annotations.
            {
                var response = _authService.Register(request);
                // Attempt to register user.
                if (response.IsSuccessful)
                    return RedirectToAction(nameof(Login));
                // On success, redirect to Login view through the
                // Login action.
                ModelState.AddModelError("", response.Message);
                // On failure, show error message in the validation summary
                // of the view.
            }
            return View(request);
            // Redisplay register view with the model and validation errors
            // or registration failure message.
        }
    }
}
```

3. Add an empty Razor View named Login.cshtml in the Views/Auth folder and implement the view as below:

```
@model APP.Authentication.Models.LoginRequest
@*
    APP.Authentication.Models namespace can also
    be added to Views/_ViewImports.cshtml therefore
    only LoginRequest can be declared as the model.
*@
```

```
@*
    Model namespace using directive should be added
    to _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "User Login";
}

<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>

<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
        /*
        TempData is used to carry extra data to the
        redirected controller action's view.
        */
        <p class="text-danger">
            @TempData["Message"]
        </p>
    }
    <form asp-action="Login" autocomplete="off">
        @Html.AntiForgeryToken()
        <div asp-validation-summary="All" class="text-danger"></div>
        @*
            Use ModelOnly instead of All to see only service response
            messages in validation summary
        *@
        <div class="row mb-3">
            <label asp-for="UserName"
                   class="col-2 col-form-label fw-bold"></label>
            <div class="col-10">
                <input asp-for="UserName" class="form-control" />
                <span asp-validation-for="UserName" class="text-danger"></span>
            </div>
        </div>
        <div class="row mb-3">
            <label asp-for="Password"
                   class="col-2 col-form-label fw-bold"></label>
            <div class="col-10">
                <input asp-for="Password" class="form-control" type="password" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
        </div>

        <hr />
        <div class="row mb-3">
```

```
            <div class="offset-2 col-10">
                <button type="submit" class="btn btn-primary">Login</button>
            </div>
        </div>
    </form>
</div>

@section Scripts {
    @*
        Can be uncommented to use client-side validation using jQuery
        instead of server-side validation.
    *@
    <partial name="_ValidationScriptsPartial" />
}
```

4. Add an empty Razor View named Register.cshtml in the Views/Auth folder and implement the view as below:

```
@model APP.Authentication.Models.RegisterRequest
@*
    APP.Authentication.Models namespace can also
    be added to Views/_ViewImports.cshtml therefore
    only RegisterRequest can be declared as the model.
*@

@*
    Model namespace using directive should be added
    to _ViewImports.cshtml.
*@

@{
    var containerDivClass = "container";
    // "container-fluid" can be used for full width
}
@{
    /*
    ViewBag and ViewData are the same collection (dictionary).
    They carry extra data other than the model from a
    controller action to its view, or between views.
    The value assigned will be shown in
    Views\Shared\_Layout.cshtml view's title tag of
    the head tag.
    */
    ViewData["Title"] = "User Register";
}

<div class="@containerDivClass">
    <h1>@ViewData["Title"]</h1>
    <hr />
</div>

<div class="@containerDivClass">
    @if (TempData["Message"] is not null)
    {
        /*
        TempData is used to carry extra data to the
        redirected controller action's view.
        */
```

```
            <p class="text-danger">
                @TempData["Message"]
            </p>
        }
        <form asp-action="Register" autocomplete="off">
            @Html.AntiForgeryToken()
            <div asp-validation-summary="All" class="text-danger"></div>
            @*
                Use ModelOnly instead of All to see only service response
                messages in validation summary
            *@
            <div class="row mb-3">
                <label asp-for="UserName"
                       class="col-2 col-form-label fw-bold"></label>
                <div class="col-10">
                    <input asp-for="UserName" class="form-control" />
                    <span asp-validation-for="UserName" class="text-danger"></span>
                </div>
            </div>
            <div class="row mb-3">
                <label asp-for="Password"
                       class="col-2 col-form-label fw-bold"></label>
                <div class="col-10">
                    <input asp-for="Password" class="form-control" type="password" />
                    <span asp-validation-for="Password" class="text-danger"></span>
                </div>
            </div>
            <div class="row mb-3">
                <label asp-for="ConfirmPassword"
                       class="col-2 col-form-label fw-bold"></label>
                <div class="col-10">
                    <input asp-for="ConfirmPassword"
                       class="form-control" type="password" />
                    <span asp-validation-for="ConfirmPassword"
                       class="text-danger"></span>
                </div>
            </div>

            <hr />
            <div class="row mb-3">
                <div class="offset-2 col-10">
                    <button type="submit" class="btn btn-primary">Register</button>
                </div>
            </div>
        </form>
    </div>

    @section Scripts {
        @*
            Can be uncommented to use client-side validation using jQuery
            instead of server-side validation.
        *@
        <partial name="_ValidationScriptsPartial" />
    }
```

5. Add Login, Register, Logout and user name links in the Authentication Links section in the nav bar (top menu) of the layout view as below:

```
@* Authentication Links: *@
@*
    Will be displayed at the right side of the menu by the below
    Bootstrap classes.
*@
<div class="navbar-text">
    <ul class="navbar-nav me-auto">
        @*
            If user is authenticated (logged in)
            show user name and log out.
        *@
        @if (User.Identity.IsAuthenticated)
        {
            <li class="nav-item">

                @* Way 1: Displaying user name text. *@
                @* <span>@User.Identity.Name</span> *@
                @*
                    Way 2: Optional link to direct the user to
                    his/her user details.
                *@
                @{
                    @*
                        Getting the user's ID from the claims to
                        direct the user to the user details with
                        id route value.
                    *@
                    var userId = @User.Claims .SingleOrDefault(
                        claim => claim.Type == "Id")?.Value;
                }
                <a class="nav-link text-dark" asp-area=""
                    asp-controller="Users" asp-action="Details"
                    asp-route-id="@userId">@User.Identity.Name</a>

            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
                    asp-controller="Auth" asp-action="Logout">Logout</a>
            </li>
        }
        else @* If user is not authenticated show register and login. *@
        {
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
                    asp-controller="Auth" asp-action="Register">Register</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
                    asp-controller="Auth" asp-action="Login">Login</a>
            </li>
        }
    </ul>
</div>
```

Notes:

- Authentication cookie existance can be checked by User.Identity.IsAuthenticated in controller actions and views returning the result of whether the user is signed in or not.

- The auhenticated user's user name can be reached by User.Identity.Name.

- The authenticated user's role claim value can be checked by User.IsInRole method which gets the role name string as parameter, or the role claim can be accessed by User.Claims.SingleOrDefault method which gets the predicate checking claim type equals role claim type as parameter.

- Examples:

a) if (User.IsInRole("Admin")) { ... }

b) Claim roleClaim = User.Claims.SingleOrDefault(

claim => claim.Type == ClaimTypes.Role);

c) if (roleClaim is not null && roleClaim.Value == "User") { ... }

d) Custom claim types' values such as Id can be accessed by:

int id = Convert.ToInt32(User.Claims?.Single(claim => claim.Type == "Id").Value);

Authorization:

1. Add the [Authorize(Roles = "Admin")] attribute for Admin role on top of the RolesController class so that all of the actions can be executed by only authenticated users with role Admin as below:

```
[Authorize(Roles = "Admin")]
// Only authenticated users with role Admin can execute all
// of the actions of this controller.
public class RolesController : Controller
```

Notes:

- The [Authorize] attribute in ASP.NET Core is used to control access to a controller's actions by requiring that the user is authenticated and, optionally, meets certain authorization requirements.

- When you decorate a controller or action with [Authorize], only authenticated users who provides the Authentication Cookie or Json Web Token (used in Web APIs) can access it. Unauthenticated requests will receive a 401 Unauthorized response.
  In MVC, they will be redirected to the LoginPath defined in the authentication configuration in Program.cs.

- You can specify roles or policies to further restrict access. For example,
  [Authorize(Roles = "Admin")]
  allows only authenticated users in the "Admin" role.

- You can apply [Authorize] at the controller level (to protect all actions) or at the action level.

Examples:

a) [Authorize]
   Any authenticated user can access.

b) [Authorize(Roles = "User")]
   Only authenticated users with role "User" can access.

c) [Authorize(Roles = "Admin,User")]
   Only authenticated users with the "Admin" or "User" role can access.

d) [AllowAnonymous]
   Can be used to override [Authorize] at the action level and allows public access, therefore gives permission to everyone for executing specific actions.

2. Add the [Authorize] attribute on top of the StatusesController class.

   For example, if the Index action is wanted to be executed by authorized and unauthorized users (eveyone), [AllowAnonymous] attribute can be defined.

   The Details action can be executed by only authenticated users since Authorize is defined at controller level.

   Add [Authorize(Roles = "Admin")] attribute on top of the Create, Edit and Delete get and post actions so that only authenticated users with role Admin can execute these actions.

3. Add the [Authorize] attribute on top of the Index action of the UsersController class so that authenticated users with all roles can execute the Index action.

   Then add [Authorize] attribute on top of the Details, Edit and Delete get and post actions to allow only authenticated users to execute. These actions also include if the user is trying to make an operation on his/her own account through IsOwnAccount method at the bottom of the UsersController class,
   or for users with role Admin if the authenticated user is in Admin role check.

   IsOwnAccount Method:

```
/// <summary>
/// For regular users can only make operations on their own accounts.
/// </summary>
/// <param name="id">User ID.</param>
/// <returns>bool</returns>
bool IsOwnAccount(int id) // private is default if not written
{
    // getting the authenticated user ID value for the claim type
    // "Id" from the user's claims and checking if it matches
    // the provided id parameter of a user
    return id.ToString() == (User.Claims.SingleOrDefault(
        claim => claim.Type == "Id")?.Value ?? string.Empty);
}
```

   Details GET Action:

```
// GET: Users/Details/5
[Authorize] // Only authenticated users can execute this action.
public IActionResult Details(int id)
```

```
{
    // Check if the user is in Admin role or trying to make
    // the operation on his/her own account.
    if (!IsOwnAccount(id) && !User.IsInRole("Admin"))
    {
        SetTempData("You are not authorized for this operation!");
        return RedirectToAction(nameof(Index));
    }

    // Get item service logic:
    var item = _userService.Single(id);
    return View(item); // return response item as model to the Details view
}
```

Edit GET Action:

```
// GET: Users/Edit/5
[Authorize] // Only authenticated users can execute this action.
public IActionResult Edit(int id)
{
    // Check if the user is in Admin role or trying to make
    // the operation on his/her own account.
    if (!IsOwnAccount(id) && !User.IsInRole("Admin"))
    {
        SetTempData("You are not authorized for this operation!");
        return RedirectToAction(nameof(Index));
    }

    // Get item to edit service logic:
    var item = _userService.Edit(id);
    SetViewData(); // set ViewData dictionary to carry extra data
                   // other than the model to the view

    return View(item); // return request as model to the Edit view
}
```

Edit POST Action:

```
// POST: Users/Edit
[HttpPost, ValidateAntiForgeryToken]
[Authorize] // Only authenticated users can execute this action.
public IActionResult Edit(UserRequest user)
{
    // Check if the user is in Admin role or trying to make
    // the operation on his/her own account.
    if (!IsOwnAccount(user.Id) && !User.IsInRole("Admin"))
    {
        SetTempData("You are not authorized for this operation!");
```

```
            return RedirectToAction(nameof(Index));
        }

        // Because we don't get the values for the Score, StatusId and
        // RoleIds properties from the user request for users not in
        // the Admin role, and Score, StatusId and RoleIds are required,
        // we must remove the ModelState errors for these properties
        // to prevent validation failures.
        if (!User.IsInRole("Admin"))
        {
            ModelState.Remove(nameof(UserRequest.Score));
            ModelState.Remove(nameof(UserRequest.StatusId));
            ModelState.Remove(nameof(UserRequest.RoleIds));
        }

        if (ModelState.IsValid)
        // check data annotation validation errors in the request
        {
            // Update item service logic:
            var response = _userService.Update(user);
            if (response.IsSuccessful)
            {
                SetTempData(response.Message);
                // set TempData dictionary to carry the message
                // to the redirected action's view
                return RedirectToAction(nameof(Details),
                    new { id = response.Id });
                // redirect to Details action with id parameter as
                // response.Id route value
            }
            ModelState.AddModelError("", response.Message);
            // to display service error message in the
            // validation summary of the view
        }
        SetViewData();
        // set ViewData dictionary to carry extra data other than
        // the model to the view

        return View(user); // return request as model to the Edit view
    }
```

Views/Users/Edit.cshtml must also be modified for the request properties Score, StatusId and RoleIds to be shown only to the authenticated users with role Admin because only authenticated users with Admin role can update them.

```
@if (User.IsInRole("Admin"))
{
    <div class="row mb-3">
        <label asp-for="Score"
            class="col-2 col-form-label fw-bold"></label>
    <div class="col-10">
        <input asp-for="Score" class="form-control" />
        <span asp-validation-for="Score" class="text-danger"></span>
    </div>
</div>
}
```

```
@if (User.IsInRole("Admin"))
{
<div class="row mb-3">
    <label asp-for="StatusId"
        class="col-2 col-form-label fw-bold"></label>
    <div class="col-10">
        <select asp-for="StatusId" class="form-select"
            asp-items="ViewBag.StatusId">
            <option value="">-- Select --</option>
        </select>
        <span asp-validation-for="StatusId" class="text-danger"></span>
    </div>
</div>
}
```

```
@if (User.IsInRole("Admin"))
{
<div class="row mb-3">
    <label asp-for="RoleIds"
        class="col-2 col-form-label fw-bold"></label>
    <div class="col-10">
        <select multiple asp-for="RoleIds" class="form-select"
            asp-items="ViewBag.RoleIds"></select>
        <span asp-validation-for="RoleIds" class="text-danger"></span>
    </div>
</div>
}
```

Delete GET Action:

```
  // GET: Users/Delete/5
  [Authorize] // Only authenticated users can execute this action.
  public IActionResult Delete(int id)
  {
      // Check if the user is in Admin role or trying to make
      // the operation on his/her own account.
      if (!IsOwnAccount(id) && !User.IsInRole("Admin"))
      {
          SetTempData("You are not authorized for this operation!");
          return RedirectToAction(nameof(Index));
      }

      // Get item to delete service logic:
      var item = _userService.Single(id);
      return View(item);
      // return response item as model to the Delete view
}
```

Delete POST Action:

```
// POST: Users/Delete
[HttpPost, ValidateAntiForgeryToken, ActionName("Delete")]
[Authorize] // Only authenticated users can execute this action.
public IActionResult DeleteConfirmed(int id)
{
    // Check if the user is in Admin role or trying to make
    // the operation on his/her own account.
    if (!IsOwnAccount(id) && !User.IsInRole("Admin"))
    {
        SetTempData("You are not authorized for this operation!");
        return RedirectToAction(nameof(Index));
    }

    // Delete item service logic:
    var response = _userService.Remove(id);
    SetTempData(response.Message);
    // set TempData dictionary to carry the message
    // to the redirected action's view

    // if the user deleted his/her own account, log out the user
    if (IsOwnAccount(id))
        return RedirectToAction("Logout", "Auth");

    return RedirectToAction(nameof(Index));
    // redirect to the Index action
}
```

Finally add [Authorize(Roles = "Admin")] attribute on top of the Create get and post actions so that authenticated users with role Admin can execute these actions.

4. We shouldn't show the links directing to unauthorized controller actions in the views. Therefore, we need to add if the user is in Admin role in the layout view for the Roles link. We also need to check if the user is in Admin or User role (checking if user is authenticated is better since we have only 2 roles) for the Users link. Statuses link will be visible to everyone. We can show the user name and Logout links if the user is authenticated, Register and Login links if the user is not authenticated.

```
@* Application Links: *@
@* Only users in Admin role can see the Roles link. *@
@if (User.IsInRole("Admin"))
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area=""
            asp-controller="Roles" asp-action="Index">Roles</a>
    </li>
}
```

```
@* Statuses link will be visible to everyone (authenticated
    and not authenticated users.) *@
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
      asp-controller="Statuses" asp-action="Index">Statuses</a>
</li>
@* Only authenticated users can see the Users and Statuses links: *@
@* Way 1: *@
@* @if (User.IsInRole("Admin") || User.IsInRole("User")) *@
@* Way 2: since we have only 2 roles, we don't need to
    check all the roles *@
@if (User.Identity.IsAuthenticated)
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area=""
            asp-controller="Users" asp-action="Index">Users</a>
    </li>
}
```

5. We also should check if the user is authenticated for the Details link in the Index view of the Statuses since only authenticated users can execute the Details action. We should also show the Create, Edit and Delete links to the users with role Admin.

6. In Statuses Details view, we should show the Edit and Delete links for the Admin role.

7. In Statuses Edit view we should show the Delete link for the Admin role.

8. In the Users Index view, we should check if the user is authenticated for the Details link. We should also check if the user is in role Admin for the Create, Edit and Delete links.

9. We should also show specific fields (Score, StatusId and RoleIds) only to the Admin users in the Users Edit view.

V) Projects Setup for Session

A) CORE Project for Session

1. Create the SessionService concrete class under CORE Project's Session/Services folder as below:

```
using Microsoft.AspNetCore.Http;
using System.Text;
using System.Text.Json;

namespace CORE.Session.Services
{
    /// <summary>
    /// Concrete class for session management.
    /// Provides methods to store, retrieve, and remove
    /// complex objects in session using JSON serialization.
    /// Implemented as a concrete class because the behaviors
    /// of this class are standard and are not expected to change.
    /// </summary>
    public class SessionService
    {
        /// <summary>
        /// Provides access to the current HTTP context,
        /// enabling session operations.
        /// </summary>
        private readonly IHttpContextAccessor _httpContextAccessor;

        /// <summary>
        /// Initializes the session service with an
        /// injected IHttpContextAccessor instance.
        /// </summary>
        /// <param name="httpContextAccessor">Accessor for the current
        /// HTTP context.</param>
        public SessionService(IHttpContextAccessor httpContextAccessor)
        {
            _httpContextAccessor = httpContextAccessor;
        }

       /// <summary>
       /// Readonly property to get the authenticated user's ID value.
       /// </summary>
       public int AuthenticatedUserId => Convert.ToInt32(
            _httpContextAccessor.HttpContext.User.Claims.SingleOrDefault(
                  claim => claim.Type == "Id").Value);

        /// <summary>
        /// Retrieves a session value by key and deserializes it to the
        /// specified type.
        /// Returns null if the key does not exist or the string value
        /// is empty.</summary>
        /// <typeparam name="T">Type to deserialize the session string
        /// value to.</typeparam>
        /// <param name="key">Unique session key.</param>
        /// <returns>Deserialized object or null.</returns>
        public T Get<T>(string key)
            where T : class // T can be any reference type.
        {
            var value = _httpContextAccessor.HttpContext
                .Session.GetString(key);
            // Retrieves the JSON string value from session by key.
            if (string.IsNullOrEmpty(value))
                return null;
            return JsonSerializer.Deserialize<T>(value);
            // Converts JSON string to object of type T and returns it.
        }

        /// <summary>
        /// Serializes the provided object and stores it in session under
        /// the specified key.
```

```
        /// Does nothing if the instance is null.
        /// </summary>
        /// <typeparam name="T">Type of the object to store.</typeparam>
        /// <param name="key">Unique session key.</param>
        /// <param name="instance">Object to serialize and store.</param>
        public void Set<T>(string key, T instance)
            where T : class // T can be any reference type.
        {
            if (instance is not null)
            {
                var value = JsonSerializer.Serialize(instance);
                // Converts object of type T to JSON string.
                _httpContextAccessor.HttpContext.Session.SetString(key, value);
                // Stores the JSON string value in session under
                // the specified key.
            }
        }

        /// <summary>
        /// Removes the session value associated with the specified key.
        /// </summary>
        /// <param name="key">Unique session key to remove.</param>
        public void Remove(string key)
        {
            _httpContextAccessor.HttpContext.Session.Remove(key);
            // Removes the session value associated with the specified key.
        }
    }
}
```

B) APP Project for Session

1. Create the UserSessionResponse model class under the APP Project's Models folder as below:

```
using System.ComponentModel;

namespace APP.Models
{
    /// <summary>
    /// The model to be used with UserSessionService to
    /// represent user session data with minimum, maximum
    /// and average score calculations.
    /// </summary>
    public class UserSessionResponse
    {
        /// <summary>
        /// The user ID of the user who is logged in.
        /// </summary>
        public int AuthenticatedUserId { get; set; }

        /// <summary>
        /// The user ID of the user whose data is being retrieved.
        /// </summary>
        public int UserId { get; set; }
```

```
        /// <summary>
        /// The user name of the user whose data is being retrieved.
        /// </summary>
        [DisplayName("User Name")]
        public string UserName { get; set; }

        /// <summary>
        /// The score of the user whose data is being retrieved.
        /// Will be used to calculate minimum, maximum and average
        /// scores.
        /// </summary>
        public double Score { get; set; }

        /// <summary>
        /// The score text of the user whose data is being retrieved.
        /// Also will be used to display minimum, maximum and average
        /// score calculations.
        /// </summary>
        [DisplayName("Score")]
        public string ScoreText { get; set; }
    }
}
```

2. Create the UserSessionService service class under the APP Project's Services folder as below:

```
using APP.Models;
using CORE.Services;
using CORE.Session.Services;

namespace APP.Services
{
    /// <summary>
    /// Concrete class for managing user session data.
    /// Provides methods to add, remove, retrieve, and
    /// clear user session data.
    /// Implemented as a concrete class because the behaviors
    /// of this class are standard and are not expected to change.
    /// </summary>
    public class UserSessionService
    {
        // Constant key value to store and retrieve user session
        // data in the session.
        // Default private if no accessibility modifier is specified.
        const string KEY = "UserSession";

        // Services to be injected for user session data management.
        private readonly SessionService _sessionService;
        private readonly IService<UserRequest, UserResponse> _userService;

        public UserSessionService(SessionService sessionService,
            IService<UserRequest, UserResponse> userService)
        {
            _sessionService = sessionService;
            _userService = userService;
        }
```

```
/// <summary>
/// Retrieves the list of user session data for the specified
/// authenticated user. Then calculates the minimum, maximum
/// and average scores of the users if there is at least one
/// user. Finally adds a calculation summary item to the list
/// before returning it.
/// </summary>
/// <returns>List of user session response items including
/// a summary item. If no users data is found in the session,
/// returns an empty list.</returns>
public List<UserSessionResponse> Get()
{
    var sessionUsers = _sessionService
        .Get<List<UserSessionResponse>>(KEY);
    if (sessionUsers is not null)
    {
        sessionUsers = sessionUsers.Where(
            su => su.AuthenticatedUserId
                == _sessionService.AuthenticatedUserId)
                    .ToList();
        if (sessionUsers.Count > 0)
        {
            var minimumScoreText = sessionUsers
                .Min(su => su.Score).ToString("N1");
            var maximumScoreText = sessionUsers
                .Max(su => su.Score).ToString("N1");
            var averageScoreText = sessionUsers
                .Average(su => su.Score).ToString("N1");
            sessionUsers.Add(new UserSessionResponse
            {
                ScoreText = $"<b>Minimum Score:</b> {minimumScoreText}, " +
                            $"<b>Maximum Score:</b> {maximumScoreText}, " +
                            $"<b>Average Score:</b> {averageScoreText}"
            });
        }
        return sessionUsers;
    }
    return new List<UserSessionResponse>();
}

/// <summary>
/// Adds the user retrieved from the Users database table by
/// the specified user ID parameter value if found and haven't been
/// added to users session before. Then sets the updated user list
/// in the authenticated user's session.
/// </summary>
/// <param name="userId">User ID to get the user from the
/// Users database table.</param>
public void Add(int userId)
{
    var user = _userService.Single(userId);
    if (user is not null)
    {
        var sessionUsers = Get();
        if (!sessionUsers.Any(su => su.UserId == user.Id
            && su.AuthenticatedUserId == _sessionService.AuthenticatedUserId))
        {
            sessionUsers.Add(new UserSessionResponse
            {
                AuthenticatedUserId = _sessionService.AuthenticatedUserId,
                UserId = user.Id,
                UserName = user.UserName,
                Score = user.Score,
                ScoreText = user.Score.ToString("N1")
            });
```

```
                    _sessionService.Set(KEY, sessionUsers);
                }
            }
        }

        /// <summary>
        /// Removes the user from the authenticated users's session
        /// then updates the user list in the session.
        /// </summary>
        /// <param name="userId">User ID to remove from the session.</param>
        public void Remove(int userId)
        {
            var sessionUsers = Get();
            var sessionUser = sessionUsers
                .FirstOrDefault(su => su.UserId == userId
                    && su.AuthenticatedUserId == _sessionService.AuthenticatedUserId);
            if (sessionUser is not null)
            {
                sessionUsers.Remove(sessionUser);
                _sessionService.Set(KEY, sessionUsers);
            }
        }

        /// <summary>
        /// Removes all users from the authenticated user's session
        /// then updates the user list in the session.
        /// </summary>
        public void Clear()
        {
            var sessionUsers = Get();
            sessionUsers.RemoveAll(
                su => su.AuthenticatedUserId == _sessionService.AuthenticatedUserId);
            _sessionService.Set(KEY, sessionUsers);
        }
    }
}
```

C) MVC Project for Session

1. In the IoC Container of Program.cs, add the sections marked as Session as below:

```
// -------
// Session
// -------
// Register session services and configure session options.
// Sets the session idle timeout to 30 minutes (default 20 minutes);
// if no activity occurs within this period, the session expires.
// Enables storing user-specific data (e.g., shopping cart, temporary users)
// across multiple requests during the session lifetime.
// Required for features that rely on session state, such as
// UserSessionService and SessionService.
builder.Services.AddSession(config =>
{
    config.IdleTimeout = TimeSpan.FromMinutes(30);
});
```

```
// -------
// Session
// -------
// Register the concrete SessionService as a scoped dependency.
// Concrete classes can also be used for registration, but using an
// abstract base class or interface allows for more flexible
// dependency injection and testing (Also better for SOLID Principles).
// Since session operations are standard and are not expected to change,
// concrete class injection is implemented.
// Scoped lifetime ensures a new instance per HTTP request,
// which is required for services accessing HttpContext.
// SessionService handles session management.
// This service registration enables constructor injection of SessionService
// to the controllers or services throughout the application.
builder.Services.AddScoped<SessionService>();

// -------
// Session
// -------
// Register the concrete UserSessionService as a scoped dependency.
// Scoped lifetime ensures a new instance per HTTP request.
// UserSessionService handles temporary users management.
// This service registration enables constructor injection of UserSessionService
// to the controllers or services throughout the application.
builder.Services.AddScoped<UserSessionService>();


// -------
// Session
// -------
// Enable session middleware in the HTTP request pipeline.
app.UseSession();
```

2. Create UserSessionController empty controller in the Controllers folder and implement as below:

```
using APP.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace MVC.Controllers
{
    /// <summary>
    /// Controller for managing temporary users session data
    /// for authenticated users.
    /// Provides actions to view, add, remove and clear temporary
    /// users session data.
    /// </summary>
    [Authorize]
    // Only authenticated users can execute this controller's actions.
    public class UserSessionController : Controller
    {
        // Constructor injection of the concrete UserSessionService instance.
        private readonly UserSessionService _userSessionService;
```

```
        public UserSessionController(UserSessionService userSessionService)
        {
            _userSessionService = userSessionService;
        }

        /// <summary>
        /// Displays the temporary users for the current authenticated user.
        /// </summary>
        /// <returns>Index view using the temporary user list model.</returns>
        public IActionResult Index() => View(_userSessionService.Get());

        /// <summary>
        /// Clears all temporary users for the current authenticated user.
        /// Sets a message and redirects to the Index action.
        /// </summary>
        /// <returns>Redirection to the Index action.</returns>
        public IActionResult Clear()
        {
            _userSessionService.Clear();
            TempData["Message"] = "All temporary users cleared.";
            return RedirectToAction(nameof(Index));
        }

        /// <summary>
        /// Removes a specific user from the temporary users for the
        /// authenticated user.
        /// Sets a message and redirects to the Index action.
        /// </summary>
        /// <param name="userId">The user ID of the user to remove.</param>
        /// <returns>Redirection to the Index action.</returns>
        public IActionResult Remove(int userId)
        {
            _userSessionService.Remove(userId);
            TempData["Message"] = "Temporary user removed.";
            return RedirectToAction(nameof(Index));
        }

        /// <summary>
        /// Adds a specific user to the temporary users for the authenticated user.
        /// Sets a message and redirects to the Index action
        /// of the Users controller.
        /// </summary>
        /// <param name="userId">The user ID of the user to add.</param>
        /// <returns>Redirection to the Users controller Index action.</returns>
        public IActionResult Add(int userId)
        {
            _userSessionService.Add(userId);
            TempData["Message"] = "Temporary user added.";
            return RedirectToAction("Index", "Users");
        }
    }
}
```

3. Add an empty Index view in the Views/UserSession folder and implement as below:

```
@model IEnumerable<UserSessionResponse>

@{
    ViewBag.Title = "Temporary Users";
}
<h1>@ViewBag.Title</h1>
```

```
@if (TempData["Message"] is not null)
{
    <p class="bg-warning text-dark">
        @TempData["Message"]
    </p>
}
@if (Model.Any()) @* if (Model.Count() > 0) can also be written. *@
{
    <table class="table">
        <tr>
            <td colspan="4">
                <a asp-action="Clear">Clear Temp Users</a>
            </td>
        </tr>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.UserName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.ScoreText)
            </th>
            <th></th>
        </tr>
        @for (int i = 0; i < Model.Count(); i++)
        {
            // Display all items except the last score summary item
            // which will be displayed separately below.
            if (i < Model.Count() - 1)
            {
                var item = Model.ElementAt(i);
                <tr>
                    <td>
                        @item.UserName
                    </td>
                    <td>
                        @item.ScoreText
                    </td>
                    <td>
                        <a asp-action="Remove" asp-route-userId="@item.UserId">
                            Remove from Temp Users
                        </a>
                    </td>
                </tr>
            }
        }
        @{
            // Display the last item which is the score summary.
            var lastItem = Model.Last();
            <tr>
                <td colspan="3" class="bg-black text-white">
                    @Html.Raw(lastItem.ScoreText)
                    @* Since lastItem.ScoreText contains HTML. *@
                </td>
            </tr>
        }
    </table>
}
else
{
    <p class="bg-success text-white">No temporary users.</p>
}
```

4. In Views/Users/Index.cshtml, create the link “Add to Temp Users” for adding the temporary user to the temporary users before the Details, Edit and Delete links as below:

```
// For temporary users session.
<a asp-action="Add" asp-controller="UserSession"
    asp-route-userId="@item.Id">
    Add to Temp Users
</a>
@: | 
```

5. Add Temp Users link at the end of the left top menu links in the layout view as below:

```
// For temporary users session.
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
        asp-controller="UserSession" asp-action="Index">Temp Users</a>
</li>
```

VI) Configuration in MVC Project

1. appsettings.json file can be used to store some configuration data for the application.

2. Add Title and App sections under the ConnectionStrings section in appsettings.json as below:

```
"Title": "Users MVC",
 "App": {
   "Version": "v1",
   "Date": "2026/08/06"
},
```

3. IConfiguration injected instance can be used in controllers or views to read configuration values from appsettings.json such as in Views/Shared/_Layout.cshtml and Views/Home/Index.cshtml.

Views/Shared/_Layout.cshtml:

On top of the view implement the following:

```
@{
    @inject IConfiguration configuration
    // This injects the IConfiguration service directly into the view making
    // configuration settings (appsettings.json) accessible without controller
    // dependencies. IConfiguration service may also be injected to the controllers.
    // configuration is the local variable name that will be used in this view
    // such as to reach the Title section of appsettings.json, configuration["Title"]
    // will be used as below.
}
```

Change the title HTML tag of the head HTML tag text as below:

```
<title>@ViewData["Title"] - @configuration["Title"]</title>
@* Changed from @ViewData["Title"] - Users MVC to use Title in appsettings.json *@
```

Change the link with class “navbar-brand” in the top menu as below:

```
<a class="navbar-brand" asp-area="" asp-controller="Home"
    asp-action="Index">@configuration["Title"]</a>
@* Changed from
   <a class="navbar-brand" asp-area=""
        asp-controller="Home" asp-action="Index">Users MVC</a>
   to use Title in appsettings.json
*@
```

Views/Home/Index.cshtml:

On top of the view implement the following:

```
@{
    @inject IConfiguration configuration
    // This injects the IConfiguration service directly into the view making
    // configuration settings (appsettings.json) accessible without controller
    // dependencies. IConfiguration service may also be injected to the controllers.
    // configuration is the local variable name that will be used in this view
    // such as to reach the Title section of appsettings.json, configuration["Title"]
    // will be used as below. configuration["App"] will also be used to reach the
    // App section of appsettings.json.
}
```

Change the h1 HTML tag text as below:

```
<h1 class="display-4">Welcome to @configuration["Title"]</h1>
@*
    Changed from
    <h1 class="display-4">Welcome to Users MVC</h1>
    to use Title in appsettings.json
*@
```

Add the p HTML tag at the bottom of the view as below:

```
@*
    Added to show App section's Version and Date
    values in appsettings.json:
*@
<p class="text-center">
    Version: @configuration["App:Version"]
    <br />
    Date: @configuration["App:Date"]
</p>
```

VII) Layouts in MVC Project

Optionally you can use the Sneat Web Template for your web application to look better: https://need4code.com/DotNet/Home/Index?path=.NET%5C00_Files%5CWeb%20Templates%5CSneat.7z

Extract the folders in the compressed file to your MVC Project.

Then modify the Views/Shared/_SneatLayout.cshtml according to your project.

Finally, change the Layout assignment in the _ViewStart.cshtml from Layout = "_Layout" to Layout = "_SneatLayout" for your views to use.

Views/Shared/_SneatLayout.cshtml:

```
<!-- Source: /wwwroot/sneat/html/layouts-fluid.html -->

@{
    @inject IConfiguration configuration
    // This injects the IConfiguration service directly into the view making
    // configuration settings (appsettings.json) accessible without controller
    // dependencies. IConfiguration service may also be injected to the controllers.
    // configuration is the local variable name that will be used in this view
    // such as to reach the Title section of appsettings.json, configuration["Title"]
    // will be used as below.
}

<!doctype html>

<html lang="en"
      class="light-style layout-menu-fixed layout-wide"
      dir="ltr"
      data-theme="theme-default"
      data-assets-path="/sneat/assets/"
      data-template="vertical-menu-template-free"
      data-style="light">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
        user-scalable=no, minimum-scale=1.0, maximum-scale=1.0" />

    <title>@ViewData["Title"] | @configuration["Title"]</title>

    <meta name="description" content="" />

    <!-- Favicon -->
    <link rel="icon" type="image/x-icon" href="/sneat/assets/img/favicon/favicon.ico" />

    <!-- Fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link href="https://fonts.googleapis.com/css2?family=Public+Sans:ital,wght@0,300;0,400;
        0,500;0,600;0,700;1,300;1,400;1,500;1,600;1,700&display=swap"
        rel="stylesheet" />
```

```
    <link rel="stylesheet" href="/sneat/assets/vendor/fonts/boxicons.css" />

    <!-- Core CSS -->
    <link rel="stylesheet" href="/sneat/assets/vendor/css/core.css"
        class="template-customizer-core-css" />
    <link rel="stylesheet" href="/sneat/assets/vendor/css/theme-default.css"
        class="template-customizer-theme-css" />
    <link rel="stylesheet" href="/sneat/assets/css/demo.css" />

    <!-- Vendors CSS -->
    <link rel="stylesheet" href="/sneat/assets/vendor/libs/perfect-scrollbar/
        perfect-scrollbar.css" />

    <!-- Page CSS -->
    <!-- Helpers -->
    <script src="/sneat/assets/vendor/js/helpers.js"></script>

    <!--! Template customizer & Theme config files MUST be included after core stylesheets
        and helpers.js in the <head> section -->
    <!--? Config:  Mandatory theme config file contain global vars & default theme options,
        Set your preferred theme option in this file.  -->
    <script src="/sneat/assets/js/config.js"></script>

</head>

<body>

    <!-- Layout wrapper -->
    <div class="layout-wrapper layout-content-navbar">
        <div class="layout-container">

            <!-- Menu -->
            <aside id="layout-menu" class="layout-menu menu-vertical menu bg-menu-theme">
                <div class="app-brand demo">
                    <a asp-area="" asp-action="Index" asp-controller="Home"
                        class="app-brand-link">
                        <i class="menu-icon tf-icons bx bx-coffee"></i>
                        <span class="app-brand-text demo menu-text fw-bold ms-2 fs-4">
                            @configuration["Title"]
                        </span>
                    </a>
                    <a href="javascript:void(0);" class="layout-menu-toggle menu-link
                        text-large ms-auto d-block d-xl-none">
                        <i class="bx bx-chevron-left bx-sm d-flex align-items-center
                            justify-content-center"></i>
                    </a>
                </div>
                <div class="menu-inner-shadow"></div>
                <ul class="menu-inner py-1">
                    <li class="menu-header small text-uppercase">
                        <span class="menu-header-text">Menu</span>
                    </li>
                    <li class="menu-item">
                        <a asp-area="" asp-action="Index" asp-controller="Statuses"
                            class="menu-link">
                            <i class="menu-icon tf-icons bx bxs-shield"></i>
                            <div class="text-truncate">Statuses</div>
                        </a>
                    </li>
                    @if (User.Identity.IsAuthenticated)
                    {
                        <li class="menu-item">
                            <a asp-area="" asp-action="Index" asp-controller="Users"
                                class="menu-link">
                                <i class="menu-icon tf-icons bx bxs-group"></i>
                                <div class="text-truncate">Users</div>
                            </a>
                        </li>
                        @if (User.IsInRole("Admin"))
                        {
                            <li class="menu-item">
                                <a asp-area="" asp-action="Index" asp-controller="Roles"
                                    class="menu-link">
                                    <i class="menu-icon tf-icons bx bxs-user-account"></i>
                                    <div class="text-truncate">Roles</div>
                                </a>
```

```
                </li>
            }
        }
    </ul>
</aside>
<!-- / Menu -->

<!-- Layout container -->
<div class="layout-page">

    <!-- Navbar -->
    <nav class="layout-navbar container-fluid navbar navbar-expand-xl
        navbar-detached align-items-center bg-navbar-theme"
        id="layout-navbar">
        <div class="layout-menu-toggle navbar-nav align-items-xl-center
            me-4 me-xl-0 d-xl-none">
            <a class="nav-item nav-link px-0 me-xl-6" href="javascript:void(0)">
                <i class="bx bx-menu bx-md"></i>
            </a>
        </div>
        <div class="navbar-nav-right d-flex align-items-center"
            id="navbar-collapse">
            <div class="navbar-nav align-items-center">
                <div class="nav-item d-flex align-items-center">
                    @if (User.Identity.IsAuthenticated)
                    {
                        <a asp-area="" asp-action="Index"
                            asp-controller="UserSession">
                            Temp Users
                        </a>
                    }
                </div>
            </div>
            <ul class="navbar-nav flex-row align-items-center ms-auto">

                <!-- User -->
                @if (User.Identity.IsAuthenticated)
                {
                    <li class="nav-item">
                        @* Way 1: *@
                        @*
                            <i class="bx bx-user bx-md me-1"></i>
                            <span>@User.Identity.Name</span>
                        *@
                        @*
                            Way 2: optional link to direct the user
                            to his/her user details
                        *@
                        <a class="nav-link ps-3" asp-area="" asp-action="Details"
                            asp-controller="Users"
                            asp-route-id="@User.Claims.SingleOrDefault(
                                claim => claim.Type == "Id")?.Value">
                            @*
                                Getting the user's Id from the claims to direct
                                the user to the user details with id route value
                            *@
                            <i class="bx bx-user bx-md me-1"></i>
                            <span>@User.Identity.Name</span>
                        </a>
                    </li>
                    <li class="nav-item">
                        <a asp-area="" asp-action="Logout" asp-controller="Auth"
                            class="nav-link ps-3">
                            <i class="bx bx-log-out bx-md me-1"></i>
                            <span>Logout</span>
                        </a>
                    </li>
                }
                else
                {
                    <li class="nav-item">
                        <a asp-area="" asp-action="Register" asp-controller="Auth"
                            class="nav-link">
                            <i class="bx bx-user-plus bx-md me-1"></i>
                            <span>Register</span>
                        </a>
                    </li>
```

```
                            <li class="nav-item">
                                <a asp-area="" asp-action="Login" asp-controller="Auth"
                                    class="nav-link ps-3">
                                    <i class="bx bx-log-in bx-md me-1"></i>
                                    <span>Login</span>
                                </a>
                            </li>
                        }
                        <!--/ User -->

                    </ul>
                </div>
            </nav>
            <!-- / Navbar -->

            <!-- Content wrapper -->
            <div class="content-wrapper">

                <!-- Content -->
                <div class="container-fluid flex-grow-1 container-p-y">
                    @RenderBody()
                </div>
                <!-- / Content -->

                <!-- Footer -->
                <footer class="content-footer footer bg-footer-theme">
                    <div class="container-fluid">
                        <div class="footer-container d-flex align-items-center
                            justify-content-between py-4 flex-md-row flex-column">
                            <div class="text-body">
                                <a asp-area="" asp-controller="Home" asp-action="Privacy"
                                    class="footer-link">Privacy</a>
                                 | 
                                <a href="/sneat/html/index.html" target="_blank"
                                    class="footer-link">Sneat</a>
                            </div>
                            <div class="d-none d-lg-inline-block">
                                <a href="https://themeselection.com/" target="_blank">
                                    ThemeSelection</a>
                            </div>
                        </div>
                    </div>
                </footer>
                <!-- / Footer -->

                <div class="content-backdrop fade"></div>
            </div>
            <!-- Content wrapper -->

        </div>
        <!-- / Layout page -->

    </div>

    <!-- Overlay -->
    <div class="layout-overlay layout-menu-toggle"></div>

</div>
<!-- / Layout wrapper -->

<!-- Core JS -->
<!-- build:js assets/vendor/js/core.js -->

<script src="/sneat/assets/vendor/libs/jquery/jquery.js"></script>
<script src="/sneat/assets/vendor/libs/popper/popper.js"></script>
<script src="/sneat/assets/vendor/js/bootstrap.js"></script>
<script src="/sneat/assets/vendor/libs/perfect-scrollbar/perfect-scrollbar.js"></script>
<script src="/sneat/assets/vendor/js/menu.js"></script>

<!-- endbuild -->
<!-- Vendors JS -->
<!-- Main JS -->
<script src="/sneat/assets/js/main.js"></script>

<!-- Page JS -->
<!-- Place this tag before closing body tag for github widget button. -->
<script async defer src="https://buttons.github.io/buttons.js"></script>
```

```
    <script>
        $(function () {
            $("#layout-menu ul.menu-inner li.menu-item").each(function () {
                if ($(this).children("a.menu-link")[0].pathname.split("/")[1] ==
                    window.location.pathname.split("/")[1]) {
                    $(this).addClass("active");
                    $(this).parent("ul.menu-sub").parent("li.menu-item").addClass("open");
                } else {
                    $(this).removeClass("active");
                }
            });
        });
    </script>

    @await RenderSectionAsync("Scripts", required: false)

</body>
</html>
```

VIII) Additional Information

1. The complete implementation of the Program.cs in the MVC Project:

```
using APP.Authentication.Services;
using APP.Domain;
using APP.Models;
using APP.Services;
using CORE.Authentication.Services;
using CORE.Services;
using CORE.Session.Services;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.EntityFrameworkCore;

// Create a builder for the web application and initialize
// configuration, logging, etc.
var builder = WebApplication.CreateBuilder(args);



// ------------------------------------------------------
// Add services to the IoC (Inversion of Control) container
// for Dependency Injections.
// ------------------------------------------------------
// Register the application's DbContext dependency
// injection by sending Db instances to the injected service
// class constructors' DbContext or Db parameter, using SQLite
// with the connection string from appsettings.json.
builder.Services.AddDbContext<DbContext, Db>(
    options => options.UseSqlite(
        builder.Configuration.GetConnectionString(nameof(Db))));
// nameof(Db) = "Db" which is the class name.

// Register the UserService dependency injection by sending
// IService<UserRequest, UserResponse> instance reference to
// the injected controller class constructor's
// IService<UserRequest, UserResponse> parameter.
// Injection through abstract class or interface is suitable
// for SOLID Principles.
builder.Services
    .AddScoped<IService<UserRequest, UserResponse>, UserService>();

// Register the RoleService dependency injection by sending
// IService<RoleRequest, RoleResponse> instance reference to
// the injected controller class constructor's
// IService<RoleRequest, RoleResponse> parameter.
builder.Services
    .AddScoped<IService<RoleRequest, RoleResponse>, RoleService>();

// Register the StatusService dependency injection by sending
// IService<StatusRequest, StatusResponse> instance reference to
// the injected controller class constructor's
// IService<StatusRequest, StatusResponse> parameter.
builder.Services
    .AddScoped<IService<StatusRequest, StatusResponse>, StatusService>();
```

```
// --------------------------
// Authentication and Session
// --------------------------
// Register IHttpContextAccessor as a singleton service.
// Enables access to the current HttpContext from classes other than
// controller classes such as services.
// Required for services like CookieAuthService that need to read
// or modify HTTP context (e.g., authentication, user info).
// Allows constructor injection of IHttpContextAccessor throughout
// the services of the application.
builder.Services.AddHttpContextAccessor();



// --------------
// Authentication
// --------------
// Register CookieAuthService as a scoped dependency for ICookieAuthService.
// Scoped lifetime ensures a new instance per HTTP request, which is
// required for services accessing HttpContext.
// CookieAuthService handles user authentication using cookies, supporting
// sign in and sign out operations.
// This service registration enables constructor injection of
// ICookieAuthService to the controllers or services throughout the application.
builder.Services.AddScoped<ICookieAuthService, CookieAuthService>();



// --------------
// Authentication
// --------------
// Register AuthService as a scoped dependency for IAuthService.
// Scoped lifetime ensures a new instance per HTTP request, which is
// required for services accessing HttpContext.
// AuthService handles user authentication using injected ICookieAuthService
// instance, supporting log in, log out and register operations.
// This service registration enables constructor injection of
// IAuthService to the controllers or services throughout the application.
builder.Services.AddScoped<IAuthService, AuthService>();



// --------------
// Authentication
// --------------
// Configure authentication services using cookie-based authentication.
// - Sets the default authentication scheme to Cookies.
// - Specifies options for cookie authentication:
//   - LoginPath: Path to redirect unauthenticated users
//     (when authentication is required).
//   - AccessDeniedPath: Path to redirect users who are authenticated
//     but lack required permissions.
//   - ExpireTimeSpan: Duration before the authentication cookie expires
//     (here, 1 hour).
//   - SlidingExpiration: Resets the expiration time on each request
//     to keep active sessions alive.
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Login";
        // changed from /Auth/Login to /Login since route will be changed for action
        options.AccessDeniedPath = "/Login";
        // changed from /Auth/Login to /Login since route will be changed for action
        options.ExpireTimeSpan = TimeSpan.FromHours(1);
        options.SlidingExpiration = true;
    });
```

```
// -------
// Session
// -------
// Register session services and configure session options.
// Sets the session idle timeout to 30 minutes (default 20 minutes);
// if no activity occurs within this period, the session expires.
// Enables storing user-specific data (e.g., shopping cart, temporary users)
// across multiple requests during the session lifetime.
// Required for features that rely on session state, such as
// UserSessionService and SessionService.
builder.Services.AddSession(config =>
{
    config.IdleTimeout = TimeSpan.FromMinutes(30);
});



// -------
// Session
// -------
// Register the concrete SessionService as a scoped dependency.
// Concrete classes can also be used for registration, but using an
// abstract base class or interface allows for more flexible
// dependency injection and testing (Also better for SOLID Principles).
// Since session operations are standard and are not expected to change,
// concrete class injection is implemented.
// Scoped lifetime ensures a new instance per HTTP request,
// which is required for services accessing HttpContext.
// SessionService handles session management.
// This service registration enables constructor injection of SessionService
// to the controllers or services throughout the application.
builder.Services.AddScoped<SessionService>();



// -------
// Session
// -------
// Register the concrete UserSessionService as a scoped dependency.
// Scoped lifetime ensures a new instance per HTTP request.
// UserSessionService handles temporary users management.
// This service registration enables constructor injection of UserSessionService
// to the controllers or services throughout the application.
builder.Services.AddScoped<UserSessionService>();



// Add support for controllers and views (MVC pattern).
builder.Services.AddControllersWithViews();

// Build the application.
var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    // Use a custom error handler in non-development environments
    // redirecting to Home controller's Error action.
    app.UseExceptionHandler("/Home/Error");

    // Enable HTTP Strict Transport Security (HSTS).
    // The default HSTS value is 30 days. You may want to change this
    // for production scenarios, see https://aka.ms/aspnetcore-hsts.
```

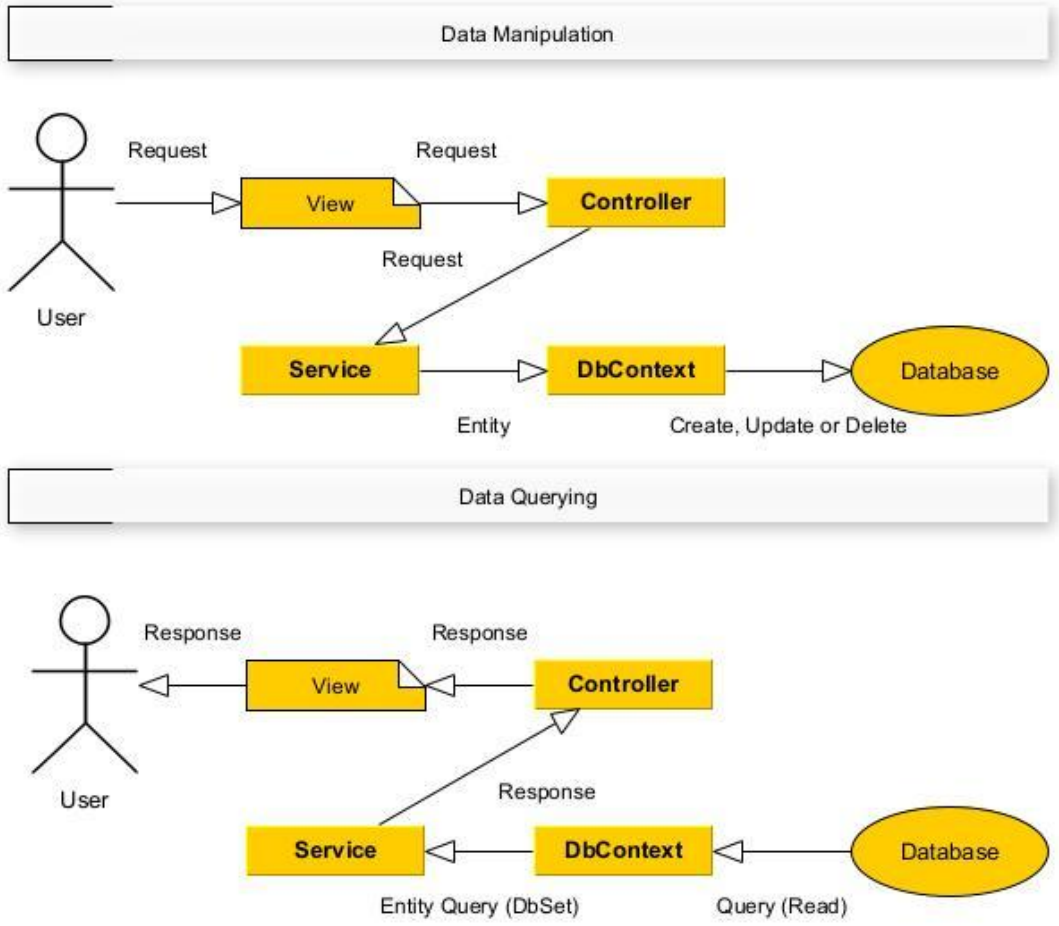
```
    app.UseHsts();
}

// Redirect HTTP requests to HTTPS.
app.UseHttpsRedirection();

// Enable serving static files (e.g., HTML, CSS, JS, images).
app.UseStaticFiles();

// Enable routing capabilities.
app.UseRouting();



// ---------------
// Authentication
// ---------------
// Enable authentication middleware so that [Authorize] works.
app.UseAuthentication();



// Enable authorization middleware.
app.UseAuthorization();



// -------
// Session
// -------
// Enable session middleware in the HTTP request pipeline.
app.UseSession();



// Configure the default route for controllers for sending requests
// to controllers' actions as
// controller/action/id where id is optional.
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

// Start the application and listen for incoming HTTP requests.
app.Run();
```

2. A simplified version of the N-Layered Architecture and Repository Pattern that include the basic concepts and structures are applied to the project. More information about N-Layered Architecture and other architectures can be found at:

   https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

3. Domain is for data access from a database, services are for business logic and MVC is for presentation. CQRS (Command Query Response Segregation) Pattern is also applied to this project.

4. MVC data querying and manipulation:



5. Controller classes have action methods that handle the incoming HTTP get or post requests, optionally interact with entity data in the database through service classes using model (DTO: Data Transfer Object) classes and generally return views with or without model class data.

6. ActionResult inheritance:

   IActionResult : general return type of actions in a controller
   |
   ActionResult: base class that implements IActionResult
   |
   ViewResult (returned by View method) - ContentResult (returned by Content method) - NotFoundResult (returned by NotFound method) - RedirectResults - HttpStatusCodeResults - JsonResult - etc.

7. You can right-click an action to go to its view with the same name such as Index action of the HomeController to go to the Index view (Index.cshtml) in Views/Home folder and Privacy action of the HomeController to go to the Privacy view (Privacy.cshtml) in Views/Home folder. You should change the HTML content in these views according to your project.

8. Razor syntax provides the usage of HTML and C# together in views:
   In order to switch to C# in HTML, @ is used.

   C# code blocks can be written in @{ ... }, for example
   ```
   @{
     var name = "Cagil Alsac";
   }
   ```
   and the value of the name variable can be printed in HTML like
   ```
   <h1>@name</h1>
   ```

   Other C# code blocks such as if, foreach, etc. can be written like
   ```
   @if (true)
   {
     <label>Success.</label>
   }
   else
   {
     <label>Error!</label>
   }

   @foreach (string item in list)
   {
     <p>@item</p>
   }
   ```

9. The folder names in the Views folder correspond to the controller names and the view file names correspond to the action names if view name is not changed in the returned View method. If view name is changed in the returned View method, the view file name must also be changed accordingly.

   The returned view of an action is first searched under Views/Home folder, then Views/Shared folder, and the view is returned if found. Therefore, views in the Shared folder can be returned from any controller's any action.
10. ViewData and ViewBag are the same collection (dictionary). They carry extra data other than the model object from a controller action to its view, or between views.

    TempData carries data from an action to the redirected action, therefore to the redirected action's view.

11. Layout view in Views/Shared folder is like a master page. It contains the common HTML elements such as head, body, Bootstrap navbar (top menu), footer, etc. The RenderBody C# method in the body tag prints the content of the returned views. The links for directing to the created contollers' actions can be added in the Bootstrap navbar.

12. Layout of a view may be changed at the top code block with one of the below assignments:

```
@{
    Layout = "_Layout";
    // _Layout.cshtml in Views/Shared folder, no need to write because defined in
    // _ViewStart.cshtml under Views folder

    Layout = "~/Views/Shared/_Layout.cshtml"; // can also be written

    Layout = null; // for no layout

    Layout = "_CustomLayout";
    // if _CustomLayout.cshtml in Views/Shared folder is created to be used with the view
}
```

13. Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor views. They look like standard HTML tags but add special attributes such as asp-controller, asp-action, asp-area, asp-route, asp-for, etc. that are processed on the server to generate dynamic content.

14. The default MVC (Model-View-Controller) route to execute controllers' actions is defined in Program.cs file of the MVC Project as: "{controller=Home}/{action=Index}/{id?}" where id value is optional, default action value is Index and default controller value is Home.

15. Getters and Setters in C#:

```
public abstract class Record
{
    // Way 1:
    // Field to store the record's ID.
    //private int id; // variable, field

    // Method to get the record's ID.
    //public int GetId() // function, method, behavior
    //{
    //    return id;
    //}

    // Method to set the record's ID.
    //public void SetId(int id)
    //{
    //    this.id = id;
    //}


    // Way 2:
    public int Id { get; set; }
}
```

16. Enums in C#:

```
namespace APP.Domain
{
    // Enums are used for predefined values and help not to remember or
    // check each element's value when being used.
    // Getting the value of an enum element:
    // int womanValue = (int)Genders.Woman;
    // Getting the text of an enum element:
    // string manText = Genders.Man.ToString();
    // Way 1:
    //public enum Genders
    //{
    //    Woman = 1, // will start from 1
    //    Man = 2 // will get the value 2 and other elements will continue
    //            // to get the next values after 2
    //}


    // Way 2:
    //public enum Genders
    //{
    //    Woman = 1, // will start from 1
    //    Man // will automatically get the next value 2 and other elements
    //        // will continue to get the next values after 2
    //}


    // Way 3:
    public enum Genders
    {
        Woman, // if no assignment, will start from 0
        Man // will automatically get the next value 1 and other elements
            // will continue to get the next values after 1
    }
}
```

17. Service Lifetimes in ASP.NET Core Dependency Injection:

1) AddScoped:
- Lifetime: Scoped to a single HTTP request (or scope).
- Behavior: Creates one instance of the service per HTTP request.
- Use case: Use when you want to maintain state or dependencies that last only during a single request.
- Example: DbContext, which should be shared across operations within a request, generally added with AddDbContext method.

2) AddSingleton:
- Lifetime: Singleton for the entire application lifetime.
- Behavior: Creates only one instance of the service for the whole app lifecycle.
- Use case: Use for stateless services or global shared data/services.
- Example: Caching services, configuration providers, logging services.

3) AddTransient:
- Lifetime: Transient (short-lived).
- Behavior: Creates a new instance every time the service is requested.
- Use case: Use for lightweight, stateless services that are cheap to create.
- Example: Utility/helper classes without state.

Notes:
- Injecting a Scoped service into a Singleton can cause issues due to lifetime mismatch.
- ASP.NET Core DI container will warn about such mismatches.

```
Summary:
| Method        | Lifetime                | Instance Created              | Typical Use Case                  |
|---------------|-------------------------|-------------------------------|-----------------------------------|
| AddScoped     | Per HTTP request        | One instance per request      | DbContext, per-request services   |
| AddSingleton  | Application-wide        | One instance for app lifetime | Caching, config, logging          |
| AddTransient  | Every time requested    | New instance each time        | Lightweight stateless helpers     |
```

18. SOLID Principles:
    1) Single Responsibility Principle (SRP):
       A class should have only one reason to change, meaning it should have only one job or responsibility.
    2) Open/Closed Principle (OCP):
       Software entities (classes, modules, functions) should be open for extension but closed for modification. You should be able to add new functionality without changing existing code.
    3) Liskov Substitution Principle (LSP):
       Subtypes must be substitutable for their base types. Derived classes should extend base classes without changing their behavior.
    4) Interface Segregation Principle (ISP):
       No client should be forced to depend on methods it does not use. Prefer small, specific interfaces over large, general-purpose ones.
    5) Dependency Inversion Principle (DIP):
       High-level modules should not depend on low-level modules; both should depend on abstractions (e.g., interfaces). This is commonly implemented in ASP.NET Core using dependency injection, as seen in Program.cs.

       Note: Concrete type object injection is not suitable for SOLID Principles (D: Dependency Inversion) and Unit Testing.

19. ASP.NET Core Environments:

    The environment is development when the MVC application is run from Visual Studio choosing a development profile from the launchSettings.json file in the Properties folder of the MVC Project.
    When the MVC application is run on a server or from Visual Studio choosing a production profile from the launchSettings.json file, the environment is production.

In launchSettings.json, http profile was defined as Development through ASPNETCORE_ENVIRONMENT section, https profile's ASPNETCORE_ENVIRONMENT value was changed to Production from Development for using the production environment.
The environments can be changed from the drop down list (selected as https) near the run button under the Visual Studio top menu when running the application.
Before, MVC must be set as startup project from the drop down list at left of the run button.
If https (production environment) is selected, sections in appsettings.json will be used for configuration, if http (development environment) is selected, sections in appsettings.Development.json will be used for configuration.
Therefore, same sections with some different values (such as connection string) must present in both files.
Environment configurations and usage is not a must for the development of applications.

IWebHostEnvironment injected instance can be used to get the web application's running environment information such as development environment or not by using IsDevelopment method.

20. var in C#:
    var can be used instead of types if there is an assignment.

```
// Way 1:
// User user = new User();
// Way 2:
var user = new User();

// Way 1:
//int number = 17;
// Way 2:
var number = 17;

// Way 1:
//string role = "Admin";
// Way 2:
var role = "Admin";
```

21. Different Generic Type implementations for classes (can also be applied to interfaces):
    Multiple different types (T1, T2, T3, ...) can also be used for example:
    public abstract class DbService<T1, T2> where T1 : class, new() where T2 : class, new()

```
// Way 1: T can be any type
public abstract class DbService<T>

// Way 2: T must be only a reference type
public abstract class DbService<T> where T : class

// Way 3: T must be only a reference type
// with a parameterless constructor.
public abstract class DbService<T> where T : class, new()

// Way 4: T must be only a type inherited from the Record class
// with a parameterless constructor.
public abstract class DbService<T> where T : Record, new()
```

22. Attributes in C#:

Attributes gain new features to the fields, properties, methods or classes. When they are used in entities or requests, they are also called data annotations which provide data validations.

Some commonly used data annotation attributes in C#:

[Required] // Ensures the property must have a value.

[StringLength] // Sets maximum (and optionally minimum) length for strings.

[Length] // Sets maximum and minimum length for strings.

[MinLength] // Specifies the minimum length for strings or collections.

[MaxLength] // Specifies the maximum length for strings or collections.

[Range] // Defines the allowed range for numeric values.

[RegularExpression] // Validates the property value against a regex pattern.

[EmailAddress] // Validates that the property is a valid email address.

[Phone] // Validates that the property is a valid phone number.

[Url] // Validates that the property is a valid URL.

[Compare] // Compares two properties for equality (e.g., password confirmation).

[DisplayName] // Sets a friendly name for the property (used in error messages/UI).

[DataType] // Specifies the data type (e.g., DateTime) for formatting/UI hints.

ErrorMessage parameter can be set in all data annotations to show custom validation error messages:

Example 1: [Required(ErrorMessage = "{0} is required!")]

where {0} is the DisplayName (used in MVC) if defined otherwise property name.

Example 2: [StringLength(100, 3, ErrorMessage = "{0} must be minimum {2} maximum {1} characters!")]

where {0} is the DisplayName (used in MVC) if defined otherwise property name, {1} is the first parameter which is 100 and {2} is the second parameter which is 3.

23. Service classes are business logic classes that first get entity data from the database through the database class (Db), which inherits Entity Framework's DbContext class for data access, convert the data to the response model object and return the response model object to the controller action for presentation (Query method).

    Secondly, service classes get request model data from the controller action, convert the data to the entity object for create and update operations, or use unique identifier (ID) for delete operation, in the database. Then they return a command response model object to the controller action to present the result of the operation (Add, Update and Remove methods).

24. Request and response model classes are also called Data Transfer Object (DTO) classes.

    Synchronous methods execute tasks one after another. Each operation must complete before the next one starts. The calling thread waits (or "blocks") until the method finishes.

    Asynchronous methods allow tasks to run in the background. The calling thread does not wait for the operation to finish and can continue executing other code. In C#, asynchronous methods often use the async and await keywords, enabling non-blocking operations (such as I/O or database calls) and improving application responsiveness.

25. Some LINQ (Language Integrated Query) methods for querying data (async versions already exists):

    Find: Finds an entity with the given primary key value. Returns null if not found.
    Uses the database context's cache before querying the database.
    Example: var group = _db.Groups.Find(5);
    Single: Returns the only element that matches the specified condition(s).
    Throws an exception if no element or more than one element is found.
    Example: var group = _db.Groups.Single(groupEntity => groupEntity.Id == 5);

    SingleOrDefault: Returns the only element that matches the specified condition(s), or null if no such element exists.
    Throws an exception if more than one element is found.
    Example: var group = _db.Groups.SingleOrDefault(groupEntity => groupEntity.Id == 5);

    First: Returns the first element that matches the specified condition(s).
    Throws an exception if no element is found.
    Example: var group = _db.Groups.First();
    Example: var group = _db.Groups.First(groupEntity => groupEntity.Id > 5 && groupEntity.Title.StartsWith("Jun");

    FirstOrDefault: Returns the first element that matches the specified condition(s), or null if no such element exists.
    Example: var group = _db.Groups.FirstOrDefault();
    Example: var group = _db.Groups.FirstOrDefault(groupEntity => groupEntity.Id < 5 || groupEntity.Title == "Senior");

    Last: Returns the last element that matches the specified condition(s).
    Throws an exception if no element is found. Usually requires an OrderBy or OrderByDescending clause.
    Example: var group = _db.Groups.OrderByDescending(groupEntity => groupEntity.Id).Last(); gets the first group from the groups descending ordered by Id.
    Example: var group = _db.Groups.OrderBy(groupEntity => groupEntity.Id).Last(); gets the last group from the groups ordered by Id.

LastOrDefault: Returns the last element that matches the specified condition(s), or null if no such element exists.
Usually requires an OrderBy or OrderByDescending clause.
Example: var group = _db.Groups.OrderBy(groupEntity => groupEntity.Id).LastOrDefault();
Example: var group = _db.Groups.OrderBy(groupEntity => groupEntity.Id).LastOrDefault(groupEntity.Title.Contains("io"));

Where: Returns the filtered query that matches the specified condition(s). Tolist, SingleOrDefault or FirstOrDefault methods are invoked to get the filtered data.
Example: var groups = _db.Groups.Where(groupEntity => groupEntity.Id > 5).ToList();

Note: SingleOrDefault is generally preferred to get single data.
Note: These LINQ methods can also be used with collections such as lists and arrays.

26. Some commonly used HTML Helpers in ASP.NET Core MVC:
We will use mostly Tag Helpers other than the HTML Helpers DisplayNameFor and DisplayFor. These helpers use lambda expressions to bind directly to model properties, providing compile-time safety and IntelliSense.

@Html.LabelFor(model => model.Property)
Generates a <label> for the specified property.

@Html.DisplayNameFor(model => model.Property)
Renders the display name of the property, using the [DisplayName] attribute if set, otherwise the property name.

@Html.TextBoxFor(model => model.Property)
Generates a <input type="text"> for the property.

@Html.TextAreaFor(model => model.Property)
Generates a <textarea> for the property.

@Html.EditorFor(model => model.Property)
Generates the most appropriate input element based on the property type and data annotations.

@Html.DisplayFor(model => model.Property)
Renders a display-only view of the property (e.g., as plain text).

@Html.CheckBoxFor(model => model.Property)
Generates a checkbox for boolean properties.

@Html.DropDownListFor(model => model.Property, selectList)
Generates a dropdown list for the property.

@Html.ListBoxFor(model => model.Property, multiSelectList)
Generates a multi-select list box for the property.

@Html.HiddenFor(model => model.Property)
Generates a hidden input field for the property.

@Html.PasswordFor(model => model.Property)
Generates a password input field for the property.

@Html.ValidationMessageFor(model => model.Property)
Displays validation error messages for the property.

@Html.Raw(string)
Renders raw HTML markup from a string (only use with trusted content to avoid XSS vulnerabilities).

27. Eager Loading and Lazy Loading with Entity Framework:

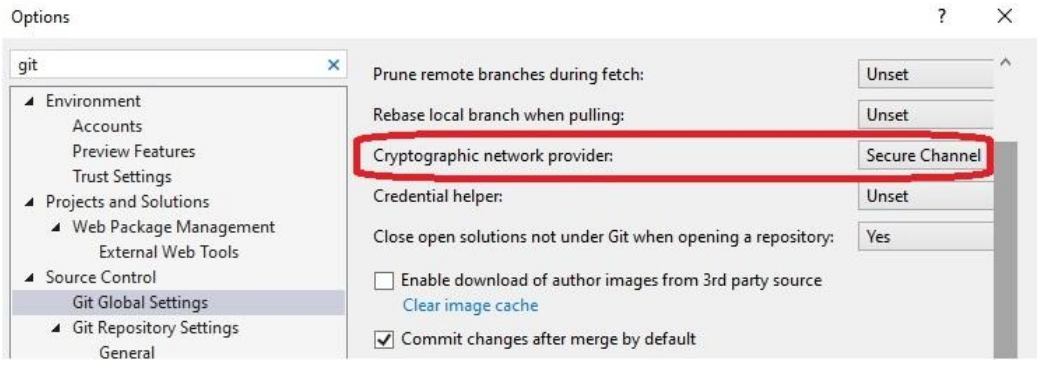
    Relational entity data can be included to the query by using the Include method (Entity Framework Eager Loading).

    If the included relational entity data has a relation with other entity data, ThenInclude method is used.

    If you want to automatically include all relational data without using Include / ThenInclude methods (Entity Framework Lazy Loading), you need to make the necessary configuration in the class inheriting from DbContext class (Db) to enable Lazy Loading (not recommended).

28. Visual Studio 2022 Community GitHub push problem solution:

From Visual Studio Menu
Tools -> Options -> Source Control -> Git Global Settings
Change Cryptographic network provider to Secure Channel



29. Html.AntiForgeryToken() in Create, Edit and Delete views generates a hidden form field containing a unique anti-forgery token to prevent Cross-Site Request Forgery (CSRF) attacks by ensuring that form submissions originate from the web application's views. [ValidateAntiForgeryToken] attribute defined for Create, Edit and Delete post actions in the controller rejects requests with missing or invalid anti-forgery tokens, thereby enhancing the security of the web application.

30. If you change the database table structures (such as adding or removing entity properties, changing entity property data types or data annotations, adding or removing entities and db sets) or relationships (such as changing Cascade delete rule to No Action) through the entities or the DbContext inheriting class, you need to update the database as below:

    - Open Package Manager Console from Visual Studio menu -> Tools -> NuGet Package Manager -> Package Manager Console
    - Right-click MVC Project and click Set as Startup Project
    - Set APP as Default Project in Package Manager Console
    - Run:
      add-migration v2
      update-database
    - For Rider, you can use the UI as described in JetBrains documentation, or from the terminal Run:
      dotnet ef migrations add v2
      dotnet ef database update -p APP -s MVC

31. Optionally database configurations such as using no action (will not allow to delete the records from the relational table when a record is deleted from the main table) instead of cascade (which will automatically delete the records from the relational table when a record is deleted from the main table) between tables may be done by overriding the OnModelCreating method in the Db database context class. Default is cascade in Entity Framework.
    Changing column configurations instead of using data annotations in entities (e.g. making a column required or setting maximum length), changing table names etc. can also be done in the OnModelCreating method among other configurations.