Software Development

Presented By:
Khaled Gamal

ASP.NET Core

MVC

Section 8

DATA ANNOTATION

Data Annotation

- What is Data Annotation?

Data annotation in ASP.NET Core MVC is a way to define metadata or attributes on model classes and properties to specify how they should be validated, displayed, or persisted in the database. It allows you to customize the data model by using attributes that specify formatting, validation, and database mapping rules. It allows you to enforce validation rules, define display names and descriptions, format data, and configure database mapping using attributes applied to model classes or properties. Data annotations are a declarative approach to define metadata for model classes and properties in the MVC application, making it easy to specify how data should be treated without writing extensive code

- To use Data Annotations, you must use this ***namespace***:

System.ComponentModel.DataAnnotations

The Key attribute

- [Key]: In ASP.NET Core MVC, This annotation is used to mark a property as the primary key of a model class. Or it denotes one or more properties that uniquely identify an entity. The primary key uniquely identifies each instance of the model in the database and is used as the unique identifier for retrieving, updating, and deleting records from the database.
- You can use the Key attribute if the entity does have its own primary key but you want to name the property something other than classnameID or ID.

C# (Students.cs)

```
public class Student
{
    [Key]
    public int ID { get; set; }
}
```

The Key attribute

- [PrimaryKey]: specifies a primary key for an entity. This attribute can be used for both keys made up of a single property and for composite keys made up of multiple properties. It can be found in *Microsoft.EntityFrameworkCore* namespace.

C# (Students.cs)

```
[PrimaryKey("ID","Email")]
public class Student
{
public int ID { get; set; }
public int Email { get; set; }

}
```

The Key attribute

- [ForeignKey]: In ASP.NET Core MVC, This annotation is used to specify a foreign key relationship between two model classes. A foreign key is a field in a database table that refers to the primary key of another table, establishing a relationship between the two tables.

C# (Enrollment.cs)

```
public class Enrollment
{
 public int EnrollmentID { get; set; }
 public Grade Grade { get; set; }
 //[ForeignKey("Student")] //name of associated navigation property
 public int StudentID { get; set; }
 //[ForeignKey("Course")] //name of associated navigation property
 public int CourseID { get; set; }

 [ForeignKey("StudentID")] //name of associated property that represents foreign key
 public virtual Student Student { get; set; }
 [ForeignKey("CourseID")] //name of associated property that represents foreign key
 public virtual Course Course { get; set; }
 }
```

The Column attribute

- You can also use attributes to control how your classes and properties are mapped to the database. Suppose you had used the name FirstMidName for the first-name field because the field might also contain a middle name. But you want the database column to be named FirstName, because users who will be writing ad-hoc queries against the database are accustomed to that name. To make this mapping, you can use the Column attribute.
- If you don't specify column names, they're given the same name as the property name.
- In the Student.cs file, add a using statement for System.ComponentModel.DataAnnotations.Schema and add the column name attribute to the FirstMidName property, as shown in the following code:

The Column attribute

C# (Student.cs)

```
using System.ComponentModel.DataAnnotations.Schema;
public class Student
{
 [Key]
 public int ID { get; set; }
 public string LastName { get; set; }
 [Column("FirstName")]
 public string FirstMidName { get; set; }
}
```

- [Column(Order =, TypeName=)]: you can also specify the order of a column in the database table and the type name in the database provider.

C# (Student.cs)

```
using System.ComponentModel.DataAnnotations.Schema;
public class Student
{
 [Key]
 public int ID { get; set; }
 public string LastName { get; set; }
 [Column("FirstName“, Order = 2,TypeName = “nvarchar”)]
 public string FirstMidName { get; set; }
}
```

The Required attribute

- [Required]: This annotation is used to mark a property as required, which means it must have a value before the model can be considered valid. If the property is null or empty, validation will fail.
- The Required attribute isn't needed for non-nullable types such as value types (DateTime, int, double, float, etc.). Types that can't be null are automatically treated as required fields.

C# (Student.cs)

```
using System.ComponentModel.DataAnnotations.Schema;
public class Student
{
 [Key]
 public int ID { get; set; }
 [Required]
 public string LastName { get; set; }
 [Column("FirstName“, Order = 2,TypeName = “nvarchar”)]
 public string FirstMidName { get; set; }
}
```

The Required attribute

- You can specify that the empty string is allowed and also an error message if the validation failed as the following:

C# (Student.cs)

```
using System.ComponentModel.DataAnnotations.Schema;
public class Student
{
 [Key]
 public int ID { get; set; }
 [Required(AllowEmptyStrings = true, ErrorMessage = "LastName is Required")]
 public string LastName { get; set; }
 [Column("FirstName“, Order = 2,TypeName = “nvarchar”)]
 public string FirstMidName { get; set; }
}
```

The StringLength attribute

- The StringLength attribute sets the maximum length in the database and provides client side and server side validation for ASP.NET Core MVC. You can also specify the minimum string length in this attribute, but the minimum value has no impact on the database schema.
- Suppose you want to ensure that users don't enter more than 50 characters for a name. To add this limitation, add StringLength attributes to the LastName and FirstMidName properties, as shown in the following example:

C# (Student.cs)

```
public class Student
{
 [StringLength(50)]
 public string LastName { get; set; }

 [StringLength(50, MinimumLength = 5, ErrorMessage = "the first Name length should be between 5 and 50 chars!")]
//you can specify also minimum length and Error Message
 public string FirstMidName { get; set; }
}
```

The RegularExpression attribute

- The StringLength attribute won't prevent a user from entering white space for a name. You can use the RegularExpression attribute to apply restrictions to the input. For example, the following code requires the first character to be upper case and the remaining characters to be alphabetical:

C# (Student.cs)

```
public class Student
{
 [RegularExpression(@"^[A-Z]+[a-zA-Z]*$", ErrorMessage = "the first char should be capital")]
 public string FirstMidName { get; set; }
}
```

- [RegularExpression]: is used to validate that a string property matches a specified regular expression pattern.
- You can also provide an error message if validation failed.

The DataType attribute

- [DataType]: This annotation is used to specify the data type of a property, which can be useful for validating and formatting data. It can be used to specify data types such as email addresses, URLs, or dates.
- The DataType attribute is used to specify a data type that's more specific than the database intrinsic type.
- The DataType Enumeration provides for many data types, such as Date, Time, PhoneNumber, Currency, EmailAddress, and more.

C# (Student.cs)

```
public class Student
{
[DataType(DataType.Date, ErrorMessage = "Enter Date")]
public DateTime EnrollmentDate { get; set; }
}
```

The DisplayFormat attribute

- The DisplayFormat attribute is used to explicitly specify the data format or how to render it on a screen.
- The ApplyFormatInEditMode setting specifies that the formatting should also be applied when the value is displayed in a text box for editing. (You might not want that for some fields -- for example, for currency values, you might not want the currency symbol in the text box for editing.)

C# (Student.cs)

```
public class Student
{
[DataType(DataType.Date, ErrorMessage = "Enter Date")]
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
public DateTime EnrollmentDate { get; set; }
}
```

The Display attribute

- [Display]: This annotation is used to specify the display name for a property, which is used as the label when rendering the form. It can also be used to specify additional properties such as the group name or the order of the property in the form.
- The Display attribute specifies that the caption for the text boxes should be "First Name", "Last Name", "Full Name", and "Enrollment Date" instead of the property name in each instance (which has no space dividing the words).

C# (Student.cs)

```
public class Student
{
[Display(Name = "Last Name")]
public string LastName { get; set; }
}
```

The DatabaseGenerated attribute

- The DatabaseGenerated attribute with the None parameter on the CourseID property specifies that primary key values are provided by the user rather than generated by the database.
- By default, Entity Framework assumes that primary key values are generated by the database by using Identity option. That's what you want in most scenarios.
- However, for Course entities, you'll use a user-specified course number such as a 1000 series for one department, a 2000 series for another department, and so on.
- The DatabaseGenerated attribute can also be used to generate computed values.

C# (Course.cs)

```
public class Course
{
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int CourseID { get; set; }
}
```

The Range attribute

- [Range]: This annotation is used to specify the allowed range of values for a numeric property, such as an integer or a decimal. If the value falls outside this range, validation will fail.

Example:

C# (Student.cs)

```
public class Student
{
    [Range(19,33)]
    public int Age { get; set; }
}
```

The EmailAddress attribute

- [EmailAddress]: This annotation is used to validate that a string property contains a valid email address.

Example:

C# (Employee.cs)

```
public class Employee
{
    [EmailAddress]
    public string Email { get; set; }
}
```

The Phone attribute

- [Phone]: This annotation is used to validate that a string property contains a valid phone number.

Example:

C# (Employee.cs)

```
public class Employee
{
    [Phone]
    public string PhoneNumber { get; set; }
}
```

The CreditCard attribute

- [CreditCard]: This annotation is used to validate that a string property contains a valid credit card number.

Example:

C# (Example.cs)

```
public class Example
{
    [CreditCard]
    public string CreditCardNumber { get; set; }
}
```

The Url attribute

- [Url]: This annotation is used to validate that a string property contains a valid URL.
- Example:

C# (Example.cs)

```
public class Example
{
    [Url]
    public string WebsiteURL { get; set; }
}
```

The Compare attribute

- [Compare]: This annotation is used to compare the values of two properties in a model. It is commonly used for password confirmation scenarios, where the user is asked to enter their password twice.

Example:

C# (Employee.cs)

```
public class Employee
{
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Compare("Password")]
    public string ConfirmPassword { get; set; }

}
```

The MaxLength and MinLength attributes

- [MaxLength]: This annotation is used to specify the maximum length of a string property.
- [MinLength]: This annotation is used to specify the minimum length of a string property.

Example:

C# (Employee.cs)

```
public class Employee
{
    [MaxLength(100)]
    [MaxLength(10)]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Compare("Password")]
    public string ConfirmPassword { get; set; }

}
```

Data Annotation

- You can put multiple attributes on one line by separating them by comma.

C# (Example.cs)

```
[DataType(DataType.Date),Display(Name = "Hire Date"),DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

Registration and Login

Registration

- Lets add a new model for Employee that will have data for authenticate him in our application.
- So, the Employee class will be like:

C# (Employee.cs)

```
[PrimaryKey("Email")]
public class Employee
{
    [Required(ErrorMessage = "You Should Enter a vaild Email"),
    EmailAddress(ErrorMessage = "You Should Enter a vaild Email")]
    public string Email { get; set; }
    [Required(ErrorMessage = "You Should Enter Your First Name"),
    Display(Name = "First Name")]
    public string FirstName { get; set; }
    [Required(ErrorMessage = "You Should Enter Your Last Name"),
    Display(Name = "Last Name")]
    public string LastName { get; set; }
    [Required(ErrorMessage = "You Should Enter Your Password"),
    DataType(DataType.Password),
    MinLength(3, ErrorMessage = "Password should be longer than 3
chars")]
    public string Password { get; set; }
    [Required, Compare("Password", ErrorMessage = "your password
doesn't match!")]
    public string ConfirmPassword { get; set; }
    [Display("Image")]
    public string ImageUrl { get; set; }
}
```

Registration

- Add in the *SchoolContext* a *DbSet* property for Employee to create a table for it in the database.

C# (SchoolContext.cs)

```
public class SchoolContext : DbContext
{
    public SchoolContext(DbContextOptions<SchoolContext> options) : base(options){ }

    public DbSet<Student> Students { get; set; }
    public DbSet<Enrollment> Enrollments { get; set; }
    public DbSet<Course> Courses { get; set; }
    public DbSet<Employee> Employees { get; set; }
}
```

- Then Add-migration and update-database

Registration

- Lets create a controller with register action and a view for registering an Employee:

C# (EmployeeController.cs)

```
public class EmployeeController : Controller
    {
private readonly SchoolContext _context;
private readonly IWebHostEnvironment _environment;
public EmployeeController(SchoolContext context, IWebHostEnvironment environment)
{
    _context = context;
    _environment = environment;
}
[HttpGet]
public IActionResult Register()
{
    return View();
}

[HttpPost]
public IActionResult Register(Employee EMP, IFormFile ImageUrl)
{
    if (ImageUrl != null)
    {
        string imgFolderPath = Path.Combine(_environment.WebRootPath, "img");
        if (!Directory.Exists(imgFolderPath))
        {
            Directory.CreateDirectory(imgFolderPath);
        }
```

C# (EmployeeController.cs)

```
string imgPath = Path.Combine(imgFolderPath,
ImageUrl.FileName);
        EMP.ImageUrl = ImageUrl.FileName;
        using (var stream = new FileStream(imgPath,
FileMode.Create))
        {
            ImageUrl.CopyTo(stream);
        }
    }
else
{
    EMP.ImageUrl = "Default.png";
}
    ModelState.Remove("ImageUrl");
    if (ModelState.IsValid)
    {
        _context.Employees.Add(EMP);
        _context.SaveChanges();
        return RedirectToAction("Index", "Home");
    }
    return View(EMP);
}
    }
```

Registration

- Lets create a controller with register action and a view for registering an Employee:

CSHTML (Register.cshtml)

```
@model Employee
@{
    ViewData["Title"] = "Registration";
}
<form class="m-2" asp-action="Register" asp-controller="Employee" method="post"
enctype="multipart/form-data">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group m-2">
        <label asp-for="FirstName" class="control-label"></label>
        <input asp-for="FirstName" class="form-control" />
        <span asp-validation-for="FirstName" class="text-danger"></span>
    </div>
    <div class="form-group m-2">
        <label asp-for="LastName" class="control-label"></label>
        <input asp-for="LastName" class="form-control" />
        <span asp-validation-for="LastName" class="text-danger"></span>
    </div>
    <div class="form-group m-2">
        <label asp-for="Email" class="control-label"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
```

CSHTML (Register.cshtml)

```
 <div class="form-group m-2">
        <label asp-for="Password" class="control-label"></label>
        <input asp-for="Password" class="form-control" />
        <span asp-validation-for="Password" class="text-
danger"></span>
    </div>
    <div class="form-group m-2">
        <label asp-for="ConfirmPassword" class="control-
label"></label>
        <input asp-for="ConfirmPassword" class="form-control" />
        <span asp-validation-for="ConfirmPassword" class="text-
danger"></span>
    </div>
    <div class="form-group m-2">
        <label asp-for="ImageUrl" class="control-label"></label>
        <input asp-for="ImageUrl" type="file" class="form-control"
/>
        <span asp-validation-for="ImageUrl" class="text-
danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
</form>
<div class="m-2">
    <a asp-action="Login">Already have an Email?</a>
</div>
```

Registration

First Name

Last Name

Email

Password

ConfirmPassword

Image

Choose File | No file chosen

Register

Already have an Email?

Login

- Lets create in the controller a login action and a view for login by an Employee:

C# (EmployeeController.cs)

```
[HttpGet]
public IActionResult Login()
{
    return View();
}

[HttpPost]
public IActionResult Login(Employee EMP)
{
    ModelState.Remove("FirstName");
    ModelState.Remove("LastName");
    ModelState.Remove("ConfirmPassword");
    ModelState.Remove("ImageUrl");
    if (ModelState.IsValid)
    {
        var user = _context.Employees.FirstOrDefault(u => u.Email == EMP.Email);

        if (user != null && user.Password == EMP.Password)
        {
            HttpContext.Session.SetString("User", user.Email);
            HttpContext.Session.SetString("img", user.ImageUrl);
            return RedirectToAction("Index", "Home");
        }
```

C# (EmployeeController.cs)

```
 else
        {
            ViewBag.error = "your Email or Password is
Invalid";
        }
    }
    else
    {
        ViewBag.error = "your Email or Password is Invalid";
    }
    return View(EMP);
}
```

Login

- Lets create in the controller a login action and a view for login by an Employee:

CSHTML (Login.cshtml)

```
@model Employee
@{
    ViewData["Title"] = "Login";
}
<form class="m-2" asp-action="Login" asp-controller="Employee" method="post">

    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
@if(ViewBag.error!=null)
{
    <span class="text-danger">@ViewBag.error</span>
}
    <div class="form-group m-2">
        <label asp-for="Email" class="control-label"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group m-2">
        <label asp-for="Password" class="control-label"></label>
        <input asp-for="Password" class="form-control" />
        <span asp-validation-for="Password" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">Login</button>
</form>

<div class="m-2">
    <a asp-action="Register">Does Not have an Email?</a>
</div>
```

Login

Email

Password

Login

Does Not have an Email?

Contoso University Home About Students Courses Instructors Departements sw@con

Welcome

Contoso University

Welcome to Contoso University

Contoso University is a sample application that demonstrates how to use Entity Framework Core in an ASP.NET Core MVC web application.

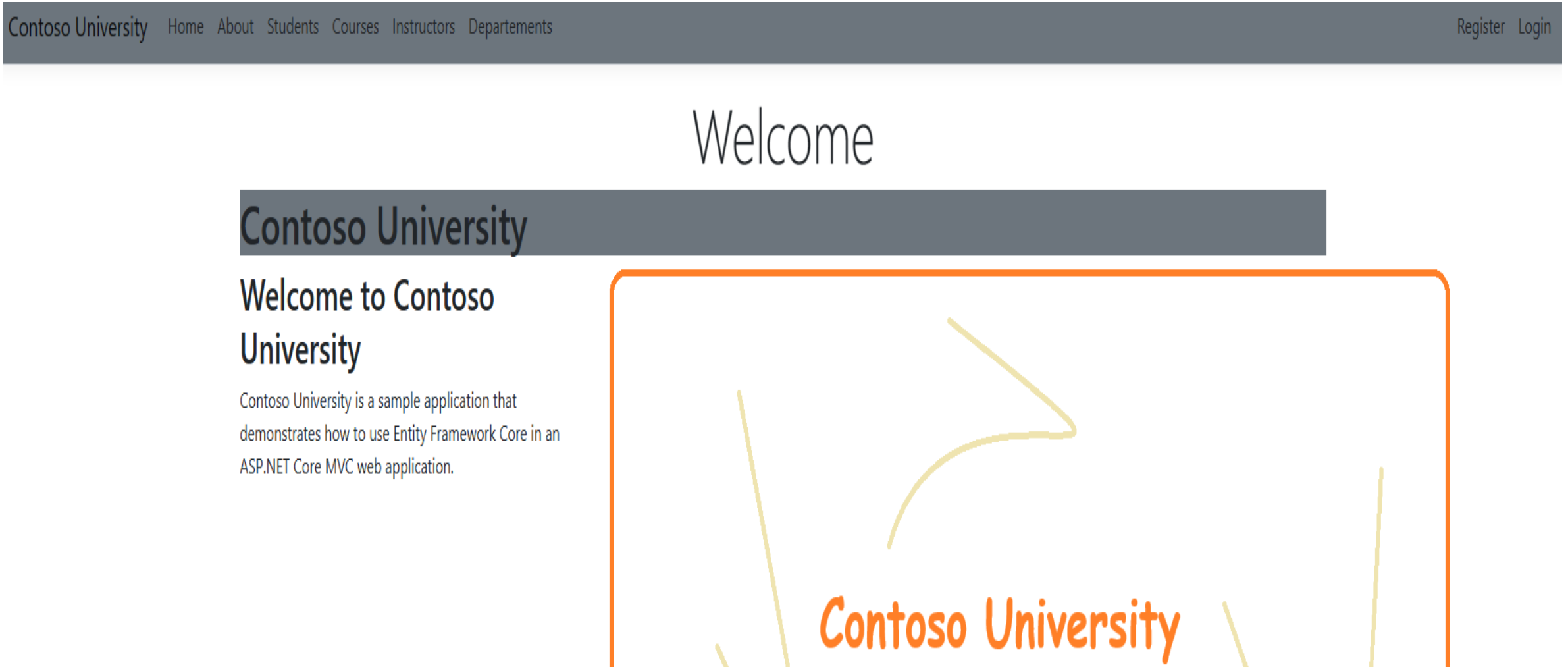
Login

- There are some changes we have made to the Layout to add register and login links, under *<ul>* type the following:

CSHTML (_Layout.cshtml)

```
<ul class="navbar-nav">
     @if(Context.Session.GetString("User") == null)
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Employee" asp-action="Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Employee" asp-action="Login">Login</a>
        </li>
    }
    else
    {
        <li class="nav-item">
            <p class="nav-link text-dark">@Context.Session.GetString("User")</p>
        </li>
        <li class="nav-item">
            <img style="width:50px;height:50px;" class="rounded-circle" src="~/img/@Context.Session.GetString("img")" />
        </li>
    }
</ul>
```

Login


Contoso University
Home
About
Students
Courses
Instructors
Departements
Register
Login
Welcome
Contoso University
Welcome to Contoso University
Contoso University is a sample application that demonstrates how to use Entity Framework Core in an ASP.NET Core MVC web application.
Contoso University

Using Session

- HTTP is a stateless protocol. By default, HTTP requests are independent messages that don't retain user values.
- State can be stored using several approaches. Let’s try session state.
- **Configure session state** in Program.cs:

C# (Program.cs)

```
 builder.Services.AddSession(ss =>
        {
            ss.IdleTimeout = TimeSpan.FromSeconds(30);
            ss.Cookie.IsEssential = true;
            ss.Cookie.HttpOnly = true;
        });
....
app.UseSession();
```

Set Session values

- You can set a session variable by using its set methods like *SetInt32* and *SetString*. as *HttpContext.Session.SetString(“key", “StringValue”);*
- As you have seen in the Login action when we set user and img variables:

C# (EmployeeController.cs)

```
HttpContext.Session.SetString("User", user.Email);
HttpContext.Session.SetString("img", user.ImageUrl);
```

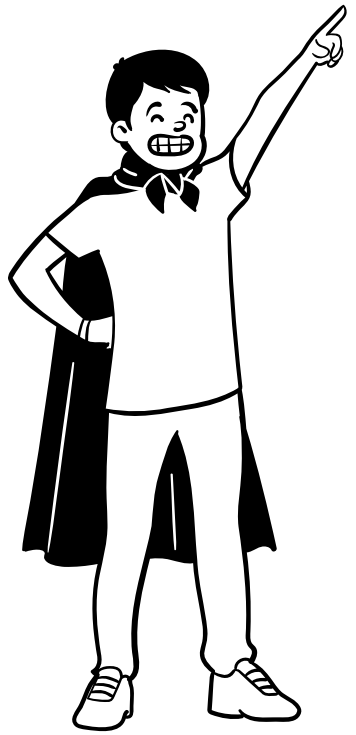
- **Note**: Session variables are stored on the server-side and are accessible to all users of your web application. It is important to keep sensitive information out of session variables to prevent security risks.

Get Session values

- You can get a session variable by using its get methods like *GetInt32* and *GetString*. as *HttpContext.Session.GetString("key");*
- As we have typed in the index action in the Home controller to get information about the logged user stored in user and img session variables:

CSHTML (_Layout.cshtml)

```
 <li class="nav-item">
            <p class="nav-link text-dark">@Context.Session.GetString("User")</p>
        </li>
        <li class="nav-item">
            <img style="width:50px;height:50px;" class="rounded-circle"
src="~/img/@Context.Session.GetString("img")" />
        </li>
```



Any Questions?


THANK
YOU!