PREGRADO

UNIDAD 3 | WEB SERVICES

ENTITY FRAMEWORK CORE

ENTITY LAYER: DATA ANNOTATIONS – FLUENT API – PART 2

SI730 | Aplicaciones Web

Al finalizar la semana, el estudiante implementa componentes para capas de persistencia de una aplicación de lado servidor, con características innovadoras, bajo una arquitectura orientada a servicios y aplicando los principios RESTful utilizando el lenguaje C# y Microsoft .NET Framework.

AGENDA

QUERIES

CONTROLLERS

VALIDATIONS

Queries

Entity Framework Core usa Language Integrated Query (LINQ) para consultar datos de la base de datos. LINQ permite usar C# para escribir consultas fuertemente tipadas basadas en el contexto derivado y las clases de entidad. Una representación de la consulta LINQ se pasa al proveedor de base de datos para que la convierta en lenguaje de consulta específico de la base de datos.

Ejemplo de consultas básicas

```
namespace EFQuerying.Basics
{
    public class Sample
    {
        public static void Run()
        {
            using (var context = new BloggingContext())
            {
                var blogs = context.Blogs.ToList();
            }

            using (var context = new BloggingContext())
            {
                var blog = context.Blogs
                    .Single(b => b.BlogId == 1);
            }

            using (var context = new BloggingContext())
            {
                var blogs = context.Blogs
                    .Where(b => b.Url.Contains("dotnet"))
                    .ToList();
            }
        }
    }
}
```

Carga de datos relacionados

Entity Framework Core permite usar las propiedades de navegación del modelo para cargar las entidades relacionados.
Carga diligente significa que los datos relacionados se cargan desde la base de datos como parte de la consulta inicial.
Usa el método **Include** para especificar los datos relacionados que se incluirán en los resultados de la consulta.

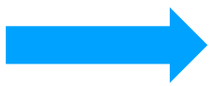
```
#region SingleInclude
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ToList();
}
#endregion

#region MultipleIncludes
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .Include(blog => blog.Owner)
        .ToList();
}
#endregion
```

Carga de datos relacionados

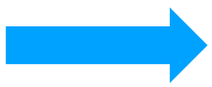
```
#region SingleThenInclude
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .ToList();
}
#endregion

#region MultipleThenIncludes
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .ToList();
}
#endregion
```



Carga de datos relacionados

```
#region MultipleLeafIncludes
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Tags)
        .ToList();
}
#endregion
```



```
#region IncludeTree
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .Include(blog => blog.Owner)
            .ThenInclude(owner => owner.Photo)
        .ToList();
}
#endregion
```

Consultas SQL sin formato

Entity Framework Core le permite descender hasta las **consultas SQL** sin formato cuando trabaja con una base de datos relacional. Esto puede resultar útil si la consulta que desea hacer no se puede expresar con **LINQ** o si usar una consulta LINQ hará que se envíen consultas de SQL ineficaces a la base de datos.

Consultas SQL básicas sin formato

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .FromSql("SELECT * FROM dbo.Blogs")
        .ToList();
}

using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .FromSql("EXECUTE dbo.GetMostPopularBlogs")
        .ToList();
}
```

Consultas SQL sin formato

Pasar parámetros

```
using (var context = new BloggingContext())
{
    var user = "johndoe";

    var blogs = context.Blogs
        .FromSql("EXECUTE dbo.GetMostPopularBlogsForUser {0}", user)
        .ToList();
}

using (var context = new BloggingContext())
{
    var user = "johndoe";

    var blogs = context.Blogs
        .FromSql($"EXECUTE dbo.GetMostPopularBlogsForUser {user}")
        .ToList();
}

using (var context = new BloggingContext())
{
    var user = new SqlParameter("user", "johndoe");

    var blogs = context.Blogs
        .FromSql("EXECUTE dbo.GetMostPopularBlogsForUser @user", user)
        .ToList();
}
```

Consultas SQL sin formato

Redacción con LINQ

```
using (var context = new BloggingContext())
{
    var searchTerm = ".NET";

    var blogs = context.Blogs
        .FromSql($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
        .Where(b => b.Rating > 3)
        .OrderByDescending(b => b.Rating)
        .ToList();
}
```

```
using (var context = new BloggingContext())
{
    var searchTerm = ".NET";

    var blogs = context.Blogs
        .FromSql($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
        .Include(b => b.Posts)
        .ToList();
}
```

AGENDA

QUERIES

CONTROLLERS

VALIDATIONS

Controllers

Los controladores se usan para definir y agrupar un conjunto de acciones. Una acción (o *método de acción*) es un método en un controlador que controla las solicitudes. Los controladores agrupan lógicamente acciones similares. Esta agregación de acciones permite aplicar de forma colectiva conjuntos comunes de reglas, como el enrutamiento, el almacenamiento en caché y la autorización. Las solicitudes se asignan a acciones mediante el enrutamiento.

Por convención, las clases de controlador:

- Residen en la carpeta **`Controllers`** del nivel de raíz del proyecto.
- Heredan de **`Microsoft.AspNetCore.Mvc.Controller`**.

Un controlador es una clase instanciable en la que se cumple al menos una de las siguientes condiciones:

- El nombre de clase tiene el sufijo "**`Controller`**".
- La clase hereda de una clase cuyo nombre tiene el sufijo "**`Controller`**".
- La clase está decorada con el atributo **`[Controller]`**.

Controllers – Definición de acciones

Los métodos públicos de un controlador, excepto los que están decorados con el atributo [NonAction], son acciones. Los parámetros de las acciones están enlazados a los datos de la solicitud y se validan mediante el enlace de modelos.

La **validación de modelos** se lleva a cabo para todo lo que está enlazado a un modelo. El valor de la propiedad **`ModelState.IsValid`** indica si el enlace de modelos y la validación se han realizado correctamente.

Las acciones pueden devolver de todo, pero suelen devolver una instancia de **`IActionResult`** (o **`Task<IActionResult>`** para métodos asincrónicos) que genera una respuesta. El método de acción se encarga de elegir el tipo de respuesta. El resultado de la acción se encarga de la respuesta.

Ejemplo de controladores

```
public class StudentsController : Controller
{
    private readonly SchoolContext _context;

    public StudentsController(SchoolContext context)
    {
        _context = context;
    }

    // GET: Students
    public async Task<IActionResult> Index()
    {
        return View(await _context.Students.ToListAsync());
    }

    // GET: Students/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var student = await _context.Students
            .FirstOrDefaultAsync(m => m.ID == id);
        if (student == null)
        {
            return NotFound();
        }

        return View(student);
    }
```

Ejemplo de controladores

```
// GET: Students/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = await _context.Students
        .FirstOrDefaultAsync(m => m.ID == id);
    if (student == null)
    {
        return NotFound();
    }

    return View(student);
}

// POST: Students/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var student = await _context.Students.FindAsync(id);
    _context.Students.Remove(student);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool StudentExists(int id)
{
    return _context.Students.Any(e => e.ID == id);
}
```

AGENDA

QUERIES

CONTROLLERS

VALIDATIONS

ASP.NET Core MVC Validations

El procedimiento para realizar las validaciones es el siguiente:

- **Agregar lógica de validación al modelo**: El espacio de nombres DataAnnotations proporciona un conjunto de atributos de validación integrados que se aplican mediante declaración a una clase o propiedad. DataAnnotations también contiene atributos de formato como DataType que ayudan a aplicar formato y no proporcionan ninguna validación.

- **Asegúrese de que las reglas de validación se aplican cada vez que un usuario cree o edite**: El método llama a **`ModelState.IsValid`** para comprobar si tiene errores de validación. Al llamar a este método se evalúan todos los atributos de validación que se hayan aplicado al objeto. Si el objeto tiene errores de validación, el método vuelve a mostrar el formulario. Si no hay ningún error, el método guarda la nueva película en la base de datos.

Ejemplo de Modelo con reglas de Validación

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z""'\s-]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}
```

Ejemplo del Controlador

```
// GET: Students/Create
public IActionResult Create()
{
    return View();
}

// POST: Students/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ID,LastName,FirstMidName,EnrollmentDate")] Student student)
{
    if (ModelState.IsValid)
    {
        _context.Add(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(student);
}
```

Ejemplo de la Vista

```
<h1>Create</h1>

<h4>Movie</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Title" class="control-label"></label>
                <input asp-for="Title" class="form-control" />
                <span asp-validation-for="Title" class="text-danger"></span>
            </div>
```

Mensajes de error

Los atributos de validación permiten especificar el **mensaje de error** que se mostrará para una entrada no válida. Por ejemplo:

```
[RegularExpression(@"^[A-Z]+[a-zA-Z""'\s-]*$")]
[Required]
[StringLength(30, ErrorMessage = "Name length can't be more than 30.")]
public string Genre { get; set; }
```

```
[RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]*$")]
[StringLength(5, ErrorMessage = "{0} length must be between {2} and {1}.", MinimumLength = 3))]
[Required]
public string Rating { get; set; }
```

RESUMEN

Recordemos

Entity Framework Core

- Consultas
- Controlador
- Validadores
- Errores

REFERENCIAS

Para profundizar

https://docs.microsoft.com/en-us/ef/core/querying/

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions?view=aspnetcore-2.2

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-2.2

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/validation?view=aspnetcore-2.2

PREGRADO

Ingeniería de Software

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe

exígete, innova