Tecnológico de Estudios Superiores de
Valle de Bravo

Ingeniería en Sistemas Computacionales

Desarrollo de Aplicaciones .NET CORE

Manual dePracticas.

M. en I.S.C. César Primero Huerta

Valle De Bravo, Estado De México, septiembre, 2023

Índice

Competencia de la materia:

Programa aplicaciones web con tecnologías de desarrollo emergente para manipular y visualizar información proveniente de una base de datos, gestionando servicios locales y en la nube.

Introducción

En este documento, abordaremos la práctica de ConstosoUniversity, donde se abordará el desarrollo de esta para que tenga un buen funcionamiento, teniendo en cuenta la "Conectividad a Base de Datos,", la integración de “API, la cual será realizada mediante la documentación de ASP.NET Core MVC con EF Core. A través de esta serie de tutoriales, exploraremos cómo diseñar controladores y vistas relacionados con una universidad. Además, se conocerán algunas funcionalidades clave, como:

- Implementar herencia en el modelo de datos.
- Realizar consultas SQL sin formato.
- Utilizar LINQ dinámico para simplificar el código.

Sin embargo, antes de comenzar, es importante conocer qué es ASP.NET. El ASP.NET es un marco web de código abierto desarrollado por Microsoft, diseñado para la creación de servicios y aplicaciones web modernas utilizando .NET.

Como se mencionó se integrará la Api la cual será realizada mediante la documentación de Creating a Web API with ASP.NET Core, y a su vez mediante

esta serie de tutoriales, exploraremos cómo diseñar una API y sus diferentes controladores y factores que la componen. Además, se conocerán algunas funcionalidades clave, como:

API	Descripción
GET /api/todoitems	Obtener todas las tareas pendientes
GET /api/todoitems/{id}	Obtener un elemento por identificador
POST /api/todoitems	Incorporación de un nuevo elemento
PUT /api/todoitems/{id}	Actualizar un elemento existente
DELETE /api/todoitems/{id}	Eliminar un elemento

Tabla 1 Funciones de API

(Microsoft, Tutorial: Creación de una API web con ASP.NET Core, 2024)

Materiales (Requisitos Previos)

Para poder implementar este proyecto se debe contar con los siguientes requerimientos:

- **Visual Studio 2022** con la carga de trabajo de ASP.NET y desarrollo web SDK de .NET 6.0: Es una herramienta de desarrollo eficaz que permite completar todo el ciclo de desarrollo en un solo lugar. Es un entorno de desarrollo integrado (IDE) completo que puede usar para escribir, editar, depurar y compilar el código y, luego, implementar la aplicación. (IDE de Visual Studio 2022: herramienta de programación para desarrolladores de software, 2023)

- **C#:** Es un lenguaje de programación orientado a componentes, orientado a objetos. C# proporciona construcciones de lenguaje para admitir directamente estos conceptos, por lo que se trata de un lenguaje natural en el que crear y usar componentes de software.
- **SQLServer:** Microsoft SQL Server es un sistema de gestión de base de datos relacional (RDBMS) producido por Microsoft. Su principal lenguaje de consulta es Transact-SQL, una aplicación de las normas ANSI / ISO estándar Structured Query Language (SQL) utilizado por ambas Microsoft y Sybase.

Desarrollo

Práctica 1: Creación de aplicación WEB

Para crear una aplicación web, primero debes abrir Visual Studio, luego, selecciona "Crear un nuevo proyecto".

A continuación, elige la plantilla Aplicación web ASP.NET Core y asigna un nombre al proyecto, en este caso, ConstosoUniversity.

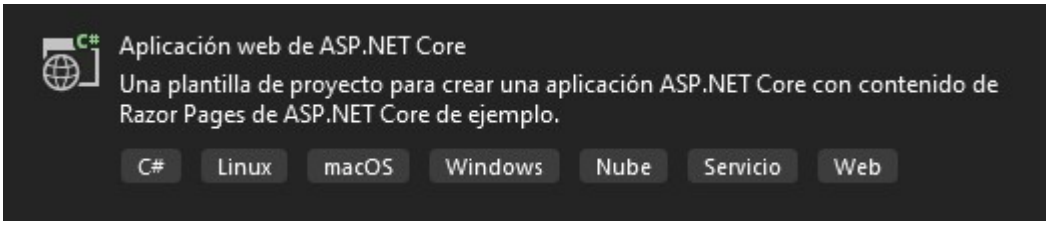

Ilustración 1 Creación de aplicación WEB

Después, haz clic en el botón "Crear". Posteriormente, debes seleccionar en los menús desplegables la versión de ASP.NET Core y

elegir la plantilla "Modelo Vista-Controlador" antes de finalmente crear el proyecto.

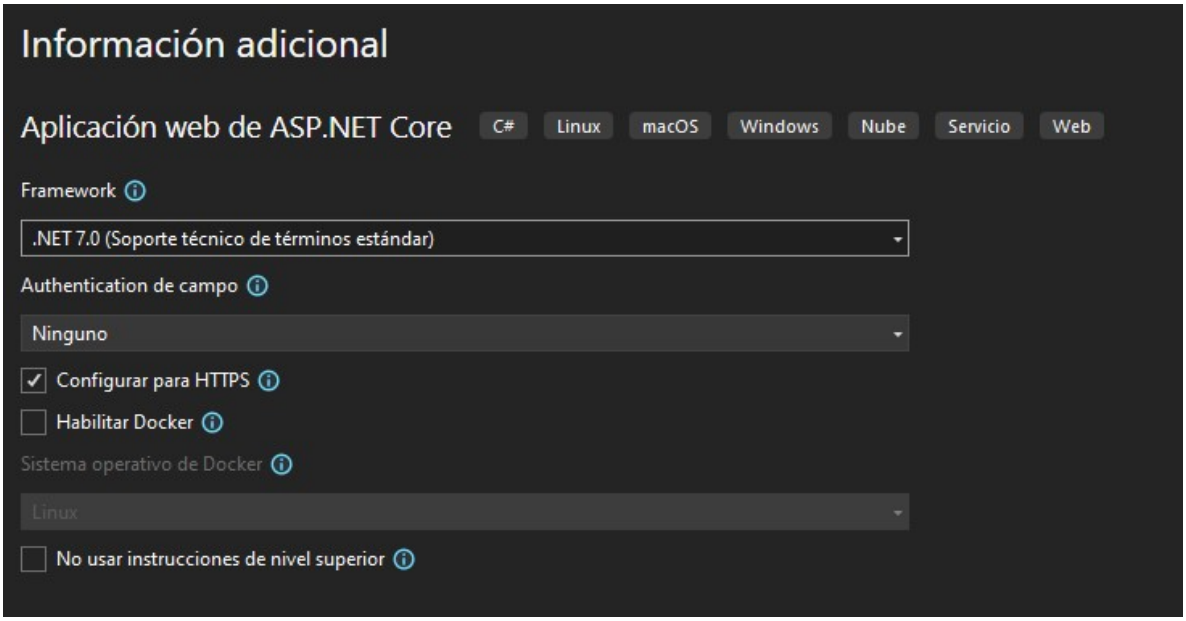

Ilustración 2Menú desplegable de la versión

Práctica 2: Configuración del estilo del sitio

Como siguiente paso, para configurar el estilo del sitio, debes navegar a la carpeta de "Views". Luego, dentro de esa carpeta, accede a "Shared" y localiza el archivo "_Layout.cshtml". En este archivo, procederás a modificar la apariencia del sitio web de "CostosoUniversity". Específicamente, deberás realizar los siguientes cambios en el menú de navegación:

- Agregar cuatro entradas de menú: About, Students,Courses,Instructors.
- Eliminar la entrada de menú Privacy.

Estos ajustes permitirán personalizar la apariencia y la estructura del sitio web según los requerimientos del proyecto CostosoUniversity.

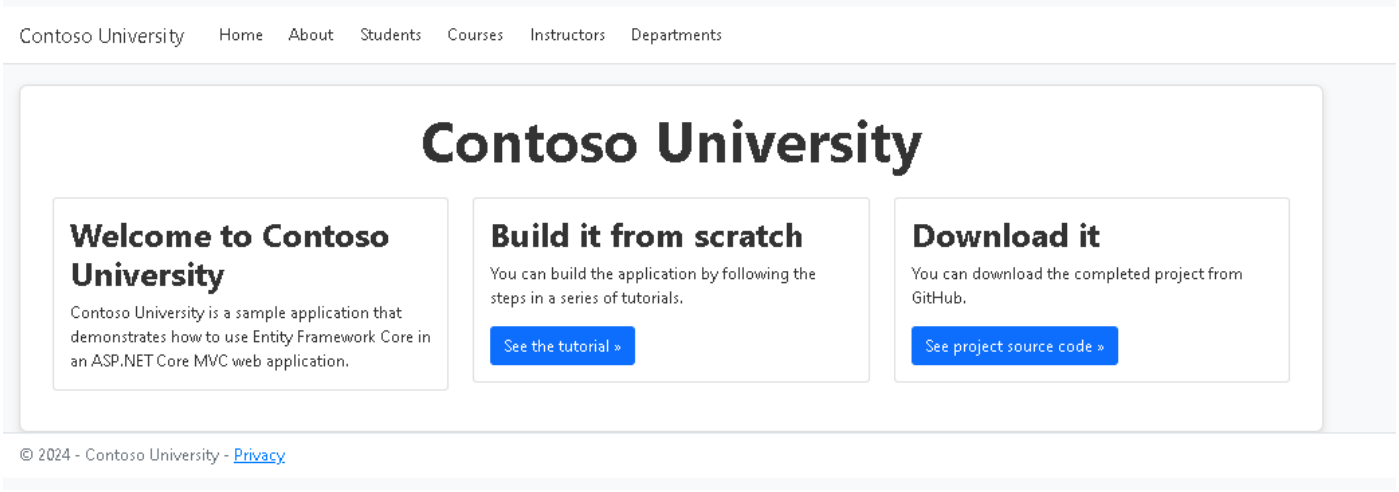

Ilustración 3 Configuración del estilo

Práctica 3: Instalación de Paquetes NuGet EF Core

- Abre Visual Studio y navega a Herramientas y Administrador de paquetes NuGet.
- Selecciona Consola del Administrador de Paquetes NuGet.
- En la consola, ingresa los comandos pertinentes.

```
Install-Package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Ilustración 4Comandos para la instalación de paquetes

Estos paquetes proporcionan middleware de ASP.NET Core para el manejo de páginas de error de EF Core, lo que es esencial para garantizar un funcionamiento adecuado en tu aplicación.

Creación del modelo de datos

Para crear el modelo de negocio de la aplicación, se deben definir las clases entidad para Cursos y Estudiantes. En este sistema, se establece una relación de muchos a muchos, lo que significa que un estudiante puede inscribirse en múltiples cursos, y, a su vez, un curso puede tener varios estudiantes inscritos.

Crear el contexto de base de datos

Para crear el contexto de la base de datos, es importante recordar que la clase principal que coordinará la funcionalidad de un modelo de

datos específico es la clase de contexto de base de datos. Esta clase debe ser ubicada en una nueva carpeta llamada Data.

Registro de SchoolContext

Para registrar el contexto de la base de datos, se debe incluir la inserción de las dependencias, como los servicios que fueron registrados en el sistema de inyección de dependencias durante el inicio de la aplicación. Esto implica proporcionar los componentes necesarios a través de parámetros de constructor.

```
using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;
namespace ContosoUniversity.Data
{
    34 referencias
    public class SchoolContext : DbContext
    {
        0 referencias
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {
        }
        26 referencias
        public DbSet<Course> Courses { get; set; }
        2 referencias
        public DbSet<Enrollment> Enrollments { get; set; }
        21 referencias
        public DbSet<Student> Students { get; set; }
        23 referencias
        public DbSet<Department> Departments { get; set; }
        28 referencias
        public DbSet<Instructor> Instructors { get; set; }
        1 referencia
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }
        1 referencia
        public DbSet<CourseAssignment> CourseAssignments { get; set; }
        0 referencias
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Student>().HasKey(s => s.ID);
            modelBuilder.Entity<Department>().ToTable("Department");
            modelBuilder.Entity<Department>().Property(p => p.RowVersion).IsConcurrencyToken();
            modelBuilder.Entity<Instructor>().ToTable("Instructor");
            modelBuilder.Entity<OfficeAssignment>().ToTable("OfficeAssignment");
            modelBuilder.Entity<CourseAssignment>().ToTable("CourseAssignment");
            modelBuilder.Entity<CourseAssignment>()
                .HasKey(c => new { c.CourseID, c.InstructorID });
        }
```

Ilustración 1 SchoolContext

Práctica 4: Inicializa la base de datos con datos de prueba

Para inicializar la base de datos con datos de prueba, es necesario crear previamente una base de datos en el gestor SQL Server. Esta acción permitirá establecer la conexión necesaria para que el proyecto pueda funcionar correctamente.

Dentro de la carpeta Data, se creará una clase llamada DBInitializer que contendrá los datos iniciales correspondientes a cada una de las tablas.

Esta clase DBInitializer será responsable de la inicialización de los datos que se relacionan con las tablas en la base de datos.

```
using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using ContosoUniversity.Models;
namespace ContosoUniversity.Data
{
    1 referencia
    public static class DbInitializer
    {
        1 referencia
        public static void Initialize(SchoolContext context)
        {
            if (context.Students.Any())
            {
                return;   // DB has been seeded
            }
            var students = new Student[]
            {
                new Student { FirstMidName = "Carson",   LastName = "Alexander",
                    EnrollmentDate = DateTime.Parse("2010-09-01") },
                new Student { FirstMidName = "Meredith", LastName = "Alonso",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Arturo",   LastName = "Anand",
                    EnrollmentDate = DateTime.Parse("2013-09-01") },
                new Student { FirstMidName = "Gytis",    LastName = "Barzdukas",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Yan",      LastName = "Li",
                    EnrollmentDate = DateTime.Parse("2012-09-01") },
                new Student { FirstMidName = "Peggy",    LastName = "Justice",
                    EnrollmentDate = DateTime.Parse("2011-09-01") },
                new Student { FirstMidName = "Laura",    LastName = "Norman",
                    EnrollmentDate = DateTime.Parse("2013-09-01") },
                new Student { FirstMidName = "Nino",     LastName = "Olivetto",
                    EnrollmentDate = DateTime.Parse("2005-09-01") }
            };
            foreach (Student s in students)
```

Ilustración 2 DBInitialize

Práctica 5: Crea un controladores, modelos y vistas.

Modelos.

Se crean los modelos a utilizar, que concuerden al contexto.

Estudiantes, donde se establecen los atributos a obtener, como lo es el id siendo el atributo que lo identificara al estudiante, el last name y frist name siendo atributos que obtienen datos importantes, Enrollment Date atributo que obtiene la fecha de enrolamiento, Full Name atributo importante, ProfilePicture siento atributo que obtiene las imágenes que

identifican a los alumnos y Enrollments que es el atributo que hace compartir datos con la tabla de courses:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace ContosoUniversity.Models
{
    37 referencias
    public class Student
    {
        32 referencias
        public int ID { get; set; }
        [Required(ErrorMessage = "Last Name is required")]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        36 referencias
        public string? LastName { get; set; }
        [Required]
        [StringLength(50)]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        23 referencias
        public string? FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        24 referencias
        public DateTime EnrollmentDate { get; set; }
        [Display(Name = "Full Name")]
        1 referencia
        public string FullName
        {
            get
            {
                return LastName + ", " + FirstMidName;
            }
        }
        [DatabaseGenerated(DatabaseGeneratedOption.None)] // Esto evita que sea una columna de identidad
        11 referencias
        public byte[]? ProfilePicture { get; set; }
        14 referencias
        public ICollection<Enrollment>? Enrollments { get; set; }
    }
}
```

Ilustración 3 Modelo students

Cursos, donde se establecen los atributos a obtener, como lo es el courseid siendo el atributo que lo identificara al curso, title y credits siendo atributos que obtienen datos importantes, departmentId que es el atributo que hace compartir datos con la tabla de departaments, Enrollments y courseassignments que son atributos que hace compartir datos con las tablas de instructor y curso asignado:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace ContosoUniversity.Models
{
    38 referencias
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name = "Number")]
        73 referencias
        public int CourseID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        47 referencias
        public string? Title { get; set; }

        [Range(0, 5)]
        21 referencias
        public int Credits { get; set; }
        14 referencias
        public int DepartmentID { get; set; }

        12 referencias
        public Department? Department { get; set; }
        3 referencias
        public ICollection<Enrollment>? Enrollments { get; set; }
        0 referencias
        public ICollection<CourseAssignment>? CourseAssignments { get; set; }
    }
}
```

Ilustración 4 Modelo course

Instructor, donde se establecen los atributos a obtener, como lo es el id siendo el atributo que lo identificara al instructor, last name y first name siendo atributos que obtienen datos importantes, hire date siendo el atributo que obtiene la fecha, full name siendo atributo de obtener datos importantes y profilepcturei que es el atributo que obtiene el dato de la imagen del instructor, officeassignment y courseassignments que son atributos que hace compartir datos con las tablas de asignación de oficina y curso asignado:

```
using System.ComponentModel.DataAnnotations.Schema;
namespace ContosoUniversity.Models
{
    37 referencias
    public class Instructor
    {
        38 referencias
        public int ID { get; set; }
        [Required]
        [Display(Name = "Last Name")]
        [StringLength(50)]
        35 referencias
        public string? LastName { get; set; }
        [Required]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        [StringLength(50)]
        19 referencias
        public string? FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Hire Date")]
        18 referencias
        public DateTime HireDate { get; set; }
        [Display(Name = "Full Name")]
        4 referencias
        public string FullName
        {
            get { return LastName + ", " + FirstMidName; }
        }
        14 referencias
        public ICollection<CourseAssignment>? CourseAssignments { get; set; }
        14 referencias
        public OfficeAssignment? OfficeAssignment { get; set; }
      [DatabaseGenerated(DatabaseGeneratedOption.None)] // Esto evita que sea una columna de identidad
        12 referencias
        public byte[]? ProfilePictureI { get; set; }
    }
}
```

Ilustración 5 Modelo de Instructor.

Departamento, donde se establecen los atributos a obtener, como lo es el departamentid siendo el atributo que lo identificara al departamento, name y budget atributos que obtienen datos importantes, start date siendo el atributo que obtiene la fecha, instructorid siendo el atributo que comparte datos con la tabla de instructores, administrator, courses y rowversion que son atributos que hace compartir datos con las tablas de administrator, courses y rowversion:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace ContosoUniversity.Models
{
    36 referencias
    public class Department
    {
        24 referencias
        public int DepartmentID { get; set; }
        [StringLength(50, MinimumLength = 3)]
        33 referencias
        public string? Name { get; set; }
        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        21 referencias
        public decimal Budget { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        21 referencias
        public DateTime StartDate { get; set; }
        20 referencias
        public int? InstructorID { get; set; }
        [Timestamp]
        9 referencias
        public byte[]? RowVersion { get; set; }
        11 referencias
        public Instructor? Administrator { get; set; }
        0 referencias
        public ICollection<Course>? Courses { get; set; }
    }
}
```

Ilustración 6 department

Práctica 6: Controladores

Home, este controlador hace que cuando yo seleccione algo de los subtemas me mande o direccione, ejemplo si quiero ir a students lo selecciono y me manda.

```
using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using Microsoft.EntityFrameworkCore;
using ContosoUniversity.Data;
using ContosoUniversity.Models.SchoolViewModels;
using Microsoft.Extensions.Logging;
namespace ContosoUniversity.Controllers
{
    3 referencias
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly SchoolContext _context;
        0 referencias
        public HomeController(ILogger<HomeController> logger, SchoolContext context)
        {
            _logger = logger;
            _context = context;
        }
        0 referencias
        public async Task<ActionResult> About()
        {
            IQueryable<EnrollmentDateGroup> data =
                from student in _context.Students
                group student by student.EnrollmentDate into dateGroup
                select new EnrollmentDateGroup()
                {
                    EnrollmentDate = dateGroup.Key,
                    StudentCount = dateGroup.Count()
                };
            return View(await data.AsNoTracking().ToListAsync());
        }
        0 referencias
        public IActionResult Index()
        {
            return View();
        }
        0 referencias
        public IActionResult Privacy()
        {
            return View();
        }
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 referencias
        public IActionResult Error()
```

Ilustración 7 home

Estudiante, Se establecerán el index que es donde se almacenara mediante una tabla los datos creados, como se ve en la imagen obtenemos todos los datos que guarda cada atributo, igual se anexa que se pueda buscar los estudiantes por nombre, apellido y fecha(viewdata).

```
public async Task<IActionResult> Index(string sortOrder, string currentFilter, string searchString,int? pageNumber)
{
    ViewData["CurrentSort"] = sortOrder;
    ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewData["DateSortParm"] = sortOrder == "Date" ? "date_desc" : "Date";

    if (searchString != null)
    {
        pageNumber = 1;
    }
    else
    {
        searchString = currentFilter;
    }
    ViewData["CurrentFilter"] = searchString;
    var students = _context.Students
     .Include(s => s.Enrollments) // Incluye las inscripciones
     .ThenInclude(e => e.Course)   // Luego incluye los cursos de las inscripciones
     .AsNoTracking();
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.Contains(searchString)
                               || s.FirstMidName.Contains(searchString));
    }
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    int pageSize = 3;
    return View(await PaginatedList<Student>.CreateAsync(students.AsNoTracking(), pageNumber ?? 1, pageSize));
}
```

Ilustración 8 acción de index

Acción de details, esta hace que obtenga todos los datos del estudiante diferenciándolos mediante el id, y ordenándolos de forma textual.

```
// GET: Students/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var student = await _context.Students
        .Include(s => s.Enrollments)
        .ThenInclude(e => e.Course)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.ID == id);
    if (student == null)
    {
        return NotFound();
    }
    return View(student);
}
```

Ilustración 9 acción details.

Acción de créate, esta hace que con los atributos que se establecieron en el modelo se obtengan los datos, dándole cada acción a estos, donde menciona que al obtenerlos los mandara al index.

```
// POST: Students/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("EnrollmentDate, FirstMidName, LastName, ProfilePicture")] Student student, int[] Enrollments, IFormFile profilePicture)
{
    if (ModelState.IsValid)
    {
        if (profilePicture != null)
        {
            using (var memoryStream = new MemoryStream())
            {
                await profilePicture.CopyToAsync(memoryStream);
                student.ProfilePicture = memoryStream.ToArray();
            }
        }
          _context.Add(student);
        await _context.SaveChangesAsync();
        if (Enrollments != null)
        {
            foreach (var courseId in Enrollments)
            {
                var enrollment = new Enrollment
                {
                    StudentID = student.ID,
                    CourseID = courseId,
                    Grade = Grade.A // Puedes establecer una nota predeterminada si es necesario
                };
                _context.Add(enrollment);
                await _context.SaveChangesAsync(); // Guarda cada Enrollment
            }
        }
        return RedirectToAction(nameof(Index));
    }
    // Recupera la lista de cursos y pásala a la vista
    var courses = _context.Courses.ToList();
    ViewData["Courses"] = courses;
    return View(student);
}
```

```
0 referencias
public IActionResult GetProfilePicture(int id)
{
    var student = _context.Students.Find(id);
    if (student?.ProfilePicture != null)
    {
        return File(student.ProfilePicture, "image/jpeg"); // Ajusta el tipo de contenido según el formato de tu imagen
    }
    // Si no hay imagen, puedes devolver una imagen predeterminada o un marcador de posición
    // return File("~/images/default-profile-picture.jpg", "image/jpeg");
    return NotFound();
}
```

Ilustración 10 acción créate

Acción edit, en esta acción tomara los datos que ya están mediante el id del estudiante y realizara las acciones de actualizar los datos de cada atributo.

```
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> EditPost(int? id, int[] selectedCourses)
{
    if (id == null)
    {
        return NotFound();
    }
    var studentToUpdate = await _context.Students
        .Include(s => s.Enrollments)
        .FirstOrDefaultAsync(s => s.ID == id);
    if (studentToUpdate == null)
    {
        return NotFound();
    }
    // Actualiza explícitamente las propiedades del modelo
    studentToUpdate.FirstMidName = Request.Form["FirstMidName"];
    studentToUpdate.LastName = Request.Form["LastName"];
    studentToUpdate.EnrollmentDate = DateTime.Parse(Request.Form["EnrollmentDate"]);
    UpdateStudentCourses(selectedCourses, studentToUpdate);
    try
    {
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    catch (DbUpdateException)
    {
        ModelState.AddModelError("", "Unable to save changes.");
    }
    ViewData["AllCourses"] = _context.Courses.ToList();
    return View(studentToUpdate);
}
```

Ilustración 11 acción edit

Acción delete, esta acción realizara la eliminación de los datos del estudiante mediante el id.

```
// POST: Students/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var student = await _context.Students.FindAsync(id);
    if (student == null)
    {
        return RedirectToAction(nameof(Index));
    }
    try
    {
        _context.Students.Remove(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    catch (DbUpdateException /* ex */)
    {
        //Log the error (uncomment ex variable name and write a log.)
        return RedirectToAction(nameof(Delete), new { id = id, saveChangesError = true });
    }
}
```

Ilustración 12 accion delete.

Cursos, Se establecerán el index que es donde se almacenara mediante una tabla los datos creados, como se ve en la imagen obtenemos todos los datos que guarda cada atributo.

```
public async Task<IActionResult> Index()
{
    var courses = _context.Courses
        .Include(c => c.Department)
        .AsNoTracking();
    return View(await courses.ToListAsync());
}
```

Ilustración 13 acción de index

Acción de details, esta hace que obtenga todos los datos de los cursos diferenciándolos mediante el id, y ordenándolos de forma textual.

```
// GET: Courses/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var course = await _context.Courses
        .Include(c => c.Department)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.CourseID == id);
    if (course == null)
    {
        return NotFound();
    }
    return View(course);
}
```

Ilustración 14 acción details.

Acción de créate, esta hace que con los atributos que se establecieron en el modelo se obtengan los datos, dándole cada acción a estos, donde menciona que al obtenerlos los mandara al index.

```
// POST: Courses/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("CourseID,Credits,DepartmentID,Title")] Course course)
{
    if (ModelState.IsValid)
    {
        _context.Add(course);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    PopulateDepartmentsDropDownList(course.DepartmentID);
    return View(course);
}
```

Ilustración 15 acción créate

Acción edit, en esta acción tomara los datos que ya están mediante el id del estudiante y realizara las acciones de actualizar los datos de cada atributo.

```
// POST: Courses/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> EditPost(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var courseToUpdate = await _context.Courses
        .FirstOrDefaultAsync(c => c.CourseID == id);
    if (await TryUpdateModelAsync<Course>(courseToUpdate,
        "",
        c => c.Credits, c => c.DepartmentID, c => c.Title))
    {
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
        return RedirectToAction(nameof(Index));
    }
    PopulateDepartmentsDropDownList(courseToUpdate.DepartmentID);
    return View(courseToUpdate);
}
4 referencias
private void PopulateDepartmentsDropDownList(object selectedDepartment = null)
{
    var departmentsQuery = from d in _context.Departments
                           orderby d.Name
                           select d;
    ViewBag.DepartmentID = new SelectList(departmentsQuery.AsNoTracking(), "DepartmentID", "Name", selectedDepartment);
}
```

Ilustración 16 acción edit

Acción delete, esta acción realizara la eliminación de los datos de los cursos mediante el id.

```
// POST: Courses/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.Courses == null)
    {
        return Problem("Entity set 'SchoolContext.Courses'  is null.");
    }
    var course = await _context.Courses.FindAsync(id);
    if (course != null)
    {
        _context.Courses.Remove(course);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

0 referencias
private bool CourseExists(int id)
{
  return (_context.Courses?.Any(e => e.CourseID == id)).GetValueOrDefault();
}
```

Ilustración 17 accion delete.

Instructor, Se establecerán el index que es donde se almacenara mediante una tabla los datos creados, como se ve en la imagen obtenemos todos los datos que guarda cada atributo.

```
3 referencias
public async Task<IActionResult> Index(int? id, int? courseID)
{
    var viewModel = new InstructorIndexData();
    viewModel.Instructors = await _context.Instructors
          .Include(i => i.OfficeAssignment)
          .Include(i => i.CourseAssignments)
            .ThenInclude(i => i.Course)
                .ThenInclude(i => i.Department)
          .OrderBy(i => i.LastName)
          .ToListAsync();

    if (id != null)
    {
        ViewData["InstructorID"] = id.Value;
        Instructor instructor = viewModel.Instructors.Where(
            i => i.ID == id.Value).Single();
        viewModel.Courses = instructor.CourseAssignments.Select(s => s.Course);
    }

    if (courseID != null)
    {
        ViewData["CourseID"] = courseID.Value;
        var selectedCourse = viewModel.Courses.Where(x => x.CourseID == courseID).Single();
        await _context.Entry(selectedCourse).Collection(x => x.Enrollments).LoadAsync();
        foreach (Enrollment enrollment in selectedCourse.Enrollments)
        {
            await _context.Entry(enrollment).Reference(x => x.Student).LoadAsync();
        }
        viewModel.Enrollments = selectedCourse.Enrollments;
    }

    return View(viewModel);
}
```

Ilustración 18 acción de index

Acción de details, esta hace que obtenga todos los datos del instructor diferenciándolo mediante el id, y ordenándolos de forma textual.

```
// GET: Instructors/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
    if (id == null || _context.Instructors == null)
    {
        return NotFound();
    }

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

    return View(instructor);
}
```

Ilustración 19 acción details.

Acción de créate, esta hace que con los atributos que se establecieron en el modelo se obtengan los datos, dándole cada acción a estos, donde menciona que al obtenerlos los mandara al index.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("FirstMidName,HireDate,LastName,OfficeAssignment, ProfilePictureI")]
Instructor instructor, string[] selectedCourses, IFormFile profilePictureI)
{
    if (selectedCourses != null)
    {
        instructor.CourseAssignments = new List<CourseAssignment>();
        foreach (var course in selectedCourses)
        {
            var courseToAdd = new CourseAssignment { InstructorID = instructor.ID, CourseID = int.Parse(course) };
            instructor.CourseAssignments.Add(courseToAdd);
        }
    }
    if (profilePictureI != null && profilePictureI.Length > 0)
    {
        using (var stream = new MemoryStream())
        {
            await profilePictureI.CopyToAsync(stream);
            instructor.ProfilePictureI = stream.ToArray();
        }
    }
    if (ModelState.IsValid)
    {
        _context.Instructors.Add(instructor);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    PopulateAssignedCourseData(instructor);
    return View(instructor);
}
0 referencias
public IActionResult GetProfilePicture(int id)
{
    var instructor = _context.Instructors.Find(id);
    if (instructor?.ProfilePictureI != null)
    {
        return File(instructor.ProfilePictureI, "image/jpeg");
    }

    // Si no hay imagen, puedes devolver una imagen predeterminada o un marcador de posición
    // return File("~/images/default-profile-picture.jpg", "image/jpeg");
    return NotFound();
}
```

Ilustración 20 acción créate

Acción edit, en esta acción tomara los datos que ya están mediante el id del estudiante y realizara las acciones de actualizar los datos de cada atributo.

```
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Edit(int? id, IFormFile profilePictureI, string[] selectedCourses)
{
    if (id == null)
    {
        return NotFound();
    }
    var instructorToUpdate = await _context.Instructors
        .Include(i => i.OfficeAssignment)
         .Include(i => i.CourseAssignments)
        .ThenInclude(i => i.Course)
        .FirstOrDefaultAsync(s => s.ID == id);

     if (await TryUpdateModelAsync<Instructor>(instructorToUpdate,"", i => i.FirstMidName, i => i.LastName, i => i.HireDate, i => i.OfficeAssignment))
    {
        if (profilePictureI != null && profilePictureI.Length > 0)
        {
            using (var memoryStream = new MemoryStream())
            {
                await profilePictureI.CopyToAsync(memoryStream);
                instructorToUpdate.ProfilePictureI = memoryStream.ToArray();
            }
        }
        if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment?.Location))
        {
            instructorToUpdate.OfficeAssignment = null;
        }
        UpdateInstructorCourses(selectedCourses, instructorToUpdate);
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            ModelState.AddModelError("", "Unable to save changes. " +
                "Try again, and if the problem persists, " +
                "see your system administrator.");
        }
        return RedirectToAction(nameof(Index));
    }
```

Ilustración 21 acción edit

Acción delete, esta acción realizara la eliminación de los datos del instructor mediante el id.

```
// POST: Instructors/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> DeleteConfirmed(int id)
{
    Instructor instructor = await _context.Instructors
        .Include(i => i.CourseAssignments)
        .SingleAsync(i => i.ID == id);

    var departments = await _context.Departments
        .Where(d => d.InstructorID == id)
        .ToListAsync();
    departments.ForEach(d => d.InstructorID = null);

    _context.Instructors.Remove(instructor);

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

0 referencias
private bool InstructorExists(int id)
{
  return (_context.Instructors?.Any(e => e.ID == id)).GetValueOrDefault();
}
```

Ilustración 22 accion delete.

Departamento, Se establecerán el index que es donde se almacenara mediante una tabla los datos creados, como se ve en la imagen obtenemos todos los datos que guarda cada atributo.

```
// GET: Departments
4 referencias
public async Task<IActionResult> Index()
{
    var schoolContext = _context.Departments.Include(d => d.Administrator);
    return View(await schoolContext.ToListAsync());
}
```

Ilustración 23 acción de index

Acción de details, esta hace que obtenga todos los datos del departamento diferenciándolo mediante el id, y ordenándolos de forma textual.

```
// GET: Departments/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
    if (id == null || _context.Departments == null)
    {
        return NotFound();
    }

    var department = await _context.Departments
        .Include(d => d.Administrator)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.DepartmentID == id);
    if (department == null)
    {
        return NotFound();
    }

    return View(department);
}
```

Ilustración 24 acción details.

Acción de créate, esta hace que con los atributos que se establecieron en el modelo se obtengan los datos, dándole cada acción a estos, donde menciona que al obtenerlos los mandara al index.

```
// POST: Departments/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("DepartmentID,Name,Budget,StartDate,InstructorID,RowVersion")] Department department)
{
    if (ModelState.IsValid)
    {
        _context.Add(department);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["InstructorID"] = new SelectList(_context.Instructors, "ID", "FullName", department.InstructorID);
    return View(department);
}
```

Ilustración 25 acción créate

Acción edit, en esta acción tomara los datos que ya están mediante el id del estudiante y realizara las acciones de actualizar los datos de cada atributo.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Edit(int? id, byte[] rowVersion)
{
    if (id == null)
    {
        return NotFound();
    }
    var departmentToUpdate = await _context.Departments.Include(i => i.Administrator).FirstOrDefaultAsync(m => m.DepartmentID == id);
    if (departmentToUpdate == null)
    {
        Department deletedDepartment = new Department();
        await TryUpdateModelAsync(deletedDepartment);
        ModelState.AddModelError(string.Empty,
            "Unable to save changes. The department was deleted by another user.");
        ViewData["InstructorID"] = new SelectList(_context.Instructors, "ID", "FullName", deletedDepartment.InstructorID);
        return View(deletedDepartment);
    }
    _context.Entry(departmentToUpdate).Property("RowVersion").OriginalValue = rowVersion;
    if (await TryUpdateModelAsync<Department>(
        departmentToUpdate,
        "",
        s => s.Name, s => s.StartDate, s => s.Budget, s => s.InstructorID))
    {
        try
        {
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateConcurrencyException ex)
        {
            var exceptionEntry = ex.Entries.Single();
            var clientValues = (Department)exceptionEntry.Entity;
            var databaseEntry = exceptionEntry.GetDatabaseValues();
            if (databaseEntry == null)
            {
                ModelState.AddModelError(string.Empty,
                    "Unable to save changes. The department was deleted by another user.");
            }
            else
            {
                var databaseValues = (Department)databaseEntry.ToObject();
```

Ilustración 26 acción edit

Acción delete, esta acción realizara la eliminación de los datos del departamento mediante el id.

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    1 referencia
    public async Task<IActionResult> Delete(Department department)
    {
        try
        {
            if (await _context.Departments.AnyAsync(m => m.DepartmentID == department.DepartmentID))
            {
                _context.Departments.Remove(department);
                await _context.SaveChangesAsync();
            }
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateConcurrencyException /* ex */)
        {
            //Log the error (uncomment ex variable name and write a log.)
            return RedirectToAction(nameof(Delete), new { concurrencyError = true, id = department.DepartmentID });
        }
    }

    0 referencias
    private bool DepartmentExists(int id)
    {
      return (_context.Departments?.Any(e => e.DepartmentID == id)).GetValueOrDefault();
    }
}
```

Ilustración 27 accion delete.

Ilustración 13 Delete

Práctica 7: Integración de API.
Creación otro proyecto dentro de la aplicación WEB CostosoUniversity.

Para crear una aplicación web, primero debes abrir el proyecto llamado CostosoUniversity en Visual Studio, luego, selecciona **"Crear un nuevo proyecto".**

A continuación, elige la plantilla API web de ASP.NET Core y asigna un nombre al proyecto, en este caso, Api1.

Ilustración 28 Creación de API web de ASP.NET Core

Después, haz clic en el botón **"Crear".** Posteriormente, debes seleccionar en los menús desplegables la versión de ASP.NET Core y finalmente crear el proyecto.

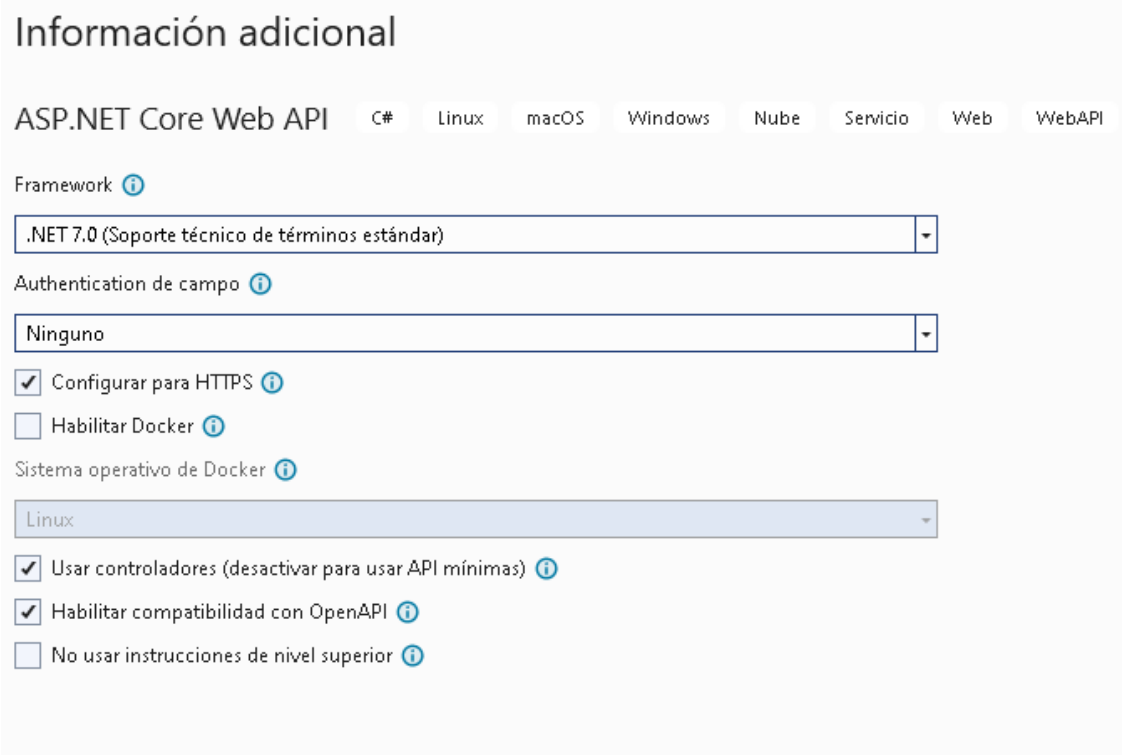

Ilustración 29 Menú desplegable de la versión

Configuración del estilo del sitio

Como siguiente paso, para configurar que el proyecto creado tome el contexto ya hecho como igual sus controladores, y así mismo su conexión a la base de datos.

Realizando primero, dando clic derecho en la carpeta del proyecto, seleccionas referencia del proyecto.

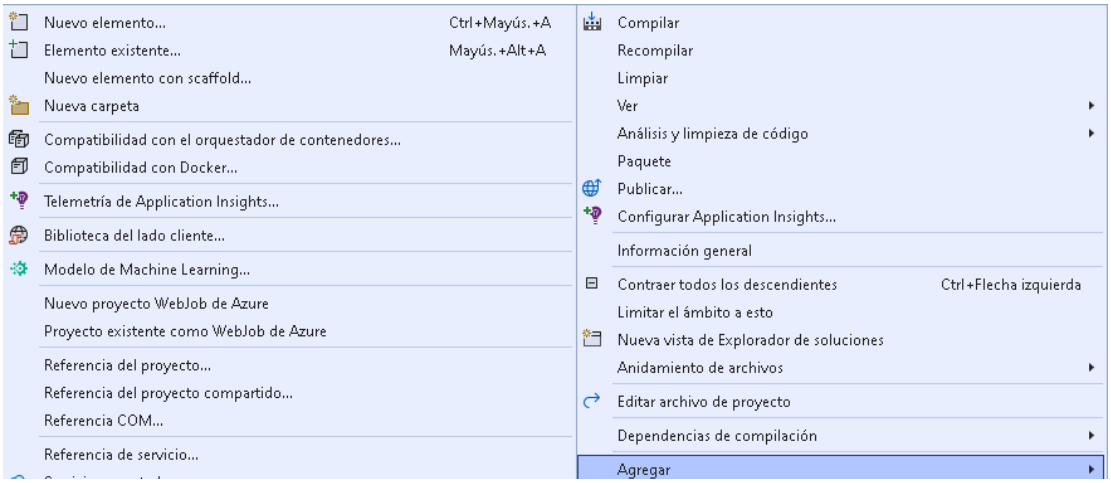

Ilustración 30 referenciar proyecto.

Continuando, se selecciona el proyecto al que quieres que se referencie el proyecto y das aceptar, esto para tomar las propiedades del proyecto.

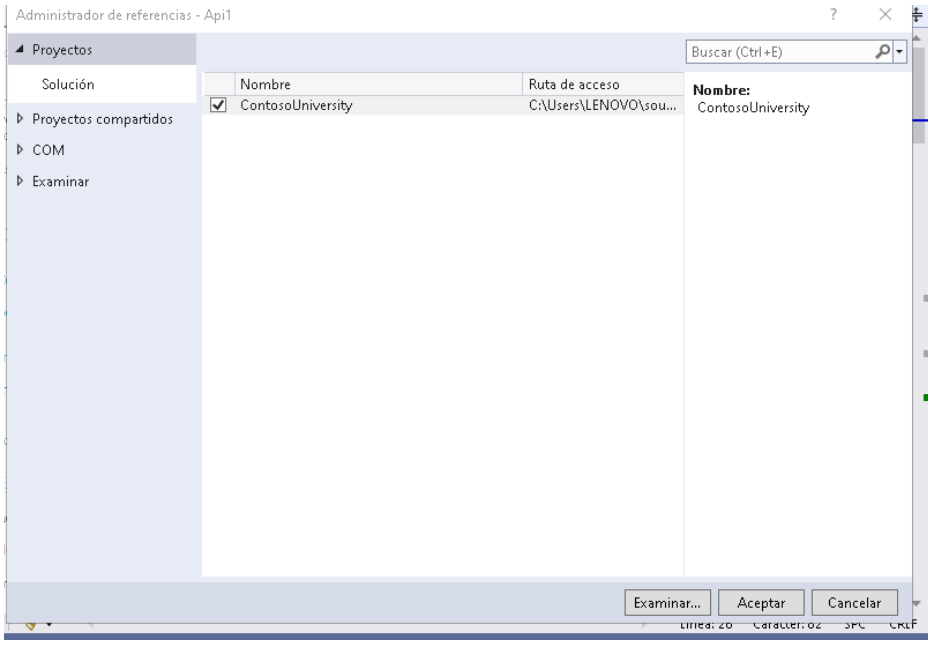

Ilustración 31 Selección del proyecto a referenciar

Para posteriormente, realizar la conexión a la base de datos, realizando unos cambios en el archivo de program.

```
using ContosoUniversity.Models;
using ContosoUniversity.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<SchoolContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("Default"),
        b => b.MigrationsAssembly("ContosoUniversity")));
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowSpecificOrigin",
        builder => builder.WithOrigins("http://localhost:7056") // Reemplaza con la URL de tu aplicación MVC
            .AllowAnyMethod()
            .AllowAnyHeader());
});

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Ilustración 32 program

Para posteriormente, realizar la conexión a la base de datos en el archivo de appsettings.

```
{
    "ConnectionStrings": {
        "Default": "Server=127.0.0.1,61250;Initial Catalog=ContosoUniversity;User Id=sa;Password=Carolina; TrustServerCertificate=true;"

    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*"
}
```

Ilustración 33 archivo appsettings

Students.

Se realiza la creación de la Api adecuada con la estructura del créate de students, para que sea coherente al introducir y guardar los datos para el usuario.
Cabe resaltar que aquí se hizo un controlador extra ya que al tomar el de Contos se repetían los datos de los estudiantes por lo cual se realizó los siguientes cambios:

Creación de modelo:

```
using ContosoUniversity.Models;

public class StudentDto
{
    public Student Student { get; set; }
    public int[] Enrollments { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
}
```

Ilustración 34 Modelo nuevo de students

Cambio en el controlador solo en el apartado de POST, agregando que solo tome los datos que queremos y que al momento de realizarlo sea por medio de FromFor:

```
[HttpPost]
public async Task<IActionResult> PostStudent([FromForm] StudentDto studentDto, IFormFile profilePicture,
    [FromHeader(Name = "From-Web")] bool fromWeb = false)
{
    try
    {
        if (ModelState.IsValid)
        {
            var student = new Student
            {
                LastName = studentDto.LastName,
                FirstMidName = studentDto.FirstMidName,
                EnrollmentDate = studentDto.EnrollmentDate,
                // Otras propiedades según sea necesario
            };

            if (profilePicture != null && profilePicture.Length > 0)
            {
                using (var memoryStream = new MemoryStream())
                {
                    await profilePicture.CopyToAsync(memoryStream);
                    student.ProfilePicture = memoryStream.ToArray();
                }
            }

            _context.Students.Add(student);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetStudent", new { id = student.ID }, student);
        }

        return BadRequest(ModelState);
    }
```

Ilustración 35 cambios en el controlador

En las demás opciones queda igual, a continuación, se muestran el resultado de crear, modificar y eliminar, y consulta un registro.

Curses

Se realiza la creación de la Api adecuada con la estructura del créate de Courses, para que sea coherente al introducir y guardar los datos para el usuario.

Cabe resaltar que aquí se hizo un controlador extra ya que al tomar el de Contos se repetían los datos de los estudiantes por lo cual se realizó los siguientes cambios:

Creación de modelo:

```
using ContosoUniversity.Models;

    1 referencia
    public class CoursesDto
    {
    //public Course Course { get; set; }
    1 referencia
    public int CourseID { get; set; }
    1 referencia
    public string Title { get; set; }
    1 referencia
    public int Credits { get; set; }
    1 referencia
    public int DepartmentID { get; set; }
    //public Department Department { get; set; }
    //public int[] Enrollments { get; set; }

}
```

Ilustración 36 Modelo nuevo de courses

Cambio en el controlador solo en el apartado de POST, agregando que solo tome los datos que queremos y que al momento de realizarlo sea por medio de FromFor:

```
[HttpPost]
0 referencias
public async Task<ActionResult> PostCourse([FromForm] CoursesDto coursesDto, [FromHeader(Name = "From-Web")] bool fromWeb = false)
{
    try
    {
        if (ModelState.IsValid)
        {
            var course = new Course
            {
                CourseID = coursesDto.CourseID,
                Title = coursesDto.Title,
                Credits = coursesDto.Credits,
                DepartmentID = coursesDto.DepartmentID,
                //Department = coursesDto.Department,
                //Enrollments = coursesDto.Enrollments,
                // Otras propiedades según sea necesario
            };

            _context.Courses.Add(course);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetCourse", new { id = course.CourseID }, course);
        }

        return BadRequest(ModelState);
    }
    catch (Exception ex)
    {
        // Log the exception or handle it according to your application's needs
        return StatusCode(500, "Internal server error");
    }
}
```

Ilustración 37 cambios en el controlador

En las demás opciones queda igual, a continuación, se muestran el resultado de crear, modificar y eliminar, y consulta un registro.

Instructors

Se realiza la creación de la Api adecuada con la estructura del créate de Instructors, para que sea coherente al introducir y guardar los datos para el usuario.
Cabe resaltar que aquí se hizo un controlador extra ya que al tomar el de Contos se repetían los datos de los estudiantes por lo cual se realizó los siguientes cambios:

Creación de modelo:

```
using ContosoUniversity.Models;
1 referencia
public class InstructorsDto
{
    //public Instructor Instructor { get; set; }
    0 referencias
    public int ID { get; set; }
    1 referencia
    public string LastName { get; set; }
    1 referencia
    public string FirstMidName { get; set; }
    1 referencia
    public DateTime HireDate { get; set; }
    0 referencias
    public ICollection<CourseAssignment>? CourseAssignments { get; set; }

}
```

Ilustración 38 Modelo nuevo de Instructors

Cambio en el controlador solo en el apartado de POST, agregando que solo tome los datos que queremos y que al momento de realizarlo sea por medio de FromFor:

```
[HttpPost]
0 referencias
public async Task<IActionResult> PostInstructors([FromForm] InstructorsDto instructorsDto, IFormFile profilePictureI,
    [FromHeader(Name = "From-Web")] bool fromWeb = false)
{
    try
    {
        if (ModelState.IsValid)
        {
            var instructor = new Instructor
            {
                //ID=instructorsDto.ID,
                LastName = instructorsDto.LastName,
                FirstMidName= instructorsDto.FirstMidName,
                HireDate=instructorsDto.HireDate,
                //CourseAssignments=instructorsDto.CourseAssignments,

                // Otras propiedades según sea necesario
            };

            if (profilePictureI != null && profilePictureI.Length > 0)
            {
                using (var memoryStream = new MemoryStream())
                {
                    await profilePictureI.CopyToAsync(memoryStream);
                    instructor.ProfilePictureI = memoryStream.ToArray();
                }
            }

            _context.Instructors.Add(instructor);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetInstructor", new { id = instructor.ID }, instructor);
        }
```

Ilustración 39 cambios en el controlador

En las demás opciones queda igual, a continuación, se muestran el resultado de crear, modificar y eliminar, y consulta un registro.

Departaments

Se realiza la creación de la Api adecuada con la estructura del créate de Departaments, para que sea coherente al introducir y guardar los datos para el usuario.

Cabe resaltar que aquí se hizo un controlador extra ya que al tomar el de Contos se repetían los datos de los estudiantes por lo cual se realizó los siguientes cambios:

Creación de modelo:

```
using ContosoUniversity.Models;
1 referencia
public class DepartmentsDto
{
    //public Department Department { get; set; }
    0 referencias
    public int DepartmentID { get; set; }
    1 referencia
    public string Name { get; set; }
    1 referencia
    public decimal Budget { get; set; }
    1 referencia
    public DateTime StartDate { get; set; }
    1 referencia
    public int InstructorID { get; set; }
   // public byte[]? RowVersion { get; set; }
  //  public Instructor? Administrator { get; set; }
}
```

Ilustración 40 Modelo nuevo de departaments

Cambio en el controlador solo en el apartado de POST, agregando que solo tome los datos que queremos y que al momento de realizarlo sea por medio de FromFor:

```
[HttpPost]
0 referencias
public async Task<ActionResult> PostDepartment([FromForm] DepartmentsDto departmentsDto,
    [FromHeader(Name = "From-Web")] bool fromWeb = false)
{
    try
    {
        if (ModelState.IsValid)
        {
            var department = new Department
            {
               // DepartmentID = departmentsDto.DepartmentID,
                Name=departmentsDto.Name,
                Budget=departmentsDto.Budget,
                StartDate=departmentsDto.StartDate,
                InstructorID=departmentsDto.InstructorID,
               // RowVersion=departmentsDto.RowVersion,


                // Otras propiedades según sea necesario
            };

            _context.Departments.Add(department);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetDepartment", new { id = department.DepartmentID }, department);
        }

        return BadRequest(ModelState);
    }
    catch (Exception ex)
    {
        // Log the exception or handle it according to your application's needs
        return StatusCode(500, "Internal server error");
    }
```

Ilustración 41 cambios en el controlador

En las demás opciones queda igual, a continuación, se muestran el resultado de crear, modificar y eliminar, y consulta un registro.

Precauciones de versiones.

Los entornos de desarrollo admiten el desarrollo de aplicaciones ASP.NET Core MVC y el uso de Entity Framework Core, pueden surgir desafíos de compatibilidad y versiones al seguir el tutorial en ambas versiones de Visual Studio.

Versiones de .NET Core y ASP.NET Core: Visual Studio 2019 inicialmente soportaba ciertas versiones de .NET Core (con las siguientes versiones 5, 3.1, 3.0, 2.2, 2.1 y 1.1.) y ASP.NET Core , mientras que Visual Studio 2022 ofrece soporte para versiones más recientes (8.0 versión preliminar, 7.0, 6.0.). Si el tutorial está basado en una versión más antigua de .NET o ASP.NET Core, podrías enfrentar desafíos al intentar ejecutar el código en una versión más reciente al igual que inversamente si se utiliza un tutorial con una versión reciente y nuestra versión es anterior. La frma de prevenir esta situación es la siguiente:

Una vez conociendo el editor que tenemos deberemos identificar con que versión de .NET Core trabajaremos para así seleccionarlo en la versión de los tutoriales y la versión con la que trabajaremos.

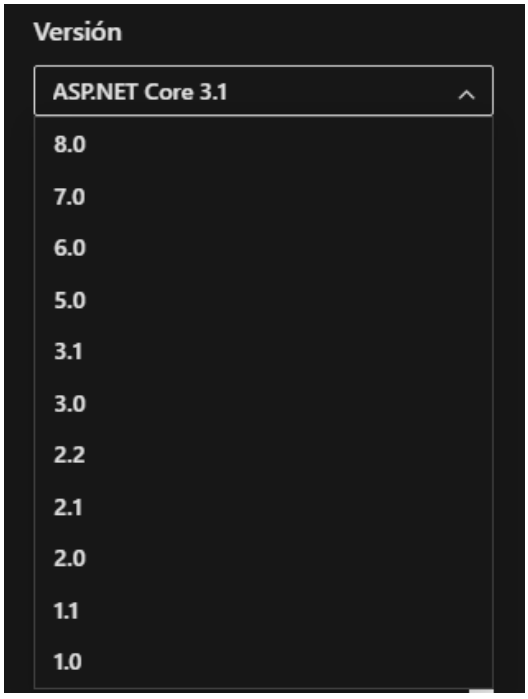

Ilustración 42 Versiones disponibles del tutorial

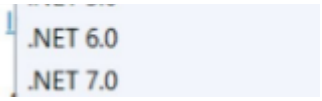

Ilustración 43 Versiones disponibles en nuestro IDE 2022 esto incluyendo la versión 8 que en algunas versiones ya está disponible

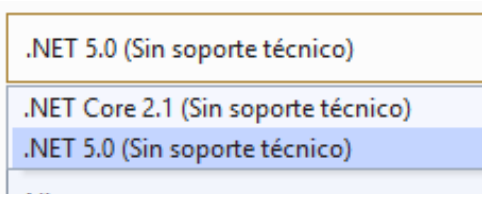

Ilustración 44 Versiones disponibles en nuestro IDE 2019

Como se puede ver en la ilustración 3 no todas las plataformas están disponibles y además de eso las versiones 2.1 y 5.0 ya no cuentan con soporte técnico lo cual provoca que diversas de las paqueterías utilizadas ya no funcionen de manera correcta.

Versiones de Entity Framework Core y paquetes relacionados: Las versiones de Entity Framework Core pueden haber cambiado entre las versiones de Visual Studio. Si el tutorial se basa en una versión específica de EF Core y se está utilizando una versión diferente en el entorno de desarrollo, puede encontrarse con diferencias en la sintaxis, comportamiento y características admitidas.

Compatibilidad de paquetes NuGet y herramientas: Las versiones de los paquetes NuGet utilizados en el tutorial podrían no ser los mismos que los disponibles en tu versión de Visual Studio. Puede resultar en incompatibilidades o cambios en las API entre versiones, lo que afectaría la forma en que se interactúa con las bibliotecas.

Si no se utilizan las versiones correctas de cada paquete o herramienta, se enfrentan problemas como:

1. Errores de compilación: Las API pueden haber cambiar entre versiones, lo que provocaría errores de compilación en el proyecto si se está utlizando utilizando una versión diferente a la del tutorial.
2. Comportamiento inesperado: Funcionalidades específicas podrían haber sido modificadas o introducidas en versiones posteriores, lo que podría llevar a un comportamiento inesperado en la aplicación si se está siguiendo un tutorial basado en una versión más antigua.

3. Problemas de migración de bases de datos: Entity Framework Core a menudo requiere migraciones de base de datos cuando se actualizan modelos. Utilizar versiones incompatibles podría hacer que las migraciones no funcionen correctamente o que la base de datos no se actualice adecuadamente.

Referencias

Tdykstra. (2023, 11 abril). Tutorial: Introducción a EF Core en una aplicación web de ASP.NET Core MVC. Microsoft Learn. https://learn.microsoft.com/eses/aspnet/core/data/ef-mvc/intro?view=aspnetcore-7.0

SQLServer. (s. f.). iessanvicente.

https://iessanvicente.com/colaboraciones/sqlserver.pdf

IDE de Visual Studio 2022: herramienta de programación para desarrolladores de software. (2023, 28 septiembre). Visual Studio.

https://visualstudio.microsoft.com/es/vs/

Microsoft. (30 de Noviembre de 2023). *Creación de API web con ASP.NET Core*. Obtenido de Microsoft: https://learn.microsoft.com/es-es/aspnet/core/web-api/?view=aspnetcore-7.0

Microsoft. (30 de Noviembre de 2023). *Elección entre las API basadas en controlador y las API mínimas*. Obtenido de Microsoft: https://learn.microsoft.com/es-es/aspnet/core/fundamentals/apis?view=aspnetcore-7.0

Microsoft. (03 de Enero de 2024). *Tutorial: Creación de una API web con ASP.NET Core*. Obtenido de Microsoft: https://learn.microsoft.com/es-es/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio

IDE de Visual Studio 2022: herramienta de programación para desarrolladores de software. (2023, 28 septiembre). Visual Studio.

https://visualstudio.microsoft.com/es/vs/