



Desarrollo Web

Documento Técnico de Proyecto Final

Alumnos:

Carlos Emmanuel Aguilar Ramirez (336277)

Daniel Roberto Barrios Martínez (261813)

Pedro Quiroz Carreón (325499)

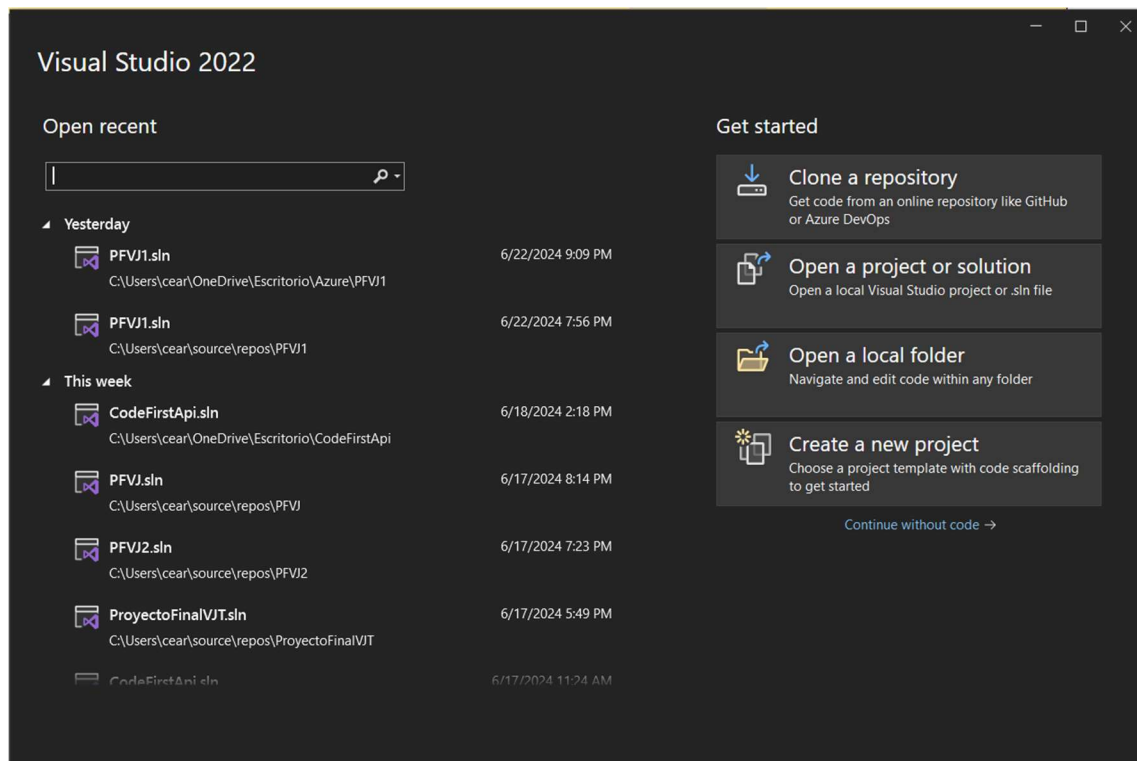
Rodolfo Eduardo Córdova Pérez (334876)

Prof.: Margarita Mondragón Arellano

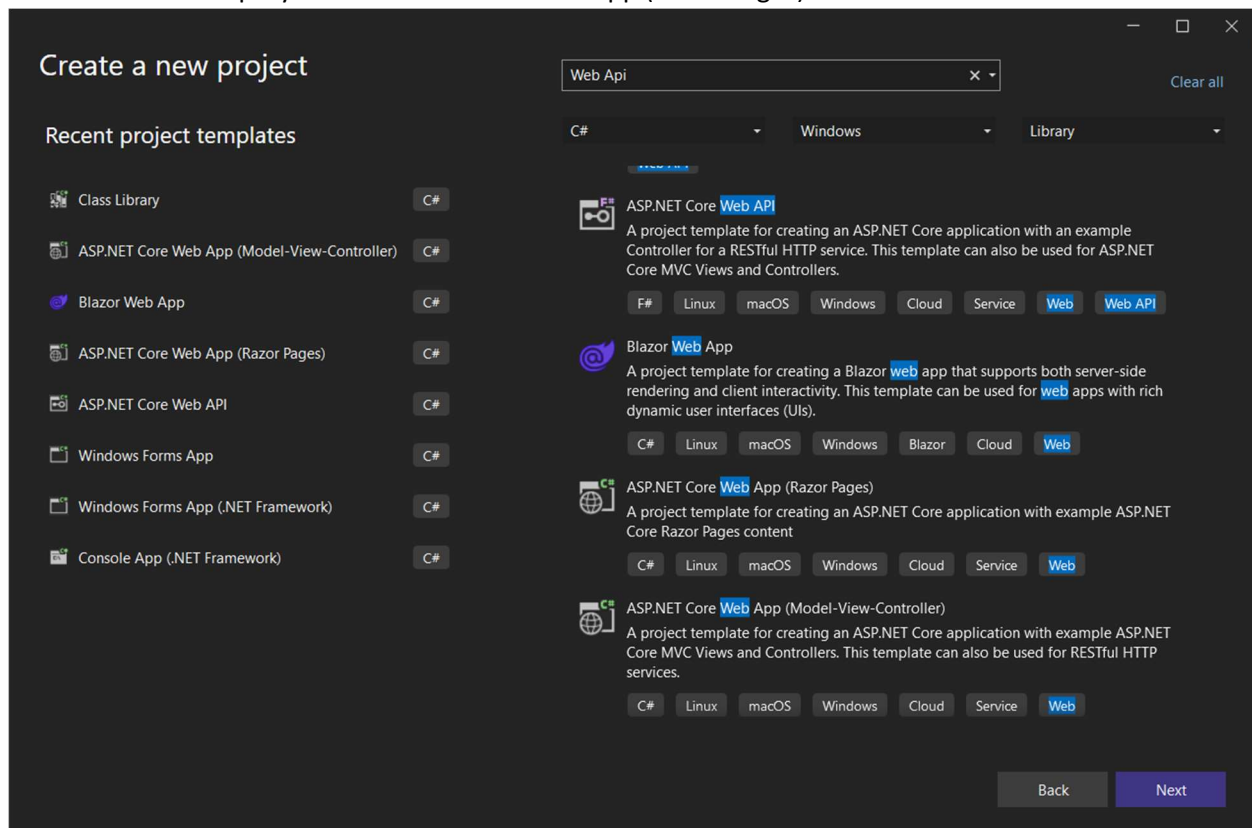
Carrera: Licenciatura en Informática y Tecnologías
Computacionales (6°A)

24/06/2024

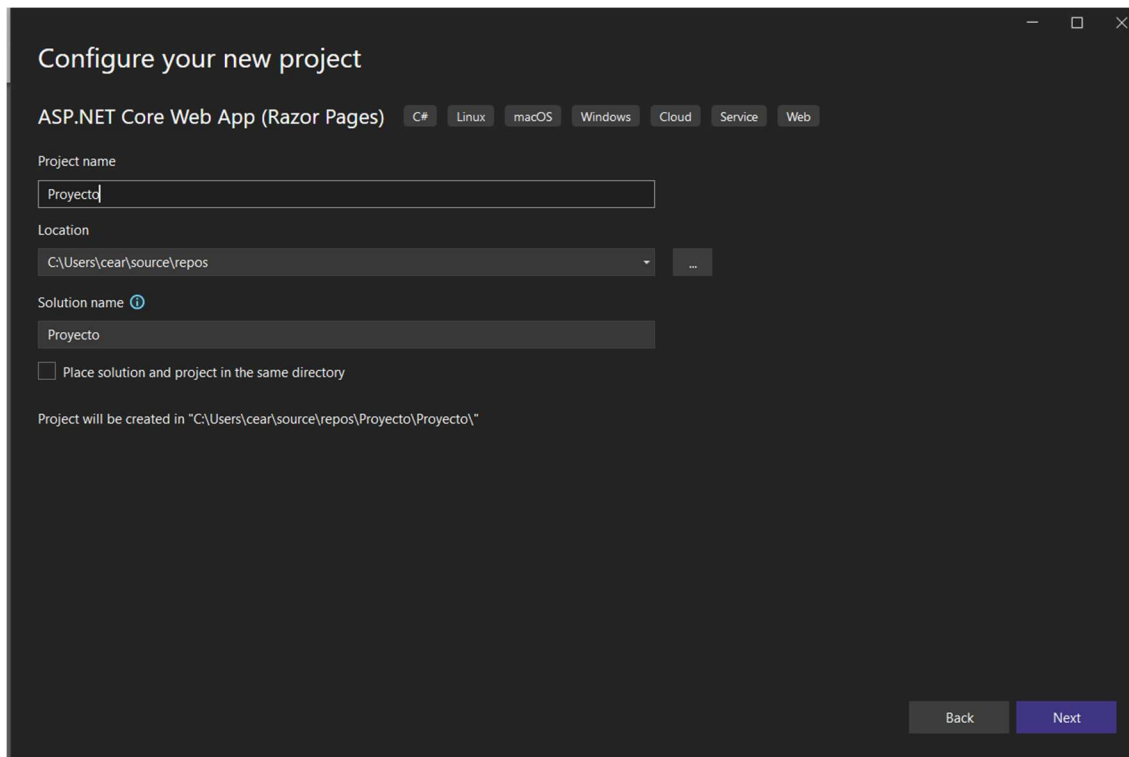
Abrimos nuestro Visual Studio 2022



Creamos un nuevo proyecto ASP .NET Core Web App (Razor Pages)



Le asignamos el nombre Proyecto



Configure your new project

ASP.NET Core Web App (Razor Pages) C# Linux macOS Windows Cloud Service Web

Project name
Proyecto

Location
C:\Users\cear\source\repos

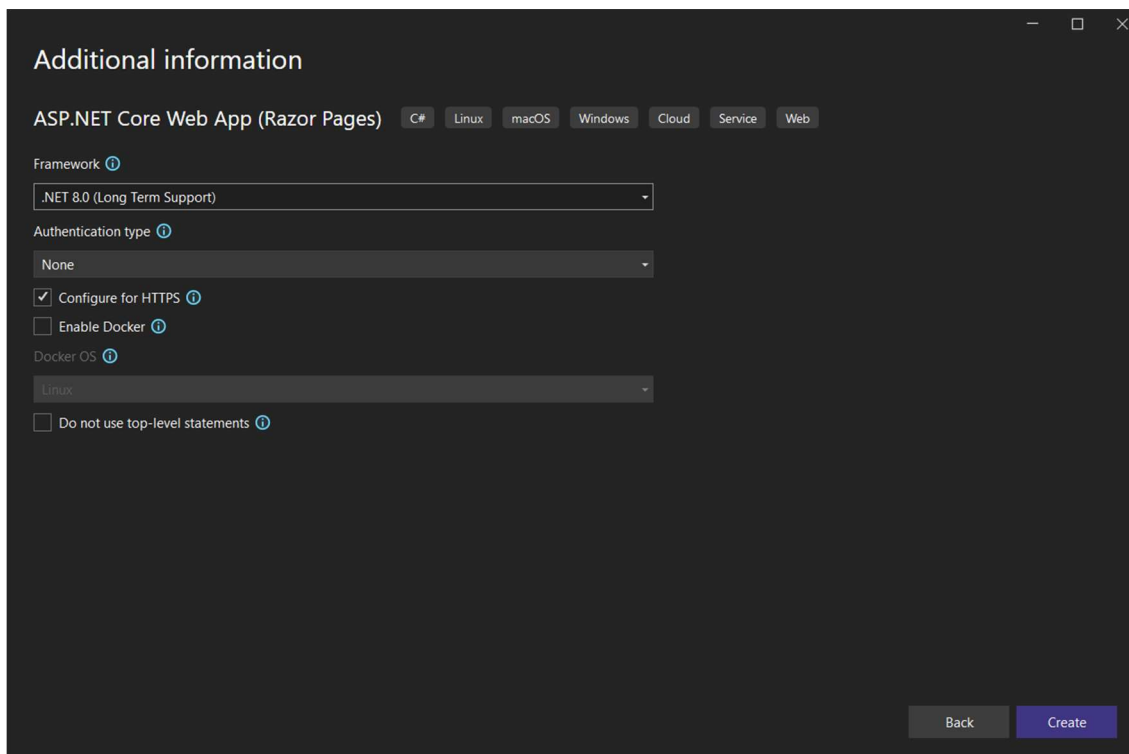
Solution name
Proyecto

☐ Place solution and project in the same directory

Project will be created in "C:\Users\cear\source\repos\Proyecto\Proyecto"

Back Next

Le asignamos el framework mas reciente ósea (.NET 8.0)



Additional information

ASP.NET Core Web App (Razor Pages) C# Linux macOS Windows Cloud Service Web

Framework
.NET 8.0 (Long Term Support)

Authentication type
None

☒ Configure for HTTPS

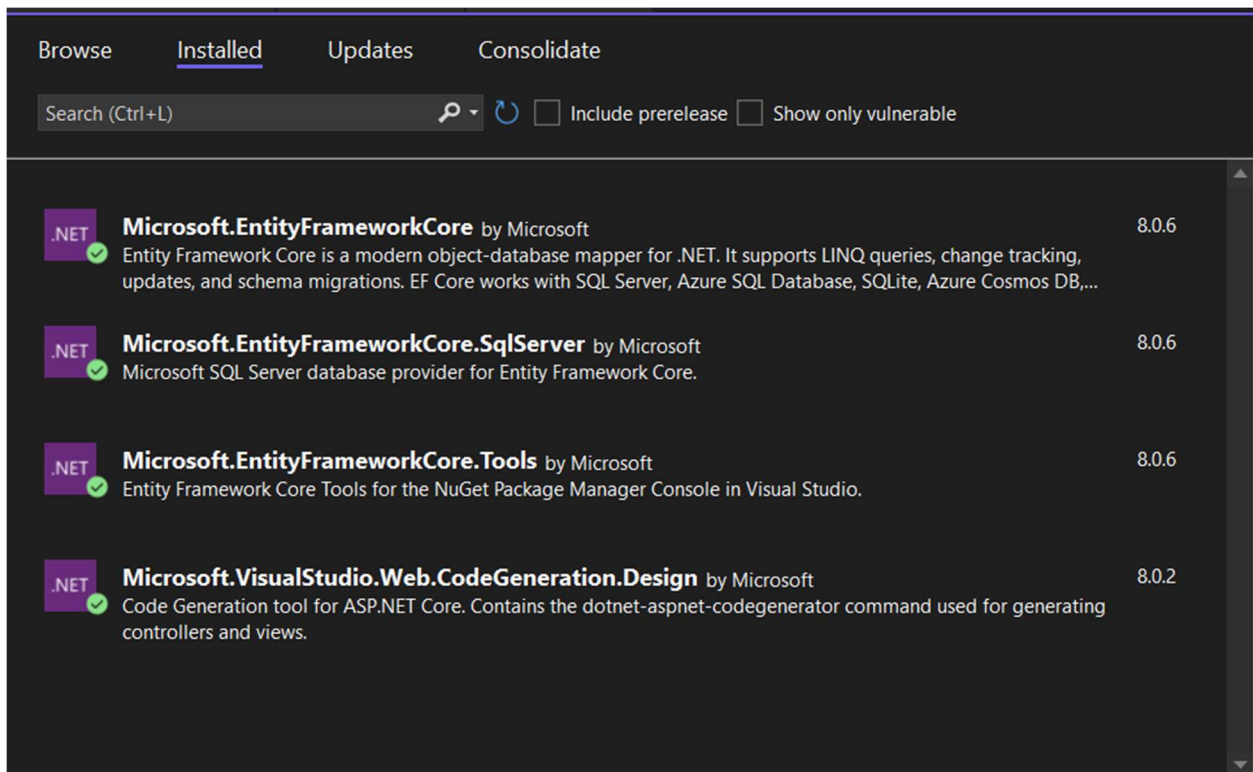
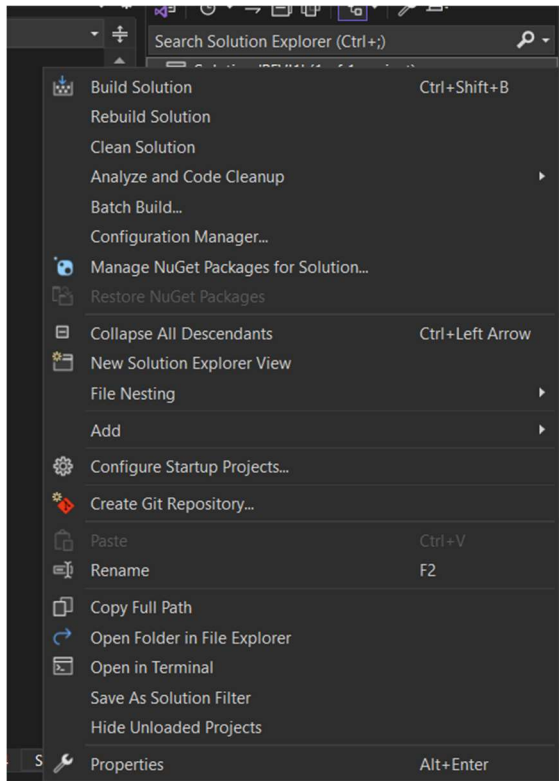
☐ Enable Docker

Docker OS
Linux

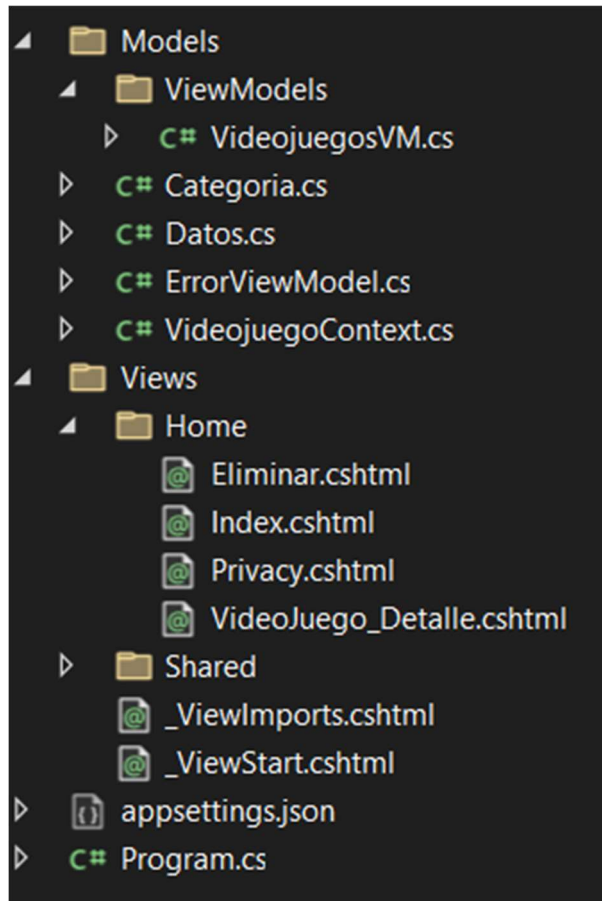
☐ Do not use top-level statements

Back Create

Después vamos a nuestro proyecto y vamos a nuestra barra de explorador de solución y vamos a instalar los siguientes paquetes Nuggets



Una vez que hayamos instalado nuestros paquetes nuggets vamos creando nuestras carpetas raíz (Models, ViewModels, Views)



Después de haber creado nuestras carpetas, creamos nuestras clases, empezaremos con crear nuestras tablas para la base de datos (Esta es la tabla de Datos) Datos tiene los siguientes parámetros IdVideojuego, Nombre, Precio, Stock, IdCategoria como llave foránea.

```
PFVJ1 PFVJ1.Models.Datos
4 namespace PFVJ1.Models
5 {
6     // Clase que representa los datos de un videojuego.
7     public class Datos
8     {
9         // Llave primaria, generada automáticamente por la base de datos.
10        [Key]
11        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
12        public int IdVideojuego { get; set; }
13
14        // Nombre del videojuego.
15        public string? Nombre { get; set; }
16
17        // Precio del videojuego.
18        public decimal? Precio { get; set; }
19
20        // Cantidad de stock del videojuego.
21        public int? Stock { get; set; }
22
23        // Identificador de la categoría del videojuego.
24        public int? IdCategoria { get; set; }
25
26        // Relación de clave foránea con la tabla de categorías.
27        [ForeignKey("IdCategoria")]
28        public virtual Categoria? VideogamesDB { get; set; }
29    }
30 }
31
```

Después esta la clase categoría el cual la tabla tiene lo siguientes parámetros CategoriaID, NombreCategoria

```
PFVJ1 PFVJ1.Models.Categoria Categoria()
1 using static System.Runtime.InteropServices.JavaScript.JSType; // Importa la clase JSType del espacio de nombres System.Runtime.InteropServices.JavaScript
2 using System.ComponentModel.DataAnnotations.Schema; // Espacio de nombres para atributos de esquema de base de datos
3 using System.ComponentModel.DataAnnotations; // Espacio de nombres para atributos de validación de datos
4
5 namespace PFVJ1.Models
6 {
7     // Clase que representa una categoría de videojuegos.
8     public class Categoria
9     {
10        // Constructor que inicializa la colección de Datos.
11        public Categoria()
12        {
13            Datos = new HashSet<Datos>(); // Inicializa la colección de Datos.
14        }
15
16        // Llave primaria, generada automáticamente por la base de datos.
17        [Key]
18        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
19        public int CategoriaID { get; set; }
20
21        // Nombre de la categoría. Puede ser nulo.
22        public string? NombreCategoria { get; set; }
23
24        // Colección de Datos relacionados con la categoría.
25        public virtual ICollection<Datos> Datos { get; set; }
26    }
27 }
28
```

Después creamos un context para poder manejar nuestras tablas y ver como serán relacionadas entre si

```
PFV1 | PFV1.Models.VideojuegoContext | VideojuegoContext(DbContextOptions<VideojuegoContext> opt)
36 // Configuración de la entidad Categoría.
37 modelBuilder.Entity<Categoría>(entity =>
38 {
39     entity.HasKey(e => e.CategoríaID).HasName("PK_IDCATEGORIA"); // Definir la llave primaria.
40     entity.ToTable("Categoría"); // Nombre de la tabla en la base de datos.
41
42     entity.Property(e => e.NombreCategoría)
43         .HasMaxLength(50) // Longitud máxima de 50 caracteres.
44         .IsUnicode(false); // No se utiliza codificación Unicode.
45 });
46
47 // Configuración de la entidad Datos.
48 modelBuilder.Entity<Datos>(entity =>
49 {
50     entity.HasKey(e => e.IdVideojuego).HasName("PK_IDVideojuego"); // Definir la llave primaria.
51     entity.ToTable("Datos"); // Nombre de la tabla en la base de datos.
52
53     entity.Property(e => e.Nombre)
54         .HasMaxLength(60) // Longitud máxima de 60 caracteres.
55         .IsUnicode(false); // No se utiliza codificación Unicode.
56
57     entity.Property(e => e.Precio)
58         .HasMaxLength(60) // Longitud máxima de 60 caracteres.
59         .IsUnicode(false); // No se utiliza codificación Unicode.
60
61     entity.Property(e => e.Stock)
62         .HasMaxLength(60) // Longitud máxima de 60 caracteres.
63         .IsUnicode(false); // No se utiliza codificación Unicode.
64
65     // Definir la relación entre Datos y Categoría.
66     entity.HasOne(d => d.VideojuegosDB)
67         .WithMany(p => p.Datos)
68         .HasForeignKey(d => d.IdCategoría)
69         .HasConstraintName("FK_Categoría"); // Nombre de la restricción de clave foránea.
70 });
71
72 // Método parcial para configuraciones adicionales en el modelo.
73 OnModelCreatingPartial(modelBuilder);
74
75 // Método parcial para permitir configuraciones adicionales en subclases.
76 // reference
77 partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
78
79
80
81
```

A continuación, Explicaremos las vistas de nuestra pagina para poder Eliminar, Crear, Editar y Consultar

En todas las clases siguientes usamos html para poder crear los cuadros de texto y los botones y referenciado a otras clases como hipervínculo

1.-Eliminar, esta clase es para eliminar nuestros elementos en la base de datos y borraríamos los datos de los videojuegos

```
7 Layout = "~/Views/Shared/_Layout.cshtml";
8 }
9
10 // Mensaje de advertencia en color rojo para confirmar la eliminación del juego.
11 <div class="text-danger"> ¿Está seguro de eliminar el Juego?</div>
12 <hr />
13 <div class="row">
14     <div class="col-sm-4">
15         <!-- Formulario para eliminar un videojuego -->
16         <form asp-action="Eliminar" asp-controller="Home" method="post">
17             <!-- Campo oculto para el ID del videojuego -->
18             <input type="hidden" asp-for="IdVideojuego" />
19
20             <!-- Campo deshabilitado para mostrar el nombre del videojuego -->
21             <div class="mb-2">
22                 <label class="form-label" asp-for="Nombre">Nombre completo</label>
23                 <input class="form-control form-control-sm" asp-for="Nombre" type="text" disabled />
24             </div>
25
26             <!-- Campo deshabilitado para mostrar el precio del videojuego -->
27             <div class="mb-2">
28                 <label class="form-label" asp-for="Precio">Precio</label>
29                 <input class="form-control form-control-sm" asp-for="Precio" type="text" disabled />
30             </div>
31
32             <!-- Campo deshabilitado para mostrar el stock del videojuego -->
33             <div class="mb-2">
34                 <label class="form-label" asp-for="Stock">Stock</label>
35                 <input class="form-control form-control-sm" asp-for="Stock" type="text" disabled />
36             </div>
37
38             <!-- Campo deshabilitado para mostrar la categoría del videojuego -->
39             <div class="mb-2">
40                 <label class="form-label" asp-for="IdCategoría">Categoría</label>
41                 <input class="form-control form-control-sm" asp-for="VideojuegosDB.NombreCategoría" type="text" disabled />
42             </div>
43
44             <!-- Botones para confirmar la eliminación o volver a la página de inicio -->
45             <div class="mb-2">
46                 <!-- Botón para confirmar la eliminación del videojuego -->
47                 <button class="btn btn-danger btn-sm" type="submit">Eliminar</button>
48
49                 <!-- Botón para volver a la página de inicio sin eliminar -->
50                 <a class="btn btn-dark btn-sm" asp-action="Index" asp-controller="Home">Volver</a>
51             </div>
52         </form>
53     </div>
```


2.- En este código podemos registrar los códigos y al igual también actualizar los códigos usando html con las barras de texto y los botones necesarios que se deberían de actualizar

```

19
20 <div class="row">
21   <div class="col-sm-4">
22     <!-- Formulario para registrar o editar un videojuego -->
23     <form asp-action="VideoJuego_Detalle" asp-controller="Home" method="post">
24       <!-- Campo oculto para el ID del videojuego -->
25       <input type="hidden" asp-for="oDatos.IdVideojuego" />
26
27       <!-- Campo para el nombre del videojuego -->
28       <div class="mb-2">
29         <label class="form-label" asp-for="oDatos.Nombre">Nombre del Videojuego</label>
30         <input class="form-control form-control-sm" asp-for="oDatos.Nombre" type="text" />
31       </div>
32
33       <!-- Campo para el precio del videojuego -->
34       <div class="mb-2">
35         <label class="form-label" asp-for="oDatos.Precio">Precio del Videojuego</label>
36         <input class="form-control form-control-sm" asp-for="oDatos.Precio" type="text" />
37       </div>
38
39       <!-- Campo para el stock del videojuego -->
40       <div class="mb-2">
41         <label class="form-label" asp-for="oDatos.Stock">Stock del Videojuego</label>
42         <input class="form-control form-control-sm" asp-for="oDatos.Stock" type="text" />
43       </div>
44
45       <!-- Campo para seleccionar la categoría del videojuego -->
46       <div class="mb-2">
47         <label class="form-label" asp-for="oDatos.IdCategoria">Categoría del Videojuego</label>
48         <select class="form-select form-select-sm" asp-for="oDatos.IdCategoria" asp-items="@Model.oLista">
49           <option selected disabled--> Seleccionar --</option>
50         </select>
51       </div>
52
53       <!-- Botones para registrar o actualizar el videojuego -->
54       <div class="mb-2">
55         @if (Model.oDatos.IdVideojuego == 0)
56         {
57           <!-- Botón para registrar un nuevo videojuego -->
58           <button class="btn btn-primary btn-sm w-25" type="submit">Registrar</button>
59         }
60         else
61         {
62           <!-- Botón para actualizar un videojuego existente -->
63           <button class="btn btn-primary btn-sm w-25" type="submit">Actualizar</button>

```

3.-En esta consultaremos y nos dará una vista completa de los datos de los videojuegos con los botones de crear, editar y eliminar los datos

```

@{
    ViewData["Title"] = "Proyecto UAA";
}

<div class="text-center">
    <h1 class="display-4">Bienvenido!</h1>
    <p>Proyecto de Tienda de VideoJuegos UAA</p>
</div>

<div class="card">
    <div class="card-header">
        <h2>Lista de VideoJuegos</h2>
    </div>
    <div class="card-body">
        <a class="btn btn-success btn-sm" asp-action="VideoJuego_Detalle" asp-controller="Home" asp-route-IdVideojuego="0">Agregar VideoJuego</a>
        <hr />
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Nombre del Videojuego</th>
                    <th>Precio</th>
                    <th>Stock</th>
                    <th>Categoría</th>
                </tr>
            </thead>
            <tbody>
                <!-- Iteración sobre cada elemento en el modelo -->
                @foreach (var item in Model)
                {
                    <tr>
                        <!-- Columnas de la tabla con los datos de cada videojuego -->
                        <td>@item.Nombre</td>
                        <td>@item.Precio</td>
                        <td>@item.Stock</td>
                        <td>@item.VideoJuegosDB.NombreCategoría</td>
                        <td>
                            <!-- Botones para editar y eliminar cada videojuego -->
                            <a class="btn btn-primary btn-sm" asp-action="VideoJuego_Detalle" asp-controller="Home" asp-route-IdVideojuego="@item.IdVideojuego">Editar</a>
                            <a class="btn btn-danger btn-sm" asp-action="Eliminar" asp-controller="Home" asp-route-IdVideojuego="@item.IdVideojuego">Eliminar</a>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>

```


En la parte de la clase program.cs es el startup del programa entonces lo primero que hacemos es conectarnos a nuestra base de datos.

```
using Microsoft.EntityFrameworkCore; // Espacio de nombres para Entity Framework Core
using PFVJ.Models; // Espacio de nombres para los modelos de la aplicación

var builder = WebApplication.CreateBuilder(args); // Crear el constructor para la aplicación web

// Agregar servicios para controladores con vistas al contenedor de servicios.
builder.Services.AddControllersWithViews();

// Configurar el contexto de la base de datos para usar SQL Server con la cadena de conexión especificada.
builder.Services.AddDbContext<VideoJuegoContext>(>options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("VideoJuegosConnection"))
);

var app = builder.Build(); // Construir la aplicación web

// Configuración para manejar excepciones en entornos que no son de desarrollo.
if (app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error"); // Usar una página de error genérica.
}

// Habilitar el uso de archivos estáticos (como archivos CSS, JS, imágenes, etc.)
app.UseStaticFiles();

app.UseRouting(); // Habilitar el enrutamiento de la aplicación.

app.UseAuthorization(); // Habilitar la autorización.

// Configurar la ruta predeterminada para los controladores.
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

// Ejecutar la aplicación.
app.Run();
```

En la clase de json esta es la connection necesaria para hacer el Azure

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "VideoJuegosConnection": "Server=tcprojectosua.database.windows.net;Initial Catalog=VideoJuegos;Persist Security Info=False;User ID=adminUA;Password=Administrador1997;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
  }
}
```

Al final tenemos nuestro HomeController donde tenemos varios métodos de HttpGet y HttpPost el cual hará la lógica de la creación, edición y eliminación de datos

```
// HomeController maneja las operaciones principales relacionadas con las entidades Videojuego.
1 referencia
public class HomeController : Controller
{
    // Contexto de base de datos para acceder a la base de datos de Videojuego.
    private readonly VideojuegoContext _DBContext;

    // Constructor para inicializar el contexto de la base de datos.
    0 referencias
    public HomeController(VideojuegoContext context)
    {
        _DBContext = context;
    }

    // Método de acción para mostrar la lista de videojuegos.
    0 referencias
    public IActionResult Index()
    {
        // Recuperar la lista de videojuegos incluyendo sus categorías relacionadas.
        List<Datos> lista = _DBContext.Datos.Include(c => c.VideojuegosDB).ToList();
        return View(lista); // Pasar la lista a la vista.
    }

    // Método GET para mostrar los detalles de un videojuego específico o un formulario para crear uno nuevo.
    [HttpGet]
    0 referencias
    public IActionResult VideoJuego_Detalle(int IdVideojuego)
    {
        // Inicializar el modelo de vista para los detalles del videojuego.
        VideojuegosVM oVideojuegoVM = new VideojuegosVM()
        {
            oDatos = new Datos(), // Crear un nuevo objeto Datos.
            oLista = _DBContext.Categoria.Select(categoria => new SelectListItem()
            {
                Text = categoria.NombreCategoria, // Nombre de la categoría.
                Value = categoria.CategoriaID.ToString() // ID de la categoría.
            }).ToList() // Convertir a una lista de SelectListItem.
        };

        // Si se proporciona un ID de videojuego existente, recuperar sus detalles.
        if (IdVideojuego != 0)
        {
            oVideojuegoVM.oDatos = _DBContext.Datos.Find(IdVideojuego);
        }
        return View(oVideojuegoVM); // Pasar el modelo de vista a la vista.
    }

    // Método POST para guardar los detalles del videojuego (crear o actualizar).
    [HttpPost]
    0 referencias
    public IActionResult VideoJuego_Detalle(VideojuegosVM oVideojuegoVM)
    {
        // Si el videojuego es nuevo (ID es 0), agregarlo a la base de datos.
        if (oVideojuegoVM.oDatos.IdVideojuego == 0)
        {
            _DBContext.Datos.Add(oVideojuegoVM.oDatos);
        }
        else
        {
            // Si el videojuego existe, actualizar sus detalles en la base de datos.
        }
    }
}
```