UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA
UNAN – LEÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
INGENIERÍA EN SISTEMAS DE LA INFORMACIÓN

AÑO LECTIVO: 2025
SEMESTRE: II

Componente Curricular: programación Orientada a la Web II

Grupo: GP1
Profesor(a): Ing.
Autor:

1. Brandon Gonzalo Altamirano Avendaño

León, Nicaragua, 31 de Agosto del 2025.

"¡A la Libertad por la Universidad!"

1. Estructura del proyecto

ProyectoVistas/

Controllers/
– ParrotController.cs # Controlador MVC para vistas HTML
– ParrotApiController.cs # Controlador API para devolver JSON
– AboutController.cs # Controlador para la página "Acerca de"

– Models/
– Parrot.cs # Modelo que representa un loro
– Parrot.json # Datos de los loros en formato JSON

- Views/
– Parrot/
– Index.cshtml # Vista principal de lista de loros
– Details.cshtml # Vista de detalle de cada loro

– Shared/
– _LayoutMio.cshtml # Layout personalizado usando Bootstrap 5.3
– About/
– Index.cshtml # Vista de "Acerca de"

– wwwroot/
– css/
– site.css # Estilos propios
– lib/
– bootstrap/
– jquery/

– Program.cs # Configuración principal de la app
– ProyectoVistas.csproj # Archivo de proyecto con referencias a paquetes
– appsettings.json # Configuración de la app

Básicamente, todo gira alrededor del JSON de loros y los controladores que lo usan para mostrar vistas o devolver datos en formato JSON.

2. Controllers
ParrotController (MVC)

Qué es: un Controller clásico de MVC.

Qué hace: sirve vistas HTML usando la información de Parrot.json.

Métodos principales:

Index() → muestra la lista completa de loros en Index.cshtml.

Find(string parrot) → busca loros por nombre y devuelve la misma vista pero filtrada.

Details(int id) → muestra los detalles de un loro específico en Details.cshtml.

ParrotApiController (API)

Qué es: un ControllerBase con [ApiController].

Qué hace: devuelve los datos en JSON, para que otras apps o front-end puedan consumirlos.

Métodos principales:

Index() → devuelve todos los loros.

Find(string parrot) → devuelve los loros que coinciden con la búsqueda.

Details(int id) → devuelve los datos de un loro específico por ID.

Nota: aquí solo cambió la forma de devolver la información (Ok(...) en lugar de View(...)), pero el resto del código se puede mantener igual que tu MVC.

AboutController

Qué es: un Controller simple.

Qué hace: muestra la página “Acerca de”.

Método:

Index() → devuelve la vista Views/About/Index.cshtml con tu mensaje.

3. Models
Parrot.cs

Qué contiene: propiedades como Id, Name, ScientificName, Family, ConservationStatus, Lifespan, Size, Weight, Habitat, Distribution, Diet, Description, Language, Country, Poster, Images.

Para qué sirve: representa un loro en C#, igual que en el JSON.

Parrot.json

Qué contiene: un array con los datos de todos los loros.

Cómo se usa: se lee con System.IO.File.ReadAllText y JsonConvert.DeserializeObject<List<Parrot>>().

Por qué es útil: mantiene los datos separados del código, lo que hace que sea fácil actualizar la información y usarla tanto en vistas como en la API.

4. Views

Index.cshtml → muestra la lista de loros con un carrusel de imágenes.

Details.cshtml → muestra un loro completo con todas sus propiedades.

About/Index.cshtml → tu página "Acerca de".

_LayoutMio.cshtml → tu layout personalizado que incluye:

Menú de navegación (Home, About…)

Referencias a Bootstrap 5.3 y jQuery

@RenderBody() para insertar las vistas dinámicamente

Esto hace que todas las vistas compartan el mismo estilo y menú.

5. wwwroot

Contiene los archivos estáticos:

Bootstrap (CSS y JS)

jQuery

Tu CSS personalizado (site.css)

6. Program.cs

Configura la app y los servicios:

MVC y Razor Pages

Runtime compilation (para ver cambios en vistas sin reiniciar la app)

Ejemplo de configuración mínima:

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews().AddRazorRuntimeCompilation();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();


app.Run();
```

Esto hace que tu app sepa dónde empezar (Parrot/Index) y pueda servir tanto vistas como archivos estáticos.

7. Paquetes NuGet (ProyectoVistas.csproj)

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.RunTimeCompilation"
Version="9.0.8" />
  <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />
</ItemGroup>
```

Razor Runtime Compilation → permite que los cambios en las vistas se vean inmediatamente.

Newtonsoft.Json → para leer y deserializar el JSON.

8. Flujo del proyecto

El JSON se carga en memoria desde los controladores.

Los controllers deciden qué devolver:

MVC → HTML (View(...))

API → JSON (Ok(...))

Las vistas muestran los datos usando Razor.

El layout mantiene el estilo y la navegación.

Program.cs arranca la app y define la ruta inicial.

Los paquetes NuGet permiten leer JSON y actualizar vistas en caliente.