R&M - Naruto Characters API

Grupo formado por Sergi Martinez y Joan Merino

1. Introducción

Este dossier se enfoca en el análisis de las API de dos sagas: Rick and Morty y Naruto. Estas API proporcionan acceso a información y recursos relacionados con estas sagas, lo que nos ha permitido exponer aspectos relevantes de los caracteres. La motivación detrás de este proyecto radica en comprender cómo estas API funcionan y el poder implementar lo conocido en una página web. A lo largo del dossier, se examina la arquitectura Modelo-Vista-Controlador (MVC) utilizada en el desarrollo de aplicaciones que interactúan con estas API. Podrás saber detalles del código implementado y si se proponen mejoras y nuevas funcionalidades. Este proyecto nos ha servido para ser autodidactas e ir aprendiendo nuevos conceptos que no sabíamos, nos ha hecho madurar y saber que podemos lograr lo que nos proponemos.

2. Justificación del Proyecto

Desde el primer momento, supimos que haríamos una api sobre Rick & Morty, ya que no teníamos mucho conocimiento sobre las diferentes características de los personajes. Al ver que había una API disponible en el cual cumpliese todas las características del Proyecto, decidimos hacerlo.

Más tarde, implementamos una API de Naruto para intentar seguir aprendiendo e intentar sacar esos puntos adicionales que sabíamos que nos iría bien. Pero la API parece estar abandonada y no funciona correctamente.

Lo menos hablado pero desde fuera más bonito, ha sido la página, con Razor/cshtml y css. Ha sido una experiencia enriquecedora de nuevos conceptos en el cual hemos aprendido mucho y desempeñado durante muchas horas un acabado que nos gusta.

3. Esquema de la arquitectura

- C# MVC
- CSS
- Razor (cshtml)
- Nugget: Newtonsoft.Json

4. Explicación detallada del código desarrollado en relación con la arquitectura modelo-vista-controlador

En el HomeController tenemos:

La función GetApiResponse para hacer la llamada http async a las apis.

```
2 references
private string GetApiResponse(string url)
{
    HttpResponseMessage response = _httpClient.GetAsync(url).Result;
    return response.Content.ReadAsStringAsync().Result;
}
```

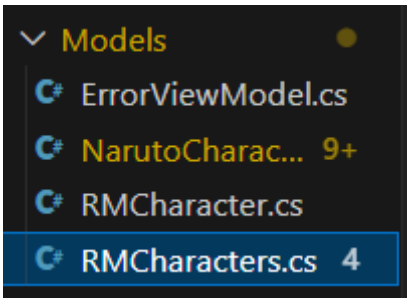
Llamada a la API de Rick y Morty (RMCharacters) inicializada en la página 1.
Llamamos a la api, deserializar de json a c# el objeto Characters. Función ViewBag para la paginación de los personajes en la página.

```
public IActionResult RMCharacters(int page = 1)
{
    string rickAndMortyApiUrl = $"https://rickandmortyapi.com/api/character?page={page}";
    string rickAndMortyApiResponse = GetApiResponse(rickAndMortyApiUrl);
    var charactersResponse = JsonConvert.DeserializeObject<Characters>(rickAndMortyApiResponse);

    ViewBag.CurrentPage = page;
    ViewBag.TotalPages = charactersResponse.info.pages;
    ViewBag.NextPage = charactersResponse.info.next != null ? page + 1 : (int?)null;
    ViewBag.PrevPage = charactersResponse.info.prev != null ? page - 1 : (int?)null;

    return View(charactersResponse.results);
}
```

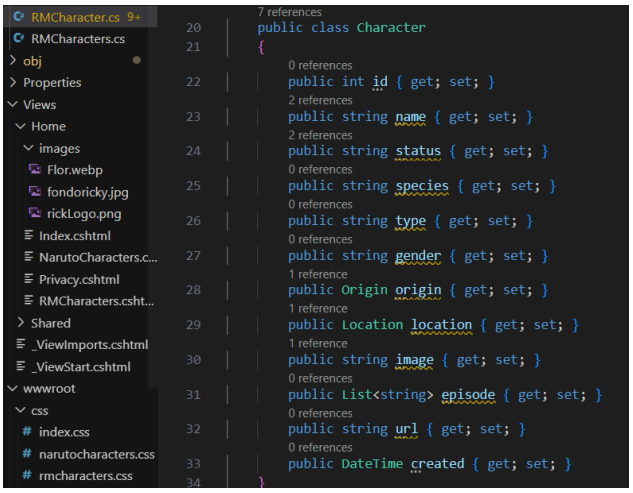
Llamada a la APi de Naruto (NarutoCharacters) inicializada en la página 1.
Llamamos a la api igual que la de Rick y Morty. ViewBag para la paginación de los personajes en la página.

```
public IActionResult NarutoCharacters(int page = 1)
{
    string narutoApiUrl = $"https://narutodb.xyz/api/character";
    string narutoApiResponse = GetApiResponse(narutoApiUrl);
    var narutoCharactersResponse = JsonConvert.DeserializeObject<Root>(narutoApiResponse);

    ViewBag.CurrentPage = page;
    ViewBag.TotalPages = (int)Math.Ceiling((double)narutoCharactersResponse.totalCharacters / narutoCharactersResponse.pageSize);
    ViewBag.NextPage = page < ViewBag.TotalPages ? page + 1 : (int?)null;
    ViewBag.PrevPage = page > 1 ? page - 1 : (int?)null;

    return View(narutoCharactersResponse);
}
```

Tenemos los modelos con los que trabajamos sacados de la api y convertido de json a c# (objetos)



Tenemos cada personaje con su id, name, status…



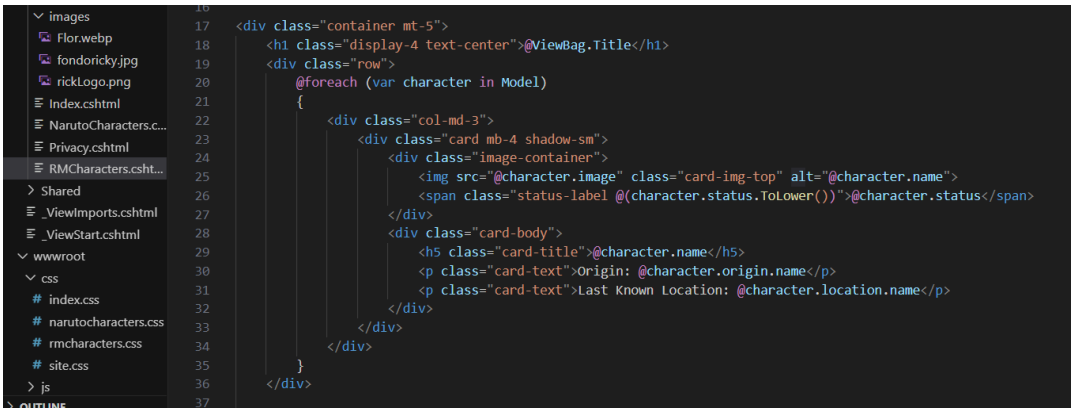
Con ello accedemos al modelo y mostramos por cada character en modelo y mostramos lo que queramos del objeto, en nuestro caso: la imagen, su status (alive, dead or unknown), el nombre del personaje, su origen y su última ubicación conocida.

Título de la página como "Rick & Morty Characters"
currentPagel y totalPages: Se obtienen de ViewBag con valores predeterminados de 1 si no están definidos.
Máximo de páginas a mostrar: Se fija en 8.
Cálculo de páginas inicial y final:
startPage: Calculada para asegurar que no sea menor que 1.
endPage: Calculada para asegurar que no sea mayor que el total de páginas.

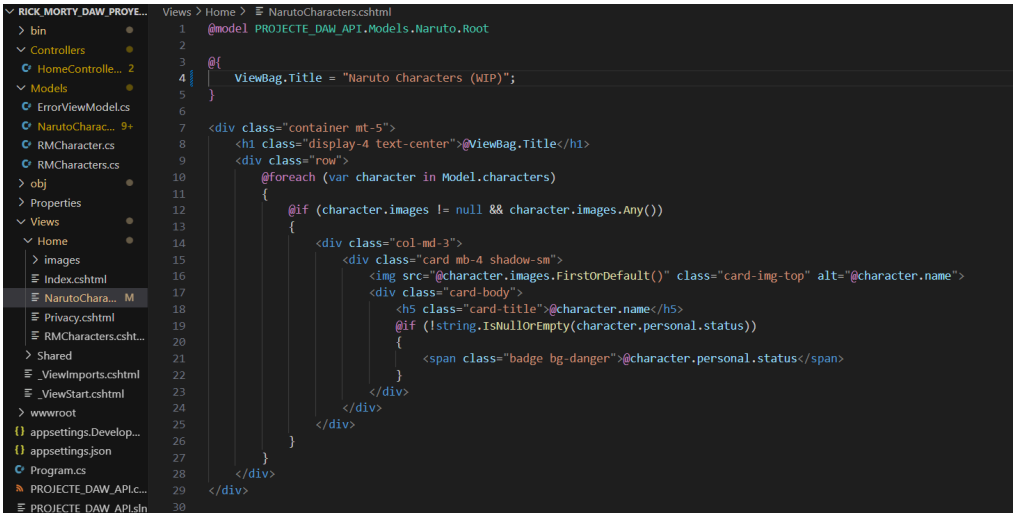
Ajuste del rango de páginas: Si el rango es menor que el máximo, se ajusta startPage para mostrar tantas páginas como sea posible dentro de los límites.

```
@model IEnumerable<PROJECTE_DAW_API.Models.Character>

@{
    ViewBag.Title = "Rick & Morty Characters";
    int currentPage = ViewBag.CurrentPage ?? 1;
    int totalPages = ViewBag.TotalPages ?? 1;
    int maxPagesToShow = 8;
    int startPage = Math.Max(1, currentPage - maxPagesToShow / 2);
    int endPage = Math.Min(totalPages, startPage + maxPagesToShow - 1);

    if (endPage - startPage < maxPagesToShow)
    {
        startPage = Math.Max(1, endPage - maxPagesToShow + 1);
    }
}
```

Contenedor y título: Configura el diseño y muestra el título centrado.
Tarjetas de personajes: Itera sobre el modelo para crear tarjetas con:
Imagen*
Estado
Nombre
Origen
Última ubicación conocida



Título: Configura y muestra el título "Naruto Characters (WIP)".
Tarjetas de personajes: Itera sobre los personajes del modelo y crea tarjetas con:
Imagen: Si el personaje tiene imágenes.
Nombre
Estado

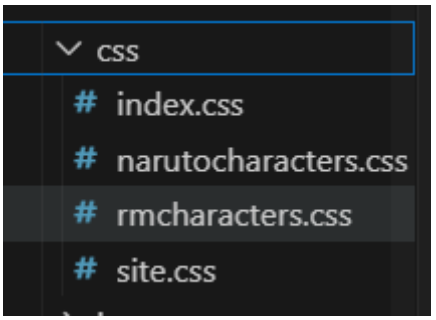


Contenedor de paginación: Usa clases de Bootstrap para centrar la paginación.
Botón "First": Deshabilitado si la página actual es la primera.
Botón "Previous": Deshabilitado si no hay una página anterior.
Números de página: Genera enlaces para cada página desde startPage hasta endPage.
Marca la página actual como activa.
Botón "Next": Deshabilitado si no hay una página siguiente.
Botón "Last": Deshabilitado si la página actual es la última.

```
<nav>
    <ul class="pagination justify-content-center">
        <li class="page-item @(currentPage == 1 ? "disabled" : "")">
            <a class="page-link" href="@Url.Action("RMCharacters", new { page = 1 })">First</a>
        </li>
        <li class="page-item @(ViewBag.PrevPage == null ? "disabled" : "")">
            <a class="page-link" href="@Url.Action("RMCharacters", new { page = ViewBag.PrevPage })">Previous</a>
        </li>
        @for (int i = startPage; i <= endPage; i++)
        {
            <li class="page-item @(i == currentPage ? "active" : "")">
                <a class="page-link" href="@Url.Action("RMCharacters", new { page = i })">@i</a>
            </li>
        }
        <li class="page-item @(ViewBag.NextPage == null ? "disabled" : "")">
            <a class="page-link" href="@Url.Action("RMCharacters", new { page = ViewBag.NextPage })">Next</a>
        </li>
        <li class="page-item @(currentPage == totalPages ? "disabled" : "")">
            <a class="page-link" href="@Url.Action("RMCharacters", new { page = totalPages })">Last</a>
        </li>
    </ul>
</nav>
```

Para poder acceder a los controladores y mostrar la página html en _Layout.cshtml lo especificamos.

Los estilos de cada página llamados en Layout.cshtml también



Para poder tener la pagina web estilada hemos usado css tambien, ademas de ir utilizando boostrap en el propio archivo de cshtml, el cual iba facilitando el trabajo.

En Index.css:

.rick-and-morty-title: Establece el color del texto en verde azulado.

.home-container: Define un contenedor flexible centrado, con padding, fondo con gradiente de color, texto blanco, alineación central y bordes redondeados.

.home-text: Configura un contenedor flexible en columna, centrado, con padding.

.home-text h1: Establece un encabezado grande con tipografía específica, margen inferior, color personalizado y sombra en el texto.

.home-text p: Define el estilo del párrafo con tamaño de fuente mediano, altura de línea cómoda y color blanco.
.home-image: Crea un contenedor flexible centrado para imágenes.

.home-image img: Estiliza la imagen para que tenga un ancho máximo del 100%, altura automática, bordes redondeados y sombra.

```css
.rick-and-morty-title {
    color: #3BB0B3;
  }

.home-container {
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 20px;
    background: linear-gradient(to right, #ffcc00, #ff6600);
    color: #fff;
    text-align: center;
    border-radius: 10px;
}

.home-text {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    padding: 20px;
}

.home-text h1 {
    font-size: 3.5rem;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    margin-bottom: 20px;
    color:  #a9f3fd;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
```

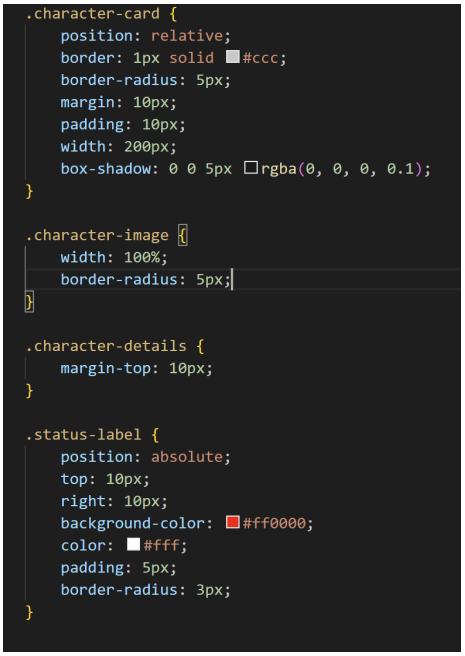
```css
.home-image {
    display: flex;
    justify-content: center;
    align-items: center;
}

.home-image img {
    max-width: 100%;
    height: auto;
    border-radius: 50%;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.3);
}
```

En narutocharacters:

.character-card: Define el estilo general de una tarjeta de personaje con borde, bordes redondeados, margen, relleno, ancho fijo y sombra suave.

.character-image: Estiliza la imagen del personaje para que ocupe todo el ancho disponible y tenga bordes redondeados.

.character-details: Ajusta el margen superior de los detalles del personaje.
.status-label: Crea una etiqueta de estado con posición absoluta en la esquina superior derecha, fondo rojo, texto blanco y bordes redondeados.



```css
.character-card {
    position: relative;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px;
    padding: 10px;
    width: 200px;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}

.character-image {
    width: 100%;
    border-radius: 5px;
}

.character-details {
    margin-top: 10px;
}

.status-label {
    position: absolute;
    top: 10px;
    right: 10px;
    background-color: #ff0000;
    color: #fff;
    padding: 5px;
    border-radius: 3px;
}
```

En rmcharacters:

.character-container: Define un contenedor flexible que permite envolver los elementos hijos con un espacio (gap) entre ellos.

.character-card: Estiliza las tarjetas de personaje con bordes, bordes redondeados, relleno, ancho fijo y posición relativa.

.display-4: Ajusta los márgenes superior e inferior de un elemento de visualización.

.image-container: Posiciona un contenedor relativo para imágenes.

.character-image: Estiliza la imagen del personaje para que ocupe todo el ancho disponible, ajuste la altura automáticamente y tenga bordes redondeados.

.status-label: Define una etiqueta de estado con posición absoluta en la esquina superior derecha, relleno, bordes redondeados, texto blanco, negrita y tamaño de fuente pequeño. Aplica diferentes colores de fondo según el estado.

.character-details: Añade relleno superior para separar los detalles del personaje del resto del contenido.

.character-details h2: Estiliza los encabezados de los detalles del personaje con márgenes ajustados y un tamaño de fuente específico.

.character-details p: Estiliza los párrafos dentro de los detalles del personaje con márgenes y tamaño de fuente específicos.
.pagination: Estiliza la sección de paginación con un margen superior, centramiento y espacio entre botones de paginación.

.pagination .btn: Ajusta el margen alrededor de los botones de paginación para un espaciado uniforme.

```css
.character-container {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
}

.character-card {
    border: 1px solid #c
    border-radius: 5px;
    padding: 10px;
    width: 200px;
    position: relative;
}

.display-4 {
margin-bottom: 55px;
margin-top: -25px;
}

.image-container {
    position: relative;
}

.character-image {
    width: 100%;
    height: auto;
    border-radius: 5px;
}
```

```css
.status-label {
    position: absolute;
    top: 10px;
    right: 10px;
    padding: 2px 5px;
    border-radius: 3px;
    color: #fff;
    font-weight: bold;
    font-size: 0.8em;
}
.status-label.alive {
    background-color: green;
}

.status-label.dead {
    background-color: red;
}

.status-label.unknown {
    background-color: gray;
}

.character-details {
    padding-top: 10px;
}

.character-details h2 {
    margin-top: 0;
    margin-bottom: 5px;
    font-size: 18px;
}

.character-details p {
    margin: 0;
    font-size: 14px;
}
```

```css
.pagination {
    margin-top: 20px;
    display: flex;
    justify-content: center;
    gap: 5px;
}

.pagination .btn {
    margin: 0 5px;
}
```

En site.css:

body: Configura el cuerpo de la página como un contenedor flexible en columna, con altura mínima del 100% del viewport, fondo claro y tipografía específica.

.navbar-brand: Estiliza la marca de la barra de navegación con márgenes y color dorado.

.navbar: Aplica un fondo con gradiente de blanco a azul y una animación de deslizamiento al aparecer.

.card-body: Establece un fondo con gradiente de blanco a verde claro para el cuerpo de las tarjetas.

.card: Incluye una animación de desvanecimiento al cargar y un efecto de aumento y sombra suave al pasar el mouse.

@keyframes fadeIn: Define la animación de desvanecimiento gradual.

@keyframes slideIn: Define la animación de deslizamiento desde la izquierda.

.footer: Aplica un fondo con gradiente de blanco a naranja y texto en blanco.

.logo-img: Estiliza la imagen del logo con tamaño fijo y margen derecho.

```css
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  background-color: #f8f9fa;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.navbar-brand {
  margin-left: 30px;
  padding-top: 10px;
  color: #C2B280;
}

.navbar {
  background: linear-gradient(to right, #FFFFFF, #66CCFF);
}

.card-body {
  background: linear-gradient(to bottom right, #fff, #dfe7cf);
}

.card:hover {
  transform: scale(1.02);
  box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.2);
}

.card {
  animation: fadeIn 1s ease-in-out;
}

@keyframes fadeIn {
  from { opacity: 0; }
  to { opacity: 1; }
}
```

```css
.navbar {
  animation: slideIn 0.5s ease-in-out;
}

@keyframes slideIn {
  from { transform: translateX(-100%); }
  to { transform: translateX(0); }
}

.footer {
  background: linear-gradient(to left, #FFFFFF, #FFA500);
  color: rgb(255, 255, 255);
}

.logo-img {
  margin-right: 10px;
  height: 40px;
  width: 40px;
}
```

LO QUE MÁS NOS HA COSTADO HA SIDO SER ORGANIZADOS CON EL CÓDIGO, YA QUE AL IR CAMBIANDO LAS CLASES CONTINUAMENTE, SE DEJAN ATRÁS ALGUNAS QUE REALMENTE NO TIENEN FUNCIÓN, EN NUESTRO CASO ESPERAMOS QUE NO HAYAN MUCHAS.

5. Propuestas de mejora y nuevas funcionalidades

Para mejorar y expandir las funcionalidades del proyecto:

Filtros Avanzados: Implementar filtros para buscar personajes por nombre, estado, origen y última ubicación conocida.
Búsqueda en Tiempo Real: Añadir una barra de búsqueda que permita buscar personajes en tiempo real mientras se escribe.
Paginación Mejorada: Optimizar la paginación para mostrar un mayor número de páginas o permitir la navegación directa a una página específica.
Mejora de UI/UX: Refinar el diseño de la interfaz de usuario para hacerla más intuitiva y visualmente atractiva.
Cacheo de Datos: Implementar un sistema de cacheo para mejorar la velocidad de carga y reducir las llamadas a la API.
Soporte para Nuevas APIs: Explorar e integrar APIs de otras sagas populares para expandir el alcance del proyecto.
Estas mejoras no solo enriquecerán la funcionalidad y usabilidad del proyecto, sino que también ofrecerán una mejor experiencia al usuario final, haciendo que la exploración de los personajes sea más agradable y eficiente.

6. Conclusiones

El desarrollo del proyecto nos ha permitido explorar y entender cómo funcionan las API de personajes de Rick & Morty y Naruto, así como implementar estas APIs en una página web utilizando la arquitectura Modelo-Vista-Controlador (MVC).

Hemos aprendido a:
Integración de API: Realizar llamadas HTTP asíncronas y deserializar respuestas JSON para trabajar con datos en C#.
Paginación: Implementar un sistema de paginación eficiente para manejar y mostrar grandes conjuntos de datos de personajes.
Desarrollo Frontend: Utilizar Razor y Bootstrap para crear una interfaz de usuario atractiva y funcional.
Manejo de Datos: Organizar y presentar datos de personajes de manera clara y accesible, mostrando imágenes, nombres, estados, orígenes y ubicaciones.
Este proyecto ha sido una experiencia enriquecedora, ayudándonos a adquirir nuevos conocimientos y habilidades, y demostrando nuestra capacidad para abordar y resolver problemas complejos. La experiencia nos ha permitido madurar en el ámbito del desarrollo web y nos ha motivado a seguir aprendiendo y mejorando.

7. Referencias

Curso MVC:
https://www.youtube.com/watch?v=9JP3PSUb5cY&list=PL9Bm8IOGYHA3X6LjUEQAw9EGxTq5EAees
ChatGPT: https://chatgpt.com/?oai-dm=1
Curso animaciones: https://www.youtube.com/watch?v=RwjgfNX41TE&t=3733s
Paginas Informacion Razor:
https://learn.microsoft.com/es-es/aspnet/core/mvc/views/razor?view=aspnetcore-8.0