

The background of the entire page is a high-resolution image of a satellite in orbit. The satellite is a complex structure with a central body and several large, rectangular solar panel arrays extending outwards. It is positioned in the lower half of the frame, appearing to move across the Earth's surface. The Earth is shown as a large, curved sphere with a deep blue ocean and white, swirling cloud patterns. The lighting suggests a bright sun, creating a slight glow around the satellite and the Earth's horizon.

PROYECTO MP4-5

CLOUDS OVER EUROPE

Natalia Soria
Natalia García



INDEX

INTRODUCCIÓN.....	pag.2
DISEÑO.....	pag.3
DESARROLLO APLICACIÓN.....	pag.5
CONTROLLERS.....	pag.5
HOMECONTROLLER.CS.....	pag.5
VIEWS/SHARED.....	pag.10
_LAYOUT.CSHTML.....	pag.10
VIEWS/HOME.....	pag.11
INDEX.CSHTML.....	pag.11
WEATHERINFORMATION.CSHTML.....	pag.12
AQIAIRINFORMATION.CSHTML.....	pag.16
SUGERENCIAS DE MEJORAS.....	pag.21
FUENTES.....	pag.22



INTRODUCCIÓN

Para este proyecto decidimos crear una web sobre el tiempo, donde pudieras seleccionar una capital europea y te mostrara el tiempo actual en esa ciudad. Para ello decidimos utilizar dos api, una desde la que extraeríamos la latitud y la longitud de la ciudad elegida y una segunda api desde la que recibiríamos la información sobre el clima y temperatura en esa localidad. De esta forma un usuario podría elegir un país de los que se mostrara en pantalla y así saber en tiempo real su temperatura, precipitaciones, hora de amanecer, entre otras características.

Para la llamada de la primera api decidimos usar la api GeoNames, que era una api que ya conocíamos y habíamos usado para una práctica de clase. Para encontrar una api sobre el tiempo que no fuera de pago y tuviera una estructura sencilla tuvimos más problemas. Estuvimos probando varias: Meteosource, meteoblue, TuTiempo.net, Open-Meteo, Vissualcrossing, OpenWeather e incluso Api.Nasa. Pero la mayoría de ellas de forma gratuita tenían muy pocas llamas o tenían una estructura en su respuesta JSON bastante complicada y no nos aclarábamos cuando teníamos que pasar la información a al html correspondiente. Por eso finalmente escogimos Weatherbit Api, que nos ofrecía un mes de prueba gratuito con 1500 llamadas diarias. Además, su respuesta JSON era más corta, pero tenía toda la información que necesitábamos.

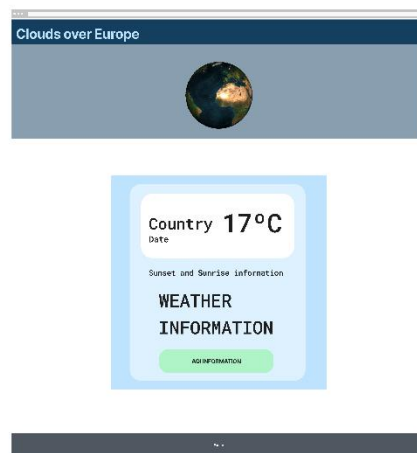
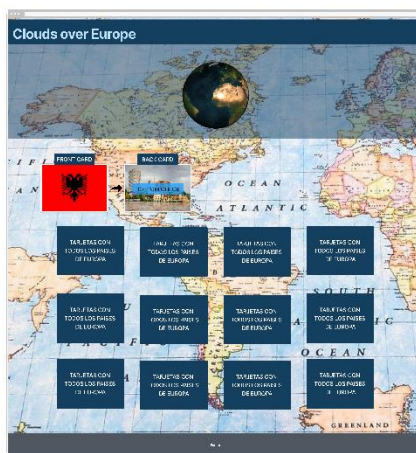


DISEÑO

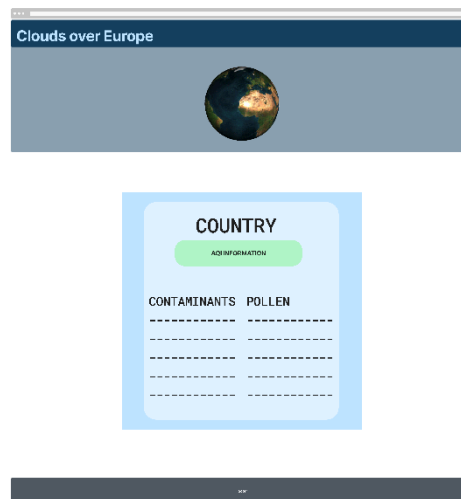
Una vez decidida la temática del proyecto, comenzamos a planear el diseño de nuestra web, como queríamos que se viera.

Planteamos una primera página donde tendríamos tarjetas giratorias con imágenes de las banderas de los países de Europa delante y detrás una imagen de la ciudad correspondiente. Al seleccionar una de las tarjetas, esta redirige a la página con la información sobre el tiempo actual en esa ciudad.

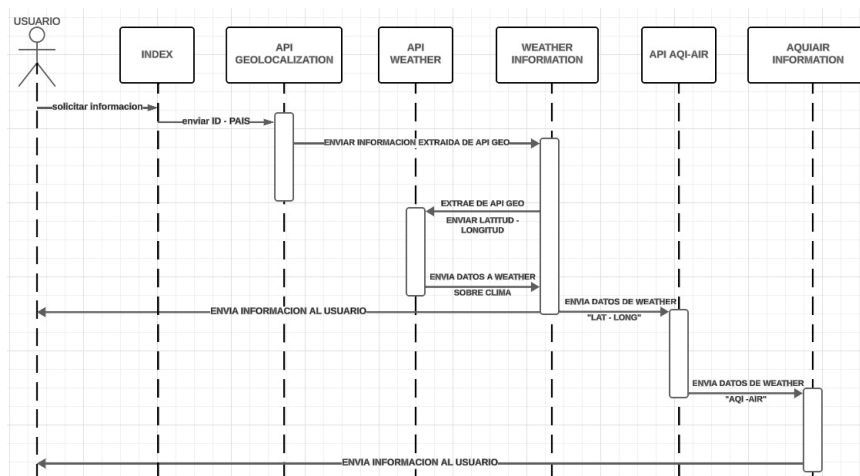
En la segunda página, a parte de los datos de clima, queríamos que, en base a sus estadísticas, cambiara el fondo dependiendo de los porcentajes en las precipitaciones, nubes, etc.



En un inicio iban a ser únicamente estas dos páginas, pero nos parecía poco contenido. Por ese motivo revisamos que más peticiones de información le podíamos hacer a la api de Weatherbit. Tenía bastantes llamadas, desde alertas temporales, mapas climatológicos, históricos por hora del clima de una localidad... Pero el que más encajaba con nuestro proyecto era completar la información sobre la calidad del aire. Por eso diseñamos una última página que mostraría la información sobre la cantidad de contaminantes que tiene el aire de esa ciudad y la cantidad de los diferentes tipos de polen.



También diseñamos un diagrama de secuencia para saber qué información debía de recibir y pasar cada bloque de nuestro programa:



Con el diseño de las páginas claro, empezamos a desarrollar nuestra aplicación web.

DESARROLLO APLICACIÓN

CONTROLLERS

HOMECONTROLLER.CS

Para iniciar la programación de nuestro proyecto, el primer paso es crear los archivos con el código C# que dan como respuesta las dos Apis que utilizaremos. Para ello hacemos una búsqueda con respuesta en formato JSON desde nuestro navegador con ambas apis y la respuesta dada la convertiremos a lenguaje C# con la página [Json2csharp.com](https://json2csharp.com).

Respuesta en formato JSON que daría una api:

```
1 {
2   "timezone": {
3     "gmtOffset": 1,
4     "timeZoneId": "Europe/Copenhagen",
5     "dstOffset": 2
6   },
7   "bbox": {
8     "east": 12.652060528479984,
9     "south": 55.61284310536006,
10    "north": 55.732711528388705,
11    "west": 12.453904564650229,
12    "accuracyLevel": 0
13  },
14  "asciiName": "Copenhagen",
15  "astergdem": 20,
16  "countryId": "2623032",
17  "fcl": "P",
18  "srtn3": 14,
19  "adminId2": "2618424",
20  "countryCode": "DK",
21  "adminCodes1": {
22    "ISO3166_2": "84"
23  },
24  "adminId1": "6418538",
25  "lat": "55.67594",
26  "fcode": "PPLC",
27  "continentCode": "EU",
28  "adminCode2": "101",
29  "adminCode1": "17",
30  "lng": "12.56553",
31  "geonameId": 2618425,
32  "toponymName": "Copenhagen",
33  "population": 1153615,
34  "wikipediaURL": "en.wikipedia.org/wiki/Copenhagen",
35  "adminName5": "",
36  "adminName4": "",
37  "adminName3": "",
38  "alternateNames": [
```

Conversión a C#

```
375 {
376   {
377     "name": "Copenhaga",
378     "lang": "th"
379   },
380   {
381     "name": "Copenhaga",
382     "lang": "bo"
383   },
384   {
385     "name": "コペンハーゲン",
386     "lang": "ja"
387   },
388   {
389     "name": "哥本哈根",
390     "lang": "zh"
391   }
392 },
393 "adminName2": "Copenhagen",
394 "name": "Copenhagen",
395 "fclName": "City, village,...",
396 "countryName": "Denmark",
397 "fcodeName": "capital of a political entity",
398 "adminName1": "Capital Region"
399 }
400
401 // Root myDeserializedClass = JsonConvert.DeserializeObject<Root>(json);
402 public class AdminCodes1
403 {
404   public string ISO3166_2 { get; set; }
405 }
406
407 public class AlternateName
408 {
409   public string name { get; set; }
410   public string lang { get; set; }
411   public bool? isShortName { get; set; }
412   public bool? isPreferredName { get; set; }
413 }
414
415 public class Bbox
416 {
417   public double east { get; set; }
418   public double south { get; set; }
419   public double north { get; set; }
420   public double west { get; set; }
421   public int accuracyLevel { get; set; }
422 }
423
424 }
```

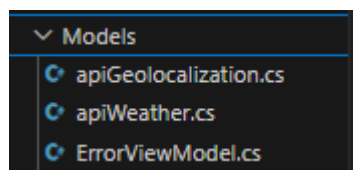



Así obtendremos un código cómo el siguiente:

```
public class ApiAqi
{
    1 reference
    public class Datum
    {
        0 references
        public int aqi { get; set; }
        0 references
        public double o3 { get; set; }
        0 references
        public double so2 { get; set; }
        0 references
        public double no2 { get; set; }
        0 references
        public double co { get; set; }
        0 references
        public int pm10 { get; set; }
        0 references
        public int pm25 { get; set; }
        0 references
        public int pollen_level_tree { get; set; }
        0 references
        public int pollen_level_grass { get; set; }
        0 references
        public int pollen_level_weed { get; set; }
        0 references
        public int mold_level { get; set; }
        0 references
        public string predominant_pollen_type { get; set; }
    }

    0 references
    public class Aqi
    {
        0 references
        public double lat { get; set; }
        0 references
        public double lon { get; set; }
        0 references
        public string timezone { get; set; }
        0 references
        public string city_name { get; set; }
        0 references
        public string country_code { get; set; }
        0 references
        public string state_code { get; set; }
        0 references
        public List<Datum> data { get; set; }
    }
}
```

Este código se debe guardar dentro de la carpeta Models, en formato .cs. Y se deberá de tener tantos documentos dentro de esta carpeta como llamadas a apis hagamos. Qué, para nuestra aplicación en un principio serán dos:



Nuestra página de inicio o Índice debía de volver un conjunto de flip-cards, tantas como países europeos hay, con una imagen de portada donde habría el nombre del país y de su capital, y una imagen de la ciudad de contraportada.

En la contraportada encontraríamos también el botón que nos redirigiría a la siguiente página, enviando la información del id del país seleccionado para poder hacer la correspondiente llamada a api.

Intentamos encontrar una Api que nos aportara toda la información necesaria, pero no tuvimos éxito en nuestra búsqueda. Por este motivo decidimos crear nuestra lista de países, donde cada país sería un dato tipo struct que almacenaría el nombre del país, de la capital, el id y las dos imágenes que necesitábamos para las flip-cards.



Creamos su estructura:

Y creamos una función donde creamos todos los registros necesarios:

```
54 references
public struct CountriesID{
    50 references
    public string countryName;
    50 references
    public string capitalCountry;

    50 references
    public string urlImagen;
    50 references
    public string urlCityImage;
    50 references
    public int id;
}
```

```
static List <CountriesID> CreateCountryList(){
    List <CountriesID> countriesIDs= new List <CountriesID> ()
    {
        new CountriesID {countryName="Albania",capitalCountry="Tirana",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Germany",capitalCountry="Berlin",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Andorra",capitalCountry="Andorra la vieja",urlImagen="Content/Images/fl
        new CountriesID {countryName="Austria",capitalCountry="Vienna",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Azerbaiyan",capitalCountry="Baku",urlImagen="Content/Images/flags/bande
        new CountriesID {countryName="Belgium",capitalCountry="Brussels ",urlImagen="Content/Images/flags/bar
        new CountriesID {countryName="Belarus",capitalCountry="Minsk ",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Bosnia and Herzegovina",capitalCountry="Sarajevo",urlImagen="Content/Im
        new CountriesID {countryName="Bulgaria",capitalCountry="Sofia",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Croatia",capitalCountry="Zagreb",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Cyprus",capitalCountry="Nicosia",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Chezchia",capitalCountry="Prague",urlImagen="Content/Images/flags/bande
        new CountriesID {countryName="Denmark",capitalCountry="Copenhagen",urlImagen="Content/Images/flags/ba
        new CountriesID {countryName="Estonia",capitalCountry="Tallinn",urlImagen="Content/Images/flags/bande
        new CountriesID {countryName="Finland",capitalCountry="Helsinki",urlImagen="Content/Images/flags/band
        new CountriesID {countryName="France",capitalCountry="Paris",urlImagen="Content/Images/flags/banderaf
        new CountriesID {countryName="Georgia",capitalCountry="Tbilisi",urlImagen="Content/Images/flags/bande
        new CountriesID {countryName="Greece",capitalCountry="Athens",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Hungary",capitalCountry="Budapest",urlImagen="Content/Images/flags/band
        new CountriesID {countryName="Iceland",capitalCountry="Reykjavik",urlImagen="Content/Images/flags/ban
        new CountriesID {countryName="Ireland",capitalCountry="Dublin",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Italy",capitalCountry="Rome",urlImagen="Content/Images/flags/banderIta
        new CountriesID {countryName="Kazakhstan",capitalCountry="Astana",urlImagen="Content/Images/flags/ban
        new CountriesID {countryName="Kosovo",capitalCountry="Pristina",urlImagen="Content/Images/flags/bande
        new CountriesID {countryName="Latvia",capitalCountry="Riga",urlImagen="Content/Images/flags/banderala
        new CountriesID {countryName="Liechtenstein",capitalCountry="Vaduz",urlImagen="Content/Images/flags/b
        new CountriesID {countryName="Lithuania",capitalCountry="Vilnius",urlImagen="Content/Images/flags/ban
        new CountriesID {countryName="Luxembourg",capitalCountry="Luxembourg(City)",urlImagen="Content/Images
        new CountriesID {countryName="Malta",capitalCountry="Valletta",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Moldava",capitalCountry="Chisinau",urlImagen="Content/Images/flags/band
        new CountriesID {countryName="Monaco",capitalCountry="Monaco",urlImagen="Content/Images/flags/bander
        new CountriesID {countryName="Montenegro",capitalCountry="Podgorica",urlImagen="Content/Images/flags/
```

Así podíamos crear nuestra primera IActionResult Index() que enviaría la lista de los países a la VIEW para que la página pudiera representar toda la información:

```
0 references
public IActionResult Index()
{

    List<CountriesID> countriesIDs=CreateCountryList();

    return View(countriesIDs);
}
```




Con la página de inicio ya definida comenzamos a trabajar sobre la segunda. Esta segunda función necesitará utilizar el id del país seleccionado para poder pasárselo a la url del llamado de la API, por eso es importante indicarla junto con el tipo de dato que será cuando inicializamos la función. Con esa url haremos la llamada y lo transformaremos en una lista de objetos del tipo de su estructura (que es la que hemos definido previamente en su archivo del Model:


```
public IActionResult WeatherInformation(string id){  
  
    var urlApiCountry = $"http://api.geonames.org/getJSON?geonameId={id}&username=nagasa";  
    var clientCountry = new HttpClient();  
    var responseCountry = clientCountry.GetAsync(urlApiCountry).Result;  
    var contentCountry = responseCountry.Content.ReadAsStringAsync().Result;  
    var informationCountry = JsonSerializer.Deserialize<ApiCountry.Country>(contentCountry);  
  
    var countryList = new List<ApiCountry.Country>{informationCountry};  
}
```

Esta primera llamada nos dará la información acerca de donde está ubicado esta ciudad, en concreto su longitud y latitud, que es la información que necesitaremos pasarle a la siguiente llamada del api.

```
var urlApiWeather = $"https://api.weatherbit.io/v2.0/current?lat={countryList[0].lat}&lon={countryList[0].lng}&key=b594f4848b42479cb1d61d4283ff8793&include=minutely";  
var clientWeather = new HttpClient();  
var responseWeather = clientWeather.GetAsync(urlApiWeather).Result;  
var contentWeather = responseWeather.Content.ReadAsStringAsync().Result;  
  
var informationWeather = JsonSerializer.Deserialize<ApiWeather.Root>(contentWeather);  
  
var listInformationWeather = new List<ApiWeather.Root>{informationWeather};
```

Volvemos a transformar la información recibida de la api en una lista del tipo Apiweather.

Esta api nos devolverá toda la información acerca del clima actual en ese país y es la que le pasaremos al VIEW.



Pero en este caso es importante chequear que la api nos da respuesta para que no se pare el programa.

```
if (listInformationWeather != null)
{
    return View(listInformationWeather);
}
else
{
    return View("no hay datos");
}
```

En la función anterior no era necesario ya que todas las id de los países las habíamos puesto nosotras de forma manual y nos aseguramos que todas daban llamadas válidas.

Llegado a este punto ya teníamos todas las funciones de nuestro HomeController creadas, pero nos parecía que el trabajo se había quedado un poco corto. Por eso decidimos hacer una última llamada para que el usuario, además de ver la información sobre el tiempo, pudiera revisar que contaminantes y polen había en el aire.

Añadimos un nuevo archivo a nuestra carpeta Models, con la información de la estructura que da nuestra nueva llamada a api. Y en este caso definimos qué a esta api le pasaremos la longitud y latitud que obtenemos de las llamada anterior. Creamos el objeto y volvemos a confirmar que la respuesta que nos da es válida (no nula) antes de pasarlo a la VIEW.

```
U references
public IActionResult AqiAirInformation(string longitude, string latitude){

    var urlApiAqi=$"https://api.weatherbit.io/v2.0/current/airquality?lat={latitude}&lon={longitude}&key=b594f4848b42479cb1d61d4283ff8793";

    var clientAqi= new HttpClient();
    var responseAqi=clientAqi.GetAsync(urlApiAqi).Result;
    var contentAqi=responseAqi.Content.ReadAsStringAsync().Result;

    var informationAqi = JsonSerializer.Deserialize<ApiAqi.Aqi>(contentAqi);

    var listInformationAqi = new List<ApiAqi.Aqi>(informationAqi);

    if (listInformationAqi != null)
    {
        return View(listInformationAqi);
    }
    else
    {
        return View("no hay datos");
    }
}
```

Con esta última IActionResult ya tendríamos nuestro HomeController listo y podemos pasar a revisar como estarían programados todos nuestros archivos .cshtml, que tenemos almacenado dentro de la carpeta VIEW.

VIEWS/SHARED

LAYOUT.CSHTML

Dentro de la carpeta views/shared encontramos los archivos. cshtml y .cshtml.css que estilarián todos los ítems comunes de las diferentes páginas que tiene nuestra aplicación, como podría ser el navbar o el footer. Aquí dejamos programado y estilados estos dos ítems y un div con un logo que escogimos, para que todas las páginas tuvieran el mismo estilo.

```
<body>
<header>
  <nav class="navbar">
    <div class="container-fluid">
      <a class="navbar-brand" asp-controller="Home" asp-action="Index">
        <p style="font-size:40px; font-weight:bold; webkit-text-stroke-width: 1px; color: #157, 216, 233);">
          
          Clouds over Europe
        </p>
      </a>
    </div>
  </nav>
  <div class="carousel slide" data-ride="carousel">
    <p></p>
  </div>
</header>
<div class="container">
  <div class="main" role="main" class="pb-3">
    @RenderBody()
  </div>
</div>

<footer class="border-top footer text-muted">
  &copy; <a style="color: #157, 216, 233); font-weight:bold;">2024 - NAT'S PROYECT </a> <a asp-area="" asp-controller="Home" asp-action="Privacy" style="color: #157, 216, 233);"/>Privacy</a>
</footer>
<script src="/lib/jquery/dist/jquery.min.js"></script>
<script src="/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
```

```
<style>
.navbar{
  margin-bottom: 0;
  background: #157, 216, 233);
  font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}
.carousel{
  margin-top: -2%;
  z-index:1000;
  background: linear-gradient(to bottom, #157, 216, 233), #0, 0, 255, 0););
  margin: 0;
  backdrop-filter: blur(3px);
  text-align: center;
  align-items: center;
  z-index:1000;
  height: fit-content;
}
.footer {
  position: relative bottom;
  bottom: 0;
  width: 100%;
  height: 20px;
  background-color: #033256;
  color: #157, 216, 233);
  text-align: center;
  z-index:1000;
  height: fit-content;
}
</style>
```

Intentamos dejar estilado desde el archivo .css externo pero no pudimos, por eso dejamos todos lo estilos definidos en el mismo archivo .cshtml, dentro del head en la etiqueta style.

VIEWS/HOME

INDEX.CSHTML

Esta sería nuestra página principal, llamamos a la lista de países que pasamos en la VIEW y dentro de ella seleccionamos a las variables de “urlImagen”, “CountryName”, “capitalCountry” y “urlCityImage” de cada componente de la lista para pasárselo a las flip-cards.

```
<div class="flip-card-front" style="background-image: url('@country.urlImagen'); background-size:cover;">
  <h1>@country.countryName</h1>
  <p class="countryInformation">@country.capitalCountry</p>
</div>
<div class="flip-card-back " style="background-image: url('@country.urlCityImage'); background-size:cover;">
  <a href="@Url.Action("WeatherInformation", "Home", new { id = country.id , background=country.urlCityImage})"
  class="button">Consultar</a>
</div>
```

Declaramos un foreach al inicio para iterar sobre cada uno y que nos muestre los datos que pedimos.

```
<body>
  <main class="container">
    @foreach (var country in Model)
    {
      <div class="row">
        <div class="flip-card">
          <div class="flip-card-inner">
            <div class="flip-card-front">
```

Utilizamos las “flip-card” para que se vean cada una de las banderas por el frente y las ciudades por detrás. Las fuimos estilando de tal manera que queden posicionadas de cuatro en cuatro y fuimos añadiendo estilos de letras, medidas, colores (botón), entre otros.

```
<div class="flip-card-back " style="background-image: url('@country.urlCityImage'); background-size:cover;">
  <a href="@Url.Action("WeatherInformation", "Home", new { id = country.id})" class="button">Consultar</a>
</div>
```

En el botón es importante dejar declarada la variable id que deberá de pasar a la siguiente página, que será la asociada a la IActionResult WeatherInformation, para que el programa pueda obtener el valor dentro de country.id y pasarlo a la url de la llamada de la api que se definió en el HomeController.

WEATHERINFORMATION.CSHTML

Comenzamos nuestra documento .cshtml de weatherInformation de la misma forma que el anterior archivo. Pero en esta página queremos crear un container que contenga el nombre del país, la fecha y la temperatura, y que el fondo de este cambie en función del clima que haga. Por eso declaramos una variable string que guardará el path donde se encuentra la imagen que queremos aplicar de fondo. Y este path variará dependiendo de las condiciones climatológicas que haya. Para este punto usamos ayuda de chatgpt.

```
string backgroundCurrentCity = datum.precip > 20 ? "~/Content/Images/iconWeather/day/rain.jpg" :  
    datum.snow > 0 ? "~/Content/Images/iconWeather/day/snow.jpg" :  
    datum.clouds >= 50 ? "~/Content/Images/iconWeather/day/cloud.jpg" :  
    "~/Content/Images/iconWeather/day/sunny.jpg";
```

Esta misma variable es la que le pasaremos de estilo background al container currentCityBox, así conseguimos que el fondo fuera en consonancia con los datos del clima. Y le pasamos las variables de los datos que queríamos que nos mostrara.

```
<div class="container">  
  <div class="currentCityBox" style="background-image: url('@Url.Content(backgroundCurrentCity)')">  
    <div class="leftSpan">  
      <h1 style="font-weight: bolder; font-size:40px;">@datum.city_name</h1>  
      <h3 style="font-weight: bolder; font-size:30px;">@datum.ob_time</h3>  
    </div>  
    <div class="rightSpan">  
      <p class="temperature">@datum.temp °C</p>  
    </div>  
  </div>  
</div>  
<div class="currentInformation" style="margin-left:15%;">
```

Dividimos en dos secciones el resto de los datos que queríamos mostrar en esta página, para que se vieran en formato de dos columnas.




La primera sección es la de “Sunrise time” con sus respectivos llamados a los datos de model y fuimos acomodando las medidas según se iba requiriendo.

```
<div class="currentInformation" style="margin-left:15%;">
  <span>
    <div style="margin-bottom:5%">
      <p class="information">
        
        SUNRISE TIME: </p>
      <p class="data">@datum.sunrise H</p>
    </div>
    <div style="margin-left:5%">
      <div><p class="information">CLOUD COVERAGE: </p><p class="data">@datum.clouds %</p></div>
      <div><p class="information">RAIN PRECIPITATION: </p><p class="data">@datum.precip mm/hr</p></div>
      <div><p class="information">SNOW PRECIPITATION: </p><p class="data">@datum.snow mm/hr</p></div>
    </div>
  </span>
</div>
```

La segunda sección es “Sunset time”. Se encuentra estructurada como la anterior, con la diferencia de que está posicionada a la derecha.

```
<span>
  <div style="margin-bottom:5%">
    <p class="information">
      
      SUNSET TIME: </p>
    <p class="data">@datum.sunset H</p>
  </div>
  <div style="margin-left:5%">
    <div><p class="information">UV INDEX: </p><p class="data">@datum.uv </p></div>
    <div><p class="information">SPEED WIND: </p><p class="data">@datum.wind_spd m/s</p> </div>
    <div><p class="information">HUMITY: </p><p class="data">@datum.rh %</p></div>
  </div>
</span>
iv>
```

Al final del container creamos un apartado que muestra la cantidad de contaminación en el aire y además nos dirige a otra página donde se muestran los datos del estado del aire. “Quality Air <datum.aqui> ICA”.



Realizamos condiciones con if para que, en caso de ser muy alto, sea considerado malo y en caso de ser bajo sea muy bueno.

Por ejemplo, en el caso de tener + de 100 sería considerado malo.

```
@if (datum.aqi>100)
{
```

Dentro del if, declaramos una clase, y un cambio de fondo de alrededor de “Quality Air” para dicha condición.

```
<div class="qualityAir" style="background-image: url('@Url.Content("~/Content/Images/iconWeather/qualityAirBAD.png")');"
```

A su vez declaramos dos clases, lo estilamos y hacemos referencia a la url a la que se navegara cuando se haga click en el enlace. Realizamos una llamada a la función especificada y desde ahí se genera la url a la página “AqiAirInformation”.

También se pasan los valores de latitud y longitud y se los asigna a dos variables.

```
<a class="qualityText stretch-out" style="margin-left:0.5% z-index:1;" href="@Url.Action("AqiAirInformation", "Home",
new { longitude = datum.lon , latitude=datum.lat})" > QUALITY AIR : @datum.aqi ICA</a>
```

Por último, dependiendo de si es malo o bueno, se agrega un icono diferente a la derecha del texto.

```
new { longitude = datum.lon , latitude=datum.lat})" > QUALITY AIR : @datum.aqi ICA</a>

iv>
```

Realizamos la misma acción en los demás elif. Para cada condición cambiara la imagen, la cantidad y el icono.



Para que el usuario pudiera ver claramente que el bloque “Quality Air” es un botón con el que podría informarse más acerca de los contaminantes del aire, pusimos el texto: “Click here for more information about the air pollution” adornado flechas.

```
<div style="align-items: center;">
  <p style="text-align: center;font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;color:#085409;font-weight: bolder;">
    
    Click here for more information about the air pollution
    
  </p>
</div>
```

Para esta página también queríamos que, además de cambiar el fondo donde se mostraba la temperatura, también el fondo de la página se adaptara dependiendo el clima de cada país (soleado, nublado, lluvia, nieve).

Realizamos una condición para cada caso y lo aplicamos dentro del “body”.

```
</div>
@if (datum.precip >=20)
{
  <body style="background-image: url('@Url.Content("~/Content/Images/lluvia.gif")')"></body>
}
else if (datum.snow > 0)
{
  <body style="background-image: url('@Url.Content("~/Content/Images/nieve.gif")')"></body>
}
else if (datum.clouds >= 50)
{
  <body style="background-image: url('@Url.Content("~/Content/Images/nublado.jpeg")')"></body>
}
else
{
  <body style="background-image: url('@Url.Content("~/Content/Images/soleado.gif")')"></body>
}
```

AQIAIRINFORMATION.CSHTML

Y por último realizamos el archivo para visualizar desde el navegador la información devuelta en la vista del IActionResult AqiAirInformation().

Antes de empezar a programar nuestro body debemos de asegurarnos que el archivo reciba información válida por parte del VIEW. Por eso primero chequeamos que el modelo pasado no sea nulo y no este vacío. Una vez asegurado que estamos trabajando con un dato válido revisaremos la información que hay dentro de la estructura para pasársela al html:

```
@if (Model != null && Model.Count > 0)
{
    var aqiModel = Model[0];

    if (aqiModel.data != null && aqiModel.data.Count > 0)
    {
        foreach (var datum in aqiModel.data)
        {
            <div class="container">

                <p class="title">@aqiModel.city_name</p>
            </div>
        }
    }
}
```

Primero añadiremos la misma condición que le dimos al bloque “qualityAir” en el modelo weatherInformation para que el fondo de ese div nos cambie según la cantidad de la calidad de aire.

```
@if (datum.aqi > 100)
{
    <div class="qualityAir" style="background-image: url('@Url.Content("~/Content/Images/iconWeather/qualityAirBAD.png")');">
        <p class="qualityText" style="margin-left:0.5%">QUALITY AIR : @datum.aqi ICA</p>
        
    </div>
}
else if (datum.aqi > 60)
{
    <div class="qualityAir" style="background-image: url('@Url.Content("~/Content/Images/iconWeather/qualityAirOK.jpg")');">
        <p class="qualityText" style="margin-left:0.5%">QUALITY AIR : @datum.aqi ICA</p>
        
    </div>
}
else if (datum.aqi < 60)
{
    <div class="qualityAir" style="background-image: url('@Url.Content("~/Content/Images/iconWeather/qualityAirGOOD.jpg")');">
        <p class="qualityText" style="margin-left:0.5%">QUALITY AIR : @datum.aqi ICA</p>
        
    </div>
}
```




Y a continuación le pasaremos los valores de los datos que queremos que se muestren en nuestra web utilizando el lenguaje Razor para llamar a las variables que necesitemos.

```
<div class="currentInformation">
  <span>
    <p class="contaminants">AIR POLLUTANTS CONCENTRATION</p>
    <p class="information">Concentration of surface O3:<br>@datum.o3 µg/m³</p>
    <p class="information">Concentration of surface SO2:<br>@datum.so2 µg/m³</p>
    <p class="information">Concentration of surface NO2:<br>@datum.no2 µg/m³</p>
    <p class="information">Concentration of carbon monoxide:<br>@datum.co µg/m</p>
    <p class="information">Concentration of large coarse particles:<br>@datum.pm10 µg/m³</p>
    <p class="information">Concentration of fine particles:<br>@datum.pm25 µg/m³</p>
  </span>
  <span>
    <p class="contaminants" style="margin-top:5%;">POLLEN LEVELS</p>
    <p style="font-size: 17px; margin-top:15px; font-family: 'Gill Sans';font-weight: bolder; color: #085409;">(0=None, 1=Low, 2=Moderate, 3=High, 4=Very High): </p>
    <p class="information" style="margin-top:5%;">Tree pollen: @datum.pollen_level_tree </p>
    <p class="information">Grass pollen: @datum.pollen_level_grass</p>
    <p class="information">Weed pollen: @datum.pollen_level_weed </p>
    <p class="information">Mold pollen: @datum.mold_level</p>
  </span>
</div>
```

Y por último volveremos a utilizar una condición if-if esle ya que queremos que al mostrar cual es el polen predominante en el aire nos muestre una imagen png diferente por cada tipo:

```
@if (datum.predominant_pollen_type=="Trees")
{
  <div class="predominant" style="margin-top:10%">
    <p class="information">PREDOMINANT POLLEN TYPE:</p>
    <p class="pollen">@datum.predominant_pollen_type </p>
  </div>
}
else if (datum.predominant_pollen_type=="Weeds")
{
  <div class="predominant" style="margin-top:10%">
    <p class="information">PREDOMINANT POLLEN TYPE:</p>
    <p class="pollen">@datum.predominant_pollen_type </p>
  </div>
}
else if (datum.predominant_pollen_type=="Molds"){
  <div class="predominant" style="margin-top:10%">
    <p class="information">PREDOMINANT POLLEN TYPE:</p>
    <p class="pollen">@datum.predominant_pollen_type </p>
  </div>
}
else if (datum.predominant_pollen_type=="Grasses"){
  <div class="predominant" style="margin-top:10%">
    <p class="information">PREDOMINANT POLLEN TYPE:</p>
    <p class="pollen">@datum.predominant_pollen_type </p>
  </div>
}
}
```

Con esto ya estarían todos los archivos necesarios para que nuestra aplicación funcione correctamente programados.

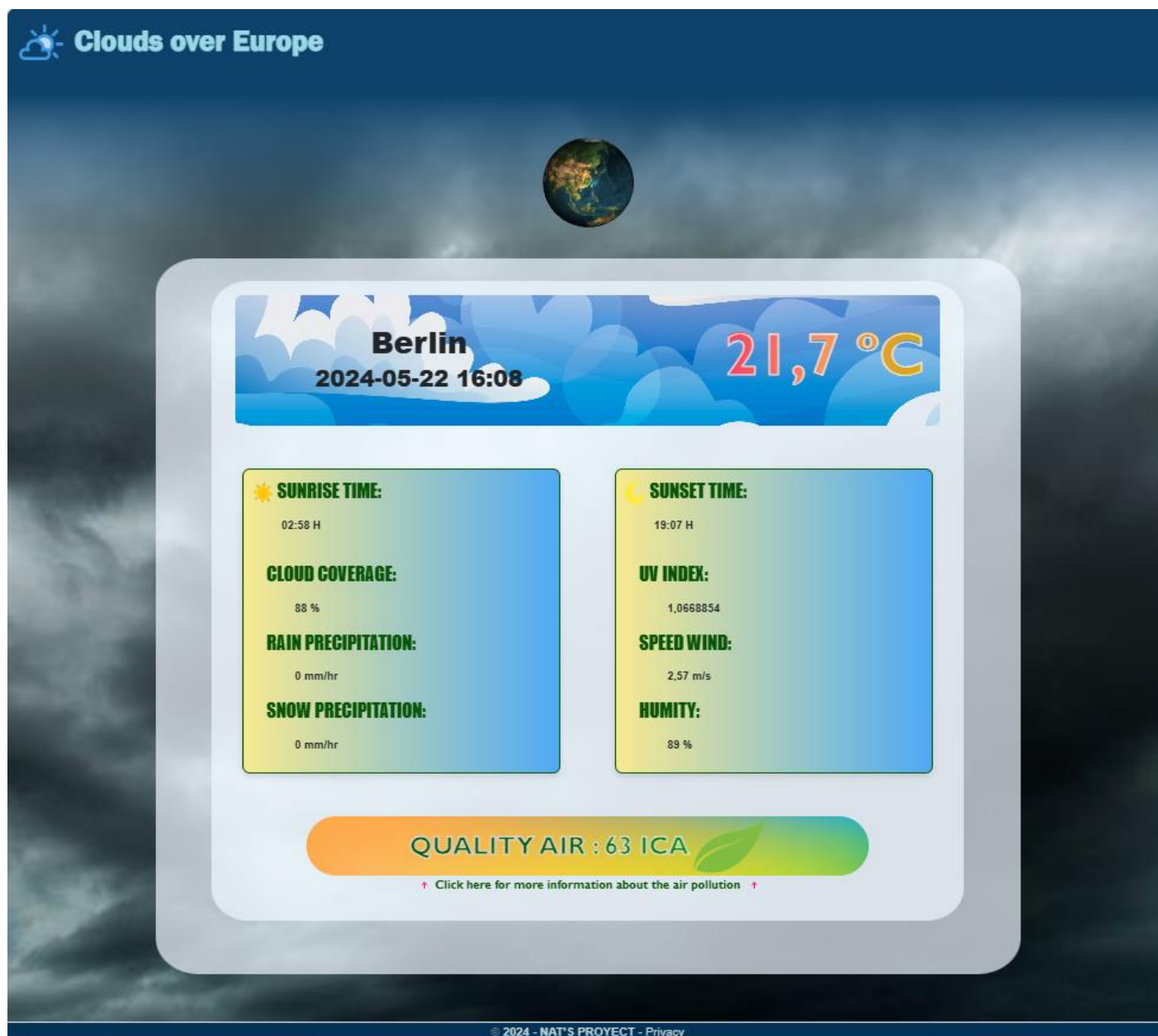


Quedaría el último paso, la comprobación del correcto funcionamiento de la aplicación:

PÁGINA INDEX:

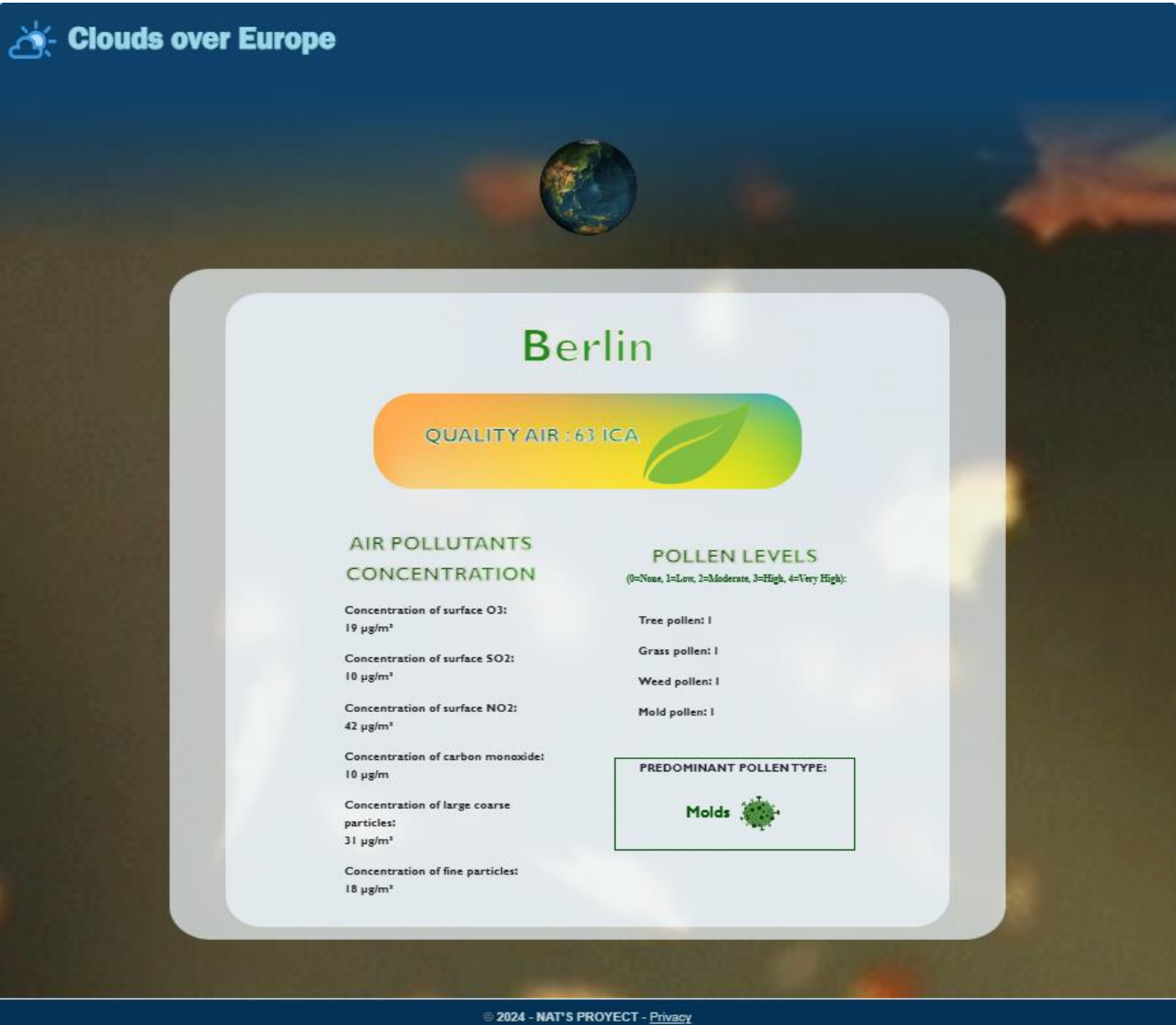


PÁGINA WEATHER INFORMATION:





PÁGINA AQI INFORMATION:





SUGERENCIAS DE MEJORAS

Como sugerencia de mejora para el proyecto, podríamos realizar más apartados, donde el usuario tenga posibilidades de ingresar a más datos relacionados con los países, clima, etc. Así como ampliar al resto de países del mundo o poder tener un mapa interactivo que te permitiera seleccionar la ciudad que quieres consultar clicando sobre él en la ubicación.

Se podría reorganizar la página del inicio de tal forma que no se vean solo las tarjetas, sino otros sectores.

Además, también sería interesante que, una vez llegados a la página con la información referente a la calidad del aire, el usuario pudiera volver hacia la página del clima/temperatura desde el navbar, igual que puede acceder a la página index.

También Y poder implementar animaciones en los botones, fondos o bordes mediante keyframes. Al no poder enlazar correctamente los archivos css al los html no hemos podido utilizar ninguno de sus recursos.

Y, por último, el tiempo actual que muestra es el del último registro de la api, no el actual de cada región. Nos hubiera gustado encontrar una forma de poder colocar el tiempo real y que se actualizara para cada ubicación, no solamente el de nuestra región meridional.



FUENTES

API'S USADAS EN LA APLICACIÓN:

- [GeoNames](#)
- [Weatherbit | API Documentation](#)

WEBS CONSULTADAS:

- [Convert JSON to C# Classes Online - Json2CSharp Toolkit](#)
- [Bootstrap en Español · La biblioteca HTML, CSS y JS más popular del mundo. \(esdocu.com\)](#)
- [CSS: Textos con color degradado | Front.id](#)
- [Bordes de color para perfilar letras: text-stroke y una alternativa | Oloblogger](#)
- [Flip Cards o Tarjetas Giratorias con CSS3 – Josetxu.com](#)
- [Llamar a una API web desde un cliente .NET \(C#\): ASP.NET 4.x | Microsoft Learn](#)
- [List of European capitals by countries \(countries-ofthe-world.com\)](#)
- https://www.w3schools.com/html/html_layout.asp
- <https://josetxu.com/flip-cards-o-tarjetas-giratorias-con-css3/>
- <https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-3>
- <https://learn.microsoft.com/es-es/aspnet/core/mvc/views/razor?view=aspnetcore-8.0>
- <https://www.byronvargas.com/web/como-centrar-una-card-en-css/>