Lab2 - HTML y CSS básico

Pepe Calderón, Fernando Sola, Daniel Ayala, Inma Hernández, Margarita Cruz, Carlos Arévalo, David Ruiz

Escuela Técnica Superior de
Ingeniería Informática

Índice

1

Objetivo

El objetivo de esta práctica es introducir al alumno a los conceptos básicos de HTML y CSS necesarios para construir una versión preliminar de la aplicación a desarrollar. El alumno aprenderá a:

- Utilizar las etiquetas básicas HTML para organizar los elementos en un sitio web.
- Usar y enlazar hojas de estilo CSS para dar formato lógico básico a los elementos mostrados.

2

Introducción

HTML5 y CSS3 son tecnologías básicas para la web que conocemos hoy en día. Si bien es cierto que el uso de frameworks que automatizan gran parte del trabajo de escritura de etiquetas HTML y maquetación CSS se ha vuelto común, el manejo básico de estos lenguajes es una competencia fundamental que todo desarrollador debe tener.

En esta práctica se mostrará una posible implementación, usando estos lenguajes, de algunos de los requisitos de la aplicación a desarrollar. Dado que el aspecto final de la interfaz queda a elección del alumno, no es necesario que ésta tenga el mismo aspecto que el mostrado en esta práctica, siempre que el acabado final cumpla con todos los requisitos funcionales solicitados en el documento de requisitos.

Es también importante recalcar que, en la mayoría de ocasiones, el maquetado final de una aplicación web estará a cargo de una o varias librerías CSS, y no únicamente de código CSS desarrollado a mano. Cuando esto ocurre, es común que el código HTML sufra cambios sustanciales para adaptarlo a las demandas de cada librería CSS concreta. Por ello, el código desarrollado en esta práctica debe interpretarse como un segundo paso en el prototipo de la aplicación a desarrollar, que ahora será ejecutable en el navegador, pero que aún estará sujeto a cambios para mejorar sustancialmente su acabado. En la siguiente práctica se mostrará como integrar una de estas librerías CSS, Bootstrap, con nuestra aplicación, así como los beneficios y desventajas que ésto conlleva.

3

Vista global de la galería

Como se explicó en la práctica anterior, la vista contenida en el archivo `index.html` es la que se mostrará por defecto al acceder a la aplicación. Es por ello que, idealmente, es este archivo el que debería contener la vista a la que se accederá cuando el usuario ingrese a la aplicación. En nuestro caso, aprovecharemos el `index.html` para mostrar las fotos que contiene la galería.

Dado que nuestra aplicación no cuenta aún con funcionalidad JavaScript, no podremos comprobar si el usuario está logueado o no, a fin de mostrarle únicamente las opciones relevantes (por ejemplo, un usuario no logueado no puede crear fotos). En esta versión incluiremos todas las opciones disponibles, y más adelante aprenderemos a ocultarlas si es necesario.

Comenzaremos por añadir un título a la vista de listado de fotos, junto con un enlace para poder crear una nueva foto, que nos llevará a una página diferente:

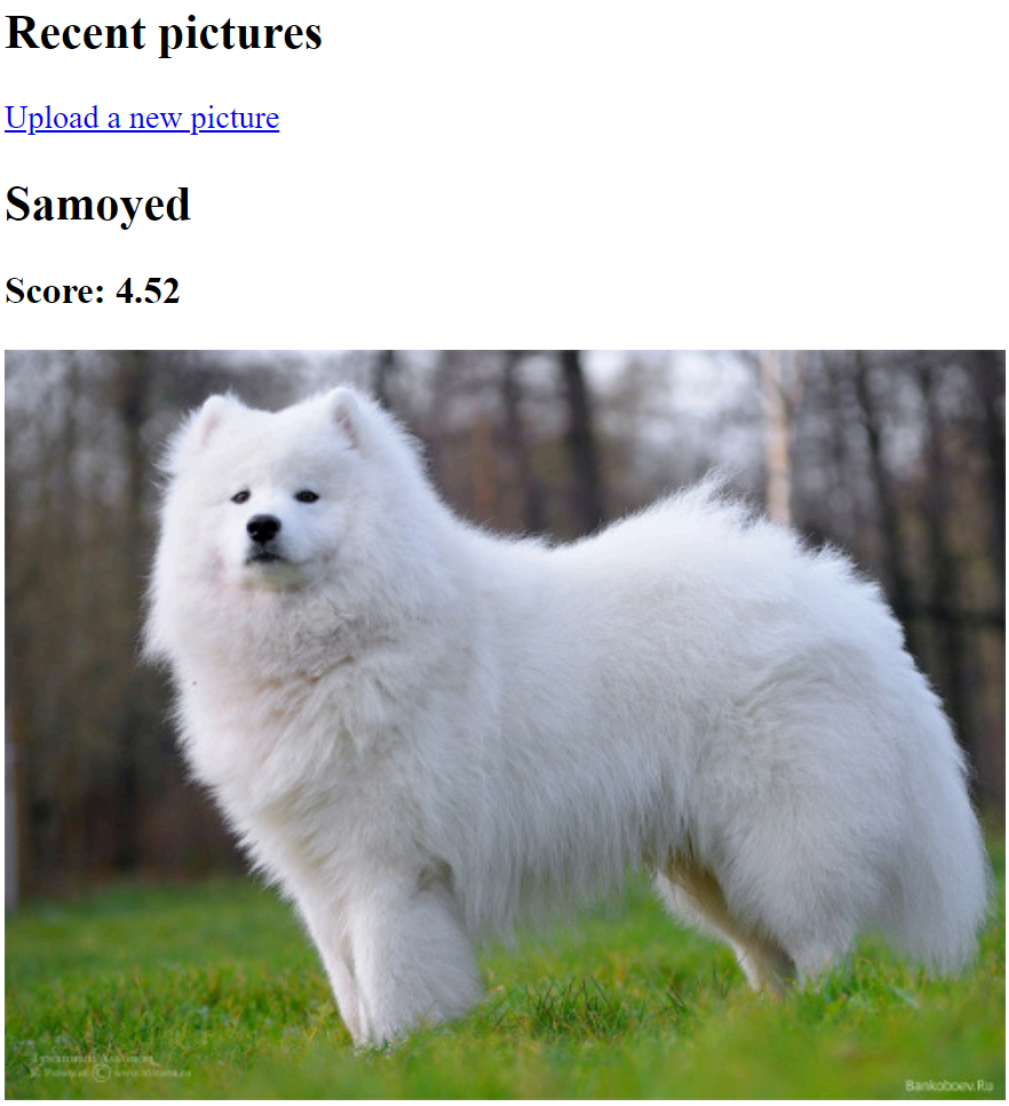
```
<body>
  <h2>Recent pictures</h2>
  <a href="upload_picture.html">Upload a new picture</a>
...
```

A continuación, crearemos un bloque lógico para mostrar una imagen junto con alguna de su información relacionada: título, descripción y puntuación media. Además, añadiremos iconos para valorar la imagen, junto con botones para editarla o borrarla:

```
<div class="photo-block">
  <div class="photo-header-block">
```

```
    <h2 class="photo-title">Samoyed</h2>
    <h3 class="photo-score">Score: 4.52</h3>
  </div>

  <div class="photo-image-block">
     <img class="photo-image" src="https://i.ibb.co/tY1Jcnc/wlZCfCv.jpg">
  </div>

  <div class="photo-metadata-block">
    <p class="photo-description">A very good boy.</p>
  </div>

  <div class="photo-edit-block">
    <button>Edit this photo</button>
    <button>Delete this photo</button>
  </div>
</div>
```

Es buena idea agrupar elementos similares dentro de bloques `<div>` y establecer clases para cada uno de ellos. Las etiquetas `<div>`, por defecto, no tienen un impacto visual directo, sino que se usan para agrupar elementos que, juntos, forman conjuntos lógicos en nuestras vistas. Así, podremos aplicarles estilos conjuntos a todos ellos simplemente referenciando el `<div>` correspondiente, en lugar de repetir el mismo estilo CSS para cada uno de ellos. El resultado visual de este código HTML es el siguiente:



Ahora, con una única foto en nuestra galería, es buen momento para afinar el estilo de cada bloque de imagen mediante CSS. Para ello, crearemos un archivo `style.css` en la carpeta `css/`, y lo enlazaremos a la vista incluyendo la siguiente etiqueta dentro de la cabecera `<head>`:

```
<link rel="stylesheet" href="/css/style.css">
```

Una vez enlazado, podemos escribir estilos en `style.css` para, por ejemplo, centrar horizontalmente todo el contenido, limitar la altura máxima de cada imagen, ajustar los tamaños y fuentes de cada texto, etcétera. Además, podemos añadir un borde alrededor de todo el bloque, para separarlos visualmente cuando haya varios. Este estilo será de aplicación en cualquier vista que enlace la hoja de estilos `style.css`. Se muestra a continuación un ejemplo de código CSS para lograr estos objetivos:

```css
div.photo-block {
    text-align: center;
    border: 1px solid gray;
    border-radius: 5%;
}

div.photo-title {
    font-size: 140%;
    font-family: Arial;
    text-decoration: underline;
}

div.photo-score {
    font-size: 130%;
    font-family: monospace;
}

img.photo-image {
    max-height: 200px;
    width: auto;
}
```

Así, resulta:

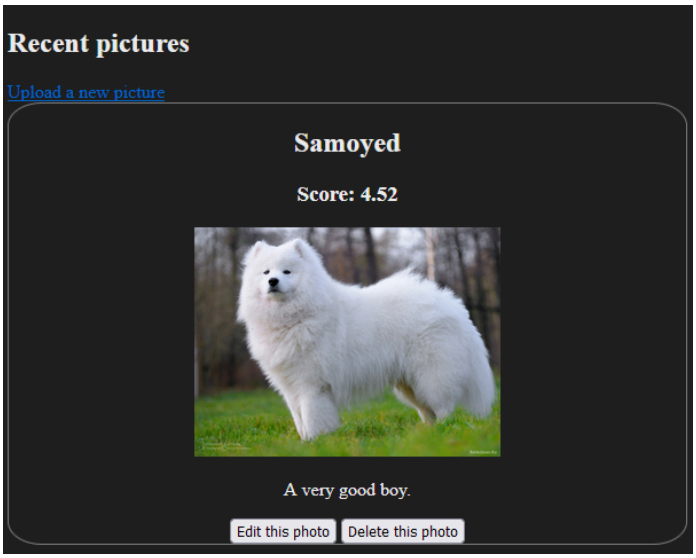
Adicionalmente, podemos hacer cambios en el estilo que afecten a toda la página. Como ejemplo, desarrollaremos nuestra galería con un tema oscuro (texto claro, fondo oscuro) para reducir la fatiga visual. Para ello, añadiremos una clase `dark` a la etiqueta `<body>` y aplicaremos el estilo CSS correspondiente:

```
<body class="dark">
...
```

Y el CSS resulta:

```
body.dark {
  background-color: #222222;
  color: #eeeeee;
}
```

Donde `background-color` y `color` son los atributos CSS que definen el color de fondo y el color del texto, respectivamente. El resultado visual es el siguiente:



Dado que la etiqueta contiene todas las demás, los estilos aplicados a ella también a todo el contenido de la página. Sin embargo, cualquier etiqueta interior que defina su propio estilo tendrá prioridad sobre el estilo de la etiqueta . Por ejemplo, los enlaces siguen siendo azules, pese a que el color de texto de ha sido cambiado a blanco.

Una vez hayamos adaptado el estilo de cada “bloque” fotográfico a nuestro gusto, podemos componer una galería reutilizando estos bloques. En el pasado, se estructuraban este tipo de páginas usando etiquetas `<table>` y organizando la vista mediante una tabla oculta, por ejemplo, mostrando tres imágenes por fila, usando tantas filas como sean necesarias. Esta práctica está considerada obsoleta y por lo tanto se desaconseja.

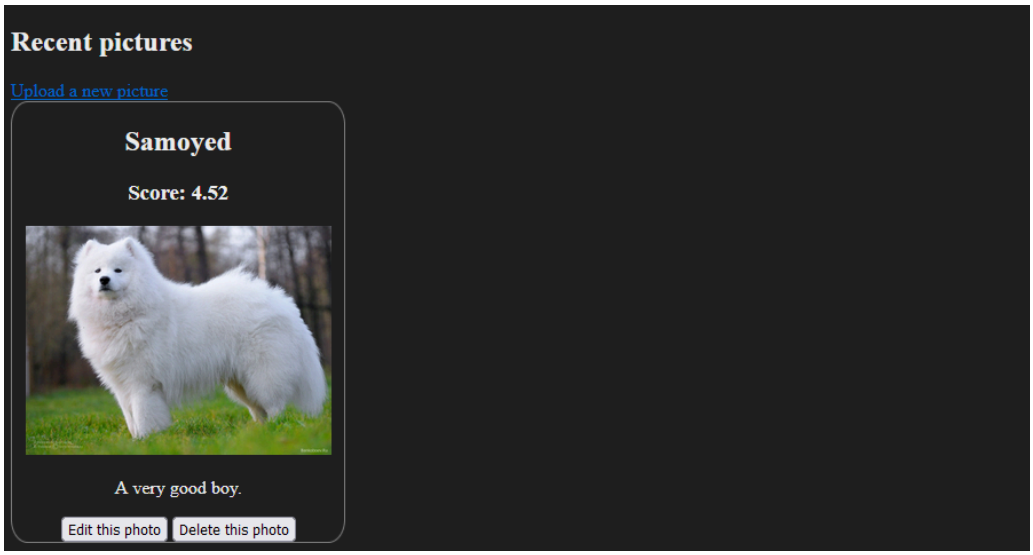
Actualmente CSS ofrece un sistema potente de tablas lógicas o `grid` para realizar una estructura similar. La principal ventaja de usar un grid CSS es que este formato no está determinado por el contenido HTML (formateado físico), y por lo tanto sus posteriores cambios serán mucho más sencillos al requerir únicamente modificaciones en la hoja de

estilos, no en el contenido HTML. Además, se adapta automáticamente a pantallas de todo tipo (responsividad), mientras que el formateado físico no lo hace.

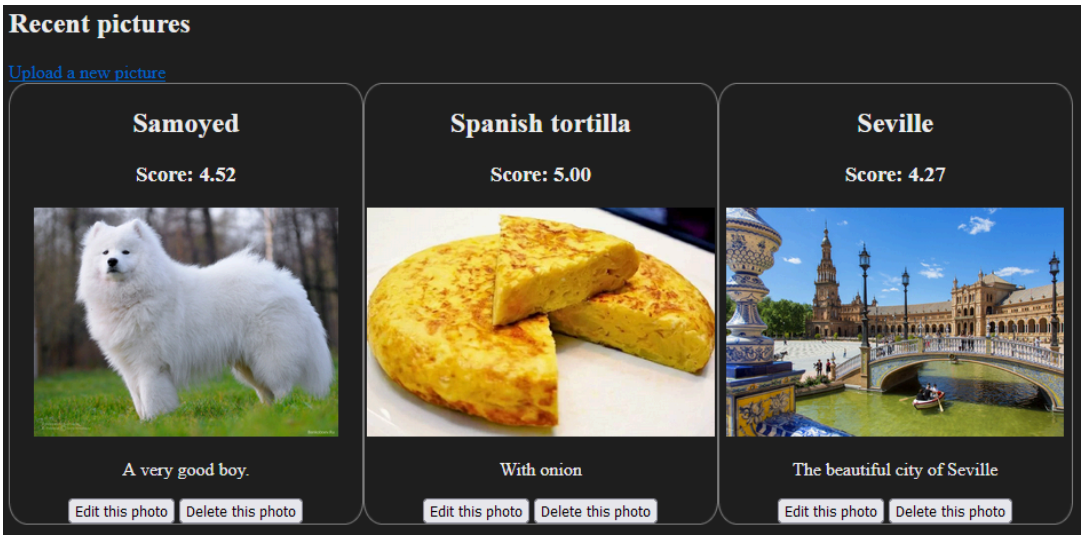
Para agrupar nuestras fotos en una grid CSS, comenzaremos por definir una etiqueta `<div>` que contendrá todas las imágenes. Esta etiqueta será el *container*. Indicaremos que los elementos dentro del *container* serán mostrados en un grid usando CSS. Además, declararemos que nuestro grid tendrá tres columnas, cada una de ellas ocupando el 33% del ancho de la página:

```
index.html:

<div class="gallery-container">
  <!-- Our photo-blocks will be here -->
</div>

style.css:

.gallery-container {
    display: grid;
    grid-template-columns: 33% 33% 33%;
}
```

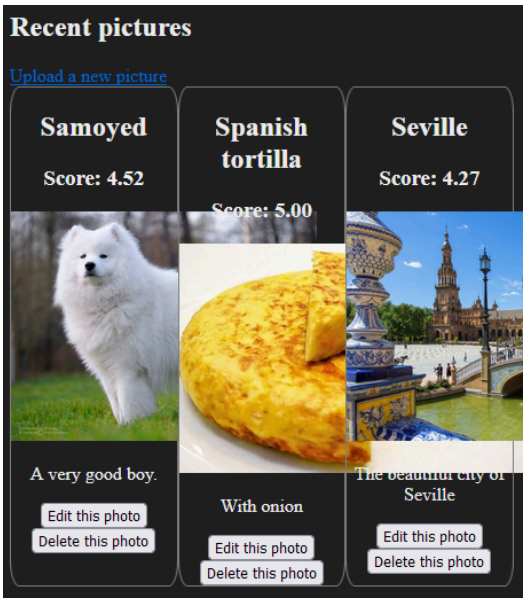
Podemos mover el `<div>` que contiene toda la información de nuestra imagen dentro de este contenedor, y el resultado será el siguiente:



Ahora, nuestro bloque de imagen ocupa la primera posición en esta tabla virtual con un ancho del 33% del total. Podemos replicar el `<div class="photo-block">` y su interior para seguir poblando nuestra galería, con datos diferentes en cada uno de ellos. Observaremos que, gracias al grid CSS, cada bloque se coloca en su lugar sin necesidad de indicarlo en el código HTML:



Si siguiéramos añadiendo fotos, dado que ya se ha rellenado una fila de 3 fotos, la siguiente se colocaría debajo de la primera en una nueva fila. Además, si redimensionamos la página, la anchura de cada columna cambia para adaptarse. Sin embargo, las fotos no se redimensionan automáticamente:



Podemos solucionar este problema añadiendo un ancho dinámico a las imágenes en cada columna: si establecemos su ancho al 100%, ésta ocupará todo el ancho de la columna, haciéndose más pequeña si es necesario. El alto se redimensionará automáticamente para mantener la proporción de aspecto:

```
img.photo-image {
    width: 100%;
    height: auto;
}
```

4

Formulario de registro

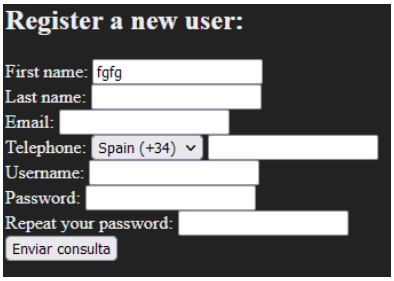
En la práctica anterior creamos una versión preliminar del formulario de registro de la aplicación. Ahora procederemos a refinarlo, añadiendo los campos adicionales que sean necesarios y perfeccionando su estilo mediante CSS:

```
<h2>Register a new user:</h2>
<form>
  <div class="input-group">
    <label for="firstname-input">First name:</label>
    <input type="text" id="firstname-input" name="firstName" required>
  </div>

  <div class="input-group">
    <label for="lastname-input">Last name:</label>
    <input type="text" id="lastname-input" name="lastName" required>
  </div>

  <div class="input-group">
    <label for="email-input">Email:</label>
    <input type="email" id="email-input" name="email" required>
  </div>

  <div class="input-group">
    <label for="telephone-input">Telephone:</label>
    <select name="prefix" id="prefix-input">
      <option value="34">Spain (+34)</option>
      <option value="1">USA (+1)</option>
      <option value="other">Other</option>
```

```
    </select>
    <input type="text" id="telephone-input" name="telephone" required>
  </div>

  <div class="input-group">
    <label for="username-input">Username:</label>
    <input type="text" id="username-input" name="username" required>
  </div>

  <div class="input-group">
    <label for="password-input">Password:</label>
    <input type="password" id="password-input" name="password" required>
  </div>

  <div class="input-group">
    <label for="password2-input">Repeat your password:</label>
    <input type="password2" id="password2-input" name="password2" required>
  </div>

  <div class="input-group">
    <input type="submit">
  </div>
</form>
```

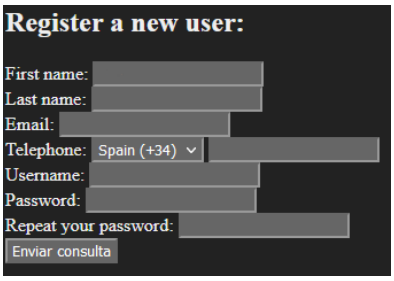
Nótese cómo se añaden a los elementos de entrada el atributo "required", que añade una capa de validación por defecto haciendo que el formulario no pueda ser enviado si el campo en cuestión está vacío. Además, especificar el tipo del campo (text, email, password...) añade detalles como verificaciones adicionales sobre emails o no mostrar la entrada en el caso de contraseñas (aunque éstas viajan como texto plano al enviar el formulario).

Cada entrada se acompaña de una etiqueta `<label>`, que muestra al usuario lo que debe introducir en cada campo. Las label deben ser vinculadas al input correspondiente por razones de accesibilidad, lo cual se consigue haciendo que coincidan el atributo `for` del `<label>` con el `id` del `<input>`. Finalmente, se agrupan las parejas label-input en `<div>` s para su formateado posterior con CSS. El resultado visual es:



Claramente, el resultado puede mejorar. Por ejemplo, los campos `<input>` y `<select>` no concuerdan visualmente con el estilo oscuro. Podemos usar CSS para modificar su color de fondo y de texto, así como el color de su borde:

```
input, select {
    background-color: #666666;
    border-color: #999999;
    color: white;
}
```



Otra posible mejora es alinear todas las etiquetas de los campos del formulario a la derecha y las entradas de texto a la izquierda para darle un aspecto más uniforme. Para ello, haremos que los `<label>` tengan un ancho fijo. Además, gracias a los `<divs>` que hemos definido para separar los grupos del formulario, podemos aumentar la separación vertical entre ellos. Para efectuar estos cambios, enlace el archivo `style.css` en el documento `register.html` de la misma manera mostrada anteriormente. Podemos entonces añadir nuevos estilos destinados a los formularios:

```
.input-group > label {
    display: inline-block;
    width: 200px;
    text-align: right;
}

.input-group {
    margin-bottom: 0.5em;
}
```

El resultado es:

Los pasos a seguir para implementar la vista de login serían muy similares, empleando un formulario con los campos relevantes. Además, se pueden reutilizar los estilos CSS ya definidos para que todos los formularios presentados en la aplicación tengan un estilo uniforme.

5

Creación de una cabecera común

Al desarrollar una aplicación Web, es deseable contar con una cabecera y un pie común a todas las páginas, con objeto de unificar su estilo. Dado que se tratará de un elemento compartido por todas las vistas de nuestra aplicación, y no es una buena práctica duplicar el mismo código en varias vistas diferentes, la definiremos en un archivo HTML separado (por ejemplo, `header.html`) y la enlazaremos a nuestras vistas:

```
<div class="title-block">
    <h1 id="title">Photo Gallery</h1>
    <h3 id="subtitle">IISSI-2 - University of Seville</h3>
</div>

<hr>
```

En este ejemplo, nuestra cabecera tendrá un título, un subtítulo y una división horizontal al final. Además, incluiremos todos los elementos de la cabecera en un bloque lógico `<div>`. Una vez hecha, debemos hacer los siguientes cambios en las vistas en las que deseemos incluir la cabecera:

```
<html>
  <head>
    ...
    <script src="/js/utils/include.js"></script>
  </head>

  <body>
    <div id="page-header"></div>
```

```
    ...

    <script>
            include("header.html", "#page-header");
        </script>
  </body>
</html>
```

Incluiremos en la etiqueta `<head>` el archivo JavaScript `include.js`, que nos permite incluir un archivo HTML externo en una etiqueta de nuestra vista (este archivo se incluye por defecto en la plantilla del proyecto). En el cuerpo de la vista, definiremos una etiqueta vacía al principio, en cuyo interior se colocará automáticamente la cabecera. Finalmente, antes de terminar el cuerpo de la página, se incluye una línea de código JavaScript que indica que el contenido de `header.html` debe volcarse dentro de la etiqueta seleccionada mediante `#page-header`, es decir, aquella cuyo atributo `id` es el especificado.

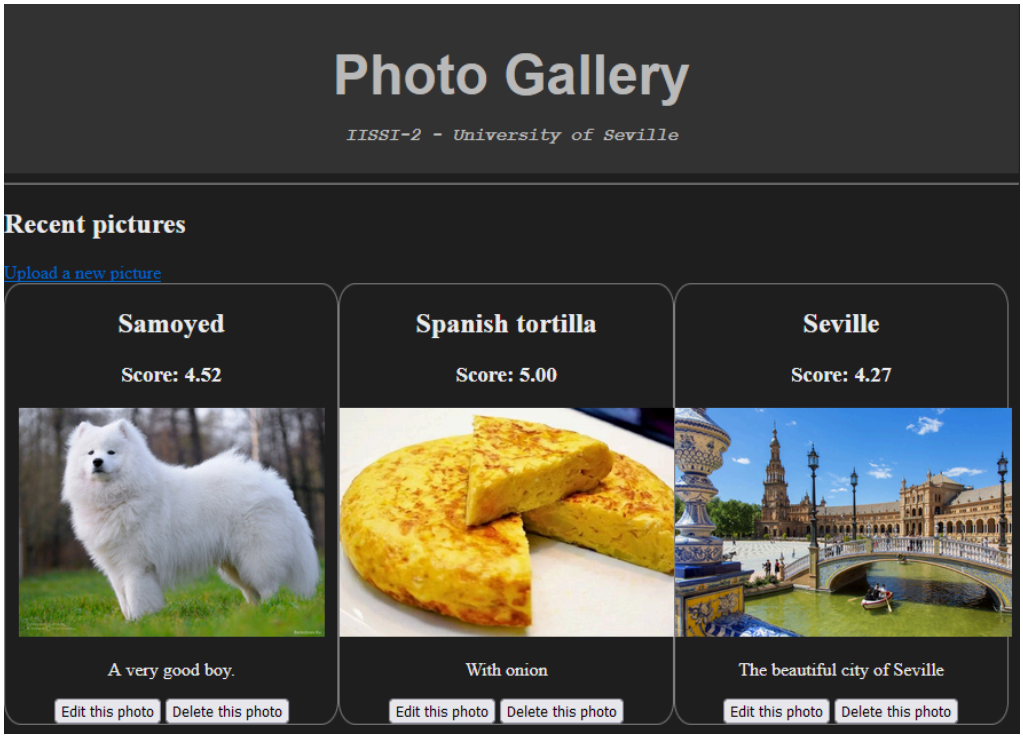
Puede repetir este proceso para incluir la cabecera que ha definido en `header.html` en tantas vistas como desee.

Sin embargo, como es de esperar, se muestra con el estilo por defecto. Mediante CSS podremos hacer cambios más finos, por ejemplo: centrar horizontalmente tanto título como subtítulo, cambiar la fuente y el estilo del texto, su color, y la distancia que cada elemento mantiene con respecto a los demás. Las modificaciones se pueden hacer sobre el archivo `style.css` provisto o crear uno nuevo, en cuyo caso se deberá enlazar con una etiqueta `<link rel="stylesheet" .../>`

```
div.title-block {
  text-align: center;
  padding-top: 1%;
  padding-bottom: 1%;
  background-color: #333333;
}

#title {
  font-size: 300%;
  text-decoration: none;
  color: #BBBBBB;
  font-family: sans-serif;
  margin-bottom: 0;
}

#subtitle {
  font-size: 120%;
  font-style: italic;
  color: #BBBBBB;
  font-family: monospace;
}
```

Mediante la etiqueta `<div>` que hemos definido podemos aplicar estilos comunes a título y subtítulo: centrado horizontal, afinado del *padding* (distancia entre los bordes del elemento y su contenido interior) y color de fondo para el bloque. Para los estilos concretos de cada elemento interior (tamaño y estilo de fuente, color, márgenes...) los referenciamos por su ID, usando el selector `#id-elemento`.

El resultado visual es el siguiente:



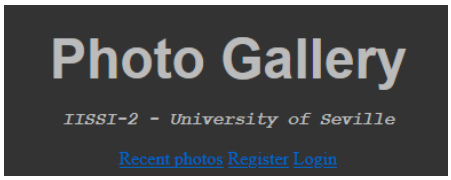
Nótese que, por defecto, las etiquetas `<h1>` tienen subrayado (`text-decoration: underline`), pero este estilo no aplica al título. Dado que existe un selector más específico (por ID) aplicando un estilo que desactiva el subrayado (`text-decoration: none`), es este último el que prevalece.

5.1. Barra de navegación

La cabecera de la aplicación es común a todas las vistas, por que es un lugar muy apropiado para añadir una barra de navegación simple, que contendrá todas las vistas a las que el usuario puede acceder. Inicialmente, podemos usar etiquetas `<a>` para implementar la navegación:

```
<div class="title-block">
    <h1 id="title">Photo Gallery</h1>
    <h3 id="subtitle">IISSI-2 - University of Seville</h3>

    <div class="navigation">
        <a href="index.html">Recent photos</a>
        <a href="register.html">Register</a>
```

```
        <a href="login.html">Login</a>
    </div>
</div>
```



Ya que se encuentra en el bloque definido anteriormente, se beneficia de su centrado horizontal. Podemos ajustar el estilo mediante CSS, para que se asemeje más a una barra de navegación clásica:

```
div.navigation > a {
    color: white;
    font-size: 120%;
    background-color: #555555;
    font-family: monospace;
    text-decoration: none;
    display: inline-block;
    min-width: 150px;
}
```

En la siguiente práctica, aprenderemos a usar la librería CSS Bootstrap para dar un acabado visual más profesional a los elementos implementados en ésta.

6

Actualización en GitHub

Actualice su proyecto en GitHub con los cambios hechos durante esta sesión. Recuerde los comandos relevantes:

- `git add .` añade los cambios efectuados en todos los archivos al próximo `commit` a efectuar.
- `git commit -m "mensaje"` crea un nuevo `commit` con los cambios efectuados a los archivos añadidos con el comando anterior, y con el mensaje indicado.
- `git push` actualiza el repositorio remoto con los cambios efectuados.

> **Info**
>
> Versión PDF disponible