Lab1 - Configuración del proyecto e introducción a HTML

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 descargar y configurar un proyecto Silence que se usará a lo largo de las posteriores sesiones de laboratorio, así como a crear contenido básico en HTML5 usando las etiquetas aprendidas en teoría.

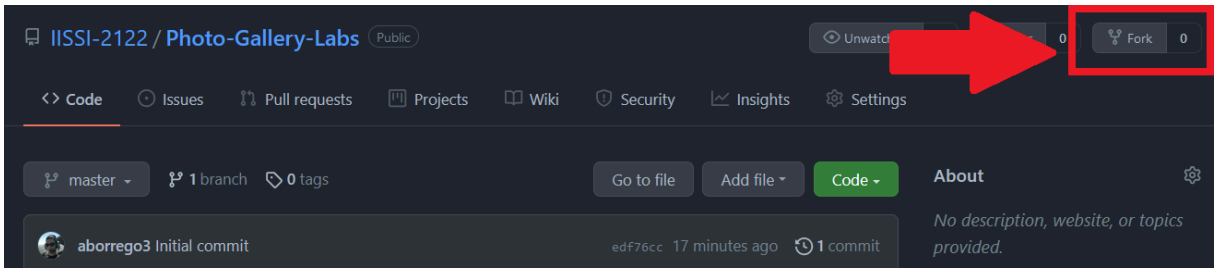
- Crear una copia de un proyecto Silence existente en GitHub.
- Descargar un proyecto en GitHub para modificarlo localmente.
- Conocer la estructura de archivos relativa a la aplicación web del proyecto.
- Usar las herramientas provistas por Silence para autogenerar archivos de configuración de endpoints y pruebas.
- Crear una nueva vista en el proyecto donde implementar contenido HTML básico.
- Subir cambios realizados localmente al repositorio GitHub.

2 Creación del proyecto

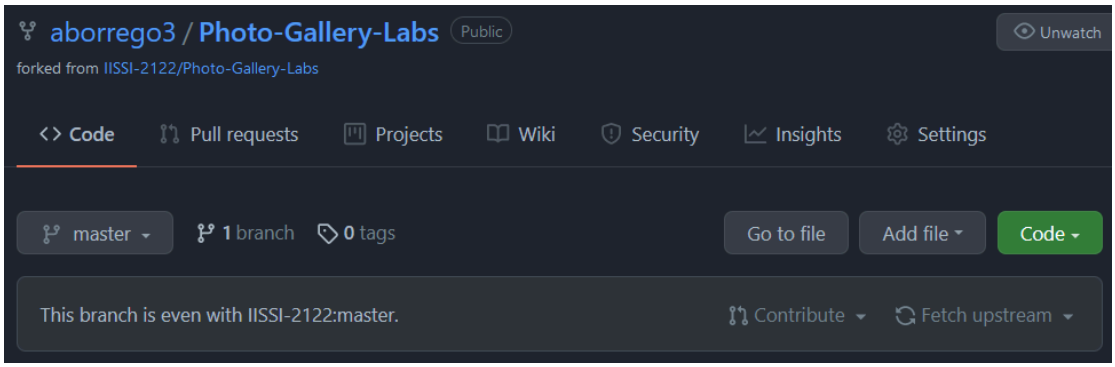
En las clases de laboratorio implementaremos un proyecto Silence que da soporte a una galería fotográfica. En la organización de la asignatura en GitHub se encuentra una versión inicial del mismo, que contiene los scripts SQL y la configuración necesaria para inicializar la base de datos.

Para trabajar en él en las siguientes clases de laboratorio, debemos hacer una copia de este repositorio en nuestro usuario de GitHub, para poder subir a nuestro repositorio los cambios realizados a lo largo de las sesiones de laboratorio.

Podemos hacer una copia del mismo accediendo a él, y pulsando en el botón "Fork":



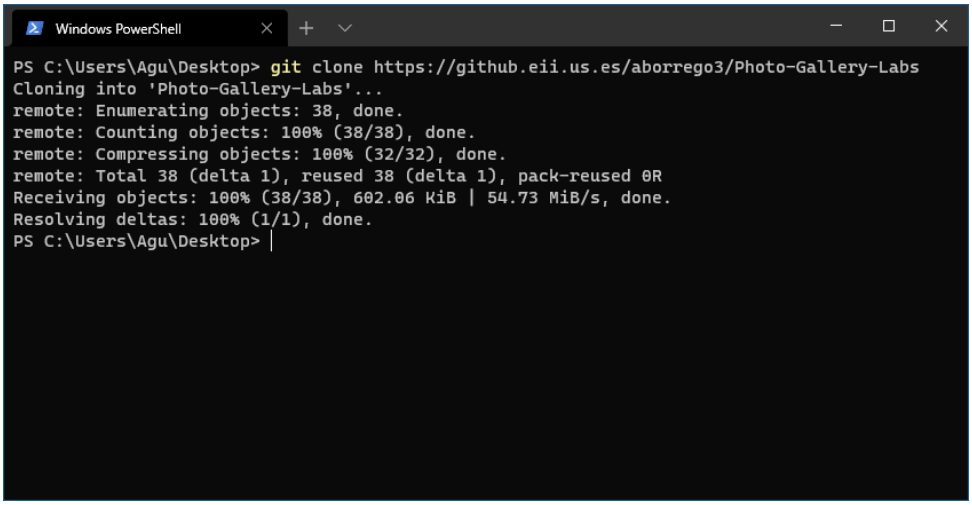
Si se nos pregunta dónde se debe realizar el fork, **seleccionaremos nuestro usuario**. Tras unos segundos de espera, la copia estará creada y podremos trabajar sobre ella:



Como se aprecia en la parte superior izquierda, se indica que este repositorio ha sido creado a partir del existente en la organización del curso. Además, se indica que en este momento, nuestra copia es idéntica al original ("*This branch is even...*"). Cuando subamos nuevos cambios, nuestra copia estará por delante de la original.

3 Descarga y configuración

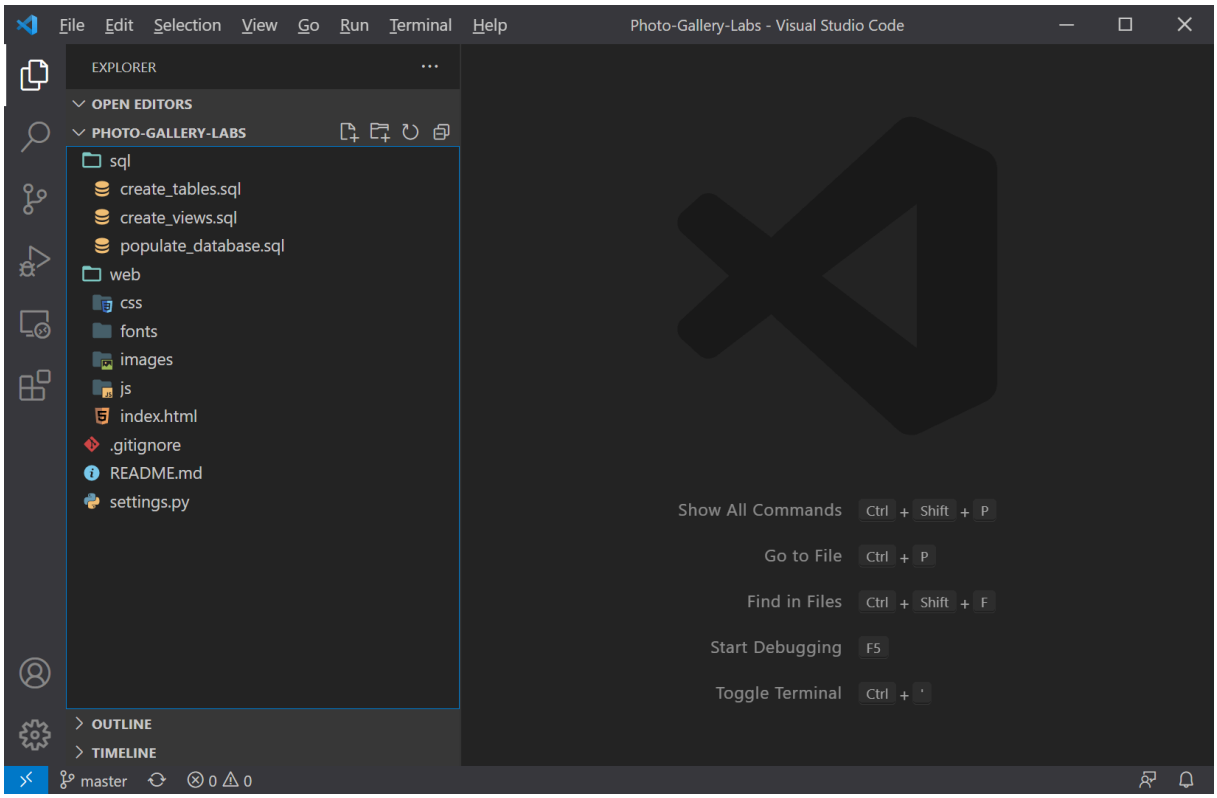
Para trabajar en el proyecto, debemos descargar el “fork” que acabamos de crear en nuestro ordenador. Para ello, usaremos el comando “git clone”:



{: .warning }

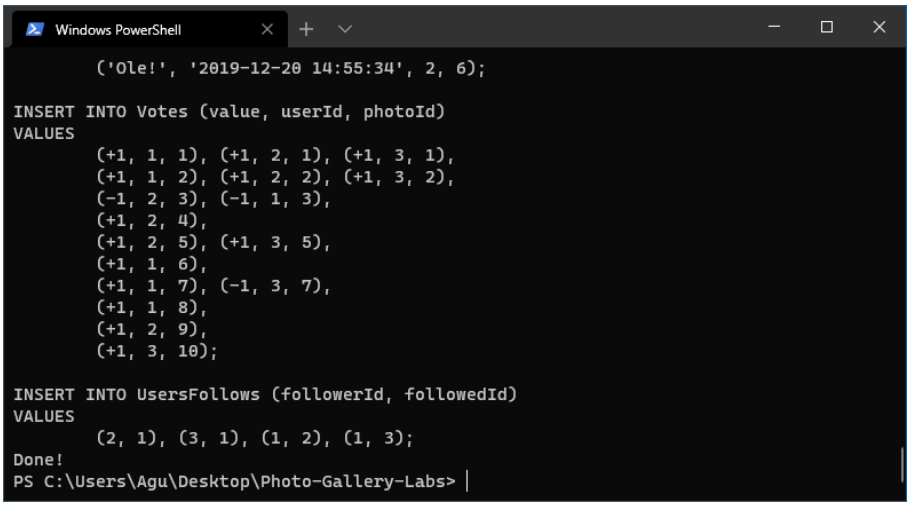
> **Info**
>
> Asegúrese de que el proyecto que está clonando es su *fork*, y no el original de la organización del curso. Su UVUS debería aparecer en la URL que se está clonando, y no el de IISSI.

Esto creará una carpeta en el directorio actual de la consola con el mismo nombre del repositorio, donde se encuentran los archivos del proyecto. Si lo abrimos con VSCode, podremos comenzar a realizar cambios en el mismo:

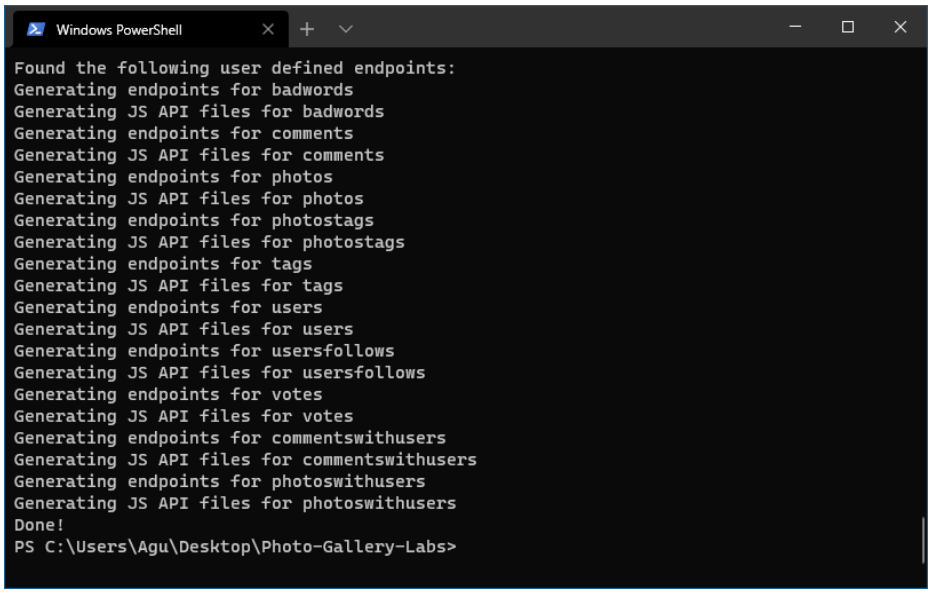


Por defecto, el proyecto está configurado para usar el usuario `iissi_user` de MariaDB, con contraseña `iissi$user`, y la base de datos `gallery`. Debemos asegurarnos de que la base de datos configurada existe, y que el usuario que se usa para la conexión tiene permisos en ella. Si alguno de estos parámetros debe modificarse, se deberá actualizar el archivo `settings.py` en la carpeta del proyecto.

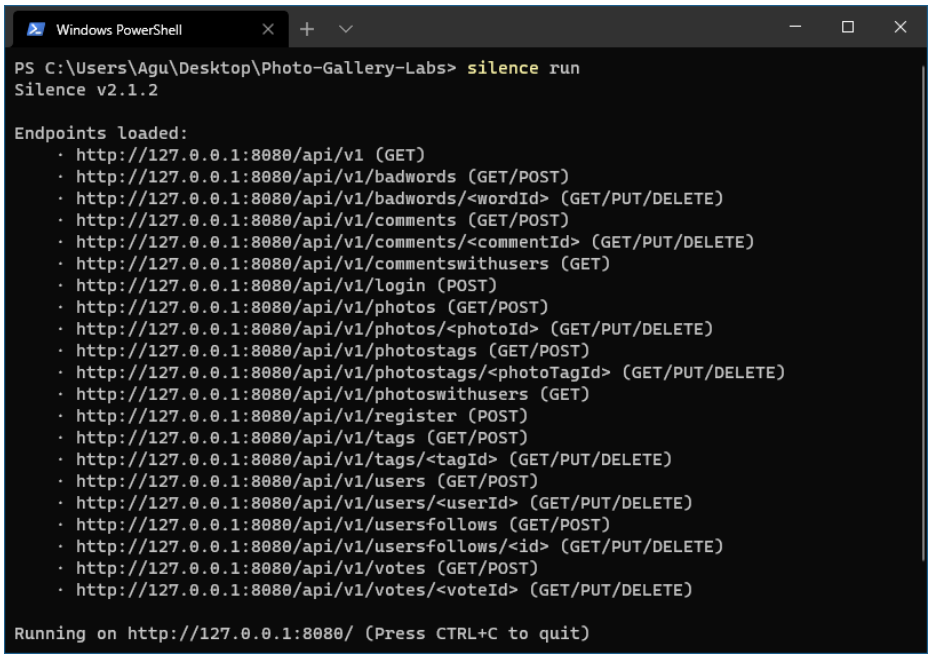
Para comprobar que la configuración es correcta, desplegaremos la base de datos de la galería fotográfica. El proyecto descargado ya incluye los ficheros SQL y la configuración adecuada, por lo que basta con ejecutar `silence createdb` en la consola. Si todo es correcto, se mostrarán las sentencias SQL ejecutadas:



Finalmente, Silence permite desplegar una API REST creada automáticamente a partir de la estructura de tablas existente en la base de datos. El comando `silence createapi` crea los archivos Python correspondientes en la carpeta `endpoints` del proyecto para definir dicha API, así como los archivos JavaScript correspondientes en la aplicación web necesarios para consumir la API creada:



Finalmente, podemos lanzar el servidor con `silence run`, donde deberían aparecer todos los endpoints disponibles en el proyecto:


Windows PowerShell
PS C:\Users\Agu\Desktop\Photo-Gallery-Labs> silence run
Silence v2.1.2
Endpoints loaded:
· http://127.0.0.1:8080/api/v1 (GET)
· http://127.0.0.1:8080/api/v1/badwords (GET/POST)
· http://127.0.0.1:8080/api/v1/badwords/<wordId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/comments (GET/POST)
· http://127.0.0.1:8080/api/v1/comments/<commentId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/commentswithusers (GET)
· http://127.0.0.1:8080/api/v1/login (POST)
· http://127.0.0.1:8080/api/v1/photos (GET/POST)
· http://127.0.0.1:8080/api/v1/photos/<photoId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/photostags (GET/POST)
· http://127.0.0.1:8080/api/v1/photostags/<photoTagId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/photoswithusers (GET)
· http://127.0.0.1:8080/api/v1/register (POST)
· http://127.0.0.1:8080/api/v1/tags (GET/POST)
· http://127.0.0.1:8080/api/v1/tags/<tagId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/users (GET/POST)
· http://127.0.0.1:8080/api/v1/users/<userId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/usersfollows (GET/POST)
· http://127.0.0.1:8080/api/v1/usersfollows/<id> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/votes (GET/POST)
· http://127.0.0.1:8080/api/v1/votes/<voteId> (GET/PUT/DELETE)
Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)

4 Estructura de archivos

La estructura general de archivos de un proyecto Silence se introdujo en la asignatura IISSI-1. En ella, existe una carpeta `web/` que contiene los archivos asociados a la aplicación web del proyecto. La estructura de esta carpeta es la siguiente:

- `css/` : Carpeta que contiene las hojas de estilo.
- `fonts/` : Carpeta que contiene los archivos de fuentes tipográficas.
- `images/` : Carpeta que contiene las imágenes usadas en la aplicación.
- `js/` : Carpeta que contiene los ficheros JavaScript.
- `*.html` : Vistas HTML de la aplicación.

A su vez, la carpeta `js/` contiene:

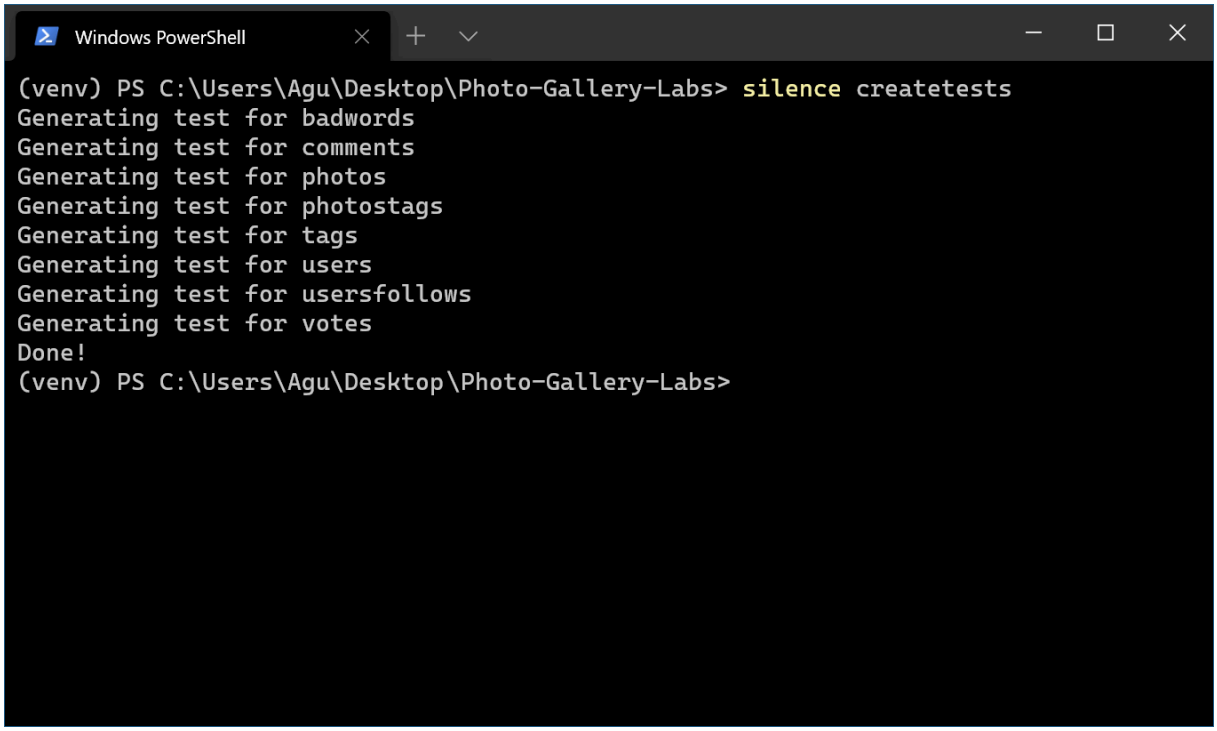
- `api/` : Módulos de acceso y consumo de la API RESTful del proyecto
- `libs/` : Librerías JS externas usadas en la aplicación
- `renderers/` : Módulos de renderizado de entidades
- `utils/` : Módulos de utilidad general
- `validators/` : Módulos de validación de formularios
- `*.js` : Archivos JS asociados a cada una de las vistas de la aplicación

En esta sesión trabajaremos con archivos `.html` para crear contenido usando este lenguaje de marcado. En posteriores sesiones de laboratorio se cubrirán el contenido y uso del resto de componentes de la aplicación web.

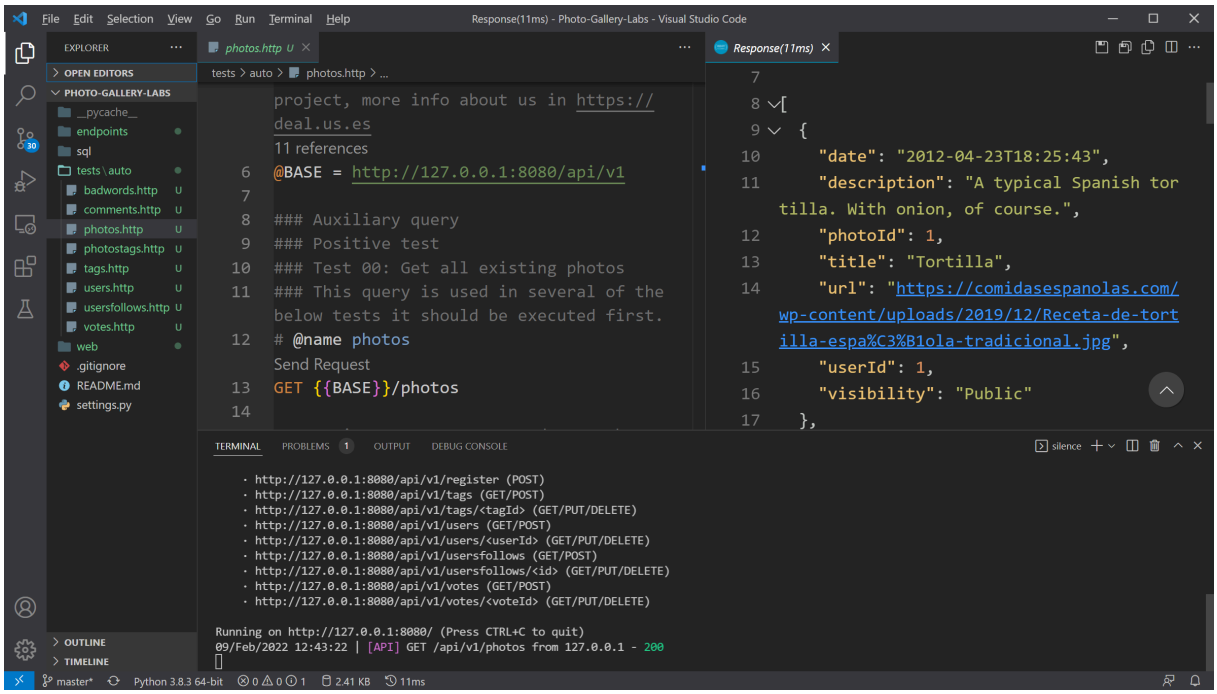
5

Autogeneración de tests

Silence es capaz de generar archivos de tests automáticamente para los endpoints generados a partir de las tablas de la base de datos. Para ello, debemos ejecutar el comando `silence createtests` en la consola:



Los tests se generan en la carpeta `tests/auto/`, y se pueden ejecutar si se tiene instalada la extensión REST Client de Visual Studio Code y el servidor se encuentra corriendo mediante `silence run`:


File Edit Selection View Go Run Terminal Help
Response(11ms) - Photo-Gallery-Labs - Visual Studio Code
EXPLORER
OPEN EDITORS
PHOTO-GALLERY-LABS
__pycache__
endpoints
sql
tests \ auto
badwords.http
comments.http
photos.http
photostags.http
tags.http
users.http
usersfollows.http
votes.http
web
.gitignore
README.md
settings.py
OUTLINE
TIMELINE
photos.http U
tests > auto > photos.http > ...
project, more info about us in https://deal.us.es
11 references
@BASE = http://127.0.0.1:8080/api/v1
Auxiliary query
Positive test
Test 00: Get all existing photos
This query is used in several of the below tests it should be executed first.
@name photos
Send Request
GET {{BASE}}/photos
Response(11ms)
[
{
"date": "2012-04-23T18:25:43",
"description": "A typical Spanish tortilla. With onion, of course.",
"photoId": 1,
"title": "Tortilla",
"url": "https://comidasespanolas.com/wp-content/uploads/2019/12/Receta-de-tortilla-espa%C3%B1ola-tradicional.jpg",
"userId": 1,
"visibility": "Public"
},
TERMINAL
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
silence
· http://127.0.0.1:8080/api/v1/register (POST)
· http://127.0.0.1:8080/api/v1/tags (GET/POST)
· http://127.0.0.1:8080/api/v1/tags/<tagId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/users (GET/POST)
· http://127.0.0.1:8080/api/v1/users/<userId> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/usersfollows (GET/POST)
· http://127.0.0.1:8080/api/v1/usersfollows/<id> (GET/PUT/DELETE)
· http://127.0.0.1:8080/api/v1/votes (GET/POST)
· http://127.0.0.1:8080/api/v1/votes/<voteId> (GET/PUT/DELETE)
Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
09/Feb/2022 12:43:22 | [API] GET /api/v1/photos from 127.0.0.1 - 200
master*
Python 3.8.3 64-bit
0 0 1
2.41 KB
11ms

6 Creación de contenido HTML

Para las siguientes secciones, deberemos haber iniciado el servidor mediante el comando `silence run`, tal y como se explicó en la anteriormente.

6.1. Elementos HTML básicos

Si accedemos en nuestro navegador a la dirección en la que se está ejecutando el servidor web de Silence, se nos presentará una página en blanco. Esto es debido a que la página por defecto es la definida en el fichero `index.html`, y éste está vacío.

Comenzaremos a darle contenido al `index.html` con la estructura común a todos los archivos HTML:

```
<!DOCTYPE html>
<html>
    <head>

    </head>

    <body>

    </body>
</html>
```

La primera línea sirve para declarar la versión HTML usada, en este caso `html` representa HTML5. Todo el documento está contenido en la etiqueta `<html>`, que a su vez contiene una

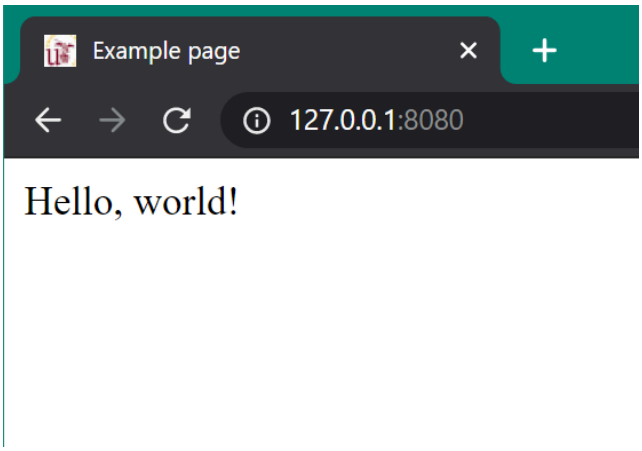
cabecera `<head>` con información para el navegador, y un contenido `<body>` que es el que se muestra al usuario.

Dentro de la cabecera podemos incluir metadatos sobre la página, por ejemplo, su título o una imagen de miniatura o *favicon* que aparecerá en la pestaña del navegador. A su vez, todo lo que se encuentre dentro del cuerpo de la página será mostrado en el navegador:

```
<head>
    <title>Example page</title>
    <link rel="icon" href="/images/favicon.ico">
</head>

<body>
    Hello, world!
</body>
```

Si refrescamos la página, aparecen los cambios realizados en `index.html`:

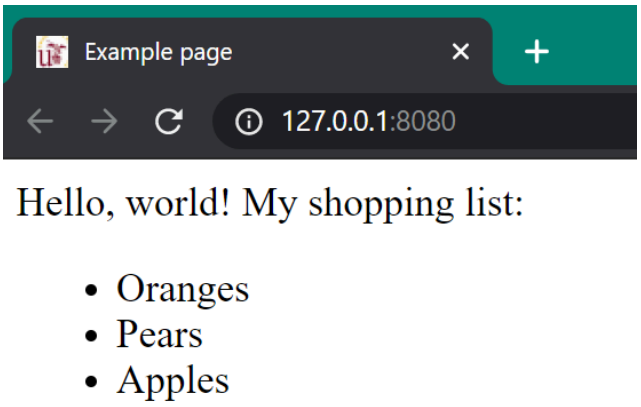


Note cómo el título y el icono de la pestaña se corresponden con lo definido dentro de la etiqueta `<head>`. A su vez, la ruta del icono hace referencia al archivo que puede encontrarse en la carpeta `images` de la aplicación web. La carpeta `docs/` del proyecto es la raíz de la aplicación web, por lo que todas las rutas relativas a archivos HTML, CSS o JS hacen referencia al contenido de esa carpeta.

Podemos seguir añadiendo contenido en el cuerpo del documento, por ejemplo, una lista no numerada mediante las etiquetas `<ul>` (*unordered list*) y `<li>` (*list item*):

```
My shopping list:
<ul>
    <li>Oranges</li>
    <li>Pears</li>
    <li>Apples</li>
</ul>
```

Si incluimos la lista después del “hola mundo”, el resultado es el siguiente:



Pruebe a cambiar la etiqueta `<ul>` por `<ol>` (*ordered list*) y observe el resultado.

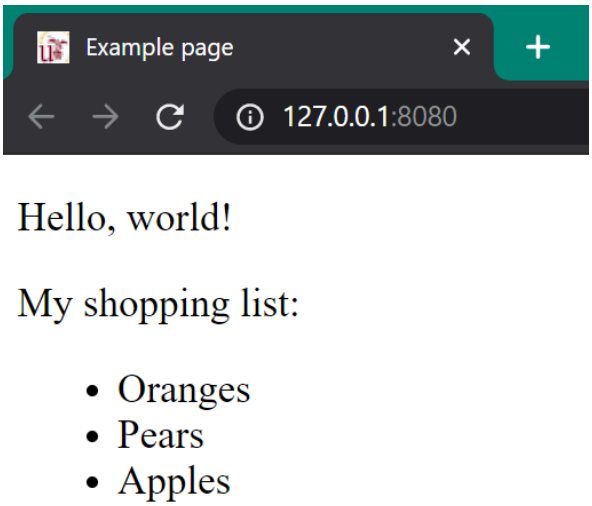
Note que, pese a separar el texto con saltos de línea, estos no se reflejan en el contenido mostrado por el navegador. Esto es debido a que los saltos de línea en un fichero HTML son generalmente ignorados por los navegadores y no influyen en la forma de organizar el contenido que se muestra.

Para forzar a que la lista de la compra aparezca bajo el "hola mundo" podemos envolver cada línea en una etiqueta de párrafo `<p>` (*paragraph*):

```
<p>Hello, world!</p>

<p>My shopping list:</p>
<ul>
    <li>Oranges</li>
    <li>Pears</li>
    <li>Apples</li>
</ul>
```

Un elemento de párrafo, que debe contener texto, ocupa todo el ancho posible, y se disponen visualmente unos sobre otros:

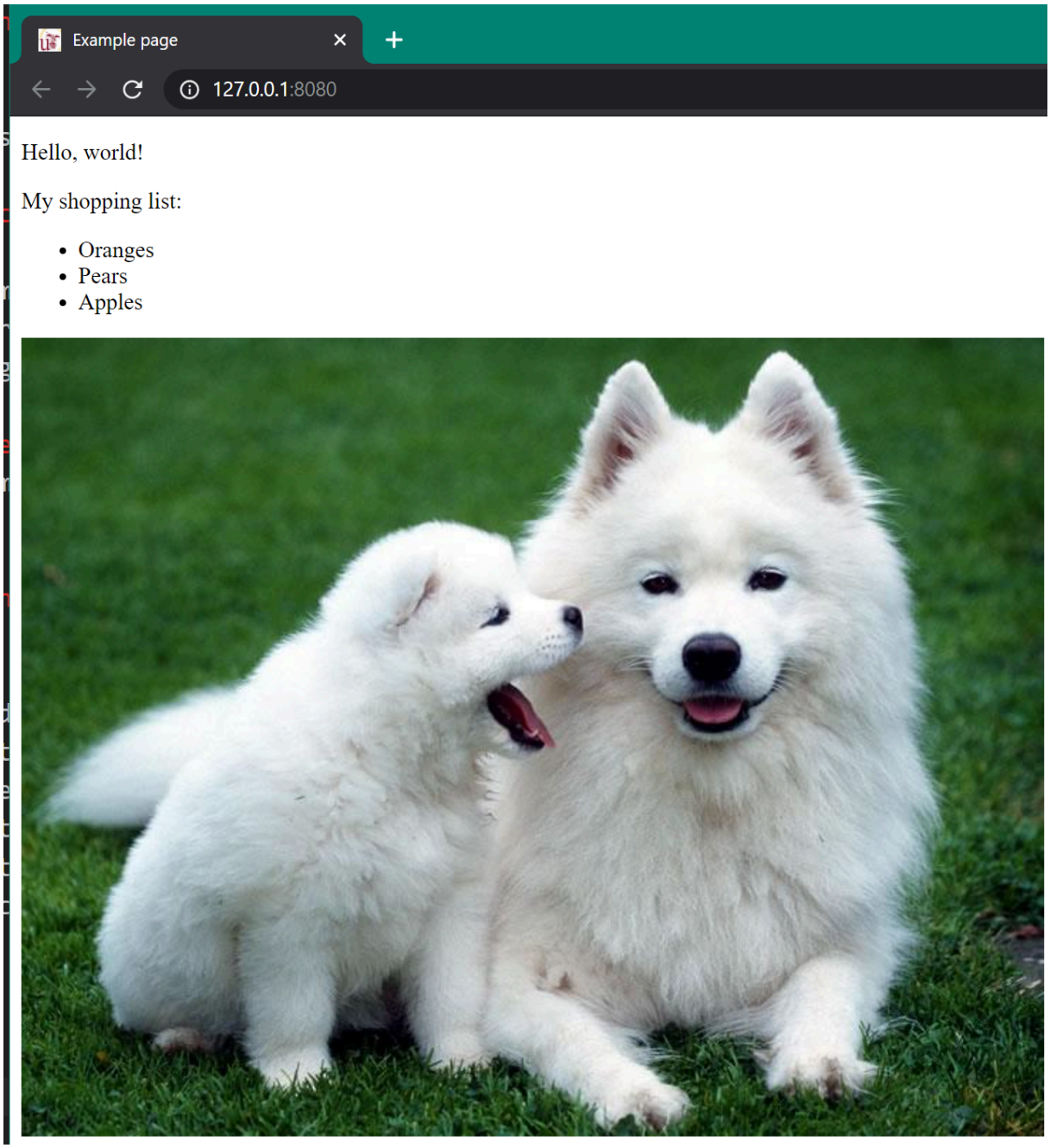


También se pueden incluir imágenes, para lo que se deben guardar en la carpeta `images` del proyecto. Suponiendo una imagen llamada `dogs.jpg`, se puede referenciar del siguiente modo:

```
<img src="/images/dogs.jpg"
     title="Two dogs"
     alt="Two dogs rest in a grass field">
```

La ruta de la imagen, relativa al directorio de nuestra aplicación web, se indica en el atributo `src`. El atributo `title` almacena el título de la foto, que se muestra al detener el ratón sobre ella, mientras que el atributo `alt` contiene una descripción de la imagen. Los dos últimos atributos no son obligatorios, pero sí altamente recomendados por motivos de accesibilidad. Note también que la etiqueta `<img>` no necesita etiqueta de cierre, ya que no es necesaria al no poder almacenar en su interior otras etiquetas.

El resultado es el siguiente:



Por defecto, las imágenes aparecen a su tamaño original. En el boletín correspondiente a estilos CSS se mostrará cómo modificar este comportamiento.

6.2. Enlaces a otros documentos

Una pantalla de la aplicación web, representada mediante un documento HTML, puede contener enlaces a otras vistas contenidas en otros documentos HTML. A modo de ejemplo, crearemos un archivo `register.html` en la carpeta `web`, junto al ya existente `index.html`. En este archivo introduciremos un formulario que podrá servir en el futuro para dar de alta a un usuario, por ejemplo.

El contenido inicial de `register.html` será el siguiente:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Register form</title>
        <link rel="icon" href="/images/favicon.ico">
    </head>

    <body>
        TO-DO
    </body>
</html>
```

Note como las cabeceras en el `<head>` de los documentos HTML de una misma aplicación suelen ser similares, mientras que el contenido del `<body>` varía.

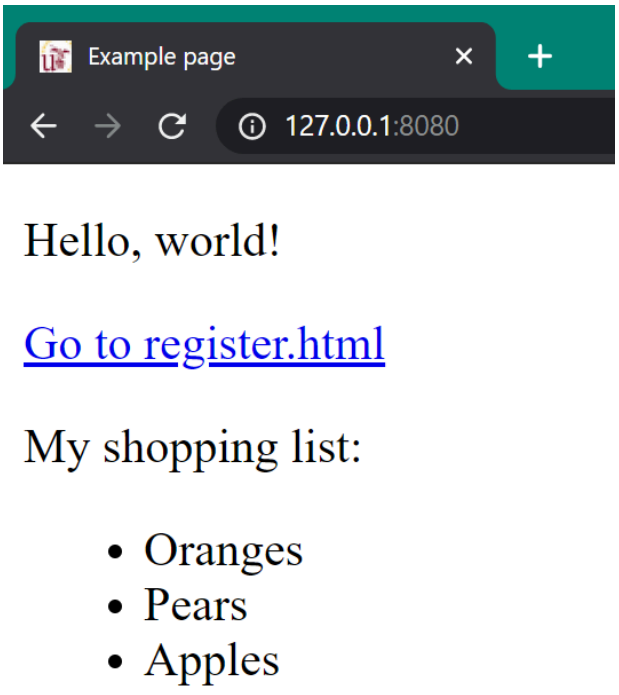
A continuación, modificaremos `index.html` para añadir un enlace al nuevo documento creado usando una etiqueta `<a>` (*anchor*):

```
(...)
<p>Hello, world!</p>

<a href="register.html">Go to register.html</a>

<p>My shopping list:</p>
(...)
```

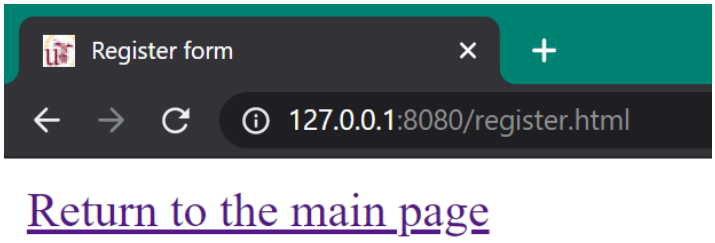
El documento al que se enlace se indica en el atributo `href`, mientras que el interior de la etiqueta contiene el texto que se mostrará como enlace:



Pulsar en este enlace nos llevará a `register.html`. Implementemos en este archivo un enlace para ir de vuelta a `index.html`:

```
<a href="index.html">Return to the main page</a>
```

Tendremos así navegación bidireccional entre ambas vistas:



6.3. Formularios

Los formularios son una parte fundamental de cualquier aplicación web, ya que permiten al usuario introducir datos para su procesamiento por parte del servidor. A modo de ejemplo, implementaremos un formulario de registro en `register.html`. Los formularios están contenidos dentro de etiquetas `<form>`:

```
<form>
    ...
</form>
```

Opcionalmente, las etiquetas `<form>` pueden tener atributos como `action` para definir la ruta del servidor que ha de procesar los datos enviados, y `method` para establecer el método HTTP a usar.

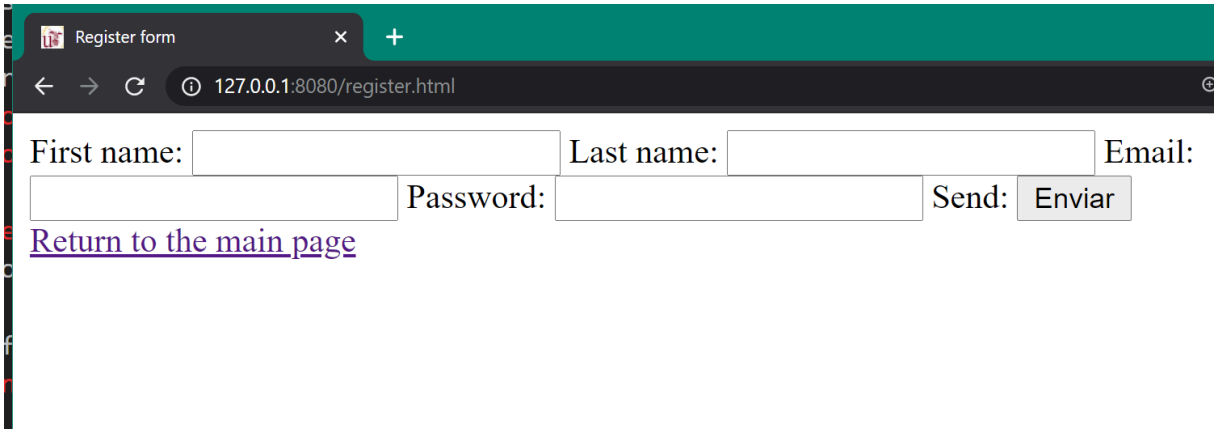
Cada campo del formulario se define mediante etiquetas `<input>`:

```
<form>
    First name: <input type="text" name="firstName">
    Last name: <input type="text" name="lastName">
    Email: <input type="email" name="email">
    Password: <input type="password" name="password">
    Send: <input type="submit">
</form>
```

Observe que los elementos `<input>` tienen un atributo `type` que indican el tipo de elemento de entrada del que se trata (dispone aquí de un listado completo de tipos de inputs), y un

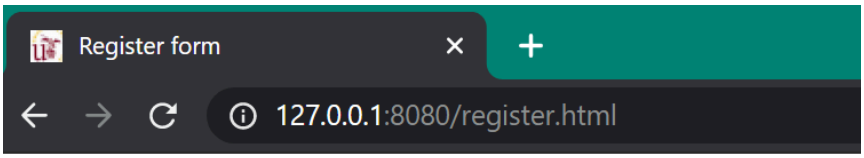
atributo `name` que determina el nombre del campo que se envía al servidor con el valor proporcionado.

El resultado que muestra el navegador es:



Como ya ocurrió antes, los saltos de línea no tienen un efecto directo en la estructura visual del documento, y los campos se muestran uno tras otro. Podemos solucionar este problema incluyendo cada campo del formulario dentro de una etiqueta `<p>`, que se muestran una encima de otra:

```
<form>
    <p>
        First name: <input type="text" name="firstName">
    </p>
    <p>
        Last name: <input type="text" name="lastName">
    </p>
    <p>
        Email: <input type="email" name="email">
    </p>
    <p>
        Password: <input type="password" name="password">
    </p>
    <p>
        Send: <input type="submit">
    </p>
</form>
```



First name:

Last name:

Email:

Password:

Send: Enviar

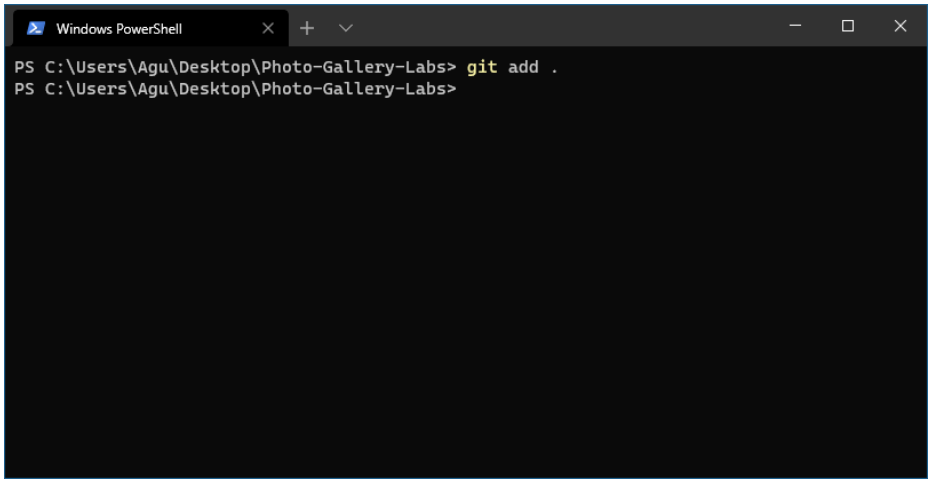
Return to the main page

7

Subida de cambios a GitHub

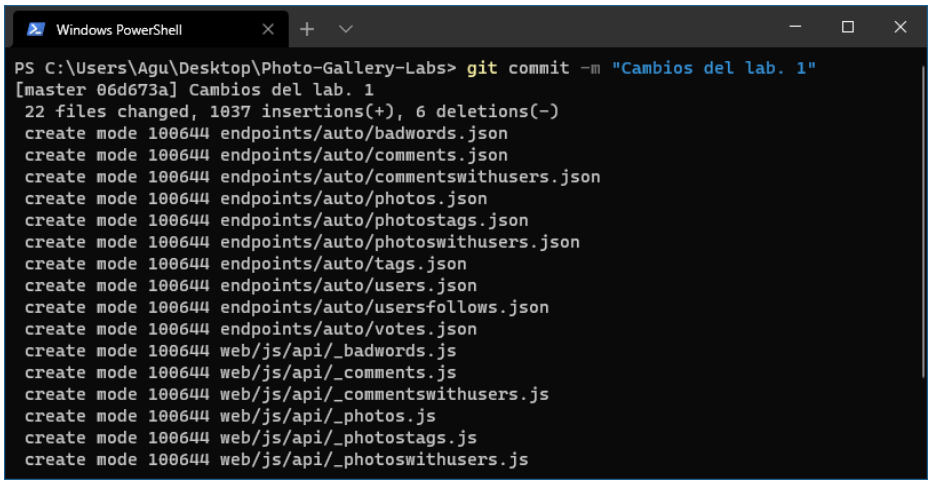
Durante el trascurso de las sesiones de laboratorio, iremos actualizando nuestro repositorio en GitHub con los cambios que hagamos en la galería fotográfica. A modo de ejemplo, realizaremos un nuevo *commit* con los cambios realizados en este laboratorio:

En primer lugar, añadiremos todos los archivos modificados al commit a realizar mediante el comando `git add .`:



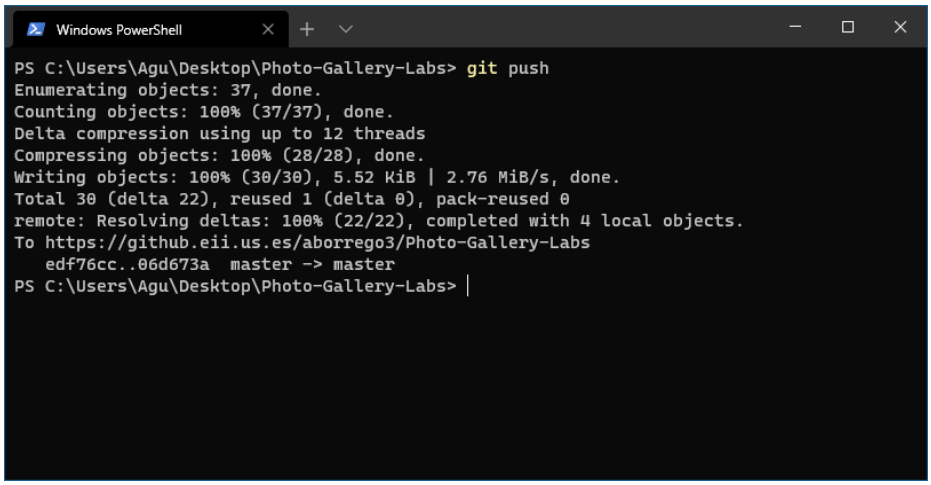
Por defecto, el comando `git add` no muestra ninguna salida si su ejecución es correcta. Si, en lugar de añadir todos los archivos modificados al commit, se desearan añadir sólo algunos en particular, se podrían especificar en el comando anterior, en lugar de usar el punto para incluirlos todos.

A continuación haremos un *commit*, con un mensaje descriptivo:

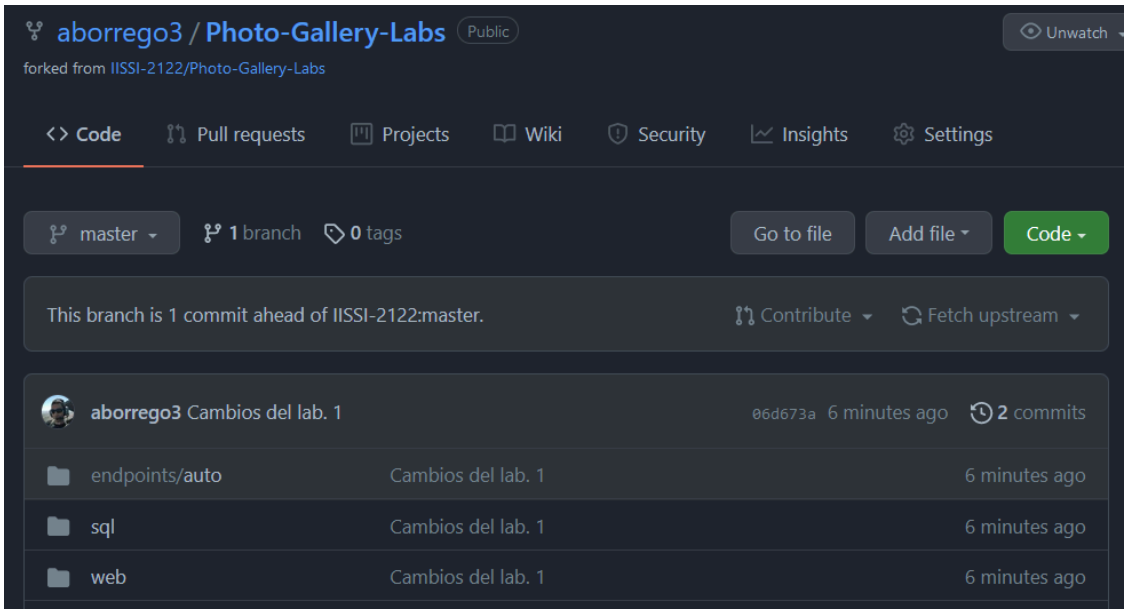


Note que la gran cantidad de archivos añadidos se deben a la ejecución del comando `silence createapi`, que creó una serie de archivos autogenerados.

Finalmente, subiremos los cambios a GitHub con `git push`:



Podremos ver que nuestro repositorio en GitHub ahora contiene los cambios más recientes:



8

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