Desarrollo de Aplicaciones para Internet
Práctica 10

David Cabezas Berrido

Grupo del Martes

dxabezas@correo.ugr.es

27 de enero de 2021

Índice

1. Resumen de la aplicación

Comenzamos proporcionando una breve descripción de la funcionalidad de la aplicación.

Al entrar en `0.0.0.0:5000` somos redirigidos a `/home`, donde recibimos un mensaje de bienvenida. A la izquierda de la barra de navegación tenemos un enlace a `/home` con el nombre de Práctica 10 (que también se utiliza como título de la pestaña) y el logo de Capsule-Corp (que también se utiliza como favicon), a la derecha de este enlace, tenemos otro con el nombre de Pokemon que nos lleva a `/pokemon`, donde podemos visualizar, consultar y editar la base de datos de Pokemon gestionada con *MongoDB*. El siguiente enlace de la barra de navegación tiene el nombre Estadísticas y abre un menú desplegable con enlaces a dos páginas donde visualizamos gráficos con estadísticas sobre los Pokemon de la base de datos. Después hay otro menú desplegable que nos lleva a probar dos mapas interactivos.

En la parte derecha de la barra de navegación tenemos un campo con la fecha y hora actual, que se actualiza cada segundo, y un botón para cambiar la apariencia de la página entre modo clásico y modo nocturno.

Por último, tenemos un footer con el nombre del autor.

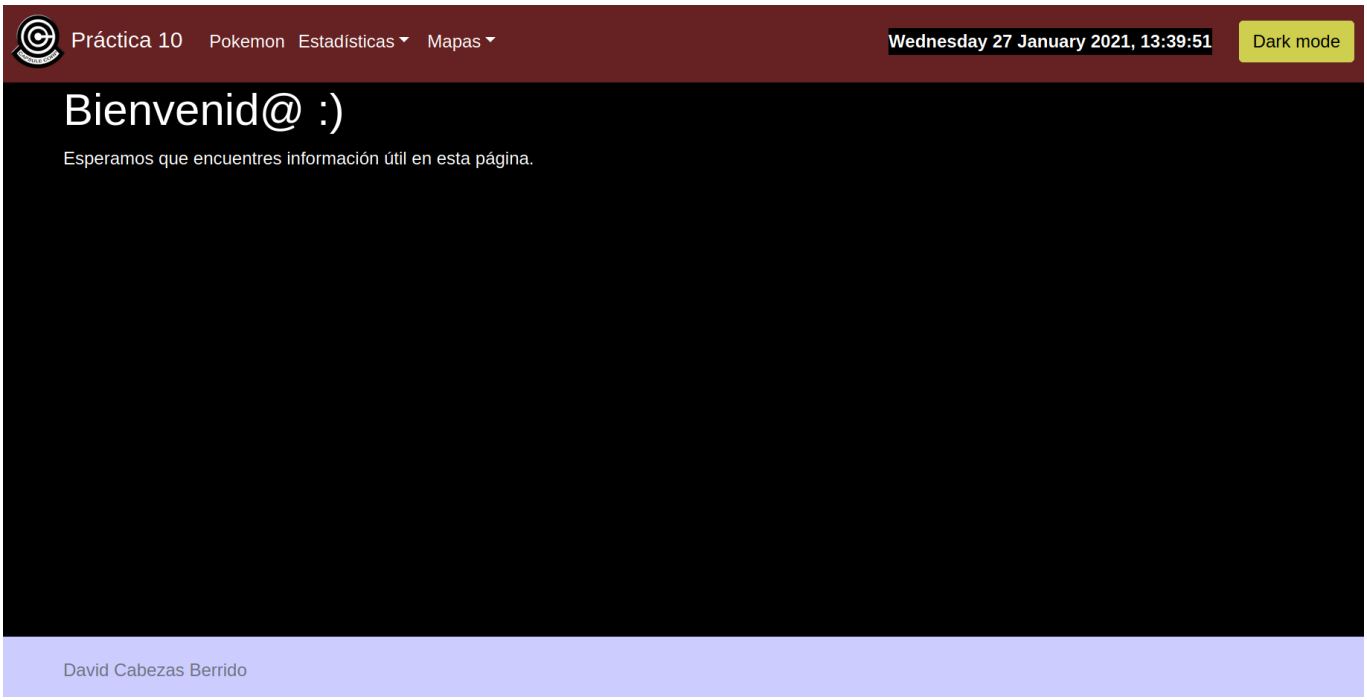

Figura 1: Página de bienvenida (modo nocturno).

En la página `/pokemom`, tenemos formularios para buscar en la base de datos de Pokemon por número (se mostrará el Pokemon correspondiente a dicho número, en caso de que exista), por nombre (se mostrarán los Pokemon cuyo nombre contenga la cadena introducida, sin diferenciar mayúsculas y minúsculas) y por tipo (análogo a nombre, pero la cadena estará contenida en alguno de los tipos de los Pokemon). También hay un formulario para buscar por nombre y tipo a la vez, realizando la intersección de las búsquedas por nombre y por tipo antes descritas. A la derecha, hay un botón que permite añadir un nuevo Pokemon a la base de datos, introuciendo sus datos en un formulario que se proporciona al pulsar el botón.

Debajo, se listan los Pokemon que satisfacen los criterios de búsqueda, cada uno con varios datos y un botón para editar o borrar el Pokemon.

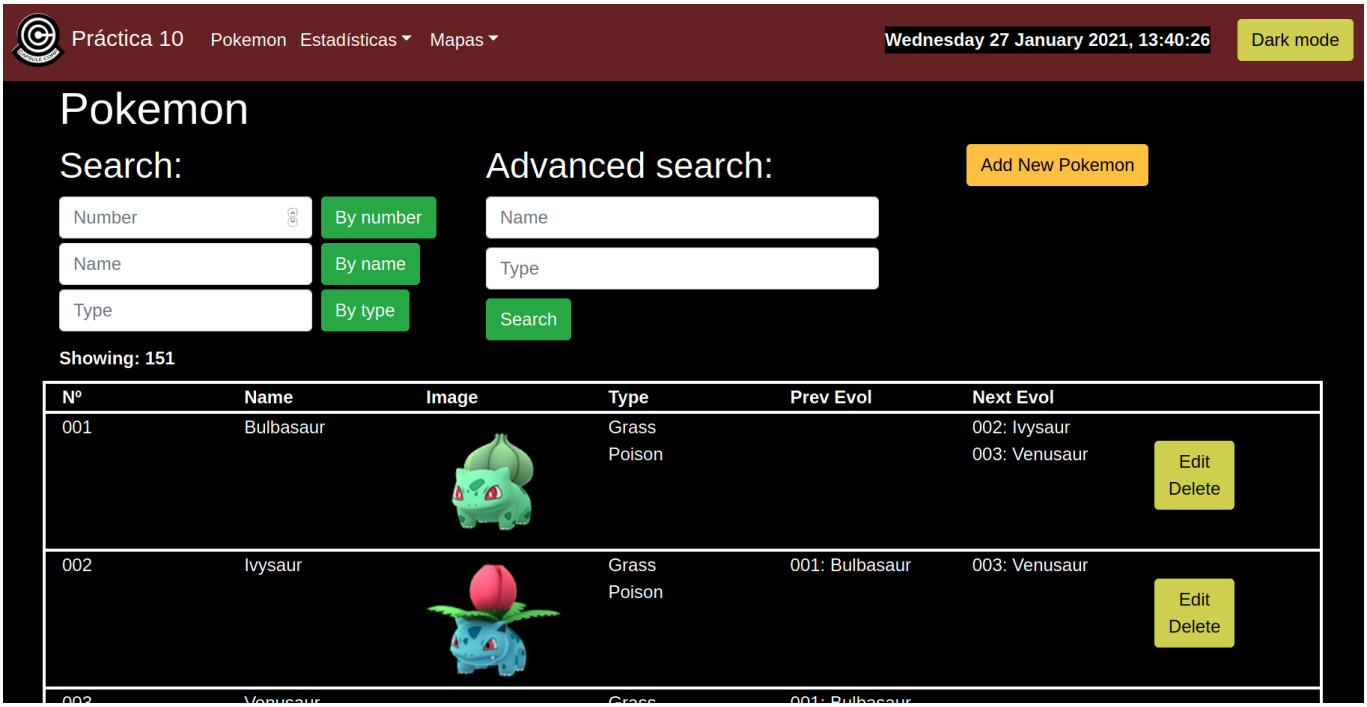

Figura 2: Base de datos de Pokemon (modo nocturno).

2. Funcionalidad front-end

Detallamos cada una de las funcionalidades que hemos implementado con Javascript en el front-end, puesto que es el principal objetivo de la práctica.

2.1. Modo nocturno

Esta funcionalidad ya fue implementada en la Práctica 8, pero la resaltamos porque hemos conseguido resolver un inconveniente que presentaba.

Implementamos esta funcionalidad en el fichero `dark-mode.js` de la carpeta `static`. Tenemos una variable llamada `mode` que utilizamos a modo de flag, tomando los valores `classic` (inicialmente) y `dark`, almacenamos esta variable en `localStorage`. La función `change_mode` se encarga de alternar el valor de esta variable, almacenarlo en el almacenamiento local y llamar a `set_mode`, que lee el valor de `mode` de `localStorage` y cambia el estilo de la página añadiendo o eliminando la clase `dark-mode` a la lista de clases del cuerpo de la página web. En el archivo `static/style.css` nos encargamos de poner el texto de los objetos con esta clase con fondo negro y texto blanco. El script se encarga también de llamar a `set_mode` cada vez que se carga la página para evitar que se restablezca el modo clásico al cambiar de ruta dentro de la página.

El botón para cambiar de modo, simplemente llama a la función `change_mode` en cada click.

En la Práctica 8, este código estaba en el mismo script que el resto del código JavaScript, que se cargaba al final del HTML, por lo que al cambiar de dirección mientras se estaba en modo nocturno, se visualizaba un "flash" blanco y volvía a fondo negro. Esto se debía a que la página cargaba en su configuración por defecto y posteriormente se ejecutaba el script que fijaba el modo nocturno. Para solucionar este problema, cargamos el script al principio de la etiqueta `body`, de tal forma que el cuerpo de la página cargue directamente con el estilo correspondiente.

2.2. Fecha y hora

Añadimos un campo en la barra de navegación que indica la fecha y hora actual. Para ello, añadimos un elemento al HTML con el id `date` e implementamos en el fichero `static/date.js` la función `printTime`. Esta función crea un objeto `Date` (que por defecto se inicializa con la fecha y hora actual), lo formatea como un string de la forma que queremos (utilizamos dos listas constantes para obtener el día de la semana y el mes como string, y fijamos

las horas, minutos y segundos a dos dígitos enteros) y modifica el `innerHTML` del elemento con id `date` para que la muestre.

Con la función `setInterval`, podemos llamar a `printTime` cada segundo para que se actualice la hora.

Esta forma de implementar un campo con fecha y hora la he conseguido modificando un ejemplo que encontré en el curso de JavaScript de *SoloLearn*.

2.3. Gráficos y estadísticas

Utilizando la librería *Highcharts* para mostrar en forma de gráficos algunas estadísticas de la base de datos de Pokemon. Otra alternativa para esta tarea sería la librería *3D.js*, pero tras observar algunas demos en sus respectivas páginas, nos decantamos por *Highcharts*, ya que los códigos para generar gráficos con esta librería son más cortos y me parecen más intuitivos. Además, los gráficos de *Highcharts* son interactivos, mientras que los ejemplos que he visto de gráficos con *3D.js* generaban simples imágenes.

El primer gráfico que generamos es un gráfico de barras que representa el **número de Pokemon de cada tipo** en la base de datos. En primer lugar, buscamos los diferentes tipos en la base de datos con la función `distinct` de Mongo:

```
types=db.samples_pokemon.distinct("type") # Lista de tipos
```

A continuación, combinamos `count` con una sencilla consulta para obtener el número de Pokemon en la base de datos que tienen cada tipo. Hay que tener en cuenta que un Pokemon puede tener uno o dos tipos.

```
for t in types:
    data[t]=db.samples_pokemon.find({"type":t}).count() # Número de Pokemon de cada tipo
```

Finalmente, renderizamos el template `stats-type` y pasamos como parámetros la lista de tipos y una lista con el número de Pokemon de cada tipo. Modificamos el código de este ejemplo para generar el siguiente gráfico de barras:

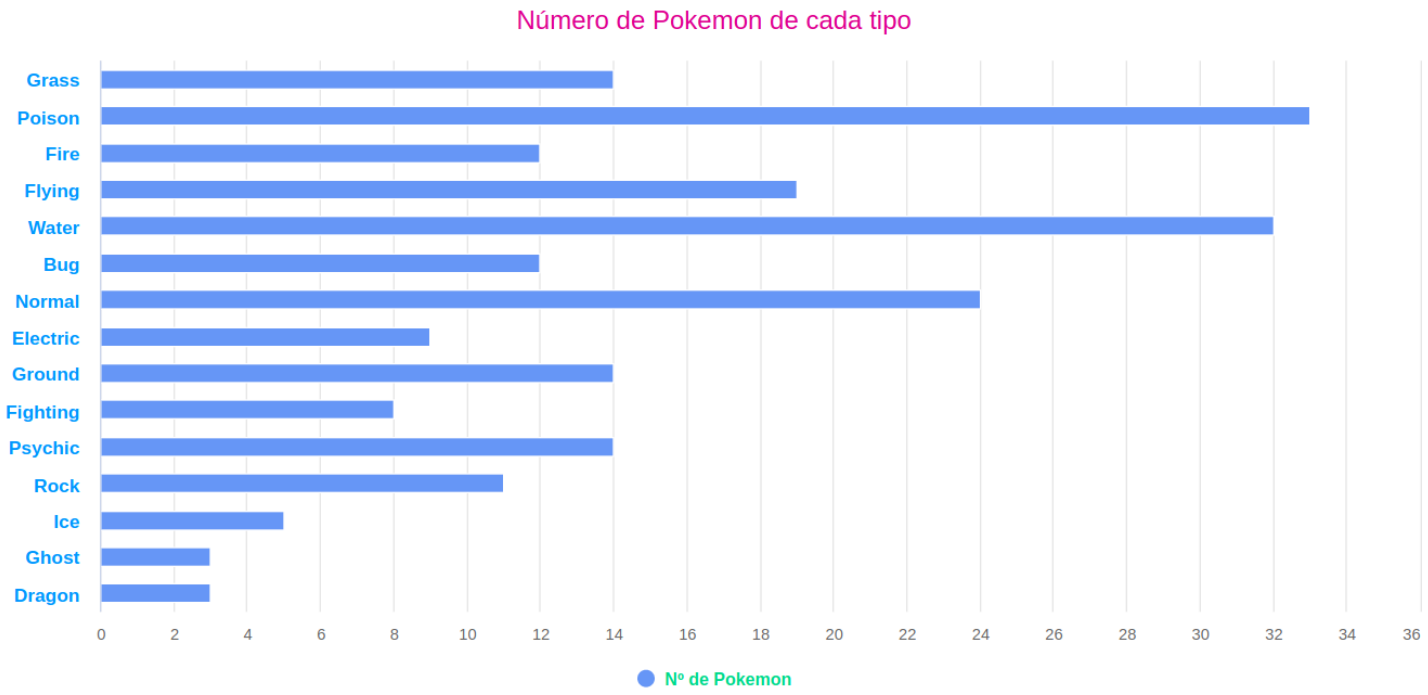

Figura 3: Gráfico con el número de Pokemon de cada tipo.

Aquí hemos tenido que resolver un importante problema: el carácter `'` se renderiza en HTML como `&#39;`. Al utilizar una lista de strings en *Flask* como entrada para `categories` (la lista de tipos), observamos (con inspaccionar elemento) que

```
categories: {{types}}
```

se renderiza como

```
categories: [&#39;Grass&#39;, &#39;Poison&#39;, &#39;Fire&#39;, &#39;Flying&#39;,...
```

y recibimos el error:

```
Uncaught SyntaxError: expected expression, got '&'
```

Para solucionar este problema, añadimos el filtro `safe` de esta forma:

```
categories: {{types|safe}}
```

como sugiere esta página.

El segundo gráfico que generamos es un gráfico de tarta que representa el **número de Pokemon en cada etapa de evolución**, separa los Pokemon en tres categorías, según si han evolucionado ninguna, una o dos veces (algunos Pokemon tienen tres fases de evolución, otros dos y otros sólo una, por lo que no evolucionan). Primero, obtenemos el número de Pokemon en cada una de estas tres categorías, consultando la longitud de la lista en el campo `prev_evolution`,

```
# Han evolucionado una vez
count[1]=db.samples_pokemon.find({"prev_evolution":{ "$exists": True, "$size": 1}}).count()
# Han evolucionado dos veces
count[2]=db.samples_pokemon.find({"prev_evolution":{ "$exists": True, "$size": 2}}).count()
# No han evolucionado nunca: lista de longitud 0 o no definida
count[0]=db.samples_pokemon.find({"prev_evolution":{ "$exists": False}}).count()
count[0]+=db.samples_pokemon.find({"prev_evolution":{ "$exists": True, "$size": 0}}).count()
```

Renderizamos ahora el template `stats-evol` con estos recuentos como parámetro, donde modificamos el código de este ejemplo para generar el siguiente gráfico de tarta:

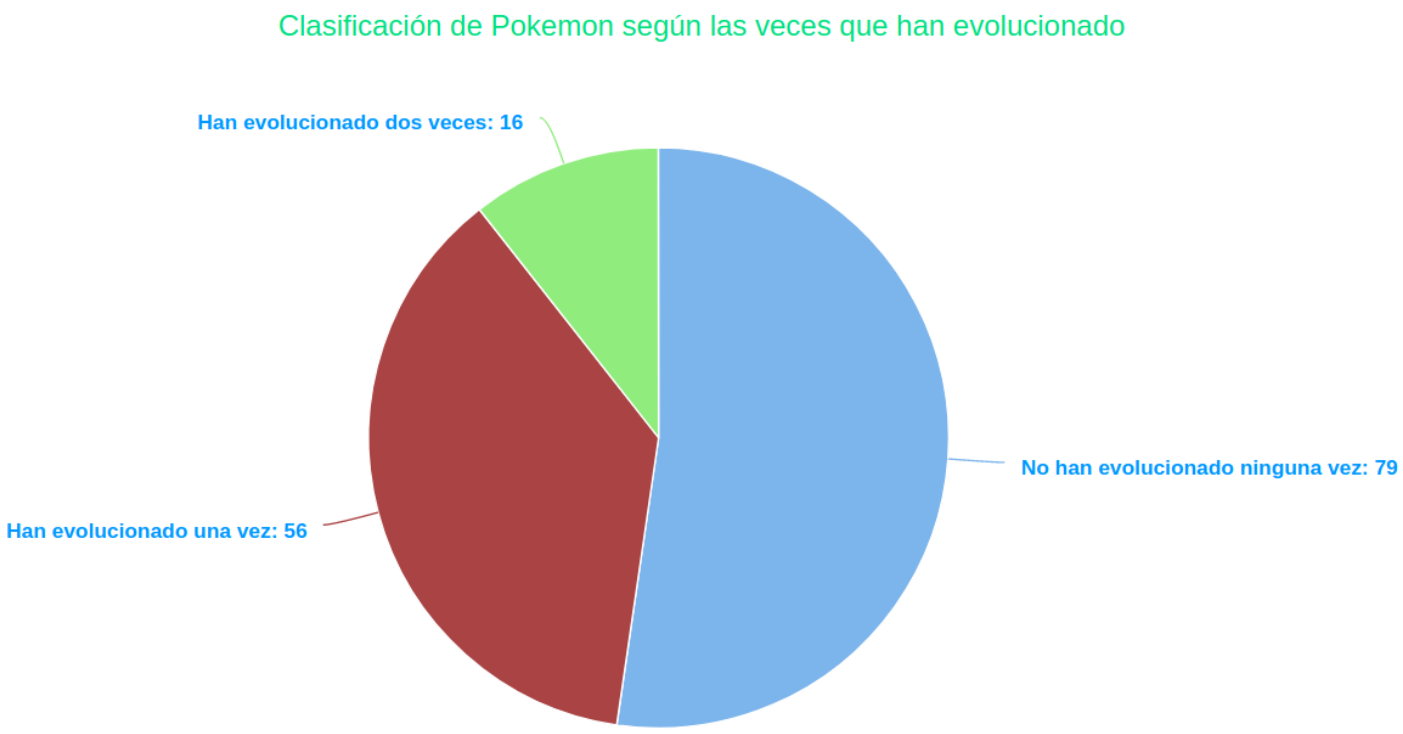

Figura 4: Gráfico con el número de Pokemon en cada etapa de evolución.

Sería apetecible que el estilo de los gráficos se adaptara al de la página (clásico y nocturno), esto presenta varias dificultades. En este post del foro de *Highcharts*, se sugiere que no es posible cambiar las opciones de *Highcharts* de forma dinámica y ofrece dos alternativas: una es destruir y volver a crear el gráfico, y la otra es utilizar `update`

para modificar el estilo del gráfico. Se puede elegir un estilo fácilmente en el momento de generar el gráfico leyendo la variable `mode` de `localStorage`, pero a la hora de cambiar de modo con un gráfico en pantalla, habría que actualizarlo, lo cual es bastante complejo. Al no encontrar ninguna solución satisfactoria, decidimos darle a los gráficos fondo transparante y colores que contrasten bien con ambos fondos (blanco y negro). Para modificar los colores, estilos y fuentes de los gráficos nos basamos en este ejemplo.

2.4. Mapas

Incluimos dos mapas interactivos, lo hacemos con la librería *Leaflet*. Hay otras alternativas, como las APIs de *Google Maps* y de *Google Street View*, pero estas dos no son gratuitas.

El primero de los mapas, en la ruta `map-markers`, permite **añadir y arrastrar marcadores** por el mapa. Lleva una cuenta de los marcadores colocados y al hacer click sobre un marcador despliega un mensaje con el número de marcador.

Este mapa se genera en el template `map-markers.html`, se inicializa en las coordenadas de la *ETSIIT*. Se añade una variable para contabilizar el número de marcadores, con cada click en el mapa (conseguimos esto con un manejador el evento `click`) se genera un nuevo, marcador arrastrable con su número correspondiente, y se acualiza el contador de marcador. También manegando el evento `dragend` de cada marcador, conseguimos que el mapa se recoloque cuando se arrastre el marcador a una posición.

Figura 5: Mapa con marcadores.

El segundo mapa, en la ruta `map-explore`, permite **explorar el mundo** introduciendo las coordenadas del lugar al que se desea viajar o arrastrando el marcador.

Este mapa se genera en el template `map-markers.html`, se inicializa en las coordenadas de la *ETSIIT*. En la parte superior, hay dos campos de entrada con la latitud y la longitud, que admiten flotantes con saltos de 0.0005 unidades (la latitud está limitada entre -85 y 85 grados). Cuenta con un único marcador, con un mensaje que muestra sus coordenadas.

Mientras se arrastra el marcador, se actualizan sus coordenadas tanto en el mensaje que muestra como en el formulario de arriba, esto lo conseguimos manejando el evento `drag`, que se desata repetidamente mientras se arrastra. Igualmente, cuando se termina de desplazar se centra el mapa manejando el evento `dragend`.

Cuando se modifica alguno de los campos de entrada de la parte superior, se desplazan el mapa y el marcador a las coordenadas indicadas. Esto se logra manejando el evento `change` de los campos de entrada.

Finalmente, incluimos dos botones en la parte superior. Cada uno de ellos desplaza el marcador y el mapa a una localización, además de actualizar las coordenadas de los campos de entrada. Uno de estos botones nos lleva a la *ETSIIT* y el otro a la Torre Eiffel.

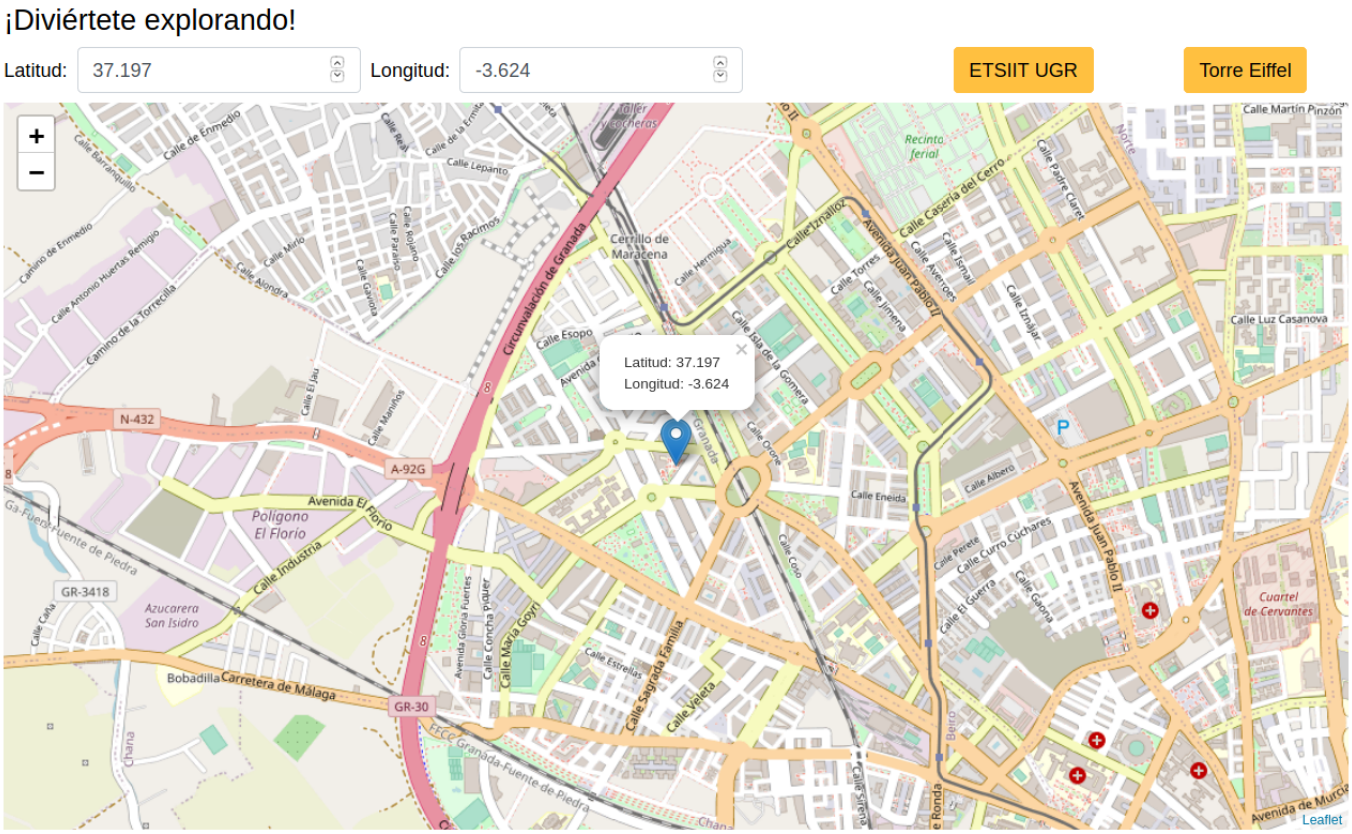

Figura 6: Mapa para explorar.