

Memoria de Prácticas

CityWeb



Realizado por
Alejandro Buitrago López
alejandro.buitragol@um.es
Guillermo Núñez Cano
guillermo.nunezc@um.es

Desarrollo De Aplicaciones Web
Grado en Ingeniería Informática

Dirigido por
Francisco Javier Bermúdez Ruiz

Mayo 2022.

Índice general

1. Introducción	1
2. Descripción de la arquitectura tecnológica	2
2.1. Servicios externos	2
2.1.1. Servicio ciudades	3
2.1.2. Servicio opiniones	5
2.1.3. API Google Maps	6
2.2. Servicios internos	6
2.2.1. Gestión de usuarios	6
2.2.2. Gestión de comercios	6
3. Descripción tecnológica de los aspectos relevantes de la funcionalidad	7
3.1. Gestión de usuarios	7
3.1.1. Registro de usuarios	7
3.1.2. Login de usuarios	8
3.1.3. Logout de usuarios	10
3.2. Gestión de comercios	11
3.2.1. Creación de comercios	12
3.2.2. Edición y borrado de comercios	14
3.2.3. Consulta de comercios	16
4. Conclusiones	19
5. Bibliografía	20

Índice de figuras

2.1. Arquitectura de la aplicación	2
2.2. Puntos de entrada del servicio ciudades	3
2.3. Dialogo para obtener los aparcamientos cercanos a un sitio de interés. .	4
2.4. Aparcamientos cercanos a un sitio de interés.	5
3.1. Registro de usuarios	8
3.2. Login de usuarios	9
3.3. Pagina de inicio	10
3.4. SweetAlert de error de permisos	11
3.5. Página de error de permisos	12
3.6. Ventana para crear un comercio	13
3.7. Ventana para gestionar los comercios	14
3.8. Ventana de confirmación para borrado	15
3.9. Ventana para editar un comercio	16
3.10. Consulta de comercios	17
3.11. Valorar un comercio	18

1. Introducción

En este trabajo se va desarrollar una web para la valoración de la accesibilidad a comercios y plazas de aparcamiento entorno a un sitio de interés de una ciudad, usando Node.js y Express, el lenguaje JavaScript, HTML y CSS lenguajes y tecnologías estudiadas en la asignatura.

En ella los usuarios, podrán registrarse y acceder a la aplicación con dos tipos de roles: un usuario administrador podrá crear,editar y eliminar comercios, mientras que un usuario que no es administrador únicamente podrá realizar consultas sobre los comercios existente. Además los usuarios pueden realizar búsquedas de sitios de interés y aparcamientos por ciudad, localizar las plazas de aparcamiento alrededor del sitio de interés, localizar comercios y, por último, valorar la accesibilidad de aparcamientos y comercios.

Para las consultas de sitios de interés, aparcamientos de una ciudad y plazas de aparcamiento alrededor del sitio de interés se ha hecho uso de las API Rest, desarrolladas en Java y usando el servidor Jetty, realizadas en el marco de la asignatura de Arquitectura Del Software.

El presente documento va a constar de una descripción de la arquitectura tecnológica planteada, una descripción tecnológica de los aspectos relevantes de la funcionalidad y por último se van a exponer las conclusiones tras realizar el proyecto. Además se mostrarán capturas de pantalla para visualizar el resultado final de la aplicación web así como se expondrán decisiones de diseño relevantes, así como los problemas más relevantes que se han presentado en la elaboración del proyecto.

2. Descripción de la arquitectura tecnológica

En la Figura 2.1 se puede apreciar la arquitectura final de la aplicación, se puede observar como es posible realizar *login/logout* de usuarios al igual que registrarse en el sistema, una vez se ha autenticado en el sistema se hace uso de dos API Rest desarrolladas en la asignatura *Arquitectura del software* para las ciudades y opiniones, ambos servicios se encuentran en el directorio */backend* del proyecto y deberán ser lanzados para el correcto funcionamiento de la aplicación. También se hace uso de la API externa de Google Maps, la comunicación es realizada mediante llamadas HTTP. Por último, también se ofrece la posibilidad de realizar operaciones *CRUD* (Crear, editar y borrar) sobre las entidades de comercios. En este capítulo se profundizará sobre los servicios mencionados

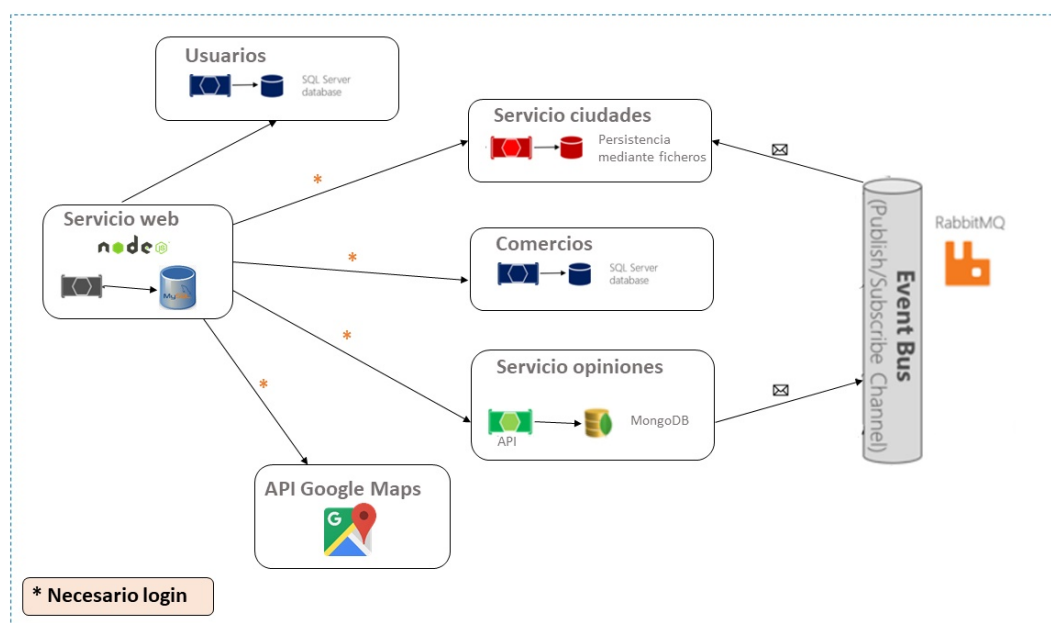


Figura 2.1: Arquitectura de la aplicación

2.1. Servicios externos

En esta sección se va a tratar en profundidad las API proporcionadas por los servicios ciudades, opiniones y la API de Google Maps.

2.1.1. Servicio ciudades

El **servicio REST Ciudades**, ha sido implementado haciendo uso del patrón servicio y repositorio, usando la especificación JAX-RS (Jersey) y desplegado en Jetty, la información será ofrecida en formato XML y JSON, además la persistencia de los datos se gestionará mediante ficheros en formato XML. El puerto en el que está escuchando es el 8082. El servicio, que utiliza el repositorio de ciudades, ofrece la siguiente funcionalidad:

- Obtener las ciudades que pueden ser consultadas.
- Conocer los sitios de interés turístico de una ciudad
- Dado un sitio de interés, obtener plazas de aparcamiento cercanas
- Obtener la información de una plaza de aparcamiento.

Para ello se realizan peticiones HTTP a los siguientes puntos de entrada:

GET	/ciudades	Ciudades disponibles
POST	/ciudades	Crear una ciudad
GET	/ciudades/{id}	Consulta una ciudad
DELETE	/ciudades/{id}	Borrar una ciudad
GET	/ciudades/{id}/aparcamientos	Aparcamientos de una ciudad
GET	/ciudades/{id}/aparcamientos/atom	Aparcamientos de una ciudad en formato ATOM
GET	/ciudades/{id}/aparcamientos/{idAparcamiento}	Consulta una plaza de aparcamiento
GET	/ciudades/{id}/sitiosInteres	Sitios de interes de una ciudad
GET	/ciudades/{id}/sitiosInteres/atom	Sitios de interes de una ciudad en formato ATOM
GET	/ciudades/{id}/sitiosInteres/{idSitio}	Consulta un sitio de interes
GET	/ciudades/{id}/sitiosInteres/{idSitio}/aparcamientos	Plazas de aparcamientos cercanas

Figura 2.2: Puntos de entrada del servicio ciudades

Quedando fuera de nuestro interés las operaciones de creación y borrado de ciudades y las operaciones que retornan ficheros en formato XML, ya que se trabaja con ficheros en formato JSON.

Por otra parte, los puntos de entrada creados en nuestra aplicación para acceder a este servicio son:

- Aparcamientos:
 - `aparcamientos/lorca`: Para acceder a los aparcamientos de Lorca.
 - `aparcamientos/malaga`: Para acceder a los aparcamientos de Málaga.
 - `aparcamientos/vitoria`: Para acceder a los aparcamientos de Vitoria-Gasteiz.
 - `aparcamientos/tenerife`: Para acceder a los aparcamientos de Santa Cruz de Tenerife.
- Sitios de interés:
 - `maps/lorca`: Para acceder a los sitios de interés de Lorca.
 - `maps/malaga`: Para acceder a los aparcamientos de Málaga.
 - `maps/vitoria`: Para acceder a los aparcamientos de Vitoria-Gasteiz.
 - `maps/tenerife`: Para acceder a los aparcamientos de Santa Cruz de Tenerife.

Todos estos puntos de entrada devuelven ficheros en formato JSON que son procesados mediante código JavaScript. Para obtener los aparcamientos cercanos a un sitio de interés en la ventana de consulta de los sitios de interés se ha añadido un botón , como se puede apreciar en la Figura 2.3 que mediante una petición HTTP GET al endpoint `/ciudades/id/sitiosInteres/id/aparcamientos` obtiene un fichero JSON que es procesado para mostrar los aparcamientos con marcadores rojos, mientras que el sitio de interés se muestra en azul, como se puede apreciar en la Figura 2.4.

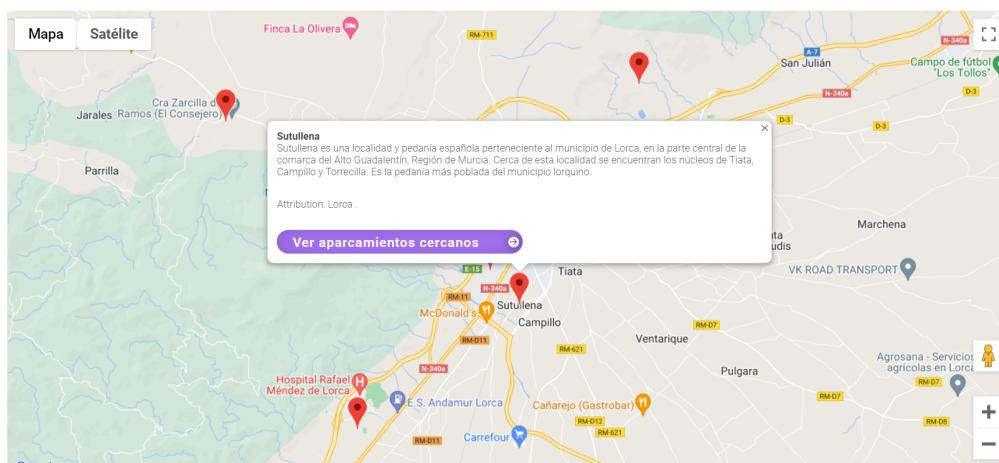


Figura 2.3: Dialogo para obtener los aparcamientos cercanos a un sitio de interés.

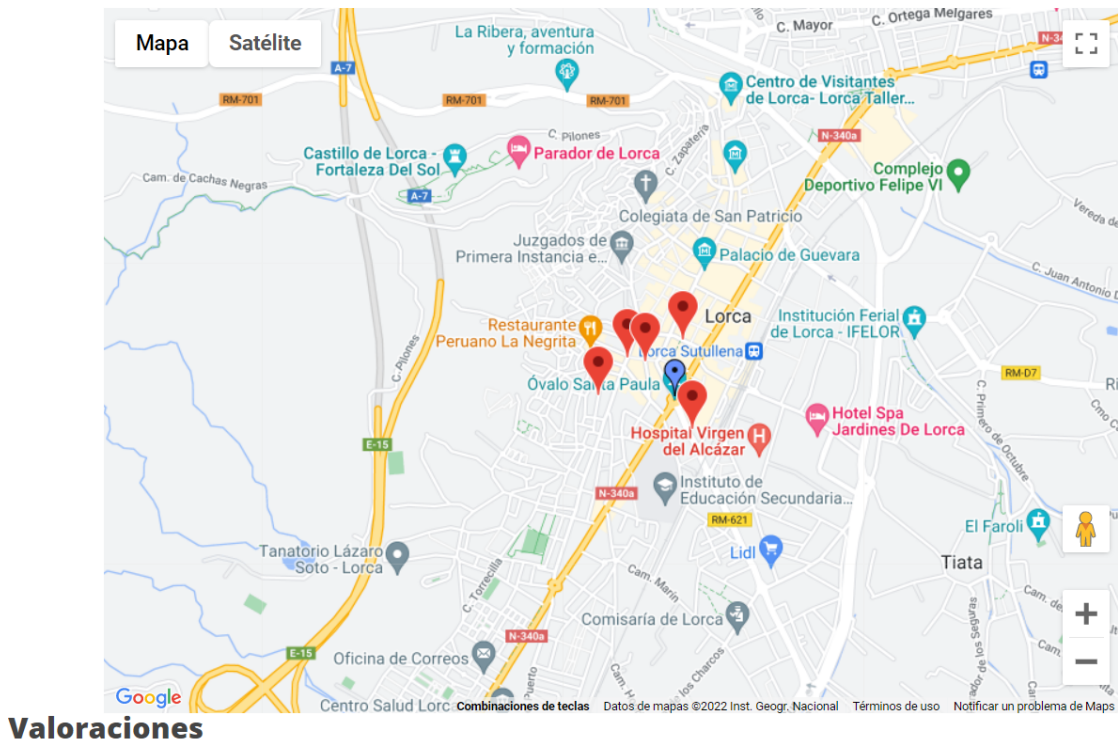


Figura 2.4: Aparcamientos cercanos a un sitio de interés.

2.1.2. Servicio opiniones

El **servicio REST Opiniones**, ha sido implementado utilizando un repositorio y desarrollado en Java con la especificación JAX-RS, la persistencia es gestionada en *MongoDB* gestionado en la nube (Atlas), y el formato de los documentos es JSON. Escuchando en el puerto 8080.

Se almacenan opiniones con la URL del recurso sobre el que se expresa una opinión, lista de valoraciones, número de valoraciones y calificación media. A su vez, una valoración constará de: correo electrónico del usuario, fecha en la que se registra la valoración, calificación de 1 a 5 y un comentario (opcional).

La funcionalidad del servicio es la siguiente:

- Registrar una URL para ser valorada (crear una opinión).
- Añadir una valoración para una URL. Si un usuario registra una segunda valoración para una misma URL, ésta reemplazará a la primera.
- Recuperar la opinión de una URL.
- Obtener un resumen de todas las opiniones guardadas.
- Eliminar una URL del servicio (eliminar una opinión y sus valoraciones).

Para lograrlo se hacen uso de los siguientes puntos de entrada (*endpoints*) API REST:

- GET `api/opiniones/<url>`: Recuperar una opinión de una url en formato JSON
- POST `api/opiniones/valoraciones/url`: Añadir una valoración (transmitida en formato JSON) a la URL especificada en el path.

2.1.3. API Google Maps

Para mostrar un mapa en la aplicación, en todas las páginas se hace una petición HTTP GET a https://maps.googleapis.com/maps/api/js?key=AIzaSyDo6BSiPH1xWfnP_b1L9MbHwBxXsoAhEMI&callback=initMap&v=weekly. También se usa para crear marcadores en los que se pueden visualizar ubicaciones de nuestro interés, indicando la latitud y longitud de la ubicación.

2.2. Servicios internos

Para hacer uso de los servicios mencionados anteriormente y del servicio comercios será necesario loguearse en el sistema.

2.2.1. Gestión de usuarios

Haciendo uso de la base de datos de MySQL se permite la creación de usuarios, con distintos tipos de roles que otorgan más posibilidades a los usuarios, una vez se ha registrado se permite al usuario tanto hacer *login* como *logout*. Se explicará en detalle el modelo de datos usado y la descripción tecnológica de esta funcionalidad en el siguiente capítulo del presente documento.

2.2.2. Gestión de comercios

Haciendo uso de la base de datos de MySQL se permite la creación de comercios, la edición de cualquier campo de estos comercios, así como la consulta de todos los comercios. Se explicará en detalle el modelo de datos usado y la descripción tecnológica de esta funcionalidad en el siguiente capítulo del presente documento.

3. Descripción tecnológica de los aspectos relevantes de la funcionalidad

En este capítulo se describirá en detalle los servicios ofrecidos internamente por la aplicación mencionados en el anterior capítulo, la gestión de comercios y de usuarios, así como se mostrarán capturas mostrando la funcionalidad mencionada.

3.1. Gestión de usuarios

Para la gestión de usuarios gestionada mediante la base de datos MySQL, se ha creado un modelo de usuario ubicado en la carpeta *models* y denominado `users.js` en el que se define que un usuario estará formado por:

- **id**: Identificador único generado por la base de datos.
- **username**: Nombre del usuario.
- **passwd**: Contraseña del usuario.
- **mail**: Correo electrónico del usuario.
- **admin**: Valor booleano que indica si un usuario es administrador o no. En caso de serlo podrá crear, editar y eliminar comercios, en cualquier otro caso únicamente podrá consultar los comercios existentes.

3.1.1. Registro de usuarios

Para realizar el registro de usuarios se ofrece la vista correspondiente a la Figura [3.1](#).

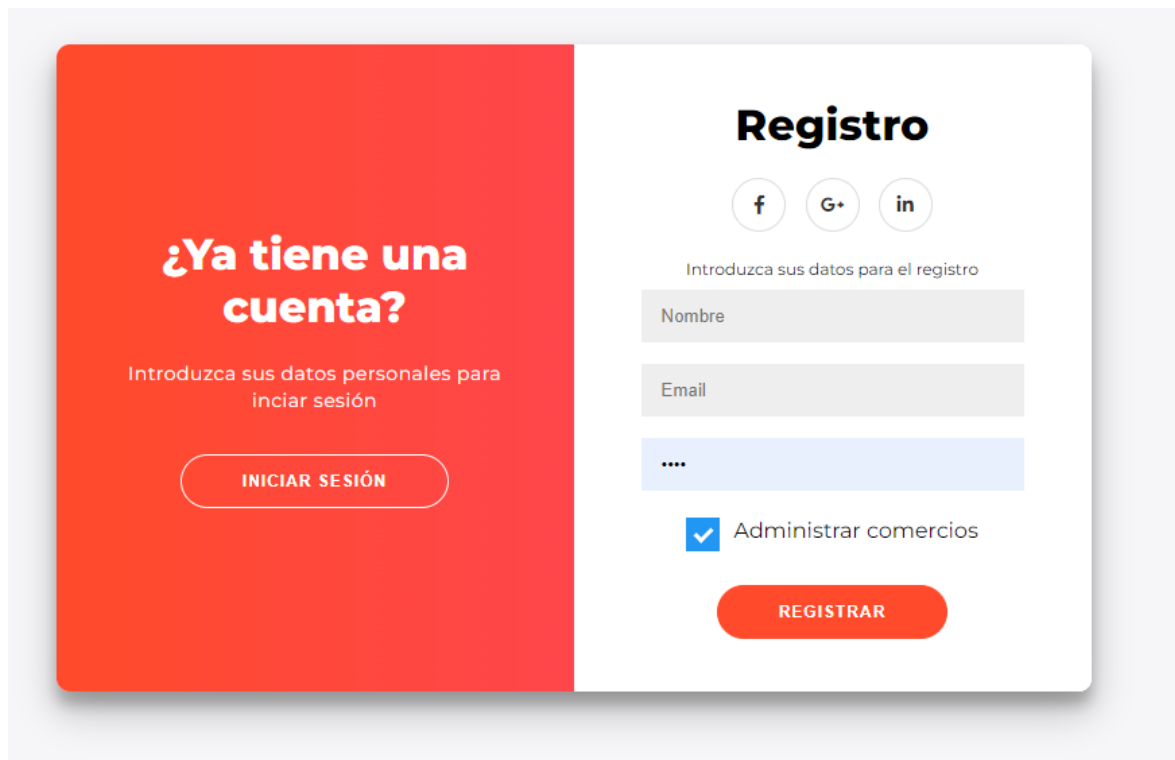
The image shows a user registration form titled 'Registro'. On the left, a red panel asks '¿Ya tiene una cuenta?' and prompts the user to enter personal data to log in, with a 'INICIAR SESIÓN' button. On the right, the registration form includes social media login options (Facebook, Google+, LinkedIn), a prompt to enter registration data, and input fields for 'Nombre', 'Email', and a password (masked with '****'). There is a checkbox for 'Administrar comercios' and a red 'REGISTRAR' button at the bottom.

Figura 3.1: Registro de usuarios

Una vez el usuario ha introducido los datos, se genera un JSON con dichos datos y mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP POST al endpoint `/users/signup` el cual se ha mapeado en el fichero `routes/users.js` que recibe el fichero JSON, extrae sus datos e invoca a la función *create* del controlador creado para los usuarios (`usersController.js`) con los campos que acaba de extraer, el controlador crea un objeto usuario (descrito anteriormente), usado como parámetro en la invocación a la función *add* del repositorio de usuarios (*UsersRepository*) el cual realiza la inserción en base de datos, si es posible, devolviendo *True* en caso afirmativo y *False* en cualquier otro caso.

Nótese que en caso de que exista un usuario con el correo electrónico igual al del usuario que se desea registrar, se mostrará un mensaje de error.

3.1.2. Login de usuarios

Para realizar el login de usuarios se ofrece la vista correspondiente a la Figura 3.2.

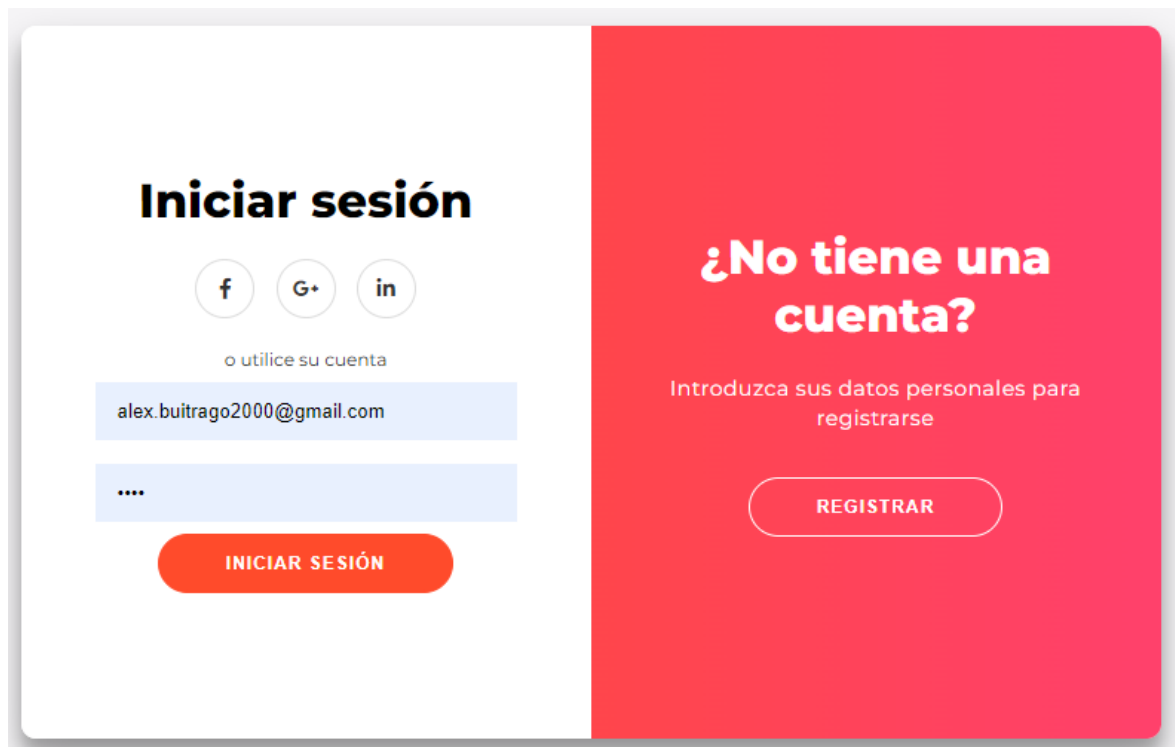


Figura 3.2: Login de usuarios

Una vez el usuario ha introducido los datos, se genera un JSON con dichos datos (correo electrónico y contraseña) y mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP POST al endpoint `/users/singin` el cual se ha mapeado en el fichero `routes/users.js` que recibe el fichero JSON, extrae sus datos e invoca a la función *login* del controlador creado para los usuarios (`usersController.js`) con los campos que acaba de extraer, el controlador llama a la función *login* del repositorio de usuarios (`UsersRepository`) el cual realiza la consulta en base de datos, devolviendo `UNDEFINED` en caso de que no exista ningún usuario con ese correo y contraseña y retornando el usuario coincidente en caso de encontrarlo. En caso de que exista se establece este usuario como usuario actual (*Current User*).

Una vez que el usuario ha accedido al sistema se le ofrece la siguiente vista.



Figura 3.3: Pagina de inicio

Las vistas se ha generado usando el sistema de plantillas [Handlebars](#) (hbs) con HTML5, CSS y Bootstrap.

3.1.3. Logout de usuarios

Para realizar el logout se deberá pulsar en la pestaña *Perfil* del navbar y posteriormente a *cerrar sesión*, redirigiendo al usuario a la vista mostrada en la Figura 3.3.

Una vez el usuario ha pulsado en la opción de *cerrar sesión*, se realiza una petición HTTP GET al endpoint `/users/logout` el cual se ha mapeado en el fichero `routes/users.js` que invoca a la función `logout` del controlador creado para los usuarios (`usersController.js`) el cual establecerá usuario actual (*Current User*) a `UNDEFINED`, posteriormente se redirige a `signin/out` para posteriormente redirigir al usuario a la vista para iniciar sesión.

3.2. Gestión de comercios

Para la gestión de comercios gestionada mediante la base de datos MySQL, se ha creado un modelo de comercio ubicado en la carpeta *models* y denominado `comercios.js` en el que se define que un comercio estará formado por:

- **nombre:** Nombre del comercio.
- **latitud:** Latitud del comercio.
- **longitud:** Longitud del comercio.
- **descripcion:** Descripción asociada al comercio.
- **tipo:** Tipo de comercio.

Es importante destacar, que tal como se ha mencionado anteriormente la creación, eliminación y edición de comercios solo está habilitada para los usuarios administradores en caso de no ser administrador se mostrará un mensaje de error usando las alertas personalizadas proporcionadas por *SweetAlerts* como se puede apreciar en la Figura 3.4 y se redirigirá al usuario a una vista en la que se muestre el error como se puede apreciar en la Figura 3.5.



Figura 3.4: SweetAlert de error de permisos

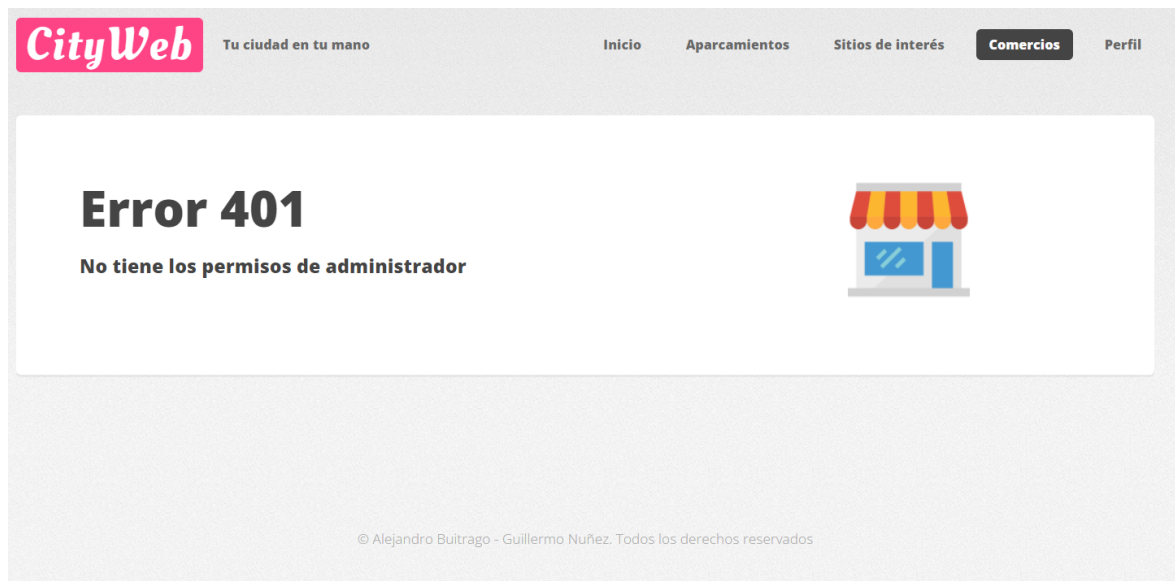


Figura 3.5: Página de error de permisos

Esto se logra, asignando al evento de carga de la ventana (*window.onload*) una función JavaScript que *fetch* se realiza una petición HTTP POST al endpoint */comercios/check* el cual se ha mapeado en el fichero *routes/comercios.js* que recibe un fichero JSON con el mail del usuario (este se guarda en el *localStorage* al iniciar sesión y es recuperado en este punto), extrae sus datos e invoca a la función *isAdmin* del controlador creado para los usuarios (*usersController.js*) con los campos que acaba de extraer, el controlador llama a la función *isAdmin* del repositorio de usuarios (*UsersRepository*) el cual realiza la consulta en base de datos, devolviendo el campo **admin** del usuario coincidente con el mail usado en la consulta.


3.2.1. Creación de comercios

Para realizar la creación de comercios se ofrece la vista correspondiente a la Figura [3.6](#).

CityWeb Tu ciudad en tu mano Inicio Aparcamientos Sitios de interés **Comercios** Perfil

Comercios

Introduzca los datos de su comercio



Nombre:


Descripción:

Tipo:

Latitud:

Longitud:

Mapa Satélite



Registrar Comercio

Figura 3.6: Ventana para crear un comercio

Se ofrece la posibilidad de introducir la latitud y longitud manualmente o arrastrando el marcador en el mapa. Esto se ha logrado, añadiendo al mapa un *listener* pasando como parámetros: el marcador que se visualiza en el mapa, la propiedad *dragend* para lograr el efecto de arrastrado y una función en la que se obtiene la latitud

y longitud del marcador una vez es arrastrado y asigna esos valores a los *input text* correspondientes.

Una vez el usuario ha introducido los datos, y pulsado el botón *Registrar comercio* se genera un JSON con dichos datos (nombre, descripción, tipo, latitud y longitud) y mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP POST al endpoint */comercios/add* el cual se ha mapeado en el fichero *routes/comercios.js* que recibe el fichero JSON, extrae sus datos e invoca a la función *createComercio* del controlador creado para los comercios (*comerciosController.js*), el controlador crea un objeto comercio (descrito anteriormente), usado como parámetro en la invocación a la función *add* del repositorio de comercios (*ComerciosRepository*) el cual realiza la inserción en base de datos. Una vez finalizado se mostrará una alerta indicando si se ha creado o no el comercio.

3.2.2. Edición y borrado de comercios

Para realizar la edición o borrado de comercios se ofrece la vista correspondiente a la Figura 3.7.

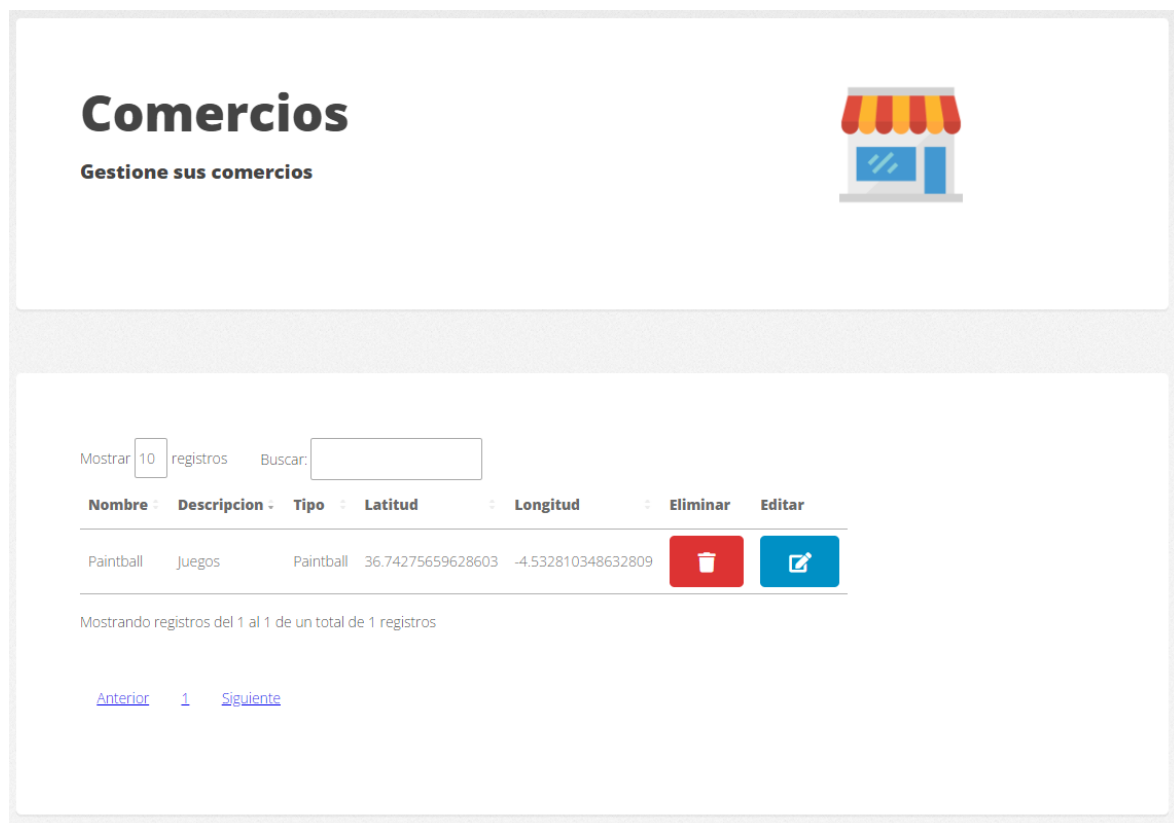


Figura 3.7: Ventana para gestionar los comercios

Para recibir y mostrar los datos, al acceder a esta vista mediante la petición HTTP GET al punto de entrada */comercios/list*, dicho punto de entrada ha sido mapeado en el fichero *routes/comercios.js* que realizará una petición al controlador creado

para los comercios (`comerciosController.js`) para obtener todos los comercios existentes mediante la función `getAllComercios`, el controlador invoca a la función `getAllComercios` del repositorio de comercios (`ComerciosRepository`) y retorna todos los comercios existentes. Para poder usarlos en la vista, al renderizarla se le pasa como parámetro la lista de comercios obtenida.

Para la tabla se ha usado un tipo de tablas de JQuery denominadas [Datatables](#) que permite la búsqueda de registros así como ordenar los registros existentes en orden natural o inverso usando sus columnas, pagina los registros, permite una navegación sencilla entre ellos y muestra el número total de ellos.

Cuando se pulsa ¹ en eliminar un comercio, se muestra una ventana de confirmación como la mostrada en la Figura 3.8, implementada con `SweetAlerts` en caso de pulsar sobre cancelar no se efectuará ningún cambio. Al pulsar sobre eliminar, mediante código *JavaScript* haciendo uso de la función `fetch` se realiza una petición HTTP POST al endpoint `/comercios/delete` generando y enviado un JSON con el nombre del comercio (nótese que no podrán existir dos comercios con el mismo nombre), dicho punto de entrada ha sido mapeado en el fichero `routes/comercios.js` que recibe el fichero JSON, extrae sus datos e invoca a la función `deleteComercio` del controlador creado para los comercios (`comerciosController.js`), el controlador invoca a la función `deleteComercio` del repositorio de comercios (`ComerciosRepository`) pasando como parámetro el nombre del comercio y borrará de base de datos el comercio que contenga dicho nombre, posteriormente se recargará la página y se podrán visualizar los cambios.

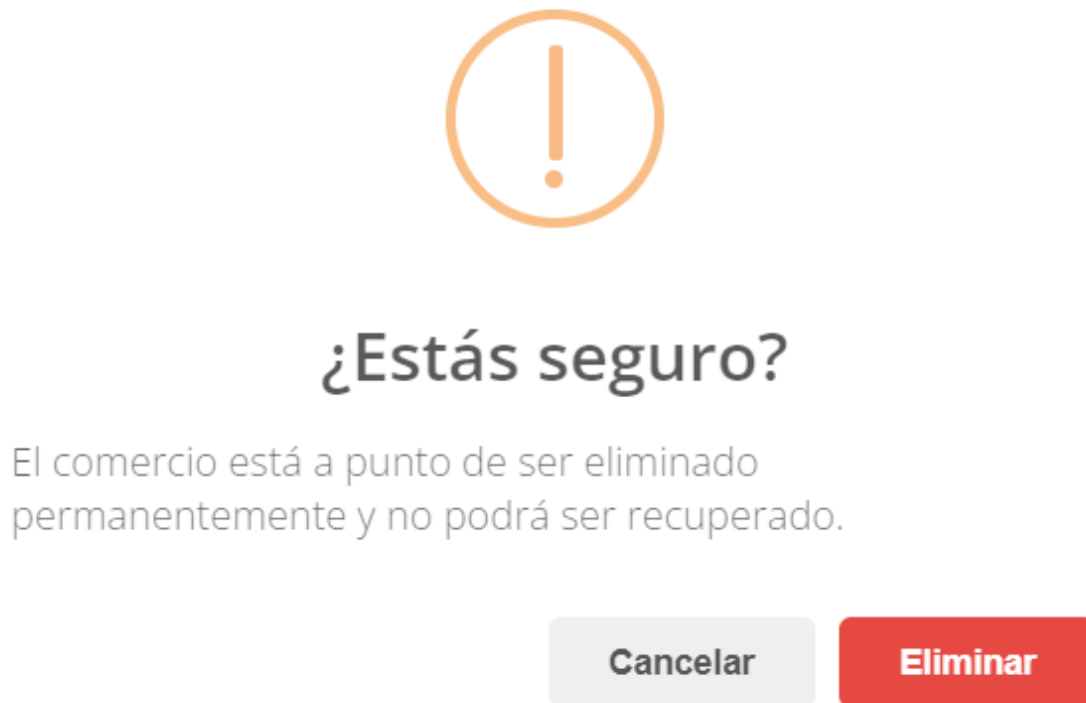
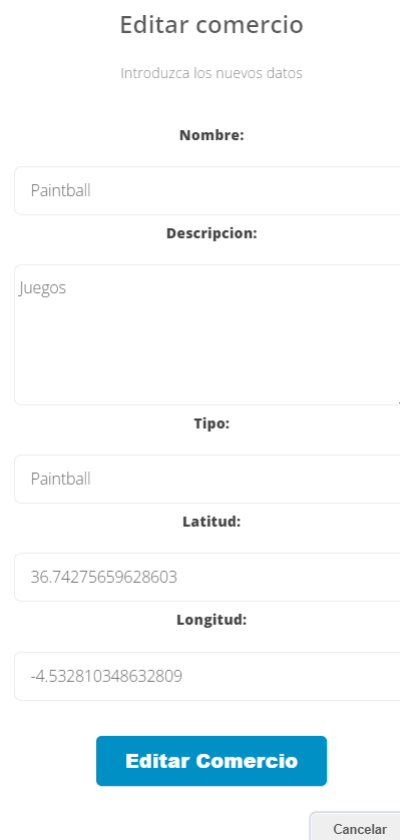


Figura 3.8: Ventana de confirmación para borrado

¹Los iconos han sido obtenidos de <https://fontawesome.com/>

Cuando se pulsa sobre editar, se muestra un formulario en forma de *pop-up* como se puede apreciar en la Figura 3.9 implementado usando SweetAlerts, como se puede observar los valores de los *input* son los actuales del comercio sobre el que se ha pulsado el botón de editar esto se ha logrado usando JQuery. Si se pulsa sobre *cancelar* no se efectuará ningún cambio, en caso de editar algún campo y pulsar sobre *Editar Comercio*, mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP POST al endpoint */comercios/change* se genera y envía un JSON con los nuevos datos (nombre, descripción, tipo, latitud y longitud), al haber mapeado dicho punto de entrada en el fichero *routes/comercios.js* que recibe el fichero JSON, extrae sus datos e invoca a la función *updateComercio* del controlador creado para los comercios (*comerciosController.js*), el controlador crea un objeto comercio (descrito anteriormente), usado como parámetro en la invocación a la función *update* además del nombre del comercio al instante de mostrarse la ventana de edición, esto es, sin modificar, del repositorio de comercios (*ComerciosRepository*) el cual realiza la actualización en base de datos. Una vez finalizado se mostrará una alerta indicando si se han actualizado los valores y se recargará la página.



El formulario, titulado "Editar comercio", solicita introducir nuevos datos. Incluye campos para el nombre ("Paintball"), una descripción ("Juegos" en un área de texto), el tipo ("Paintball"), la latitud ("36.74275659628603") y la longitud ("-4.532810348632809"). Al final, hay dos botones: "Editar Comercio" (azul) y "Cancelar" (gris).

Figura 3.9: Ventana para editar un comercio

3.2.3. Consulta de comercios

Para realizar la consulta de comercios se ofrece la vista correspondiente a la Figura 3.10 la cual es muy similar a las vistas de *aparcamientos* y *sitios de interés*. El

mapa se muestra haciendo uso de la API de Google Maps y se muestra un diálogo mostrando los datos del comercio al pulsar en el marcador situado en la ubicación del comercio.

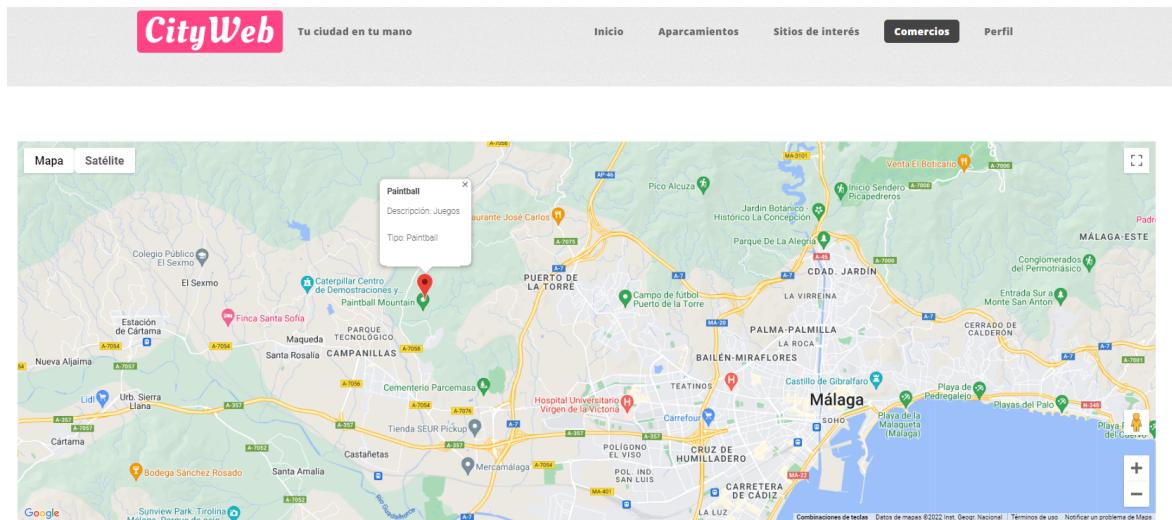


Figura 3.10: Consulta de comercios

Para ello mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP GET al endpoint `/comercios/all` al iniciar el mapa, al haber mapeado dicho punto de entrada en el fichero `routes/comercios.js` que invoca la función al controlador creado para los comercios (`comerciosController.js`) para obtener todos los comercios existentes mediante la función `getAllComercios`, el controlador invoca a la función `getAllComercios` del repositorio de comercios (`ComerciosRepository`) y retorna todos los comercios existentes en formato JSON para extraer sus datos y mostrarlos en el mapa.

Valorar un comercio

En la ventana de consulta (al igual que en aparcamientos y sitios de interés) al pulsar sobre un comercio en la parte inferior podremos realizar una valoración sobre el mismo y consultar las existentes.

Para obtener las valoraciones existentes, mediante código *JavaScript* haciendo uso de la función *fetch* se realiza una petición HTTP GET al endpoint `http://localhost:8080/api/opiniones?url=url` obteniendo todas las valoraciones de ese recurso en formato JSON.

Si por el contrario se desea emitir una nueva valoración, se deberán introducir en el apartado habilitado para ello como se puede observar en la Figura 3.11 los detalles y pulsar sobre *Enviar*, entonces se mostrará una ventana de confirmación implementada con SweetAlerts para estar seguros de la valoración que se va a emitir, si se confirma, haciendo uso de la función *fetch* se realiza una petición HTTP POST al endpoint `http://localhost:8080/api/opiniones/valoraciones?url=url` pasando como da-

tos un JSON generado con los datos de la valoración (calificación, comentario, correo y la fecha de valoración).

Valoraciones

Comentario:

Introduzca un comentario sobre este aparcamiento

Calificar:

★★★★★

Enviar

“

*akjsehtdug howh eoif wea efj soi aoih
doifhoiau rjoijasoihf oisahhf ojaoid sfh
8ahe tjolo fhoahdsoi fhoh
aerthoihodhf98 ahofhoa h8rf 8a.*

”

ADMIN@UM.ES

★★★★★

Figura 3.11: Valorar un comercio

4. Conclusiones

Hemos buscado realizar el trabajo de la manera que más nos podría convenir a los dos en función del tiempo disponible por cada alumno, además para realizar el trabajo conjuntamente de manera telemática hemos usado aplicaciones de mensajería como **Telegram**, para hacer videollamadas para ayudarnos a realizar el código o preguntarnos dudas **Discord** y para el control de versiones se ha usado **GitHub**.

En nuestra opinión, el trabajo nos ha ayudado para comprender mejor de manera práctica la funcionalidad y las aplicaciones de lo visto en las clases de teoría, además de aprender como desarrollar una aplicación web de manera completa usando tecnologías como HTML y CSS, y el desarrollo de la arquitectura interna de la aplicación usando JavaScript con NodeJs y Express. Vemos que la realización de este trabajo es sumamente importante para asentar los conocimientos de la asignatura.

5. Bibliografía

- <https://nodejs.dev/learn>
- <https://sweetalert2.github.io/>
- <https://datatables.net/>
- <https://htmlcheatsheet.com/>
- <https://fontawesome.com/>
- <https://developers.google.com/maps/documentation?hl=es-419>
- <https://uiverse.io/>