GESTOR DE PUNTOS DE INTERÉS VIGNEMALE

Proyecto Sistemas y Tecnologías Web

Jorge Sanz Alcaine 680182
Ana Roig Jiménez 686329
Beatriz Pérez Cancer 683546

Índice

1. Resumen del proyecto

El objetivo del proyecto es la creación de una aplicación para la gestión de puntos de interés (POIs) utilizando la tecnología stack MEAN. Los usuarios pueden acceder al sistema mediante su registro, con el cual pueden optar a realizar todas las funcionalidades presentes, o sin necesidad de registrarse, con el cual solo pueden acceder a algunas funcionalidades de obtención de datos.

La interacción del usuario puede producirse a través de un mapa basado en Google Maps en el cual podrá almacenar tanto puntos de interés como rutas entre ellos. Además, el usuario que está registrado puede interactuar con otros usuarios del sistema, pudiendo suscribirse a su perfil, poder ver sus puntos de interés y rutas, replicarlos y compartirlos.

Además, se han realizado las partes opcionales para las cuales se ha documentado el API utilizando Swagger y se ha subido a AWS como se explica en los apartados 6 y 10.

2. Propuestas similares

Esta aplicación web tiene una gran base en la funcionalidad de Google Maps, porque de hecho la utiliza para alcanzar algunos de sus objetivos. Pero además de servir opciones de interacciones con el mapa, proporciona una interfaz sencilla para gestionar usuarios del sistema, pudiendo utilizarse también como una red social. Un usuario puede, entre otras funciones, seguir a otros usuarios para suscribirse a ellos, poder recomendar POIs por email a otras personas y observar estadísticas del sistema en ese momento. Estas funciones la hacen diferente a otras aplicaciones existentes en el mercado actualmente, porque une muchas opciones en una sola aplicación.

3. Arquitectura de alto nivel

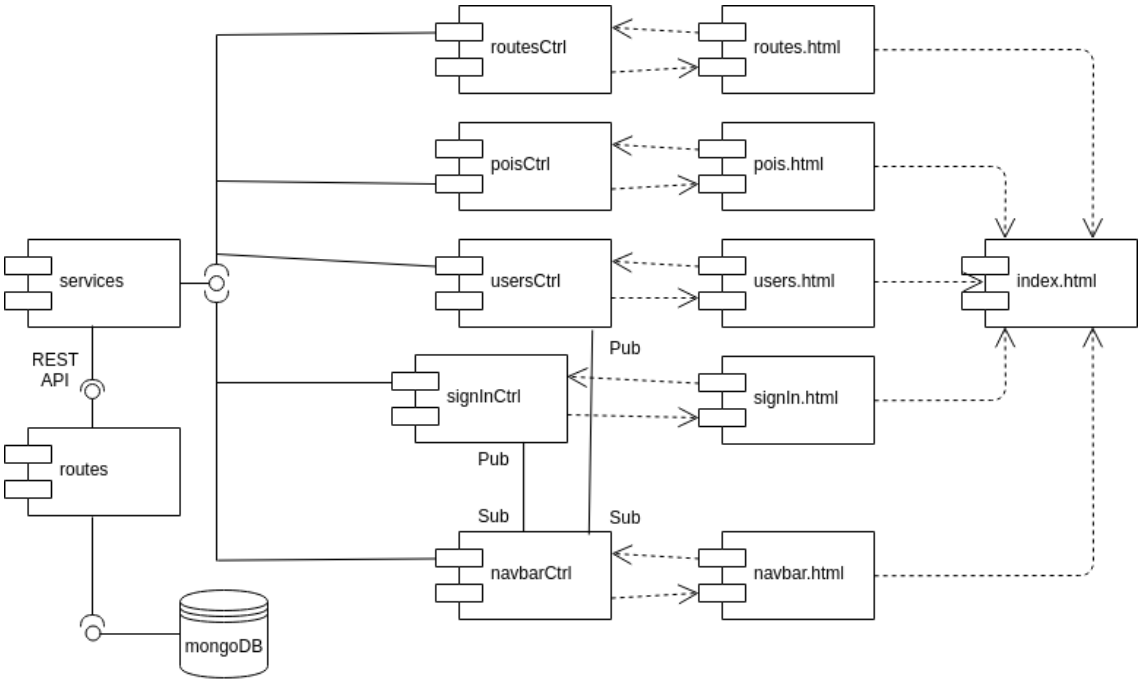

Ilustración 1. Diagrama componente y conector del sistema simplificado.

El componente *routes* que se encuentra en el servidor es el único capaz de interactuar con la base de datos. Además, ofrece una interfaz REST que es utilizada en el lado del cliente por el componente *services*. Los *controladores* utilizan las funciones de *services* para que realicen las peticiones HTTP y devuelvan los datos que necesitan. Cada controlador tiene una vista asociada con la que comparte una serie de variables. Cuando estas variables del controlador cambian, Angular se encarga de propagar los cambios a la vista y viceversa. Las vistas de los controladores se inyectan sobre index.html en función del estado de la aplicación.

La aplicación sigue un patrón cliente-servidor con tres capas modelo, lógica y presentación. La capa de presentación consiste en los ficheros HTML, los controladores de Angular y services para solicitar los datos al servidor. La capa de la lógica de negocio contiene únicamente el fichero routes.js. La capa del modelo consiste en la base de datos de Mongodb, y el fichero users.js para facilitar el acceso a los datos de usuario.

La mayoría de los controladores no están conectados, los únicos que sí lo están son el controlador de usuarios y de inicio de sesión con el de la barra de navegación para modificarla cuando se inicia o se cierra sesión. Estos controladores están conectados mediante el patrón publicación-subscripción en dos canales diferentes.

4. Detalle de los componentes arquitecturales

Para facilitar su legibilidad, el diagrama del apartado anterior no representa todos los componentes del sistema. En este apartado se describirán con detalle todos los componentes del sistema.

Por un lado se encuentra la parte del servidor, la cual contiene el componente *routes* y el almacenamiento de la base de datos. El componente *routes* abarca todo el API REST de la aplicación, por lo tanto se encarga de recibir las peticiones que llegan del cliente (concretamente, del componente *services*), realizar operaciones con la información que incluyen éstas peticiones y devolver una respuesta al cliente.

Todas operaciones llegan y se devuelven en formato json, que es el utilizado constantemente en la comunicación cliente-servidor por ser el formato más ligero y sencillo de mantener. La información que llega al servidor siempre se presenta en forma de un objeto en el cuerpo de la petición donde se almacena la información con la que tratar, y el identificador del elemento (usuario, POI, ruta…) llega en la propia url. En la respuesta al cliente se envía un mensaje de éxito o error, posible información consultada de la base de datos, y también el estado de la respuesta, para que el componente *services* pueda separar respuestas correctas de las incorrectas.

Habitualmente, estas operaciones que realiza el componente *routes* requieren tanto consultar como actualizar información de la base de datos, la cual es el otro componente del lado del servidor. En ella se refleja toda la información almacenada del sistema, y gracias a ella se permite la interacción consistente de elementos entre cliente-servidor. La base de datos se ha gestionado con Mongoose, que permite fácilmente realizar operaciones CRUD sobre la base de datos desarrollada con MongoDB. Se han creado diferentes esquemas según los recursos que se han localizado en la aplicación para dirigir las funciones que corresponden a cada recurso por separado. Estos esquemas pertenecen a usuarios, POIs, rutas, urls acortadas, valoración de POIs, y recomendaciones realizadas.

Además de estos esquemas que permiten relacionar todos los recursos, se han añadido dos índices tanto al esquema de usuarios como al de POIs, para buscar por nombre, apellido y email en el caso de usuarios, y por palabras clave, nombre, ciudad y país en el caso de los POIs. Se ha decidido de esta manera para realizar las búsquedas de una manera más rápida por los campos que se han creído relevantes en el momento de ejecutar búsquedas de ambos elementos.

Por otro lado, se presenta el lado del cliente, el cual está separado en diferentes componentes: plantillas HTML, controladores, y services, cada uno con una función determinada. Para cada estado definido en la aplicación, se determina una url, una plantilla HTML y un controlador asociado.

Las plantillas HTML se encargan de mostrar la vista al usuario en cada momento. Todas plantillas contienen un apartado donde se muestra el éxito o error de las posibles operaciones que se puedan realizar con ellas.

Los controladores se ocupan de producir todo el dinamismo de las plantillas para ofrecer una interacción con feedback al usuario. Permiten realizar un data binding con las variables de las plantillas para aportar esta movilidad, ya que los controladores envían solicitudes al services para que éste remita estas solicitudes al servidor, obtener la respuesta y devolverla a los controladores, y es con esta respuesta con la cual cambian información de la vista que obtiene el usuario. Todos los controladores contienen operaciones para poder enseñar o esconder al usuario diferente información según la situación en la que se encuentra (una operación ha ido bien o mal, está o no loggeado, debe mostrar rutas y no POIs…).

Por último, se encuentra el componente *services*, el cual se encarga de unir la comunicación entre cliente-servidor. Manda las peticiones requeridas por los controladores al API REST del *routes*, recibe la respuesta de éste y realiza la acción que los controladores le han formulado. De esta manera, se actualiza la información en los controladores para poder ofrecer una vista distinta al usuario. El *services* también permite cambiar de estado (por ejemplo, cuando un usuario cierra sesión, cambia al estado inicial de la aplicación para mostrar la pantalla original), y diferenciar entre peticiones en las que todo ha ido bien, y en las que algo ha fallado. Esto es muy importante para realizar unas acciones u otras y cambiar la vista al usuario correctamente para presentarle un feedback adecuado y que el proyecto siga el curso estimado.

El componente index.html también es una plantilla HTML, pero está separada del resto, porque se encarga de incluir todas las dependencias que se utilizan en la aplicación respecto a la parte del cliente, tales como Angular, Bootstrap, librerías de Google Maps… Es la primera página que se carga al acceder a la aplicación, y solamente contiene la barra de navegación, que será constante en todas las demás plantillas. El resto del cuerpo de la página lo van proporcionando las plantillas descritas anteriormente, dependiendo del estado en el que se encuentra.

5. Modelo de datos

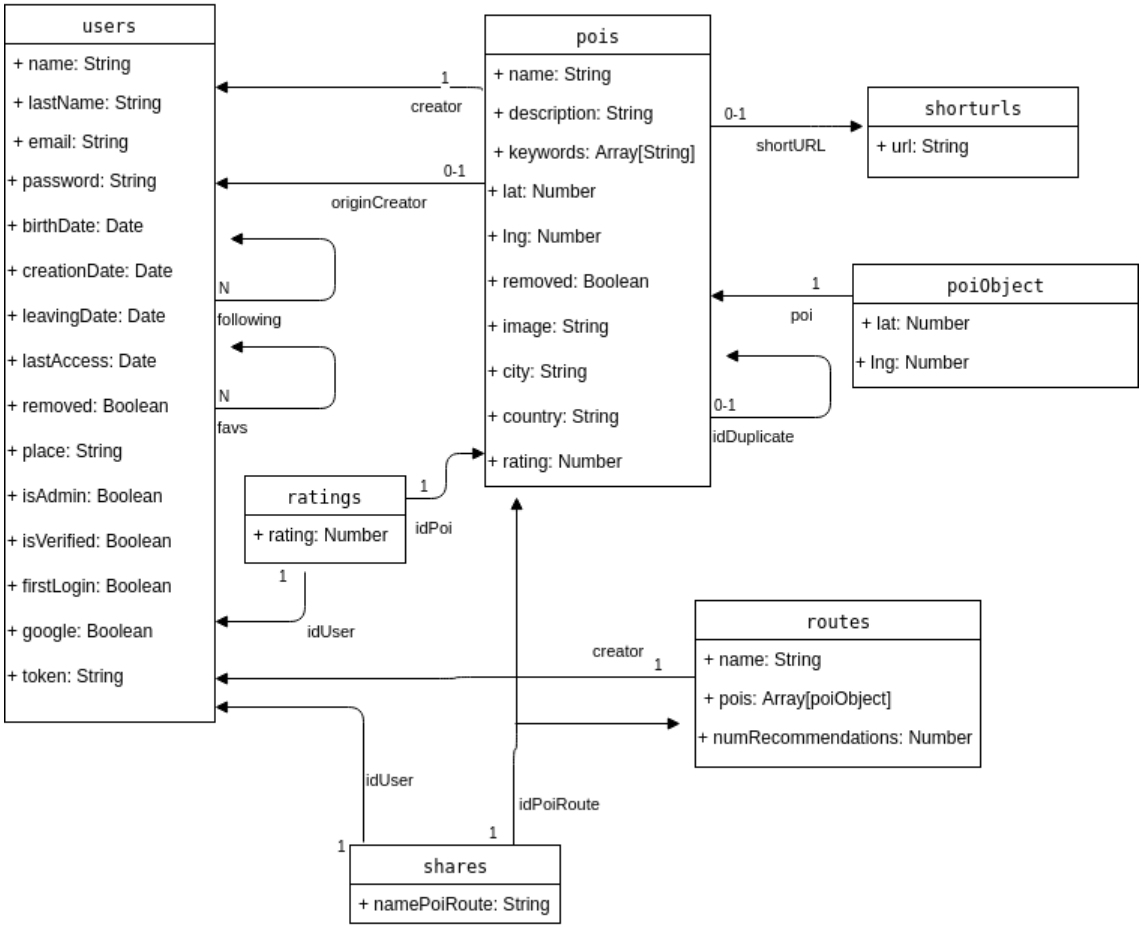

Ilustración 2. Modelo de datos de la aplicación

Para la realización del modelo de datos se ha decidido utilizar Mongoose para seguir un esquema a la hora de realizar las operaciones con los datos de la aplicación, ya que la mayoría de tuplas de cada colección van a tener siempre los mismos atributos. Para este modelo de datos se ha decidido crear las siguientes colecciones:

5.2 Colección de usuarios

Esta colección permite almacenar la información de los usuarios que acceden a la aplicación. En ella se almacenan sus datos básicos y otros que se han considerado de interés para estadísticas:

- **name, lastName**: String → Nombre, apellidos
- **birthDate, place**: Date, String → Fecha de cumpleaños y lugar donde vive para realizar estadísticas según las edades de los usuarios y de donde son. Estos datos no se almacenan si el usuario accede con una cuenta de Google, ya que Google no proporciona esta información sobre los usuarios.
- **email**: String → Cuenta de correo con la que se registra el usuario, la cual debe ser única para que no pueda haber problemas de que varios usuarios se registren con el mismo email y se produzcan conflictos con sus datos de usuario, pois y rutas.
- **password**: String → Contraseña, de la cual solo se almacena el hash para mayor seguridad y que esta no pueda ser descubierta si alguien accede a la base de datos.

- **creationDate, lastAccess**: Date → Fecha de creación y de último acceso para llevar un control de los usuarios y poder realizar estadísticas.
- **removed, leavingDate**: Boolean, Date → Indica si esa cuenta de usuario ha sido borrada, en cuyo caso no se permitirá que este acceda a su cuenta pero no se elimina del todo para mantener información del usuario por aspectos legales, y en caso de que se haya eliminado se almacena la fecha de baja.
- **isAdmin**: Boolean → Indica si el usuario es un usuario administrador o no.
- **isVerified**: Boolean → Indica si el usuario ha verificado su cuenta para no dejar que acceda si no la ha verificado anteriormente.
- **firstLogin**: Boolean → Indica si es la primera vez que el usuario accede a la aplicación para obligarle a cambiar su contraseña.
- **google**: Boolean → Indica si ha entrado con la cuenta de Google.
- **token**: String identificador de dentro del JWT.
- **following**: Array → Contiene los ids usuarios a los que sigue el usuario.
- **favs**: Array → Contiene los ids de los pois que ha guardado como favoritos.

5.3 Colección de POIs

Almacena información sobre los puntos de interés que los usuarios crean en la aplicación y contiene la siguiente información:

- **name**: String → El nombre que el usuario asigna al poi, este no tiene por qué ser único, pueden existir varios puntos de interés creados por diferentes usuarios que tengan el mismo nombre.
- **description**: String → Descripción del punto de interés.
- **keywords**: Array → Palabras clave que se utilizarán para la búsqueda de pois.
- **lat, lng**: Number → Latitud y longitud del punto de interés.
- **shortURL**: String, → Url acortada de la url que introduce el usuario, donde la url original se almacena en otra tabla para guardar la relación entre ambas. Esta url es opcional al crear un poi.
- **image**: String → Imagen que introduce el usuario, opcional.
- **city**, country: String → Ciudad y país donde se encuentra el poi, obtenido de la información que proporciona Google y que se va a utilizar para estadísticas.
- **creator**: objectId → Id del usuario que ha creado el poi.
- **rating**: Number → Valoración media que se va actualizando cada vez que un usuario da una valoración al poi.

5.4 Colección de rutas

Almacena todas las rutas que se crean en la aplicación y tiene los siguientes campos:

- **name**: String → Nombre que el usuario ha asignado a la ruta.
- **pois**: [{poi: objectId, location: {lat: Number, lng:Number}}] → Array que guarda la información en formato JSON del id de los pois que están en la ruta y sus coordenadas.
- **numRecommendations**: Number → Número de recomendaciones para cada ruta para la realización de estadísticas.
- **creator**: objectId → Id del usuario que ha creado la ruta.

5.5 Colección de relación de URLs

Esta colección almacena la relación entre las urls acortadas y las urls que introduce el usuario de tal forma que el id es parte de la url acortada y el atributo url es un string que almacena la url original. Cuando el usuario quiere acceder a la url, se busca la relación entre el id de la url acortada y la url original, y se le redirige.

5.6 Colección de valoraciones

Esta colección guarda las valoraciones para poder llevar el control de que usuarios han valorado y qué pois han valorado para evitar así que un poi pueda ser valorado por el mismo usuario más de una vez y que no se produzcan valoraciones incorrectas. Este esquema almacena:

- **idUser**: String → id del usuario que ha realizado la valoración
- **idPoi**: String → id del poi valorado
- rating: Number → valoración que ha dado al poi correspondiente para la posible realización de estadísticas

5.7 Colección de recomendaciones

Esta permite almacenar las recomendaciones realizadas en el sistema. Esta información se ha guardado para poder realizar estadísticas asociadas a las recomendaciones:

- **namePoiRoute** → Nombre del POI o ruta recomendada.
- **idUser** → Id del usuario que ha hecho la recomendación.
- **idPoiRoute** → Id del POI o la ruta recomendada

Además de las colecciones anteriores, se han añadido dos índices en la base de datos para permitir las búsquedas de usuarios según su nombre de forma más rápida, para lo que se ha creado un índice en el atributo nombre de la colección de usuarios, y se ha creado otro índice sobre los atributos de palabras clave, nombre de los pois, ciudad y país para poder realizar búsquedas rápidas según estos campos cuando el usuario introduce una palabra en el buscador de pois.

6. API REST

El API REST se ha documentado utilizando Swagger, al cual se puede acceder mediante: localhost:8888/api-docs.

A pesar de ello, a continuación se explica la función que realiza cada endpoint.

6.1 Gestión de usuarios:

/users/:id

Get → Devuelve la información del usuario: nombre, apellidos, pois, rutas, siguiendo, favoritos

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/'+id,
 }).success(function (data) {
```

```
            callbackSuccess(data);
        }).error(function (data) {
          callbackError(data);
        });
```

Delete → Elimina el usuario, pero no lo borra de la base de datos, se modifica el atributo removed del usuario para seguir manteniendo su información en la base de datos por cuestiones legales. Además, se realiza el mismo proceso con todos los POIs y rutas que tenía ese usuario, además de sus POIs favoritos y de ser borrado en usuarios en los cuales fuera seguido.

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
            method: 'DELETE',
            url: '/users/'+id,
            headers: {
               'Authorization': auth.getToken()
            }
        }).success(function (data) {
            callbackSuccess(data);
        }).error(function (data) {
            if (data ==="Unauthorized") $state.go('unauthorized');
            else callbackError(data);
        });
```

Put → Modifica la información del usuario, en este caso el usuario solo puede modificar su contraseña, para lo que es necesario que introduzca su contraseña antigua para verificar que es él y que introduzca la nueva contraseña dos veces para evitar errores

Parámetros:

id(requerido)
contraseña antigua(requerido)
contraseña nueva(requerido)
contraseña nueva repetición(requerido)

Ejemplo de uso:

```
$http({
            method: 'PUT',
            url: '/users/'+userObject.id,
            data: $httpParamSerializer(userObject),
            headers: {
               'Content-Type': 'application/x-www-form-urlencoded',
               'Authorization': auth.getToken()
            }
        }).success(function (data) {
            callbackSuccess(data);
        }).error(function (data) {
            if (data ==="Unauthorized") $state.go('unauthorized');
            else callbackError(data);
```

```
});
```

/users/:id/verifyAccount

Get → Al realizar la petición a este recurso se verifica la cuenta del usuario con id, ya que la url solo la puede conocer el propio usuario, porque se le envía al email que ha indicado al registrarse. Para verificar la cuenta, se cambia un atributo isVerified del usuario a true que al principio tenía valor false. Además genera una contraseña alfanumérica aleatoria que se le enviará en un nuevo mail al usuario para proporcionar seguridad.

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/'+id+'/verifyAccount'
  }).success(function (data) {
    callbackSuccess(data);
  }).error(function (data) {
    callbackError(data);
  });
```

/users/:id/password

Put → A este recurso sólo tiene acceso el usuario concreto ya que se le redirige la primera vez que se loguea y además requiere autenticación. Recibe como parámetro la nueva contraseña y la actualiza en la base de datos. En la base de datos se guarda el hash de la contraseña para no almacenarla en texto plano.

Parámetros:

id(requerido)

contraseña(requerido)

Ejemplo de uso:

```
$http({
    method: 'PUT',
    url: '/users/'+user.id+'/password',
    data: $httpParamSerializer(user),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Authorization': auth.getToken()
    }
  }).success(function (data) {
    console.log("Services email " + data.email);
    var userObject = {
      email: data.email,
      password: data.password
    };
    $state.go('signIn', {email: userObject.email}, {password: userObject.password});
    callbackSuccess(userObject);
```

```
}).error(function (data) {
  if (data ==="Unauthorized") $state.go('unauthorized');
  else callbackError(data);
});
```

/resetPassword

post → Permite cambiar la contraseña al usuario en caso de haberla olvidado. Envía un email al correo proporcionado con una nueva contraseña disponible.

Parámetros:

email(requerido)

Ejemplo de uso:

```
$http({
    method: 'POST',
    url: '/resetPassword',
    data: $httpParamSerializer(email),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    }
}).success(function (data) {
    callbackSuccess(data);
}).error(function (data) {
    callbackError(data)
})
```

/users/:id/favs

Get → Obtiene toda la lista de pois favoritos asociada a un usuario.

Parámetros

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/'+ idUser + '/favs'
}).success(function (data) {
    callbackSuccess(data);
}).error(function (data) {
    callbackError(data);
});
```

Post → Añade un POI que recibe en el cuerpo de la petición como favorito a un usuario, esta operación requiere autenticación, solo permite añadir el poi como favorito una vez.

Parámetros:

id(requerido)

idPoi(requerido)

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/users/'+ idUser + '/favs',
        data: $httpParamSerializer(idPoi),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

Delete → elimina un POI que recibe en el cuerpo de la petición de la lista de POIs de un usuario, requiere autenticación.

Parámetros:

id(requerido)

idPoi(requerido)

Ejemplo de uso:

```
$http({
        method: 'DELETE',
        url: '/users/'+ idUser + '/favs',
        data: $httpParamSerializer(idPoi),
        headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

/users/:id/follow

Post → Añade a la lista de seguidos del usuario que realiza la petición, el cual se obtiene del token, el usuario cuyo id se pasa como parámetro en la url, requiere autenticación.

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/users/' + idFollow + '/follow',
```

```
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

/users/:id/unfollow

Post → Elimina de la lista de seguidos del usuario que realiza la petición, el cual se obtiene del token, el usuario cuyo id se pasa como parámetro en la url, requiere autenticación

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/users/' + idUnfollow + '/unfollow',
        //data: $httpParamSerializer(idsUsers),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

/users/:id/routes

Get → Devuelve las rutas dado un usuario

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/users/'+id+'/routes'
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        console.log("error");
      });
```

/users/:id/following

Get → devuelve la lista de usuarios que sigue un usuario

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/'+id+'/following'
}).success(function (data) {
    callbackSuccess(data);  //en data tiene q ir [following]
}).error(function (data) {
    console.log("error");
});
```

/users/:id/isfollowed

Get → Dado un usuario concreto como seguidor y otro como usuario seguido, devuelve en formato json la respuesta que contiene true en caso de que se esté siguiendo al usuario, y false en caso contrario.

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/' + idFollowed + '/isfollowed/',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': auth.getToken()
    }
}).success(function (data) {
    callbackSuccess(data);
}).error(function (data) {
    if (data ==="Unauthorized") $state.go('unauthorized');
    else callbackError(data);
});
```

/users/:id/pois

Get → Devuelve toda la información de los POIs que tiene el usuario.

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/users/'+id+'/pois'
}).success(function (data) {
    callbackSuccess(data);
```

```
}).error(function (data) {
  console.log("error");
});
```

/share

Post → Envía un mail al usuario indicado para recomendar la ruta o el POI. Este es uno de los métodos que acepta como entrada XML.

Parámetros:

recomendación(requerido)

Ejemplo de uso:

```
$http({
    method: 'POST',
    url: '/share',
    data: $httpParamSerializer(recommendation),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    }
  }).success(function (data) {
    callbackSuccess(data);
  }).error(function (data) {
    callbackError(data);
  });
```

Ejemplo de uso XML (postman):

```
<share>
    <email>
        jorge.sanz.alcaine@gmail.com
    </email>
    <idOrigin>
        591c9ba612bd3b1084662d32
    </idOrigin>
    <idPoiRoute>
        591ca18112bd3b1084662d4e
    </idPoiRoute>
    <isPoi>
        true
    </isPoi>
    <message>
        Mira este poi
    </message>
    <userLastNameOrigin>
        Sanz
    </userLastNameOrigin>
    <userNameOrigin>
        Jorge
```

```
        </userNameOrigin>
</share>
```

/search/users/:words

Get → Devuelve una lista con todos los usuarios que contienen las palabras indicadas por el usuario en el nombre del usuario. Los usuarios con el atributo removed a true, no se muestran porque están borrados del sistema.

Parámetros:

palabras a buscar(requerido)

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/search/users/'+words
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        callbackError(data);
      });
```

6.2 Gestión de autenticación:

/signUp

Post → Recibe los parámetros que ha introducido el usuario en el formulario de registro, que son todos obligatorios: nombre, apellidos, email, fecha de cumpleaños y ciudad donde vive. Los campos adicionales a la práctica se han incluido para poder hacer posteriores estadísticas según diferentes datos de los usuarios. Además en este método se actualiza en la base de datos la fecha de creación se indica que el nuevo usuario no ha verificado su cuenta ni ha hecho login nunca. También se comprueba que este mail no existe en la base de datos para evitar repeticiones ya que, a pesar de que se tiene como clave un id que genera mongo, se necesita que el mail no se repita para no enviar emails que van dirigidos a un usuario a varios mails. Este es uno de los métodos que acepta como entrada XML.

Parámetros:

nombre(requerido)
apellido(requerido)
email(requerido)
fecha nacimiento(requerido)
lugar de vivienda(requerido)

Ejemplo de uso;

```
$http({
        method: 'POST',
        url: '/signUp',
        data: $httpParamSerializer(userObject),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      }).success(function (data) {
```

```
            callbackSuccess(data);
          }).error(function (data) {
            callbackError(data);
          });
```

Ejemplo de uso XML (Postman);

```
<user>
        <birthDate>
                12/04/1995
        </birthDate>
        <email>
                pepeSTW2017@gmail.com
        </email>
        <lastName>
                perez
        </lastName>
        <name>
                pepe
        </name>
        <place>
                Zaragoza
        </place>
</user>
```

/signIn

post → Para comprobar que el logging se puede realizar, se comprueba si el usuario y la contraseña existen en la base de datos. Como se almacena el hash de la contraseña, es necesario comparar estos valores. También se realizan las comprobaciones necesarias y no se deja que el usuario acceda a su cuenta si no la ha verificado, si no ha cambiado su contraseña por defecto o si ha cancelado su cuenta. Además si el usuario accede correctamente a su cuenta, se actualiza la fecha de último acceso.

Parámetros:

email(requerido)

contraseña(requerido)

Ejemplo de uso:

```
var that = this;
          $http({
            method: 'POST',
            url: '/signIn',
            data: $httpParamSerializer(userObject),
            headers: {
              'Content-Type': 'application/x-www-form-urlencoded'
            }
          }).success(function (data, status, headers, params) {
            that.authenticate(headers().authorization);
```

```
        callbackSuccess(data);
    }).error(function (data) {
        if(data.message === "You must change your password") {
            var userObject = {
                id: data.id,
                email: data.email
            };
            $state.go('password', {id: userObject.id}, {email: userObject.email});
        }
        callbackError(data);
    });
```

/googleSignIn

Post → Verifica el token recibido y actualiza en la base de datos la información que llega del cliente para vincularlo con un usuario si existe ya en el sistema, y si no lo crea con la información recibida.

Parámetros

token(requerido)

Ejemplo de uso:

```
var that = this;
    $http({
        method: 'POST',
        url: '/googleSignIn',
        data: $httpParamSerializer(token),
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        }
    }).success(function (data, status, headers) {
        that.authenticate(headers().authorization);
        callbackSuccess(data);
    }).error(function (data) {
        callbackError(data);
    });
```

/getIdFomToken

Get → Recupera el id del usuario que está realizando una petición determinada.

Parámetros:

token(requerido)

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/getIdFromToken',
        headers: {
            'Authorization': token
        }
```

```
}).success(function (data) {
   callbackSuccess(data);
});
```

6.3 Gestión de POIs:

/pois

Get → obtiene todos los POIs que se almacenan en la base de datos

Parámetros: ninguno

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/pois/'
    }).success(function (data) {
        callbackSuccess(data);
    }).error(function (data) {
        callbackError(data);
    });
```

Post → añade un POI en la base de datos con la información que recibe en en cuerpo de la peticion. Este es uno de los métodos que acepta como entrada XML.

Parámetros:

- nombre(requerido)
- descripcion(requerido)
- palabras clave(requerido)
- latitud(requerido)
- longitud(requerido)
- valoración(requerido)
- creador
- url
- identificador de POI original si es duplicado
- identificador de creador

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/pois',
        data: $httpParamSerializer(poi),
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
            'Authorization': auth.getToken()
        }
    }).success(function (data) {
        callbackSuccess(data);
    }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
```

```
            else callbackError(data);
          });
```

Ejemplo de uso XML (Postman con autenticación):

```
<poi>
        <creator>
                5922f6fdda056126af141cfb
        </creator>
        <description>
                Soy un poi
        </description>
        <keywords>
                Poi
        </keywords>
        <lat>
                41.649699288606264
        </lat>
        <lng>
                -0.8712673187255859
        </lng>
        <name>
                Poi
        </name>
        <rating>
                3
        </rating>
        <shortURL></shortURL>
</poi>
```

/pois/:id

Get → Obtiene la información de un poi concreto y la devuelve en formato json

Parámetros:

idPoi(requerido)

Ejemplo de uso:

```
$http({
            method: 'GET',
            url: '/pois/' + id
          }).success(function (data) {
            callbackSuccess(data);
          }).error(function (data) {
            callbackError(data);
          });
```

Put → modifica los campos del poi que se le pasan en formato json en el cuerpo de la peticion
Parámetros:

- idPoi(requerido)
- nombre(requerido)

descripcion(requerido)

- palabras clave(requerido)
- latitud(requerido)
- longitud(requerido)
- valoración(requerido)
- creador
- url
- identificador de POI original si es duplicado
- identificador de creador

Ejemplo de uso:

```
$http({
        method: 'PUT',
        url: '/pois/' + newPoi.idPoi,
        data: $httpParamSerializer(newPoi),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

Delete → Elimina el poi de la base de datos, y en caso de que este se encuentre en los favoritos de algún usuario, se elimina también.
Parámetros:

- idPoi(requerido)

Ejemplo de uso:

```
$http({
        method: 'DELETE',
        url: '/pois/'+idPoi,
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
```

```
});
```

/pois/:id/rating

Post → añade la valoración de un usuario a un poi concreto, obteniendo el id del usuario del token, si ya lo ha valorado anteriormente, no deja volver a valorarlo

Parámetros:

idPoi(requerido)

valoración(requerido)

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/pois/'+idPoi+'/rating',
        data: $httpParamSerializer(rating),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

/pois/:id/isfav

Get → dado un poi concreto y un usuario, devuelve en formato json la respuesta que contiene true en caso de que el poi sea favorito para dicho usuario, y false en caso contrario

Parámetros:

idPoi(requerido)

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/pois/' + idPoi + '/isfav',
        headers: {
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

/short/:id

Get → redirige a la url original asociada a la url acortada

Parámetros:

id(requerido)

/search/pois/:words

Get → devuelve una lista con todos los pois que contienen las palabras indicadas por el usuario en el nombre del poi o como palabras clave. Los POIs con atributo removed a true no se muestran, porque están borrados del sistema.

Parámetros:

palabras a buscar(requerido)

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/search/pois/'+words
  }).success(function (data) {
    callbackSuccess(data);
  }).error(function (data) {
    callbackError(data);
  });
```

6.4 Gestión de rutas:

/routes

Get → Devuelve todas las rutas que se encuentran en la base de datos

Parámetros: ninguno

Ejemplo de uso:

```
$http({
    method: 'GET',
    url: '/routes'
  }).success(function (data) {
    callbackSuccess(data);
  }).error(function (data) {
    callbackError(data);
  });
```

Post → Añade una nueva ruta.

Parámetros:

nombre(requerido)

array de pois(requerido)

creador(requerido)

Ejemplo de uso:

```
$http({
    method: 'POST',
    url: '/routes',
    data:route,
    headers: {
```

```
            'Content-Type': 'application/json',
            'Authorization': auth.getToken()
          }
        }).success(function (data) {
          callbackSuccess(data);
        }).error(function (data) {
          if (data ==="Unauthorized") $state.go('unauthorized');
          else callbackError(data);
        });
```

/routes/:id

Get → devuelve la información asociada a una ruta concreta

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
          method: 'GET',
          url: '/routes/' + id
        }).success(function (data) {
          callbackSuccess(data);
        }).error(function (data) {
          callbackError(data);
        });
```

Delete → borra la ruta

Parámetros:

id(requerido)

Ejemplo de uso:

```
$http({
          method: 'DELETE',
          url: '/routes/'+idRoute,
          headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
            'Authorization': auth.getToken()
          }
        }).success(function (data) {
          callbackSuccess(data);
        }).error(function (data) {
          if (data ==="Unauthorized") $state.go('unauthorized');
          else callbackError(data);
        });
```

6.5 Gestión de administrador:

/admin/usersList

get → Devuelve los usuarios que no han sido eliminados para q el administrador pueda realizar operaciones sobre ellos.

Parámetros: ninguno

Ejemplo de uso:

```
$http({
        method: 'GET',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        },
        url: '/admin/usersList'
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
    }
```

/sendMail/:email

Post → el usuario envía un correo al administrador.

Parámetros:

email(requerido)

mensaje texto(requerido)

Ejemplo de uso:

```
$http({
        method: 'POST',
        url: '/sendMail/' + userObject.email,
        data: $httpParamSerializer(userObject),
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      }).error(function (data) {
        if (data ==="Unauthorized") $state.go('unauthorized');
        else callbackError(data);
      });
```

6.6 Gestión de estadísticas

/users/:id/statistics/n

Get → Se obtienen las estadísticas de los usuarios donde n varía de 1 a 10, cada una presenta una gráfica diferente en relación al usuario.

Parámetros:

id usuario(requerido)

id estadística(requerido)

Ejemplo de uso:

```
$http({
        method: 'GET',
        url: '/users/'+idUser+'/statistics/'+id,
        headers: {
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      })
```

/admin/statistics/n

Get → donde n varía de 1 a 10, cada una presenta una gráfica diferente en relación a toda la información del sistema para que el administrador tenga una visión global.

Parámetros:

id estadística(requerido)

Ejemplo de uso:

```
 $http({
        method: 'GET',
        url: '/admin/statistics/'+id,
        headers: {
          'Authorization': auth.getToken()
        }
      }).success(function (data) {
        callbackSuccess(data);
      })
```

7. Analíticas

Para desarrollar las estadísticas del sistema, se ha hecho uso de la librería Chart.js, la cual ha hecho sencillo el uso y ha permitido representar la información que se pretendía con distintos formatos.

Se han realizado 10 estadísticas correspondientes al usuario, en las cuales se centran tanto en la información del propio usuario como en los usuarios a los que sigue. Se ha creído relevante mostrar información de las personas a las que se ha suscrito para obtener una representación visual de los aspectos más llamativos. Además, se han realizado distintos tipos de gráficas, con el objetivo de variar tanto la vista de la presentación de unas a otras como la información a observar:

1. Muestra las fechas de creación del propio usuario y de los usuarios que sigue. Se ha querido representar una línea temporal de evolución en los usuarios al registrarse en el sistema. La gráfica elegida se representa con barras.
2. Muestra las fechas de último acceso del propio usuario y de los usuarios que sigue para poder ver la actividad de los usuarios, si los usuarios a los que siguen se han metido hace poco significa que la aplicación la siguen utilizando. La gráfica elegida se representa con barras.
3. Muestra los POIs que el usuario ha recomendado de los usuarios que sigue. Se ha pretendido representar los POIs más recomendados por un usuario. La gráfica elegida se representa con barras.
4. Muestra el número de recomendaciones de los pois originales que el usuario ha duplicado. Se ha pretendido representar los POIs duplicados en función de sus recomendaciones para ver si los POIs que ha duplicado el usuario son populares. La gráfica elegida se representa con barras.
5. Muestra los POIs que ha duplicado el usuario de los usuarios a los que sigue. Se han querido representar los POIs más deseados por el usuario o que más duplicados tienen gracias a él. La gráfica elegida se representa con barras.
6. Muestra los POIs del usuario divididos por países. Se ha pretendido representar una visión global de los sitios respecto a los diferentes países en los que puede tener interés. La gráfica elegida se representa con círculos.
7. Muestra los POIs de los usuarios que sigue divididos por países. De la misma manera que la estadística anterior, se pretende mostrar una visión global de los países en los que han marcado por lo menos un punto de interés los usuarios a los que se suscribe el usuario. La gráfica elegida se representa con círculos.
8. Muestra los usuarios que sigue el usuario divididos por edades. Se ha querido dar una visión de las edades de los usuarios a los que está suscrito un usuario para poder conocer el rango en el que se encuentran. La gráfica elegida se representa con círculos.
9. Muestra la actividad de los usuarios que sigue el usuario, indicando para cada usuario que sigue en número de POIs que tiene cada uno. Ha resultado interesante presentar de esta manera el dinamismo de los usuarios a los que se está suscrito. La gráfica elegida se representa con barras.
10. Muestra los POIs de los usuarios que sigue el usuario divididos por valoración. Se ha creído relevante exponer una gráfica que represente la valoración de los puntos guardados por los usuarios a los que se está suscrito, porque la valoración es la media de las valoraciones que se le han dado a ese punto. La gráfica elegida se representa con radar.

También se han realizado una serie de estadísticas a las que solo tiene acceso el usuario administrador, ya que engloban información global de todo el sistema, teniendo en cuenta información de todos los usuarios registrados en él:

1. Representa los usuarios por edades dividiéndolos en diferentes rangos. Se ha pretendido dar una visión global de las edades de los usuarios registrados en cada momento en el sistema. La gráfica elegida se representa con doughnut.
2. Representa los usuarios en función de los lugares donde viven. Se ha pretendido mostrar una visión global de los sitios originales de los usuarios del sistema (esta información solamente se ha requerido para estadísticas). La gráfica elegida se representa con doughnut.
3. Representa la relación entre usuarios existentes y usuarios activos del sistema. Se ha pretendido mostrar la correlación entre ambos aspectos de una manera global para observar qué índice de actividad existe en el sistema. Se ha dividido en tres partes: Inactividad total (usuarios que no tienen ningún POI); Actividad media (usuarios que tienen menos de 5 POIs pero al menos 1); y Actividad alta (usuarios que tienen más de POIs). La gráfica elegida se representa con barras horizontales.
4. Representa la relación entre usuarios registrados con Google y usuarios registrados con la cuenta del sistema. Al añadir la opción de registro mediante Google, se vio relevante mostrar una visión de los diferentes registros de los usuarios. La gráfica elegida se representa con círculos.
5. Representa lugares que no se han encontrado en Google Maps. Esto es, el sitio que cada usuario indica como lugar de vivienda al registrarse en el sistema, se comprueba en Google Maps para conocer si tiene información de ese lugar. Como la entrada de texto de esa información es libre, se ha pretendido mostrar qué lugares no se han conseguido encontrar con Google Maps. La gráfica elegida se representa con círculos.
6. Representa la relación entre usuarios y número de recomendaciones de sus POIs. Se ha pretendido dar una visión del número de recomendaciones que ha tenido un usuario mediante sus POIs para poder ver a simple vista la popularidad de cada usuario del sistema. La gráfica elegida se representa con círculos.
7. Representa la relación entre usuarios y número de duplicados de sus POIs. Se ha pretendido dar una visión global del número de duplicados que se han realizado de cada usuario, lo que muestra su divulgación en el sistema por parte de otros usuarios. La gráfica elegida se representa con círculos.

8. Implementación

Para el desarrollo de la aplicación en MEAN, se ha utilizado el patrón MVC (Modelo-Vista-Controlador), en el cual se ha hecho uso del tipo Single-Page Application. De esta manera, se ha dividido el desarrollo en Controllers, que dan dinamismo al frontend y envían la información que indica el usuario al Services. El Services se encarga de enviar esta información al servidor, recoger su respuesta y enviarla de nuevo a los Controllers para estructurar la información de un modo u otro para presentarlo al usuario.

También se ha hecho uso de Directives para controlar la barra de navegación de la aplicación, y para hacer uso del drag-drop en la creación de rutas.

La estructura se ha dividido en un directorio *public*, en el cual se almacena la documentación swagger (directorio api-docs), los ficheros css (directorio css), los controladores, services y directives (directorio js), las librerías utilizadas (directorio lib), y las plantillas html que muestran la información al usuario (directorio templates). Al directorio *public* también pertenece el fichero app.js, en el cual se representan todos los estados de la aplicación, y el fichero index.html, donde únicamente se realiza toda la inclusión de librerías utilizadas y se muestra la barra de navegación.

Por otro lado, está el directorio *server* que contiene toda la información referente al servidor. Guarda la información relacionada con la base de datos de Mongo, así como sus esquemas (directorio models), y el fichero donde se encuentra el API REST para poder responder las peticiones del cliente (directorio routes).

Por último, también se han creado varios ficheros en los cuales se encuentra la configuración de claves y contraseñas del sistema, pertenecientes a la clave de generación de tokens, la contraseña de la base de datos, y la clave del API de Google Maps utilizada.

La clase principal, llamada server.js, se encuentra en la raíz de la aplicación, y en ella se realizan las inclusiones de todas tecnologías utilizadas, además de definir swagger, iniciar passport, y lanzar la aplicación en el puerto 8888. También crea el usuario administrador si aún no ha sido creado anteriormente. Para gestionar el manejo de dependencias se ha utilizado el fichero package.json, también en la raíz de la aplicación, en el cual aparece toda la información referente a sus repositorios, scripts de lanzamiento, repositorios utilizados, autores, y dependencias.

8.1 Servicios

A continuación, se van a explicar en detalle todos los servicios que utiliza el sistema en el fichero service.js

Autenticación

Es el componente necesario para realizar la autenticación segura en el sistema. Este consiste en:

Sign up, donde el usuario se registra en el sistema aportando los campos requeridos, se envían al servidor y éste guarda al usuario en la base de datos para futuras autenticaciones. Si todo va bien, se envía un correo a la cuenta especificada en el registro en el que aparece un link de confirmación de la cuenta. Una vez confirmada, el usuario recibirá un nuevo correo que contendrá su contraseña inicial (generada aleatoriamente en el servidor), con la cual podrá realizar su primera autenticación y que tendrá que cambiar en cuanto realice ese proceso.

Sign in, donde el usuario se loguea en el sistema mediante su email y contraseña y el servidor genera un token asociado a ese usuario. Si es la primera vez que se loguea con la contraseña proporcionada por el sistema, deberá cambiarla por la suya propia, indicando dos veces la nueva contraseña, y pudiendo realizar el log in con ella. Si no es la primera vez, directamente se le muestra su pantalla de Home.

Google sign in, donde el usuario puede realizar log in sin necesidad de utilizar su cuenta del sistema, sino que accede a través de credenciales de Google. Si es la primera vez que accede mediante este procedimiento, se le enviará al usuario un correo con la contraseña aleatoria creada por si quiere utilizar la cuenta del sistema en otra ocasión. Cabe decir que en ambas cuentas el usuario tendrá la misma información almacenada, gracias al vínculo entre el id proporcionado por Google y el id creado al registrarse. La petición enviada al servidor contiene el token facilitado por Google.

Reset password, con el cual el usuario puede utilizar una nueva contraseña para loguearse en el sistema porque ha olvidado/perdido la suya antigua. Si el usuario ya existe en el sistema, el servidor genera una nueva contraseña aleatoria que será enviada por correo al usuario y con la que podrá hacer posteriores log in.

Get id from token, con el cual el servidor devuelve el id del usuario que está realizando una petición determinada a través de la decodificación del token facilitado. Este procedimiento es necesario para conocer si deben aparecer o no ciertas funciones en cada momento. Por ejemplo, si un usuario está viendo su perfil, no puede aparecer el botón de Follow pero sí el de crear POIs o rutas, pero si está viendo el perfil de otro usuario debe aparecer el botón de Follow pero no poder crear elementos. Esto se comprueba mediante esta función.

Log out, con el cual el usuario cierra sesión en el sistema, eliminando el token generado y trasladando la vista a la plantilla Starter, donde aparece el mapa con la posición actual.

Get token, mediante el cual se comprueba que existe un token guardado en el Local Storage del navegador, y si es así se devuelve. Si no existe ninguno, se envía a un estado de Unauthorized para el usuario, porque intenta acceder a un recurso para el cual no está logueado.

Authenticate, con el cual se crea la sesión en el Local Storage del navegador mediante el jwt (json web token) asociado a la sesión del usuario.

Usuarios

Es uno de los componentes más importantes de la aplicación, el que aporta un concepto social al proyecto. Consiste en:

Get user, el cual pide la información de un usuario concreto al servidor, lo busca en la base de datos, y devuelve su información al cliente. Si se intenta la búsqueda de un usuario que ha sido dado de baja en el sistema, se lleva a la pantalla inicial de Starter.

Modify user, el cual recibe del cliente datos del usuario para actualizarlos en la base de datos, y devuelve un mensaje de que todo ha ido correcto, o en caso contrario, se traslada a la pantalla inicial de Starter si no se tiene privilegios de edición. En esta aplicación, el usuario solamente puede cambiar su contraseña, por lo que esta función recibirá la contraseña vieja, y la nueva, para realizar el cambio con seguridad. Además de esta información, también se pasa al servidor el token asociado al usuario.

Delete user, el cual recibe el identificador de usuario desde el cliente y el token asociado, y realiza una actualización en la base de datos del atributo removed, en lugar de borrarlo del sistema, por motivos legales de información. Si no se tiene derecho de borrado, se traslada a la pantalla inicial de Starter.

Verify account, el cual recibe el identificador de usuario y realiza una petición al servidor para generar una contraseña inicial, actualizarla en la base de datos y devolver al usuario un mensaje de que todo ha ido correcto, en cuyo caso mandará un correo a su email con la nueva contraseña. Si ha habido algún error, aparecerá un mensaje de error.

Password, con el cual el usuario puede cambiar la contraseña la primera vez que se loguea en el sistema. Envía al servidor la contraseña nueva, la actualiza en su base de datos y si todo ha ido bien, traslada al usuario a la pantalla de sign in, donde podrá iniciar sesión con la nueva contraseña indicada anteriormente. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Get user pois, el cual recibe el identificador de usuario y busca en la base de datos todos los POIs cuyo creador tiene ese identificador, y los devuelve al cliente. Este proceso es el primero que se realiza siempre que el usuario inicia sesión en la aplicación, para mostrarle inicialmente sus puntos de interés.

Get user routes,el cual recibe el identificador de usuario y busca en la base de datos todos las rutas cuyo creador tiene ese identificador, y las devuelve al cliente.

Get user follows, el cual recibe el identificador de usuario y busca en la base de datos todos los identificadores de usuario a los que sigue ese usuario, busca sus nombres de usuario, y los devuelve al cliente.

Get user favs, el cual recibe el identificador de usuario y busca en la base de datos todos los POIs que son favoritos del usuario, y los devuelve al cliente.

Add fav, el cual recibe el identificador de usuario y la información del POI que desea añadir como favorito, además del token asociado, y en la base de datos se actualiza esa adicción en el usuario especificado. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Delete fav, el cual recibe el identificador de usuario y la información del POI que desea añadir como favorito, además del token asociado, y en la base de datos se actualiza el borrado de ese POI en la lista de favoritos del usuario especificado. Si no se tiene derecho de borrado, se traslada a la pantalla inicial de Starter.

Follow user, el cual recibe el identificador del usuario a seguir, y se actualiza en la base de datos añadiendolo a la lista del atributo following. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Unfollow user, el cual recibe el identificador del usuario a abandonar, y se actualiza en la base de datos quitándolo de la lista del atributo following. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Is followed, con el cual se comprueba si el identificador recibido está en la lista de Siguiendo por el usuario que realiza la petición, además de recibir el token. Este método sirve para comprobar si al ver un perfil de un usuario, se debe poder seguir/abandonar (que aparezca el botón Follow/Unfollow). Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Search, con el cual se reciben palabras proporcionadas por el cliente para buscar usuarios, se buscan coincidencias en la base de datos con respecto al nombre, y se devuelven los resultados que han encajado con la búsqueda.

SendMail, en el cual se recibe el correo del usuario al que enviar un email, y el texto que debe ir en él, además del token. Esta función es llamada por el administrador si quiere enviar algún correo a algún usuario. Realiza el envío del correo al usuario especificado. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Get statistics, en el cual se recibe el identificador de usuario, el identificador de la estadística, y el token asociado. Esta función es llamada para obtener todas las estadísticas del usuario, y recibe del servidor toda la información necesaria para representar cada estadística. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

POIs

Es el componente esencial de la aplicación, el que da nombre al proyecto, y con el que más interactúa el usuario. Contiene:

Create poi, el cual recibe información de un POI indicada por el usuario, además del token, y lo actualiza en la base de datos, indicando que su usuario creador es el que ha realizado la petición. Si todo ha ido bien, devuelve un mensaje de que ha ido correcto, y si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter. La imagen del poi no puede ser demasiado grande (unos 30Kbytes).

Get poi, el cual recibe el identificador de un POI, y busca en la base de datos ese identificador para devolver toda la información asociada a ese POI.

Edit poi, el cual recibe información de un POI actualizada por el usuario en el cliente, además del token asociado, y reemplaza esa información de ese punto en la base de datos. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Delete poi, el cual recibe el identificador del POI a borrar, además del token asociado, actualizando en la base de datos el atributo removed del punto. Igual que en el caso de los usuarios, los POIs no se borran completamente del sistema, sino que se cambia un atributo para indicarlo. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Rate poi, el cual recibe el identificador del POI a valorar y el valor de puntuación dado, además del token. Se actualiza en la base de datos realizando una media entre la valoración actual y la valoración añadiendo la nueva puntuación. Se comprueba aquí que cada usuario solo pueda votar un determinado POI una sola vez. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Is fav, el cual recibe un identificador de POI, además del token asociado, y comprueba en la base de datos si ese determinado punto está como favorito del usuario. Este método sirve para poder mostrar la función de añadir/borrar como favorito un determinado POI en cada momento. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Search, el cual recibe una lista de palabras del cliente, y permite buscar los POIs que coincidan con su nombre, keywords, o país. Estos tres atributos se han creído relevantes en cuanto a una búsqueda de puntos.

Rutas

Este componente permite ofrecer distintas funciones con las rutas entre POIs que tiene el usuario en su sistema. Contiene:

Get route, el cual recibe un identificador de ruta, busca en la base de datos esa ruta y devuelve la información asociada a ella para mostrarsela al cliente.

Create route, el cual recibe los POIs que componen la ruta, y el nombre de ésta, además del token asociado, y se crea un registro en la base de datos indicando que el creador es el usuario que realiza la petición. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Delete route, el cual recibe un identificador de ruta, la busca en la base de datos y borra el registro correspondiente a esa ruta.

Administrador

Este componente permite observar desde una visión global la información del sistema, a través de sus usuarios y sus estadísticas. Contiene:

List users, es la pantalla principal del administrador, en la cual aparece una lista con todos los usuarios del sistema, indicando su nombre, su id, su email, su fecha de creación y su fecha de último acceso. Con cada usuario, el administrador puede o bien mandarle un correo, o bien borrarlo del sistema.

Get admin statistics, en el cual se recibe el identificador de la estadística, y el token asociado. Esta función es llamada para obtener todas las estadísticas del administrador, y recibe del servidor toda la información necesaria para representar cada estadística. Si no se tiene derecho de edición, se traslada a la pantalla inicial de Starter.

Recomendaciones

Este componente es utilizado cuando un usuario, registrado o no en el sistema, decide recomendar un POI o una ruta a alguien, enviando por correo el link del elemento recomendado. Contiene:

Share, el cual recibe datos de si es un POI o una ruta, su identificador, su nombre y el creador de ese elemento, además del texto y la cuenta de correo a enviar. Se envía un mensaje desde el servidor al correo especificado conteniendo el link del POI o ruta recomendado, y se devuelve un mensaje de éxito o de error al cliente.

Google Maps

Este componente es de los más relevantes del sistema, ya que mediante él se realiza la interacción sobre el mapa y la gestión de vistas de los diferentes puntos de interés. Comprende varias funciones que inicializan el mapa del Home de usuario, añaden y borran marcadores a los POIs, traza el camino de las rutas del usuario, y las esconde.

8.2 Controladores

usersController: En este controlador se definen los objetos que contienen los datos del nuevo poi a crear, nueva ruta etc. Gran parte de sus métodos consisten en llamar a un servicio para realizar la petición y en el callback de dicho servicio mostrar los datos devueltos por el servidor.

Al cargarse el controlador, se ejecuta el método getUser que se encarga de inicializar el controlador creando el mapa y mostrando los pois del usuario de la url.

Para controlar qué es lo que se muestra en la vista, se ha definido una variable llamada show. En función del nombre de esa variable se muestran unas cosas u otras.

Como este componente tiene asociado un mapa para las rutas y los pois, es necesario gestionar lo que se muestra en él. Se han definido dos métodos para ocultar pois y rutas. Estos métodos deben utilizarse al modificar la variable show (Al cambiar de vista) si la vista seleccionada no necesita que se muestren.

La creación de rutas mediante drag-and-drop también se realiza desde este componente, por lo que son necesarios métodos para inicializar los contenedores drag y drop. En la inicialización del contenedor drop se ha creado el método "onSort" al que se llama cada vez que se modifica el orden del contenedor (Añadir, quitar o ordenar pois) para que se muestre la ruta actual en el mapa. Es necesario un proceso de traducción entre los pois del drag-and-drop y lo que se necesita para mostrarlos. La traducción la lleva a cabo la función "getWaypts".

El servidor guarda las imágenes como un String, por lo que cuando se crea o edita un poi es necesario extraer la imagen como un String mediante un objeto FileReader.

Cuando un usuario borra su propia cuenta se cierra la sesión y para que la barra de navegación se modifique, se lanza un evento con título "logout" para que el controlador nabvar lo capture.

signInController: Este controlador tiene dos funciones para iniciar sesión normal o con google. Se han definido las variables email y password para usarse entre el formulario y la función de inicio de sesión normales. Al cargar la página, se realiza una llamada a iniciar sesión con google sin haber pulsado ningún botón. Para evitar este comportamiento se ha definido el booleano click. La función para el inicio de sesión con google solo tendrá efecto si la variable click es igual a true, que solo ocurre al pulsarse el botón. El resto de métodos y variables se encargan de devolver el feedback al usuario. Cuando un usuario se registra con éxito se crea un evento signIn para que la barra de navegación lo capture.

navbarController: Al cargarse el controlador se llama al metodo setId que comprueba si se ha iniciado sesión y en función de eso modifica las variables para mostrar lo que es necesario. Al pulsar sobre el botón Home, se ejecuta el método goHome que redirige a la vista principal del administrador o a la de un usuario normal en función del tipo de usuario. El método logout llama al servicio auth para que cierre sesión y modifica las variables de las vistas a sesión cerrada. Este controlador dispone de dos métodos que se encargan de capturar los eventos signIn y logOut que desencadenan llamadas a setId y logout respectivamente.

signUpController: Recoge los datos del usuario del formulario de registro, llama al servicio auth para que inicie sesión y devuelve feedback al usuario.

resetPasswordController: Recoge el email del formulario y se lo pasa al servicio users para que resetee su contraseña.

passwordController: Recoge la nueva contraseña del formulario, compueba que se ha escrito correctamente y se la pasa al servicio users para que la cambie.

verifyAcountController: Al cargarse llama al servicio users para que verifique la cuenta del usuario de la url y devuelve feedback al usuario.

starterController: Es el controlador que se carga al iniciar la aplicación, pero no tiene ninguna funcionalidad.

searchPoisController: Contiene los métodos para cargar los pois en función de la búsqueda y el resto de funcionalidades asociadas a un poi. Utiliza el servicio maps para cargar los pois en el mapa.

searchUsersController: Cargar los usuarios en función de la búsqueda. Los usuarios se guardan en la variable "foundUsers".

poisController: muestra la información del poi de la url. Utiliza el servicio maps para mostrarlo.

routesController: Muestra la información de la ruta de la url. Utiliza el servicio maps para mostrarla.

adminListController: Al cargarse utiliza el servicio adminList para cargar todos los usuarios del sistema y guardarlos en la variable listOfUsers. Ofrece las funcionalidades de borrar y mandar un aviso a un usuario. Como son acciones importantes requieren una confirmación que se realiza desde un modal. Al pulsar sobre una de estas acciones se ejecuta el método select, que modifica la variable send a true si es una advertencia y a false si es una eliminación. El método select también modifica el valor de index con el índice de usuario seleccionado en la tabla.

usersStatisticsController: Al cargarse el controlador utiliza el servicio users para obtener los nombres y los valores para la gráfica y el usuario de la url.

adminStatisticsController: Al cargarse el controlador utiliza el servicio users para obtener los nombres y los valores de la gráfica para el usuario administrador.

8.3 Autenticación

La tecnología utilizada para la gestión de autenticación ha sido JWT (Json Web Tokens). JWT es un método simple para aportar seguridad en una API REST, basado completamente en peticiones de token JSON entre cliente y servidor. El proceso es el siguiente:

Cliente hace una petición mandando sus credenciales de login. El servidor valida sus credenciales, y si todo ha ido bien, devuelve al cliente un JSON con un token que codifica los datos del usuario logueado en el sistema. El cliente al recibir el token puede guardarlo de la manera que quiere, mediante LocalStorage, Cookie u otros mecanismos de almacenamiento en el lado del cliente. En este caso, se ha utilizado el LocalStorage.

Todo el tiempo que el cliente acceda a una ruta que requiera autenticación, mandará su token a la API para autenticarse y poder consumir los datos que pretende. El servidor siempre validará este token para permitir o bloquear la petición llegada.

Como intermediario en este proceso se ha hecho uso de Passport, un framework muy flexible y fácil de usar, así como modular. Permite trabajar con las principales estrategias de autenticación, entre ellas, JWT. Es por ello que se ha elegido incluirlo en el proyecto, y que sea él el que sepa cuándo un método necesita validar el token.

Se ha decidido que la aplicación mostrará información tanto para usuarios registrados en el sistema como no registrados, sin embargo, las funcionalidades que se presentan en cada caso son distintas. Un usuario que no está registrado en el sistema podrá buscar POIs y usuarios y ver su información, así como poder recomendar sus POIs o rutas, pero está limitado a esas funciones. Un usuario registrado puede acceder a todas las funcionalidades explicadas anteriormente, gracias a la validación del token asociado a su sesión. Es por ello que las funciones de edición, creación, borrado, suscripción a usuarios y adicción de POIs a favoritos sólo se podrán realizar si un usuario está registrado en el sistema.

En base a estos dos tipos de usuarios, también cambia la barra de navegación, mostrando poder registrarse o loguearse a un usuario que no lo está, y mostrando el log out para uno que sí está logueado.

No obstante, se ha añadido que si el usuario por algún motivo llega a un punto donde se le permite realizar alguna operación no autorizada (mediante herramientas como Postman), esta se bloquea y se le remite a una pantalla de error que le informa de lo ocurrido, indicando que el acceso no está autorizado porque necesita estar logueado para hacer dicha operación.

Por último, cabe indicar que los métodos que se han elegido para consumir XML son el procedimiento de Sign up, el proceso de Post poi (crear POI), y la función Share POI/Route. Se han decidido estos por no ser triviales, todos son tipo POST para que tengan información, y por ser de diferentes ámbitos. Los DTD asociados están en el propio código de cada procedimiento indicado, pasándolo a un validador de xml para comprobar si es o no correcto.

8. Modelo de navegación

La navegación por la aplicación se muestra en el siguiente mapa de navegación:

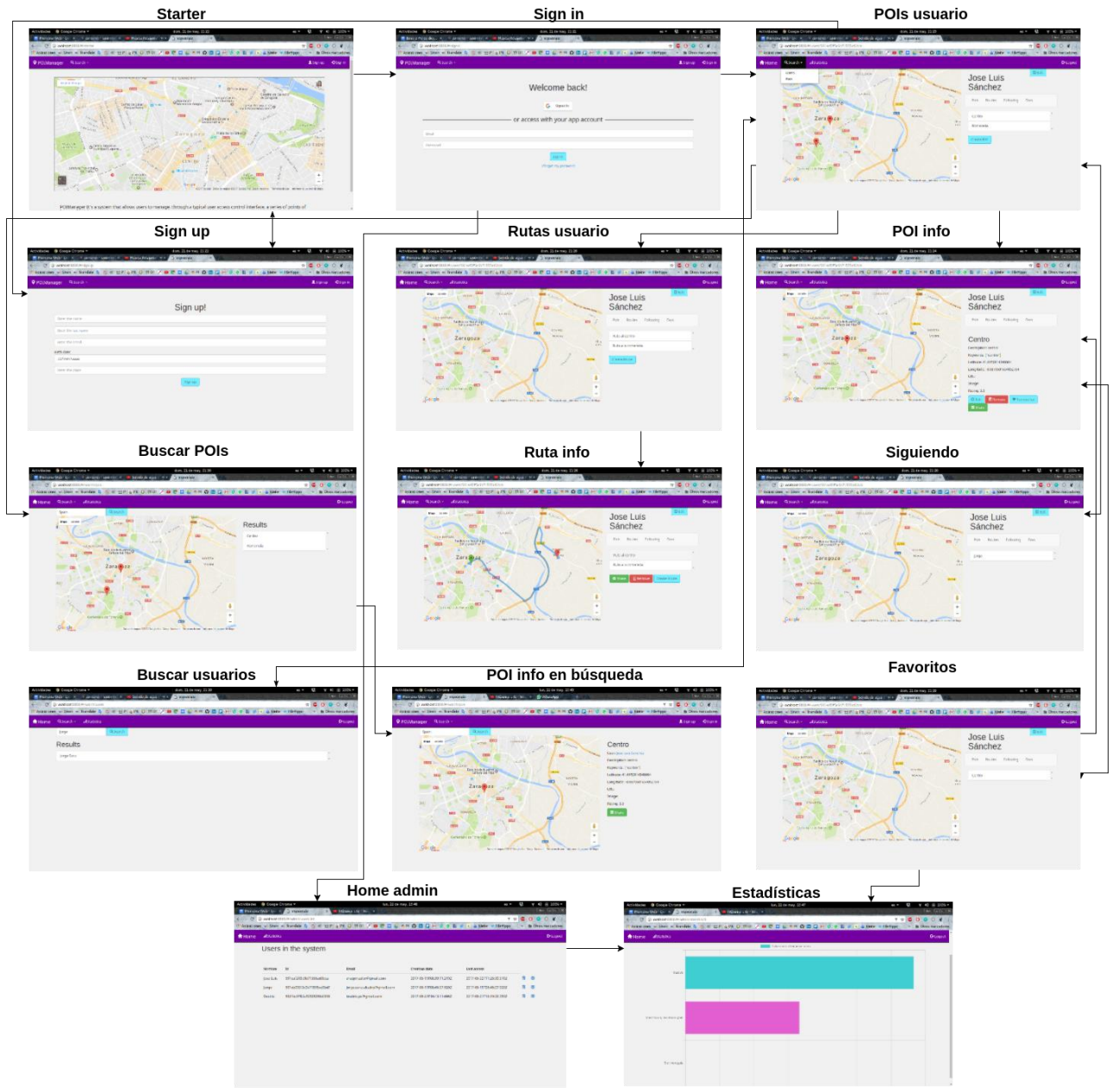

Ilustración 3. Mapa de navegación

Además, cabe destacar que desde cualquier punto de la aplicación se puede volver al home del usuario o se pueden visitar las estadísticas, ya sea usuario normal o administrador, debido a que los botones Home y Statistics siempre están visible en la barra de navegación.

Para la búsqueda, también se puede acceder desde cualquier punto de la aplicación ya que también se encuentra en la barra de navegación, excepto si el usuario logueado es el usuario administrador ya que se ha considerado que no tiene necesidad de buscar usuarios ni pois en concreto. Sin embargo, un usuario que no esté logueado también puede acceder a la búsqueda ya que, como se ha comentado anteriormente, este puede visualizar la información del sistema relacionada con usuarios y pois, aunque no puede realizar modificaciones en el sistema.

Cuando un usuario está logueado en el sistema, también puede cerrar su sesión en cualquier momento desde el botón de logout que se encuentra en la barra de navegación, y volverá a la página principal de la aplicación.

Respecto a los usuarios normales del sistema, siempre que se encuentra un usuario en la página principal, ya sea suya o de otro usuario, se puede acceder en cualquier momento a sus POIs, rutas, usuarios seguidos y POIs favoritos aunque todas estas interacciones no se han representado en el mapa de navegación anterior por simplificación del mismo.

Además, la vista de las pantallas se ha incluido en el anexo para poder visualizarlas con mayor detalle.

10. Despliegue del sistema

Para desplegar el sistema en local y partiendo de que ya se encuentra instalado NodeJS y MongoDB, es necesario seguir los siguientes pasos:

1. Copiar el proyecto completo a la máquina local
2. Lanzar MongoDB: mongod --dbpath /path/donde/se/quiera/poner/mongo
3. Situarse dentro del directorio del proyecto, Vignemale, y dentro instalar todos los módulos que tiene como dependencias ejecutando: npm install
4. Lanzar la aplicación ejecutando: npm start

Además, la aplicación se ha desplegado en una instancia de AWS (Amazon Web Services), perteneciente a Ubuntu 16.04. Para ello ha sido necesario instalar NodeJS y MongoDB en la máquina, y a continuación realizar los pasos explicados anteriormente.

11. Validación

Para la validación del sistema no se ha considerado necesario la creación de pruebas automáticas puesto que el proyecto dura algo más de dos meses y emplear mucho tiempo en ellas puede ser contraproducente. Es por ello que se ha decidido realizar sólo pruebas manuales con Postman para validar las funciones del servidor y el API REST.

Cuando se crea un nuevo endpoint se ha utilizado Postman para comprobar su correcto funcionamiento. Una vez se han terminado las pruebas, deben guardarse las peticiones realizadas con Postman para repetirlas en un futuro si se modifica dicho endpoint y detectar posibles fallos.

También se ha podido comprobar el correcto funcionamiento de la aplicación a través del API de la documentación proporcionada por Swagger, y poder descubrir posibles fallos que no se habían encontrado hasta el momento.

Para la validación en el lado del cliente no se ha utilizado ninguna herramienta específica, sino a base de pruebas de todo el sistema mediante la interfaz proporcionada. Si un miembro del grupo detecta un fallo, se apunta como tarea pendiente para que se arregle lo antes posible.

12. Análisis de problemas

En las fases iniciales del proyecto, se realizaron varias reuniones del equipo para elaborar los esquemas de componentes, relación entre las vistas, un primer esquema del modelo de datos, los endpoints que se iban a proporcionar mediante el API y toda información que pudiera ocurrirse en fases de análisis y diseño.

Inicialmente, se plantearon varias dudas por no saber la dificultad que podían suponer las funciones pensadas, sobre todo relacionadas con la parte del frontend. Por ello se hizo un primer diseño de las pantallas sobre las que posteriormente se han realizado algunos cambios para facilitar su implementación.

Todos los problemas encontrados durante todo el proceso de la aplicación se han comentado entre el equipo para poder solucionarlo lo más rápido posible.

13. Problemas encontrados durante el desarrollo

Algunas de las tecnologías empleadas en el proyecto eran nuevas para nosotros y nos ha costado su tiempo acostumbrarnos a ellas.

Otro de los problemas fue que gran parte de la funcionalidad estaba conectada y resultaba difícil. Al mostrar POIs, rutas, followings y favoritos había que seguir manteniendo la información del usuario al que pertenecían por lo que se decidió utilizar un único controlador para gestionar los POIs, rutas, followings y favoritos de un usuario específico. Este controlador es bastante grande en comparación con el resto.

Nos ha costado bastante encontrar una librería que funcionase bien tanto en sobremesa como en móvil, ya que la mayoría de ellas no funcionaba para movil. Al final encontramos rubaxa/Sortable.js que si que funcionaba.

Al principio del proyecto se utilizaba el tipo de dato objectId para las referencias entre tablas. Utilizar este tipo de datos generaba algunos problemas en búsquedas y comparaciones. Aunque se use este tipo de dato, mongoDB no verifica la integridad de las referencias, por lo que se decidió utilizar Strings para las referencias.

Se tuvieron problemas ejecutando la aplicación desde AWS porque no enviaba los correos que se tenían configurados. Para solucionar esto, se descubrió que Gmail bloquea las direcciones IP de AWS por defecto, por lo que se visitó la url https://accounts.google.com/DisplayUnlockCaptcha una vez hecho login desde el correo de administrador, y consiguió solucionarse el problema, pero llevó mucho tiempo descubrirlo.

14. Análisis de problemas potenciales

La aplicación está pensada para dar servicio a un gran número de usuarios, puesto que las tecnologías utilizadas lo permiten. Mongo es una base de datos que tiende a trabajar con cantidades de información considerables, teniendo una velocidad de respuesta muy alta. Node JS muestra su alto rendimiento a través de JavaScript, y no se han visto afectadas funciones en ningún caso.

Se ha podido probar que hasta 10 usuarios el sistema funciona correctamente. Para que el sistema funcionase con muchos más usuarios, sería necesario que el sistema pudiese escalar de forma automática, por ejemplo, realizando la configuración adecuada en AWS.

Además, si el sistema fuera utilizado por muchos usuarios al mismo tiempo podrían producirse fallos o retenciones en el envío de correos de recomendaciones o de registro de usuarios, los cuales serían bastante críticos ya que provocaría que un usuario tuviera que esperar mucho hasta recibir los correos de confirmación y su contraseña.

15. Distribución de tiempo

El proyecto se ha llevado a cabo con una metodología Scrum, en la cual se dividieron las tareas principales en tareas más pequeñas para poder distribuir entre los miembros más fácilmente los trabajos. Esto se ha apoyado con la gestión de tareas a través de la herramienta Trello, en la cual se han ido trasladando las tareas En lista, creadas al principio, a la fase En proceso, indicando qué miembro realizaba cada tarea para saber qué se estaba haciendo en cada momento. Cuando se terminaba cada tarea, se pasaba a la fase Listo, y se volvía a asignar una tarea para realizar.

Con este modelo, respaldado con una hoja Excel del equipo en la que se indican las horas dedicadas al proyecto cada día y en qué, ha sido sencillo llevar a cabo una distribución flexible del trabajo, así como una cuenta productiva de todo el equipo.

Se adjunta el Excel en el que va la dedicación en horas de cada parte y de cada miembro en el mismo directorio en el que está esta memoria.

16. Conclusiones

Este proyecto ha servido a todos los miembros del equipo a realizar un trabajo de tamaño considerable en cuanto a una aplicación web desarrollada con Stack Mean. Ha sido la primera vez que se ha hecho uso de esta tecnología por parte de todos miembros del equipo, y ha resultado una experiencia satisfactoria. Se han adquirido una gran cantidad de conceptos nuevos, y se han mejorado otros muchos que se conocían vagamente.

Para el equipo, es de gran importancia haber aprendido a realizar tareas que hoy en día son muy solicitadas en el ámbito de desarrollo web, por ser tecnologías modernas y sencillas para el desarrollador. Es un trabajo adecuado a la asignatura, aunque hay aspectos que no se han llegado a pulir del todo como se habría querido por falta de tiempo, como el despliegue bien realizado en Cloud.

No obstante, el equipo ha sabido coordinarse y distribuirse correctamente las tareas mediante una metodología que no se había usado anteriormente por parte del equipo.

17. Conclusiones del proyecto

Es un proyecto en el que se trabajan multitud de aspectos, no sólo el propio desarrollo web, si no un análisis y diseño previo que se ha conseguido implementar tal y como se propuso al principio, además de una distribución y coordinación de las tareas de gran

relevancia en todo el proceso. La comunicación entre los miembros del grupo es fundamental para llevarlo a cabo.

Es un gran trabajo que requiere una enorme dedicación en el que se manejan muchos aspectos no desarrollados tan profundamente anteriormente y que, sin embargo, es de vital importancia en aplicaciones del mercado, como es toda la gestión correcta de usuarios y su seguridad. La repercusión que supone realizar una adecuada relación entre todos los componentes del sistema es principal, y se ha sabido llevarla a cabo sin graves problemas.

18. Valoración personal del grupo

Los tres miembros del grupo han aprendido a realizar gran cantidad de funciones que no sabían anteriormente, y coinciden en que es un proyecto que toca muchos temas de una aplicación real. Gestionar un proyecto de este tamaño equivale a mucho tiempo de administración y desarrollo por parte de todo el equipo, además de coordinación en las tareas para avanzar lo más rápido posible.

Se echa en falta un poco más de tiempo para terminar de elaborar tareas más pesadas, y todo el equipo coincide en que piden demasiadas tareas para el tiempo del que se dispone. No obstante, es la manera de aprender y de saber solucionar problemas, por lo que se ha visto necesario de un modo u otro.

19. Valoración personal de cada miembro

Beatriz Pérez: La realización de este trabajo me ha resultado muy útil para aprender muchos aspectos relacionados con las tecnologías web, más en concreto del stack MEAN. Se han aprendido nuevos aspectos y se han afianzado algunos que se conocían anteriormente, y ha resultado interesante para aprender a trabajar en grupo en un trabajo que requiere la división de tareas y posterior unión, donde todo tiene que funcionar correctamente. Sin embargo, ha faltado algo de tiempo para elaborar la parte opcional relacionada con AWS que me parecía bastante importante e interesante, porque se ha tenido que invertir demasiado tiempo en el resto de las funcionalidades. Por este motivo, me ha parecido que hubiera resultado más interesante reducir alguna de las funcionalidades básicas que al final resultan repetitivas para poder incluir el despliegue y la escalabilidad en AWS.

Ana Roig: ha resultado ser un trabajo considerable en menos tiempo del que nos habría gustado para poder finalizar alguna de las tareas más correctamente. Aún así, es un proyecto del que me ha gustado mucho formar parte, porque he aprendido completamente cómo funciona toda la comunicación cliente-servidor en una aplicación con tecnología MEAN. Me ha servido para afianzar algunos aspectos de desarrollo web que antes no tenía muy claros, además de aprender gran cantidad de recursos que veo muy necesarios para las aplicaciones web

actuales. En conclusión, me ha gustado mucho poder elaborar una aplicación como ésta porque, además de aprender cómo funciona todo a bajo nivel, le veo utilidad en un futuro. Aunque me habría gustado tener algo más de tiempo.

Jorge Sanz: En general ha sido un proyecto bastante interesante. Hemos trabajado con tecnologías muy recientes y algunas de ellas facilitan bastante el desarrollo web. Al comienzo del proyecto decidimos cómo iban a ser muchos de los aspectos de la aplicación y la mayor parte de lo que decidimos entonces se ha mantenido hasta el final. Creo que hemos trabajado bien juntos y no ha habido ningún problema entre los miembros del grupo. El tiempo se ha quedado un poco corto, ya que si hubiéramos tenido algo más de tiempo podríamos haber mejorado algunos detalles.

Anexos

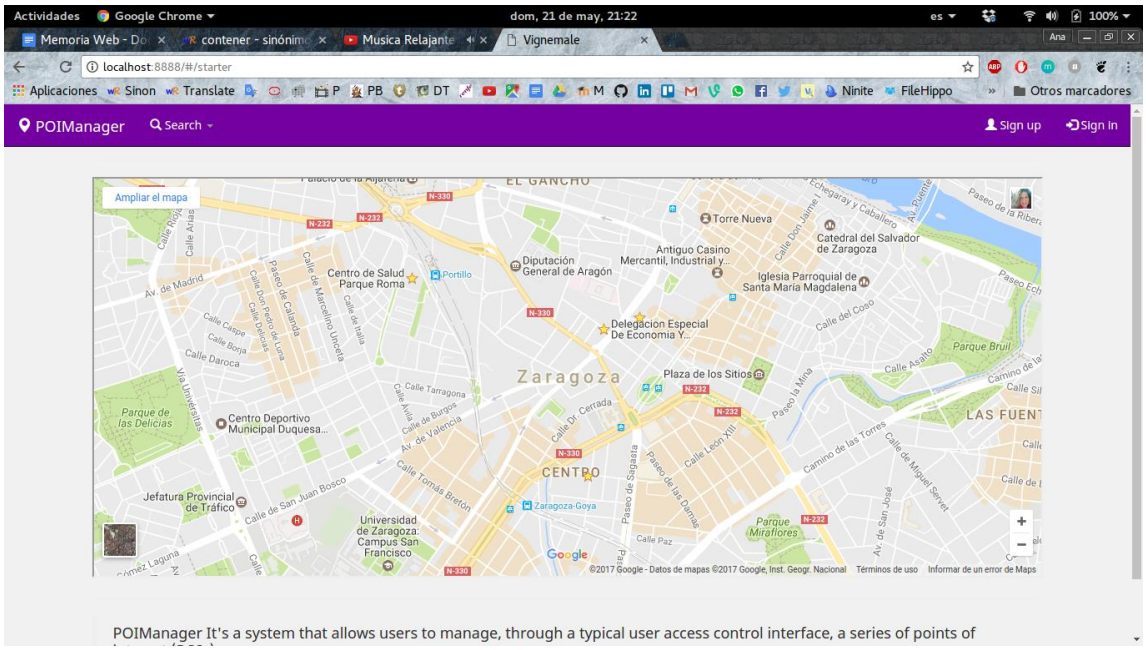

Ilustración 4. Starter

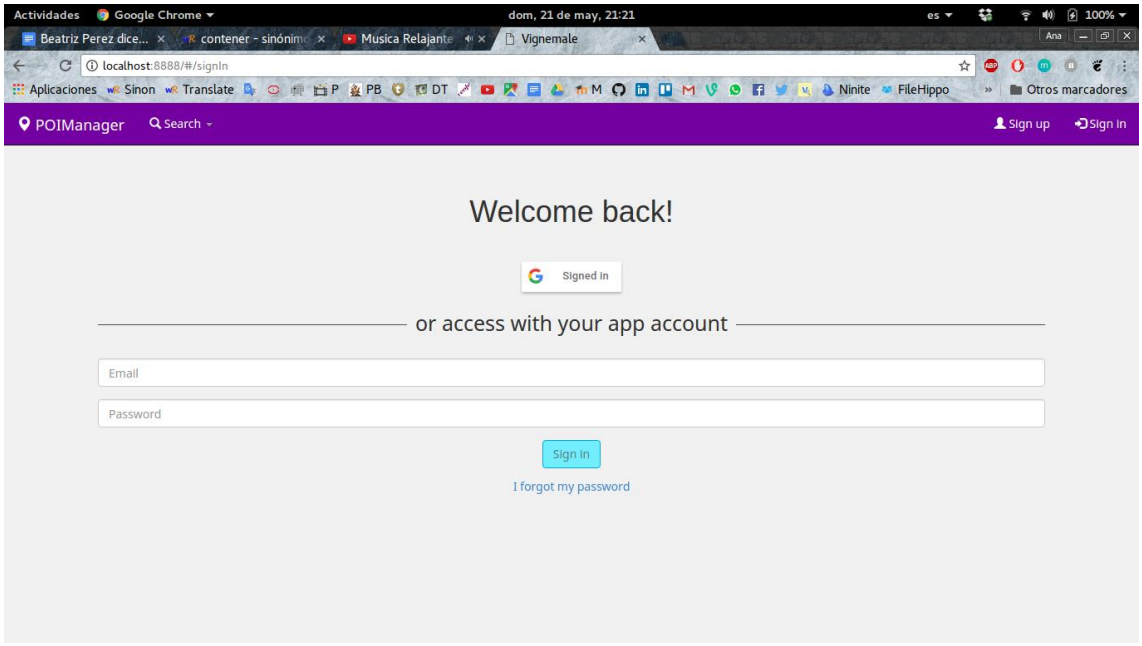

Ilustración 5. Sign in

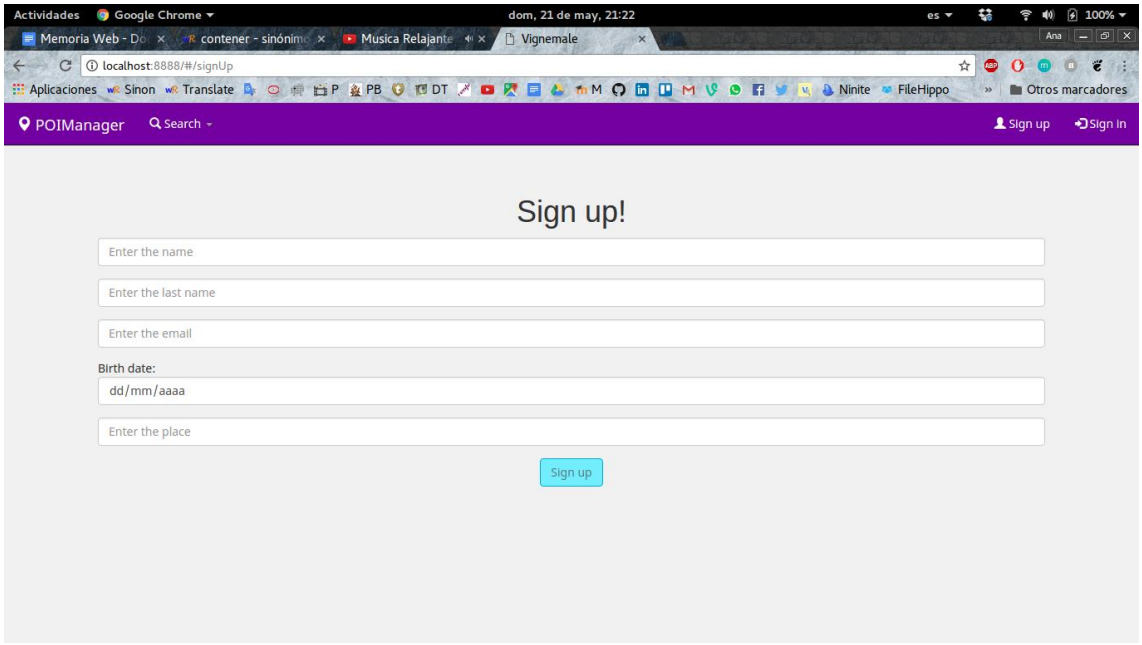

Ilustración 6. Sign up

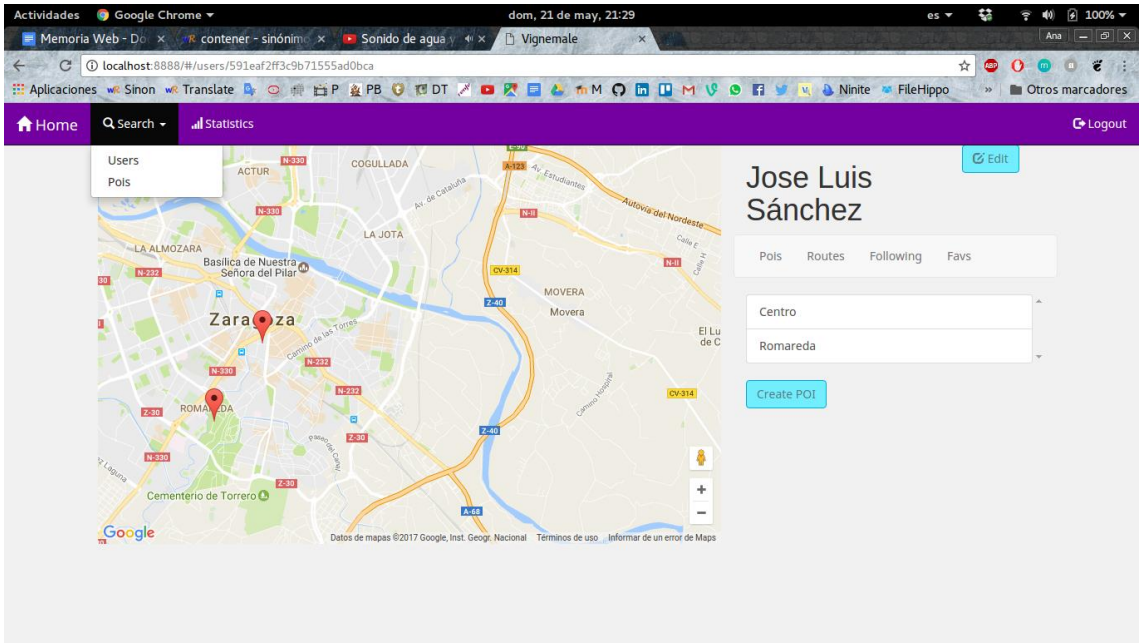

Ilustración 7. POIs usuario

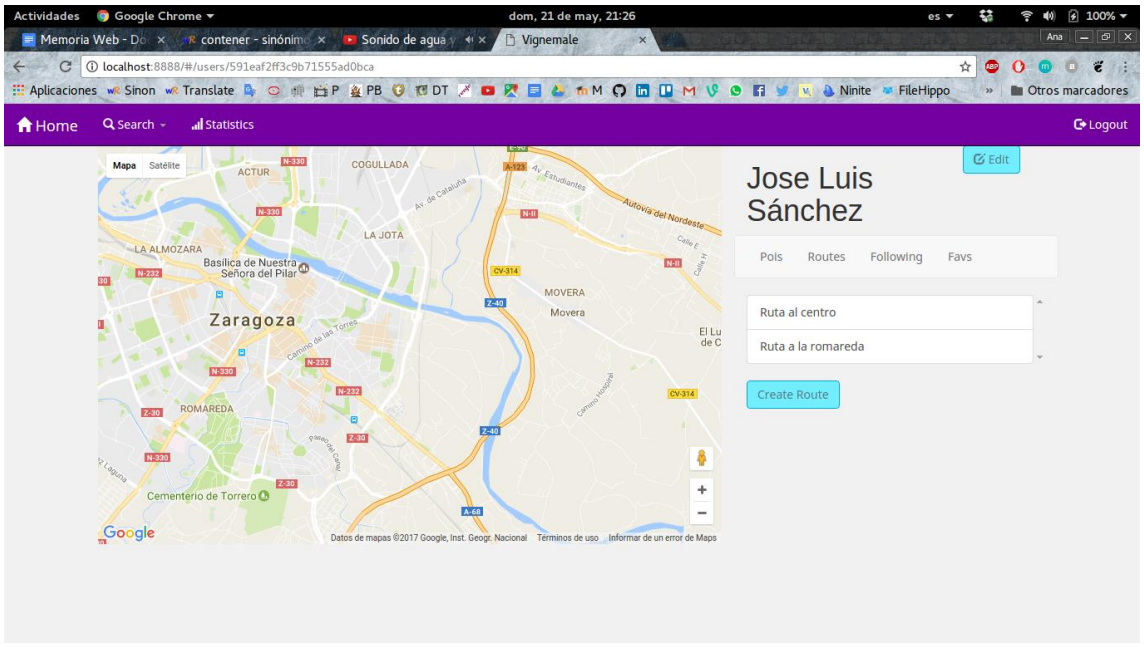

Ilustración 8. Rutas usuario

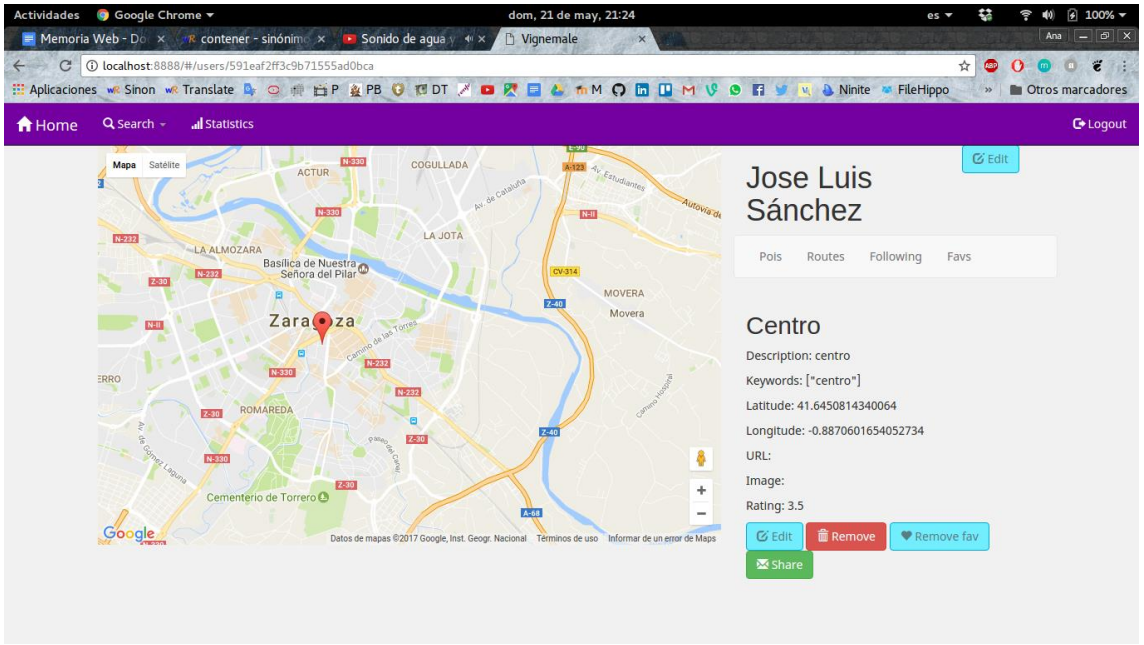

Ilustración 9. POI info

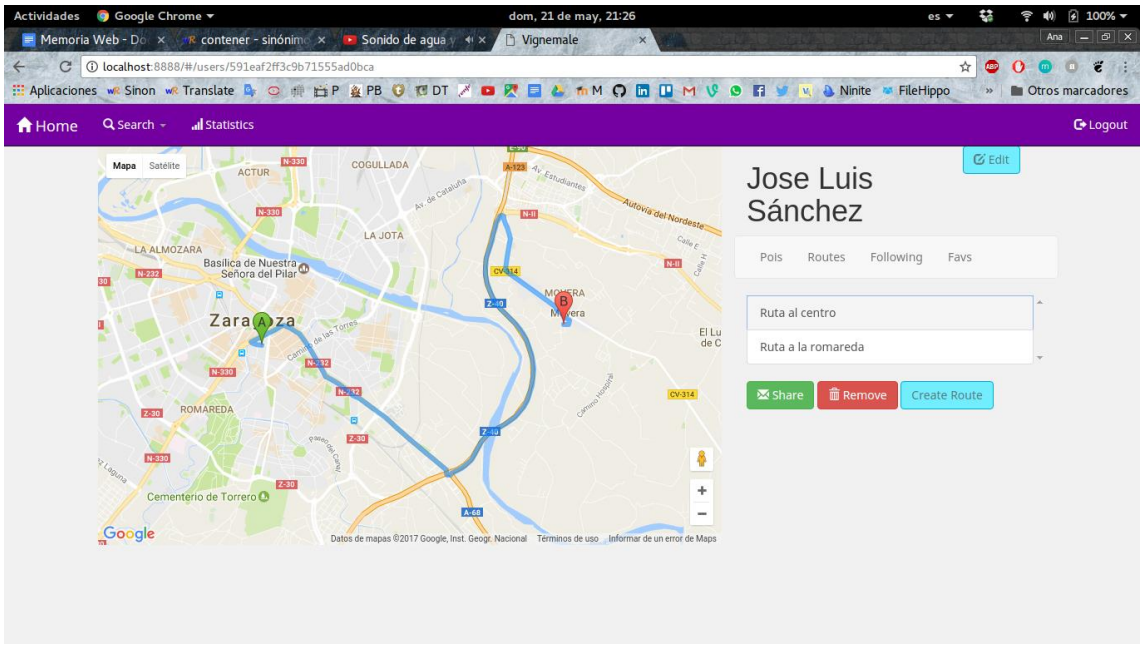

Ilustración 10. Ruta info

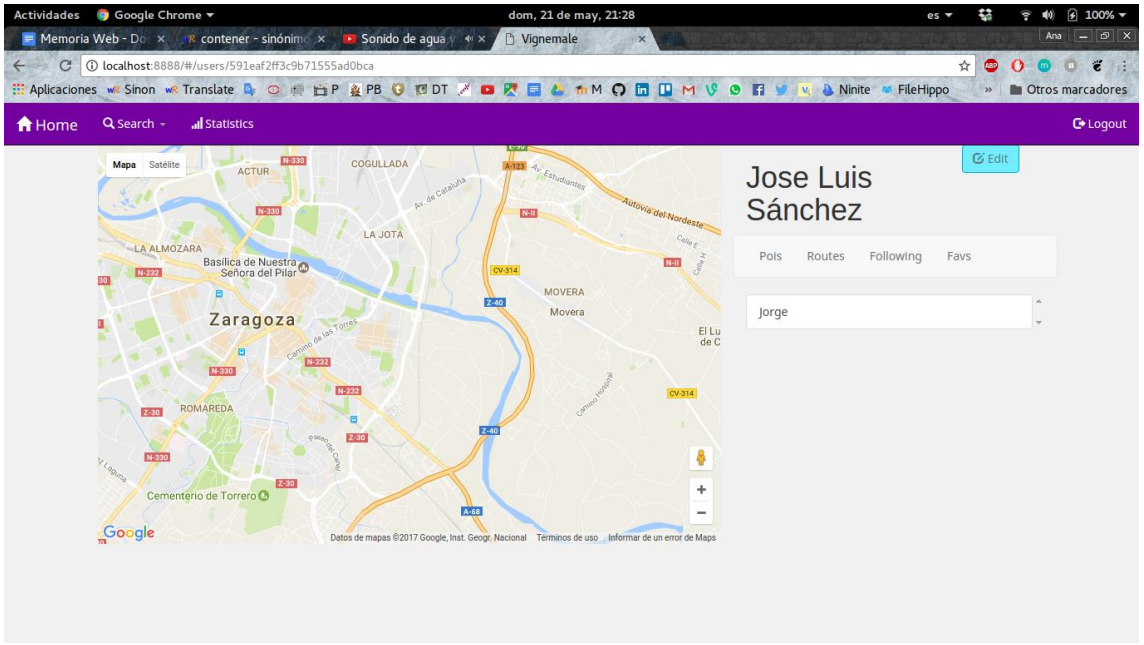

Ilustración 11. Siguiendo

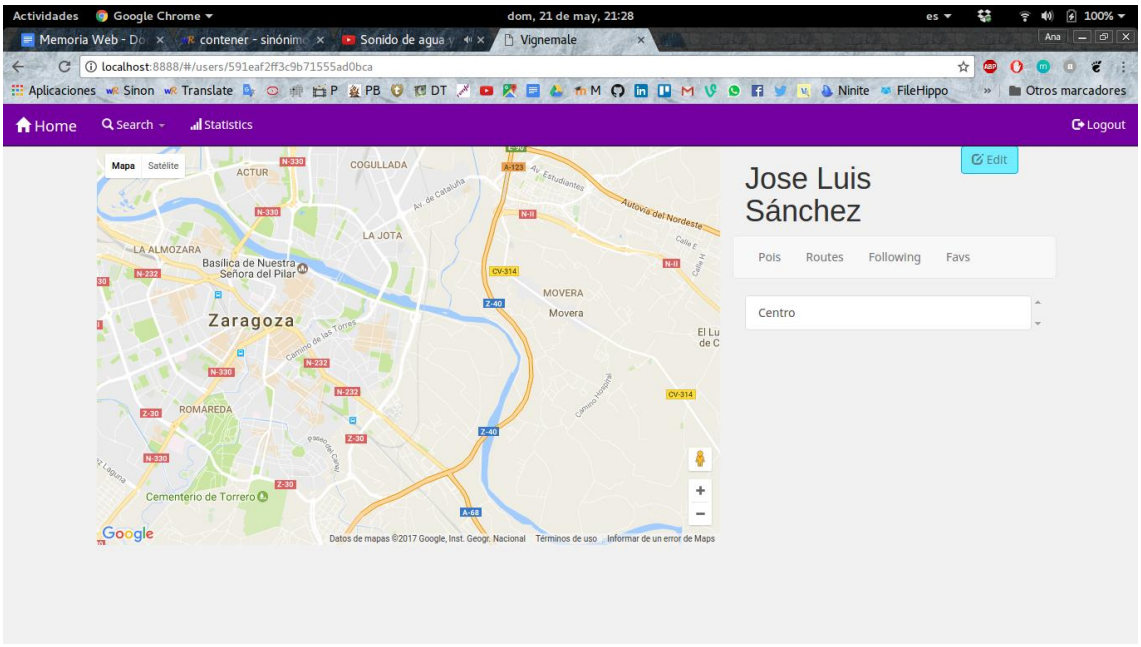

Ilustración 12. Favoritos

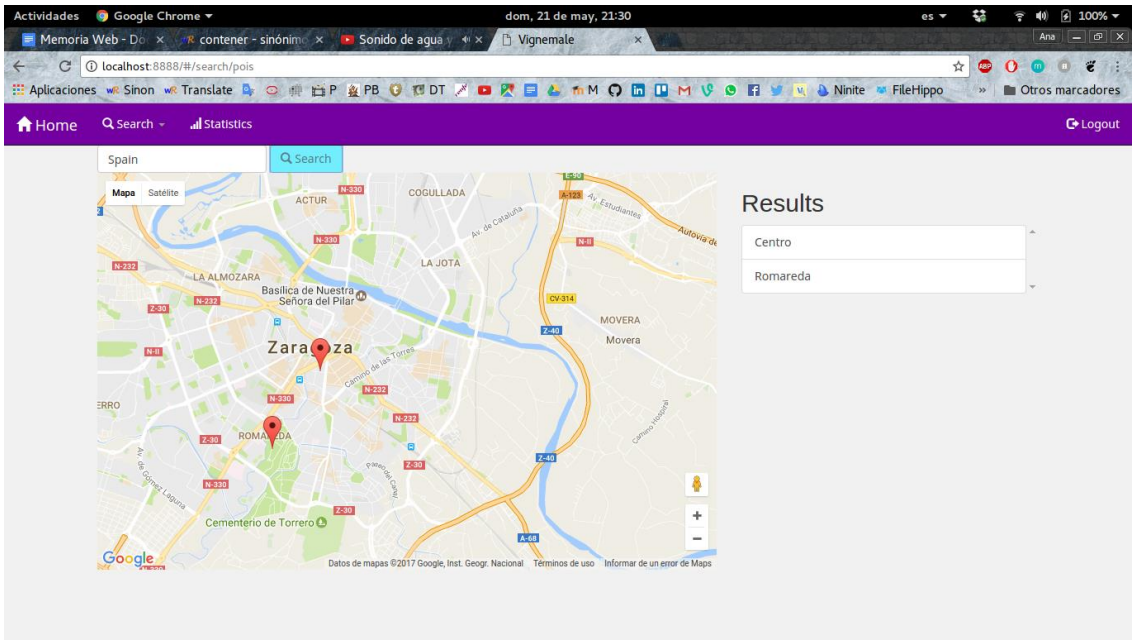

Ilustración 13. Buscar POIs

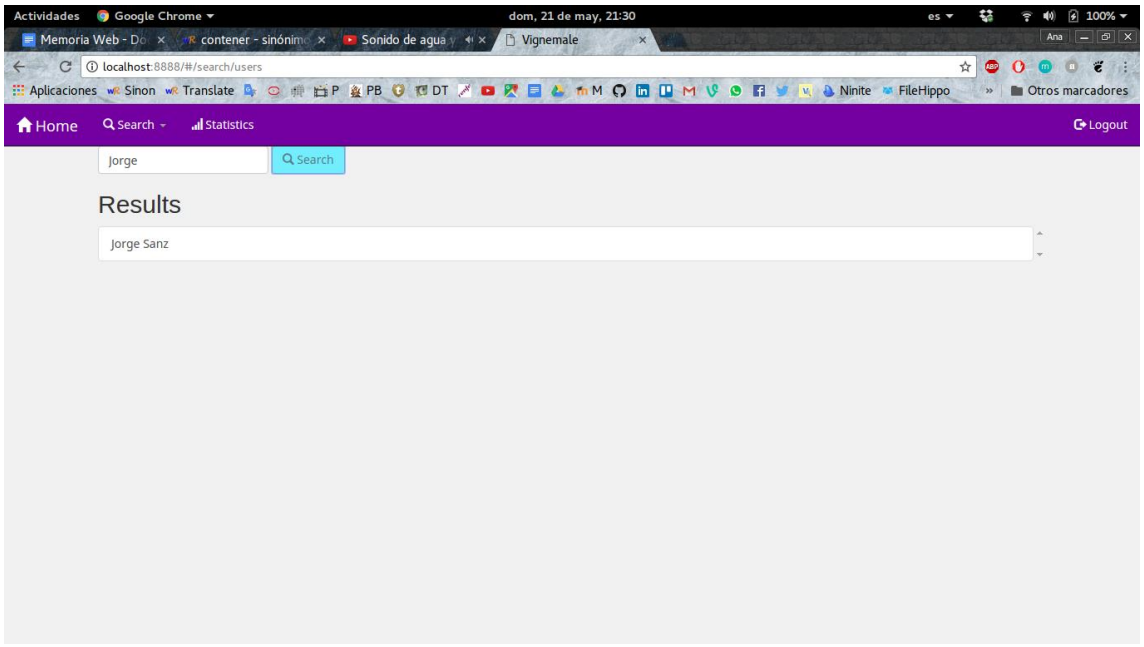

Ilustración 14. Buscar usuarios

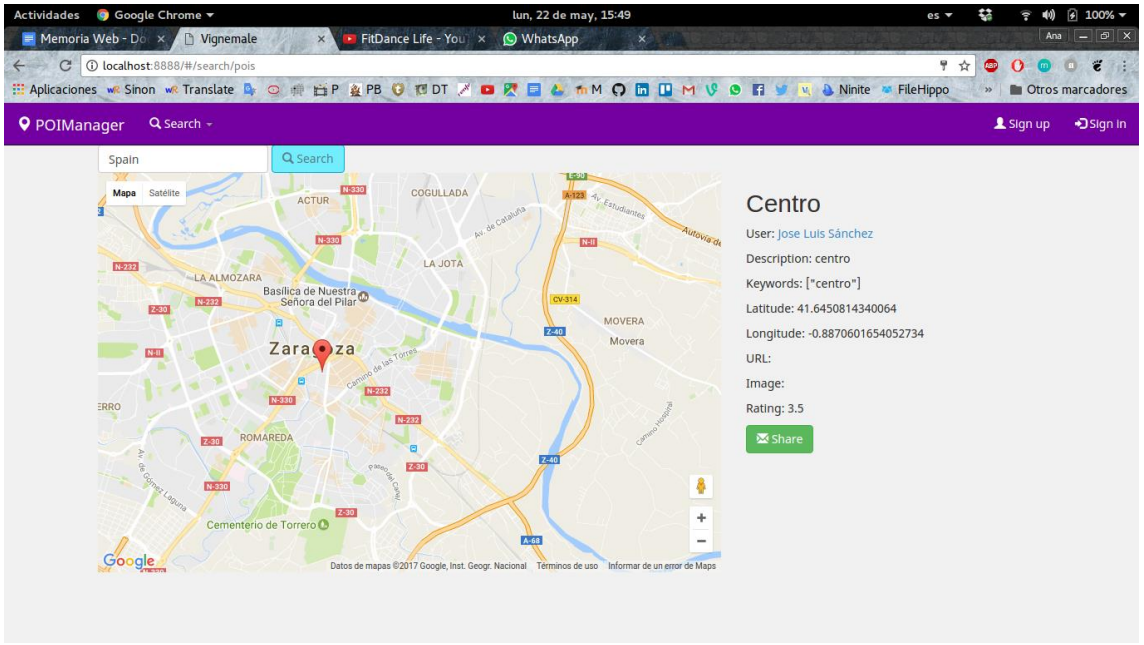

Ilustración 15. POI info en búsqueda

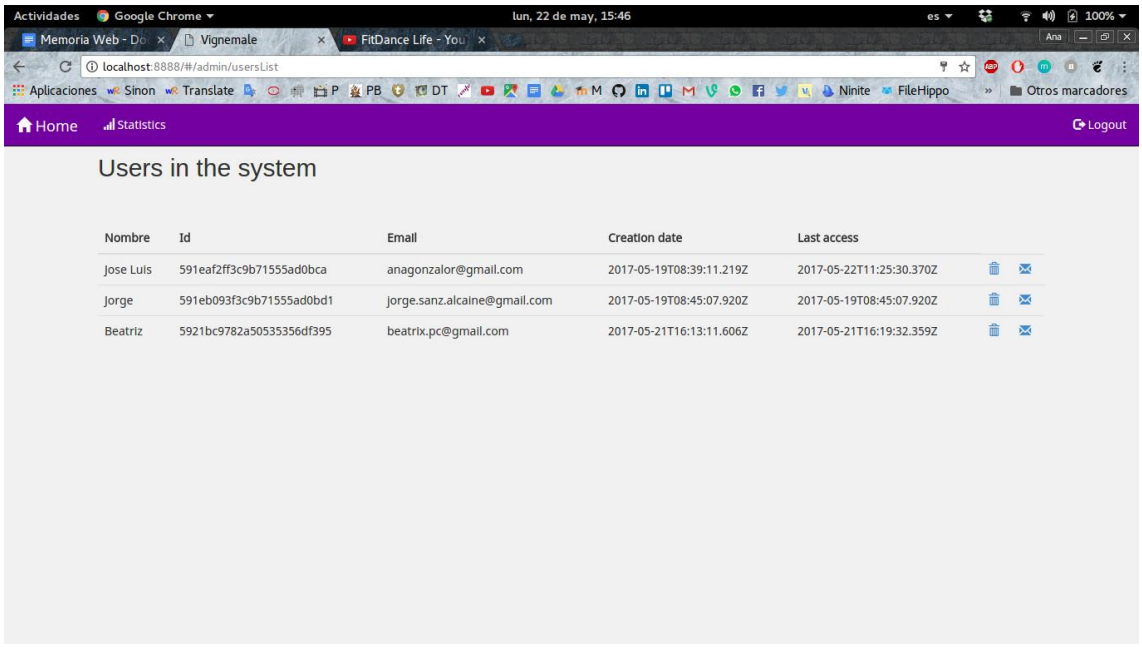

Ilustración 16. Home admin

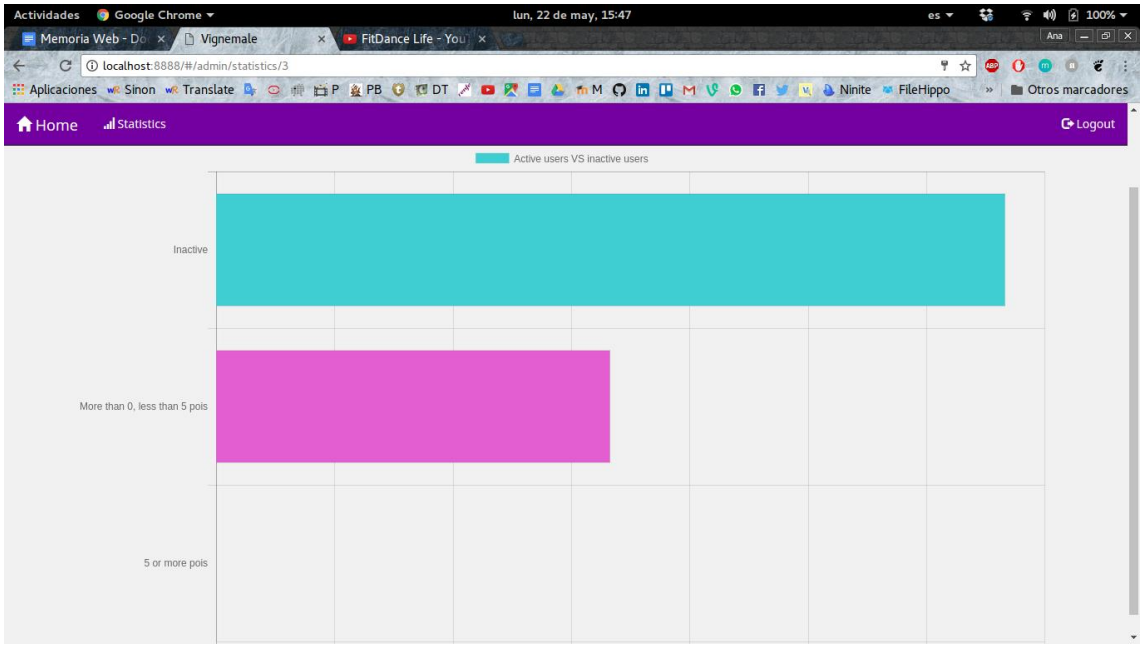

Ilustración 17. Estadisticas