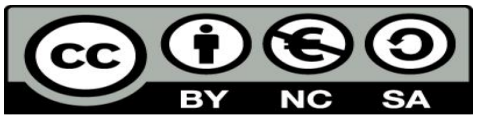
Desarrollo de Aplicaciones Web

Proyecto web conjunto: Gestión y Valoración de Restaurantes Desarrollo de API REST en Node.js

Despliegue de Aplicaciones Web

Pablo Garramone
Arturo Bernal

IES San Vicente



Índice de contenidos

1. Introducción

En este proyecto deberemos, en lo que a a la parte de Node.js se refiere, definir una serie de servicios REST como respuesta a las diferentes peticiones que nos puedan llegar de la aplicación cliente. Estos servicios se agrupan en 3 categorías principales:

- Autenticación y registro de usuario: contra la propia base de datos, o contra Facebook/Google.
- Gestión de restaurantes: listado, alta, borrado y modificación de restaurantes y comentarios
- Perfil del usuario: consulta del perfil de usuario, así como modificación de algunos datos (e-mail, nombre, password y avatar)

En este documento se pretenden dar unas orientaciones sobre cómo desarrollar los servicios propuestos, y qué atributos o campos puede devolver cada uno, pero no es obligatorio seguir este esquema, siempre que el esquema elegido sea también razonable y esté debidamente documentado en la entrega.

1.1. La base de datos

En principio, se trabajará sobre una base de datos MySQL cuyo script de importación se puede descargar en los recursos del aula virtual para este proyecto. Al importarla, creará automáticamente la base de datos también, con tres tablas:

- Tabla "**user**": contiene la información de cada usuario registrado: id (autonumérico), nombre, email, password, avatar, latitud y longitud de geolocalización
- Tabla "**restaurant**": con información sobre cada restaurante: id (autonumérico), nombre, descripción, días abierto, teléfono, imagen, tipo de cocina (italiana, argentina...), creador (clave ajena a la tabla "user" que indica el usuario que creó el restaurante), estrellas, latitud y longitud del restaurante (para geolocalización) y dirección
- Tabla "**comment**": relación de las dos anteriores, indicando los comentarios que hace un usuario sobre qué restaurante. Como campos tiene un id (autonumérico), el id del usuario, el id del restaurante (es una relación M:M, donde un usuario solo puede comentar 1 vez cada restaurante), el comentario realizado y la fecha del mismo.

Además de estas tablas, la base de datos cuenta con otros tres elementos a tener en cuenta:

- Por un lado, tiene definido un *trigger* que, en cuanto añadamos un comentario en la tabla "*comment*" para calcular las estrellas del restaurante comentado, automáticamente se actualizará el campo "*stars*" de la tabla "*restaurant*" indicando la nueva media de estrellas del mismo.
- Un segundo trigger que insertará la fecha del comentario automáticamente
- Por otro lado, cuenta con un procedimiento implementado, llamado **haversine** que recibe como parámetros la latitud y longitud de un usuario y la de un resaturante (primero el usuario y luego el restaurante, o al revés), y devuelve la distancia que hay desde el usuario al restaurante. Por ejemplo:

```
SELECT user.name, restaurant.name, haversine(user.lat, user.lng, restaurant.lat, restaurant.lng) as distance
FROM user, restaurant
WHERE user.id = XXXXX AND restaurant.id = XXXXX
```

1.2. Estructura general de los servicios a desarrollar

Como indicamos en esta introducción, no es obligatorio seguir el esquema que se indica a continuación, pero sí es buena idea dotar a todos los servicios de un mismo formato de respuesta, para que el código en servidor y cliente sea más homogéneo. Por ejemplo, podemos indicar una serie de campos comunes de respuesta, como:

- **error**, que será *false* si el resultado que se envía es correcto, o *false* si ha habido algún error (por ejemplo, validación de usuario incorrecta, o *id* no encontrado).
- **errorMessage**, que sólo se enviará si el anterior campo es *true*, y contendrá información sobre el error producido.
- **result**, con el resultado a enviar, en el caso de que error sea *false*. Este campo puede que no sea necesario en algunos servicios en concreto, como por ejemplo al autenticar el usuario.

1.2.1. Sobre las URIs asociadas a los servicios

En los servicios que se exponen a continuación, se indica para cada uno qué URI tiene asociada. Sin embargo, algunas URIs pueden reestructurarse de otra forma, si se prefiere. Por ejemplo, en el caso de los restaurante, hay una URI para ver todos los restaurantes (/restaurants), otra para ver los restaurantes cuyo creador es el usuario actual (/restaurants/mine) y otra para ver un restaurantes dado por id (/restaurants/:id). Podemos plantearlo así, o como una única URI /restaurants que reciba un parámetro diferente para cada tipo concreto de listado:

- /restaurants?show=all
- /restaurants?show=mine
- /restaurants?show=id

2. Autenticación y registro (I)

En este apartado comentaremos los servicios básicos a desarrollar para registrar y autenticar a los usuarios de la aplicación. Para ello, nos centraremos exclusivamente en la tabla "*user*" de la base de datos, sin tener (de momento) en cuenta fuentes externas de autenticación, como Facebook o Google. Para acceder a cualquier servicio de este apartado (salvo al de validación de token) no será necesario haber conseguido un token previo (es decir, se puede acceder a ellos sin estar previamente autenticado).

2.1. Registro de usuarios

Para registrar usuarios accederemos por **POST** a la URI **/auth/register**. Enviaremos como datos los campos del registro (nombre, e-mail, contraseña, latitud, longitud y avatar del usuario), y obtendremos como respuesta un objeto JSON con la información correspondiente. Por ejemplo, los atributos error, errorMessage y result comentados en la introducción, conteniendo este último los datos completos del usuario añadido, o bien el *id* que se le ha asignado (como se prefiera).

2.2. Autenticación local de usuarios

Para loguear o autenticar a un usuario de forma local contra nuestra base de datos, accederemos por **POST** a la URI **/auth/login**. Enviaremos el e-mail y contraseña, y obtendremos como respuesta un objeto JSON, que puede contener los atributos error, errorMessage y token, este último con el token generado en el caso de que la validación haya sido correcta. Podéis emplear la palabra secreta y el tiempo de vida de token que consideréis oportuno. También convendría que el token contuviera información para identificar al usuario, como por ejemplo su *id* (y también su e-mail si se quiere).

2.3. Validación de token

Contaremos también con la URI **/auth/validate** por **GET**, para validar si el token que le llegue (en la cabecera *Authorization*) es válido. Este servicio puede devolver un simple campo error indicando si es correcto o no.

3. Gestión de restaurantes

En este apartado comentaremos los servicios a desarrollar para la gestión de los restaurantes y usuarios que los crean y los comentarios que hacen sobre ellos. Para acceder a todos estos servicios, es necesario estar logueado previamente, por lo que deberemos enviar el token para validar, a través de la cabecera "Authorization". En caso de no estar autorizado, devolveremos un código de estado 403, con algún mensaje de error si se quiere.

3.1. Información adicional a añadir en algunos resultados

En los tres listados de restaurante devuelto en cada resultado, deberemos añadir una información adicional que no está incluida en la tabla "*restaurant*":

- La puntuación media del mismo (stars)
- La distancia en Km desde el usuario hasta el restaurante (usando la función *haversine* comentada en la introducción).
- Si el restaurante es del usuario actual o no (un booleano mine)
- El el caso del listado por id, también se obtendrá la información sobre el usuario creador del mismo

En el caso de que el servicio solicitado devuelva (o pueda devolver) un vector, lo podemos incluir en el campo result de la respuesta (añadiendo para cada restaurante la información adicional comentada antes). Si el servicio devuelve un único restaurante, añadiremos dicho restaurante en el result, junto con su información adicional también.

3.2. Listados de restaurantes

3.2.1. Listado general de restaurantes

Si accedemos por **GET** a la URI **/restaurants** obtendremos un listado de todos los restaurantes de la tabla "*restaurant*" ordenados por distancia respecto al usuario autenticado. Podemos devolver los elementos error, errorMessage y result, conteniendo este último el vector de restaurantes encontrado. En el caso de que el vector sea vacío, podéis decidir si devolver un vector vacío sin error, o un error a *true*.

3.2.2. Restaurantes creados por el usuario actual

Si se accede por **GET** a la URI **/restaurants/mine**, obtendremos un listado restringido a los restaurantes creados por el usuario actual (tendremos que procesar el token de la cabecera para averiguar el usuario). Ordenados por distancia también.

3.2.3. Datos de un restaurante en concreto

Accediendo por **GET** a la URI **/restaurants/:id** obtendremos el restaurante cuyo id recibe por parámetro en la url. Obtendremos también el usuario creador del mismo.

3.2.4. Comentarios sobre un restaurante

Accediendo por **GET** a la URI **/restaurants/:id/comments** obtendremos todos los datos de los comentarios en un array sobre el restaurante cuyo id recibe por parámetro en la url, además de los datos del usuario que lo hace.

3.3. Modificaciones de restaurantes

3.3.1. Creación de restaurantes

Accediendo por **POST** a la URI **/restaurants**, insertaremos los datos del restaurante que nos lleguen desde el cliente (nombre, descripción, días que abre, tipo de cocina, teléfono, dirección, latitud, longitud e imagen). Podemos devolver el error, errorMessage, y como resultado podemos devolver el registro añadido, o su *id* asignada.

3.3.2. Añadir comentario sobre un restaurante

Accediendo por **POST** a la URI **/restaurants/:id/comments** , insertaremos un comentario al restaurante indicado en el id. Como datos del POST indicaremos la descripción del comentario y las estrellas que deseamos darle, internamente el trigger le añadirá la fecha. Como resultado podemos devolver el error, el mensaje y el *id* generado en caso de inserción correcta.

3.3.3. Modificación de restaurantes

Si accedemos por **PUT** a la URI **/restaurants/:id***,* podremos modificar los datos del restaurante que indiquemos en la URI. Recibirá los mismos datos que el POST para creación de restaurantes, y como respuesta, en lugar de devolver el *id*, si todo es corecto devolveremos el objeto modificado (o el antiguo antes de modificarse, para permitir posibles operaciones de deshacer).

3.3.4. Borrado de restaurantes

Si accedemos por **DELETE** a la URI **/restaurants/:id** procederemos a borrar el restaurante con el *id* se indique. La base de datos cuenta con un borrado en cascada que automáticamente borrará las entradas de la tabla "*comments*" vinculadas al restaurante borrado.

Es IMPORTANTE que, tanto en la modificación como en el borrado, controlemos que el restaurantes a modificar y borrar lo haya creado el usuario actual (para que nadie pueda alterar restaurantes creados por otros).

4. Gestión de usuarios

El tercer gran módulo de la aplicación hace referencia a la gestión de usuarios: obtener el perfil del usuario autenticado, de un usuario concreto y modifcar partes del usuario. Todos los servicios de esta sección requieren un token enviado como cabecera, y por tanto, que el usuario se haya logueado previamente. En caso contrario, igual que en la sección anterior, enviaremos un código 403 como respuesta.

4.1. Listados

4.1.1. Ficha del usuario actual

Accediendo por **GET** a la URI **/users/me** se devolverán los datos del usuario actualmente logueado. Podemos emplear la misma estructura error, errorMessage y result, dejando en este último el objeto con los datos del usuario actual.

4.1.2. Ficha de un usuario concreto

Si accedemos por **GET** a la URI **/users/:id** obtendremos, de forma similar al servicio anterior, los datos del usuario cuyo *id* se especifique en la URI.

4.2. Modificaciones

Además de los listados anteriores, el cliente también podrá modificar mediante **PUT** algunos datos de su perfil. Dependiendo de la estructura de la URI, se modificarán unos u otros datos:

- **/users/me** servirá para actualizar el e-mail y el nombre del usuario
- **/users/me/avatar** servirá para actualizar el avatar o imagen del usuario
- **/users/me/password** servirá para actualizar el password del usuario

En cualquiera de los casos, enviaremos en el cuerpo de la petición los datos correspondientes, y devolveremos un objeto JSON, que puede contener los campos error, errorMessage y result, este último con el objeto usuario con sus datos actualizados (o antes de actualizarse, como se prefiera).

5. Autenticación y registro (II)

En esta segunda parte sobre la autenticación y registro vamos a ver algunos detalles sobre cómo validar usuarios contra sus cuentas de Google o Facebook, a través de los servicios correspondientes.

5.1. Paso previo: cambios en la base de datos

Para dar cabida a estas nuevas validaciones, necesitamos hacer unos pequeños cambios en la base de datos, en concreto en la tabla de usuarios:

- En primer lugar, haremos que el campo *password* admita nulos
- En segundo lugar, añadiremos los campos *id_google* e *id_facebook*, de tipo cadena (varchar(100), por ejemplo) y que admitan nulos.

Esto nos servirá para que, cuando un usuario quiera registrarse con su cuenta de Google o Facebook, se le almacene el *id* correspondiente en el campo correspondiente, y se deje su password vacío. Por otra parte, cuando un usuario ya registrado acceda con el botón de Google o Facebook y obtengamos su e-mail e *id*, veremos que ya está registrado y no se procederá a registrarlo de nuevo.

5.2. El caso de Google

En el caso de Google existe una forma muy sencilla de obtener los datos de un usuario. Consiste en acceder a esta URL, proporcionando el token de acceso que nos llegue desde la parte cliente de nuestra aplicación:

https://www.googleapis.com/plus/v1/people/me?access_token=XXXXXX

Como resultado, obtendremos un objeto con varios datos. En el cuerpo (body) de dicho objeto están los datos del usuario, en formato JSON. Para acceder a esta información desde Node podemos procesarla con la librería *http* (o *https* en este caso, al ser peticiones seguras):

```
const http = require('http');
const https = require('https');
const request = require('request');

let atenderPeticion = (req, resp) => {

    if (req.url === '/auth/google' && req.method === 'GET') {
        let token = req.headers['authorization'];
        https.request('https://www.googleapis.com/plus/v1/people/me?
        access_token='+token)
        .on('response', function(res) {
            let body = '';
            res.on('data', function(chunk) {
                body += chunk
            }).on('end', function() {
                let datos = JSON.parse(body);
                console.log(datos);
            });
        })
        .end();
    }
```

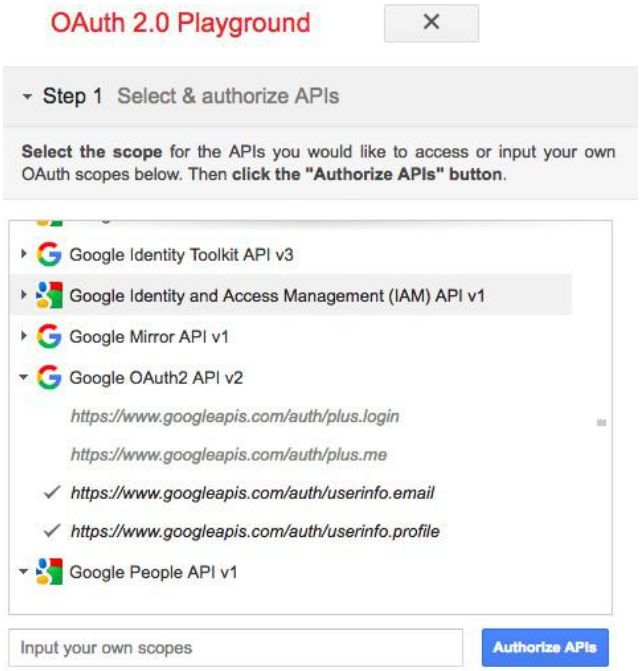
```
};
```

Observa los datos que se devuelven en ese perfil, y trata de obtener los que interesan: *id* de Google, nombre, e-mail, imagen... Para el tema de la imagen, deberás acceder a la URL que se proporciona para la misma, y procesar los bytes que se devuelven como respuesta para guardarla. Puede servirte repetir el proceso anterior en otra función similar con el campo datos.image.url que se obtenga, guardando los datos binarios del cuerpo como imagen:

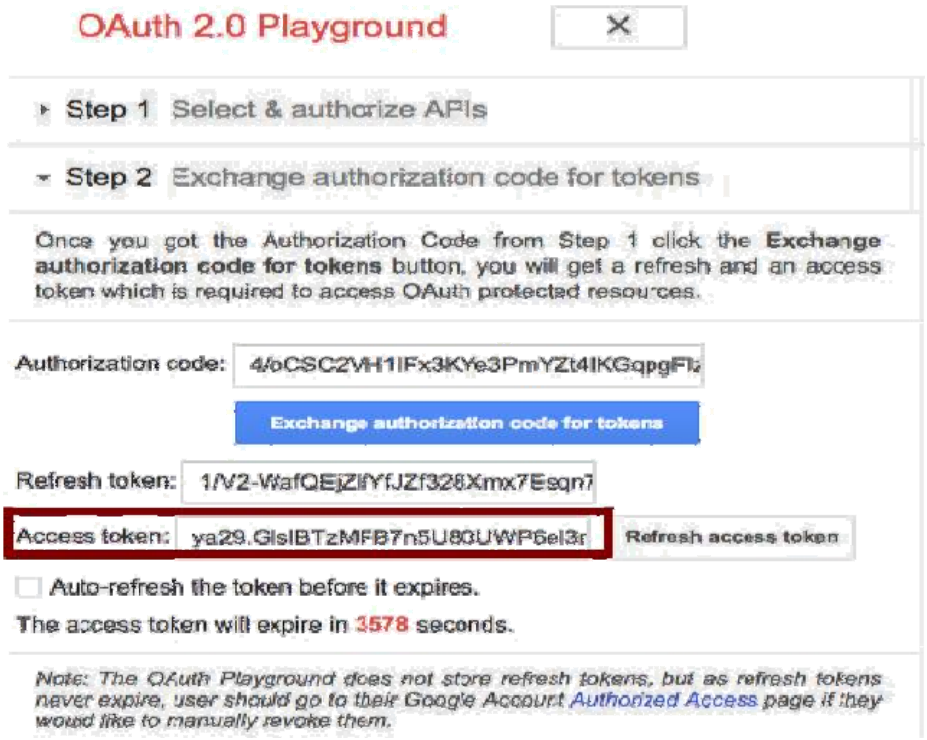
```
let guardarImagen = url => {
	https.request(url)
	.on('response', function(res) {
		let body = '';
		res.setEncoding('binary');
		res.on('data', function(chunk) {
			body += chunk
		}).on('end', function() {
			fs.writeFileSync("nombreArchivo.png", body, 'binary');
		});
	})
	.end();
}
```

5.2.1. Obtener token de acceso de prueba

Para simular que un cliente ha conectado con Google y obtenido el token de acceso para enviar al servidor Node, vamos a emplear una web de Google llamada *OAuth Playground*. Elegiremos la API a autorizar (*Google OAuth2 API v2*) y lo que queremos obtener con ella (el perfil del usuario, y también su e-mail, por ejemplo), y pulsamos en *Authorize APIs*:



En el segundo paso, tras indicar qué cuenta de Gmail vamos a utilizar, podemos obtener el token de acceso pulsando el botón de *Exchange authorization code for tokens*:



El token que aparece en el campo *Access token* es el que tendremos que enviar al servidor Node para que se comunique con Google y obtenga los datos del usuario (en este caso, del usuario que ha simulado la petición desde este *playground*).

5.3. El caso de Facebook

Para Facebook, emplearemos una API propia llamada Graph para acceder al perfil de los usuarios. En cualquier caso, seguiremos unos pasos similares: podemos conectar a una URL y proporcionar un token de acceso para obtener los datos del usuario, pero debemos especificar qué datos son los que queremos obtener concretamente. En esta página podemos consultar la información disponible del usuario, y cómo se llama cada campo. En concreto, nos interesan los campos del *id* de Facebook, nombre del usuario, imagen y e-mail, que se encuentran en los campos id, name, picture y email, respectivamente.

Podemos obtener estos datos con una petición desde nuestro código Node, accediendo a la URL siguiente, e indicando en el parámetro *fields* los datos a obtener, y en el parámetro *access_token* el token de acceso que nos envíe el cliente:

https://graph.facebook.com/v2.11/me?fields=id,name,email,picture&access_token=XXXX

Uniéndolo todo en Node, empleando la librería *https*, nos queda algo así (muy similar al ejemplo de Google):

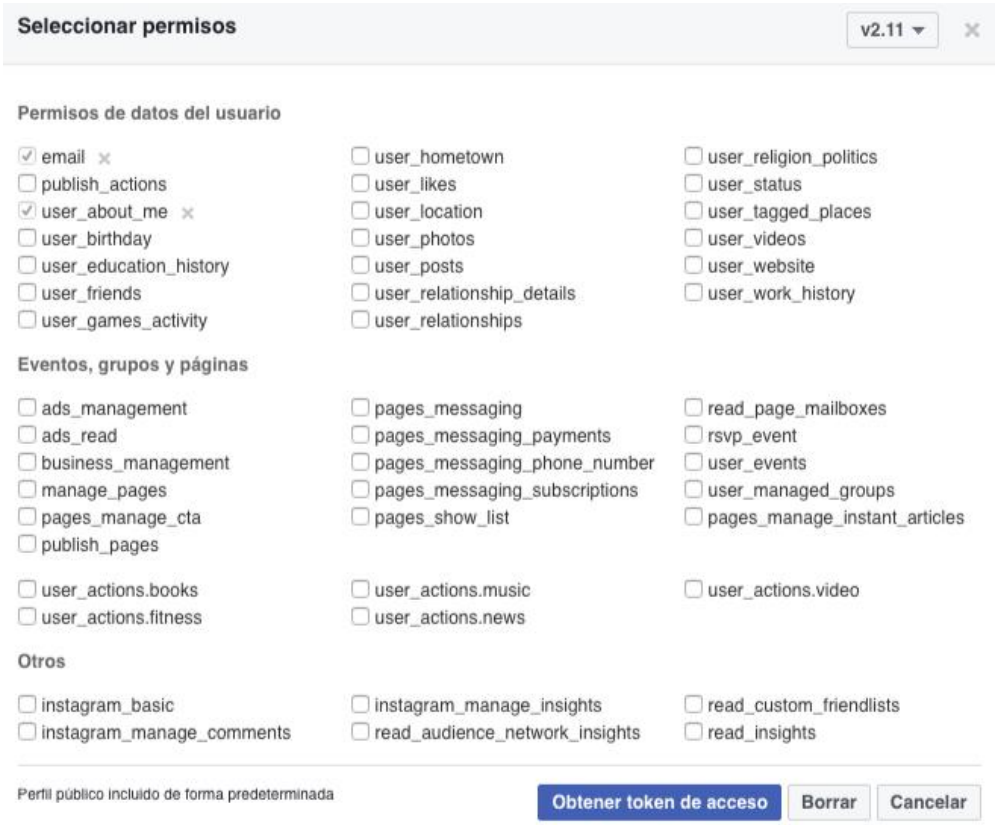
```
if (req.url === '/auth/facebook' && req.method === 'GET') { let token =
        req.headers['authorization'];
        https.request('https://graph.facebook.com/v2.11/me?
        fields=id,name,email,picture&access_token=' + token)
        .on('response', function(res) {
                let body = '';
                res
                .on('data', function(chunk) {
                        body += chunk
                })
                .on('end', function() {
```

```
                let datos = JSON.parse(body);
                console.log(datos);
            });
    })
    .end();
}
```

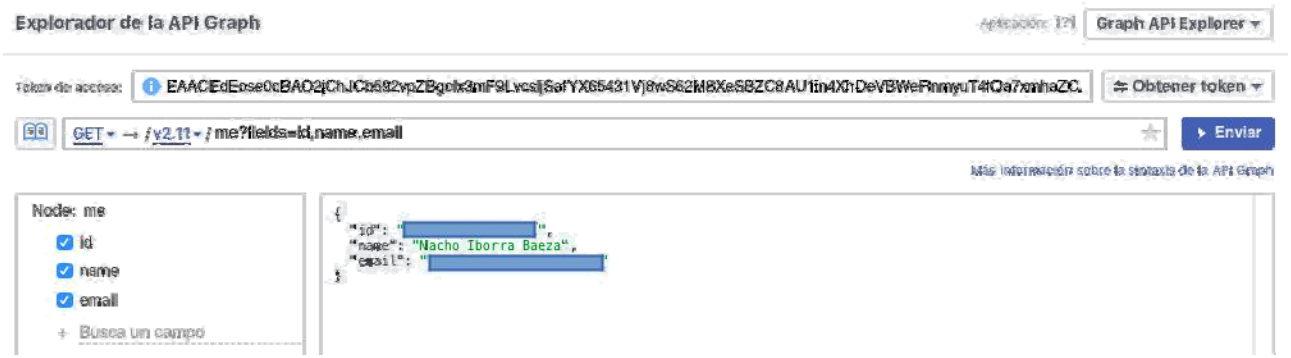
5.3.1. Obtener token de acceso de prueba

Podemos también obtener un token de acceso de prueba para un usuario a través del explorador de Graph:

Si pulsamos en el botón de "Obtener token" de la parte derecha, podemos indicar qué elementos queremos obtener con el token (en principio, nos basta con el email y los datos generales sobre el usuario):



Al pulsar en *Obtener token de acceso* aparecerá el token en la barra superior de la ventana anterior. Nos faltará indicar en la URL qué información queremos obtener. En nuestro caso, el id, nombre, imagen y email:



En el caso de la imagen (*picture*), deberemos acceder al atributo picture.data.url para obtener la URL de la imagen, y podemos emplear la función guardarImagen vista para el caso de Google para guardarla a partir de dicha URL.