Práctica de Passport

Autenticación de usuarios usando google

En esta práctica usaremos Passport para autenticarnos a través de la API de google. Además, veremos el flujo que se requiere para mantener la sesión.

Descripción de la práctica

Realizar una aplicación básica que implique el uso de autenticación por terceros en este caso Google.

Autenticación por OAuth 2.0

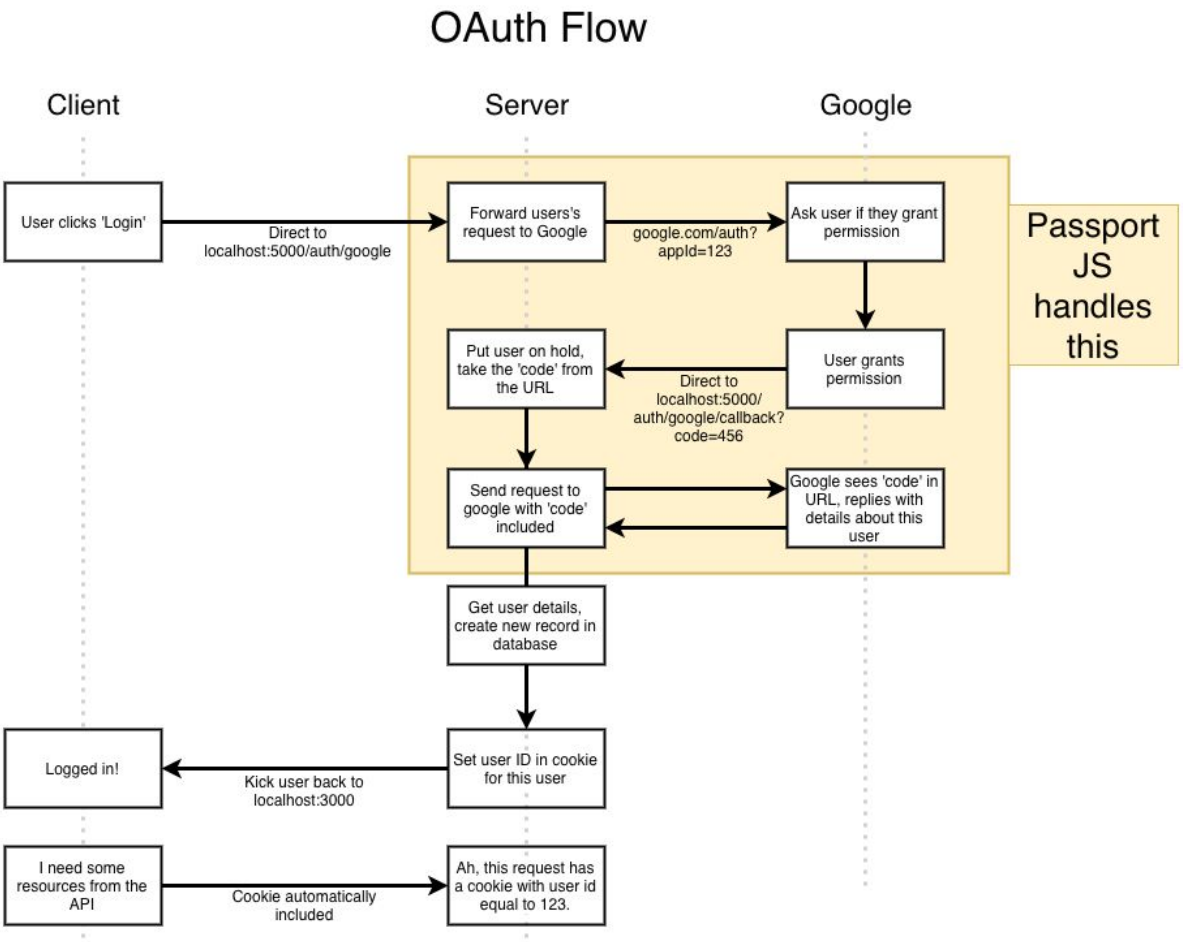
1. Estructura de carpetas y configuración inicial
2. Obtener credenciales de google
3. Configuración inicial de passport
4. Callback y almacenamiento de usuario
5. Serializar/deserializar y cookies
6. Flujo completo logout
7. Añadir estilo a ventanas (ver perfil con imagen)
8. Completar con base de datos
9. Conclusiones

Autenticación a Google por OAuth

El esquema de OAuth2 (Open Autorization) es una estructura de autorización que permite que una aplicación tener acceso a cuentas de terceros como Facebook, Google, etc. La autenticación es realizada por el tercero, quien proporciona las credenciales que validan al usuario.

No es lo mismo autenticación que autorización, un usuario puede estar autenticado por Google (Google reconoce que es su usuario) pero no está autorizado para usar Google Drive. En esta práctica nos enfocamos en autenticación.

Para entender a grandes rasgos lo que vamos a hacer debemos ver la siguiente imagen que refleja los pasos en los que estaremos trabajando.



https://docs.mikelgoig.com/nodejs/autenticacion.html#google-oauth

Conviene durante el transcurso de la práctica ubicarte en este diagrama para que vayas comprendiendo que vamos haciendo.

Passport.js es una librería que nos permite autenticarnos creando flujos conocidos como estrategias, donde cada API de terceros tiene su propia estrategia.

Passport.js también permite que tu crees tu propia estrategia y te facilita mucho la autenticación en tu aplicación.

Emplearemos también en esta práctica:

Cookie-session: que nos ayuda a mantener la sesión de nuestra aplicación encriptando el id del usuario y usando cookies.

1. Estructura básica de la aplicación

Para esta aplicación dividiremos los archivos en diferentes carpetas de la siguiente forma

El proyecto se llama adopt-me para ello es necesario con la práctica anterior.

1. Prueba que funciona

   Para facilitar las cosas primero usaremos un archivo .json como base de datos pero al final deberás cambiarlo por tu base de datos.

   En **auth.js** crear

- GET /auth/login render (login.hbs que tiene un formulario dummy y un link para acceder por google que redirige a /auth/google)
- GET /auth/google/login regresa un texto
- GET /auth/google/redirect regresa un texto
- GET /auth/logout regresa texto

En **profile.js** crear

- GET /profile (renderiza profile.hbs puedes crearlo de una vez)

En **index.js**

- GET / (renderiza home.hbs)

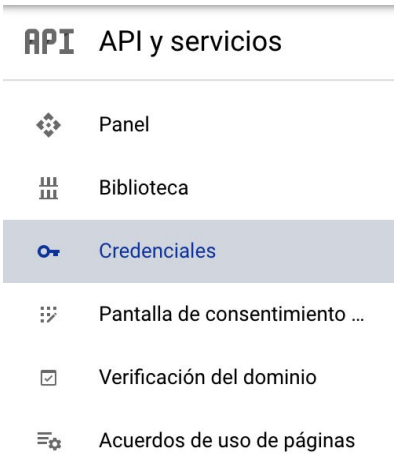
Finalmente, en index.js añade el router de auth a la ruta /auth.

Para probar crea un pequeño navbar que puedan navegar a las diferentes rutas.

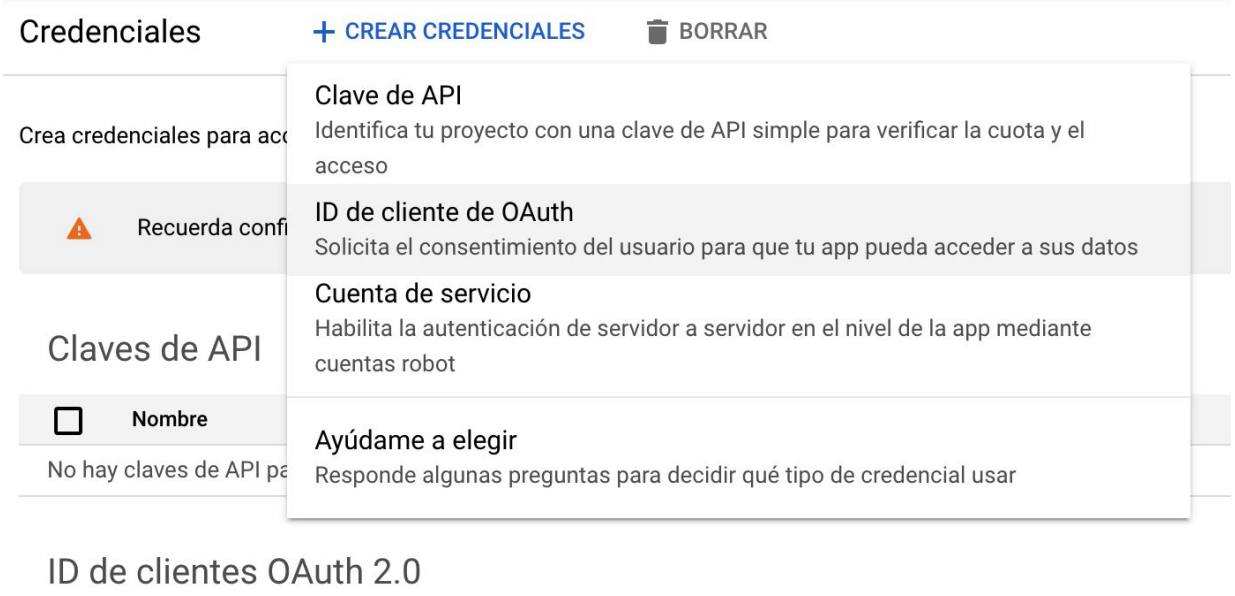
2. Obtener credenciales de google

Accede a https://console.developers.google.com/ y registra un proyecto (ponle nombre) y presiona crear.

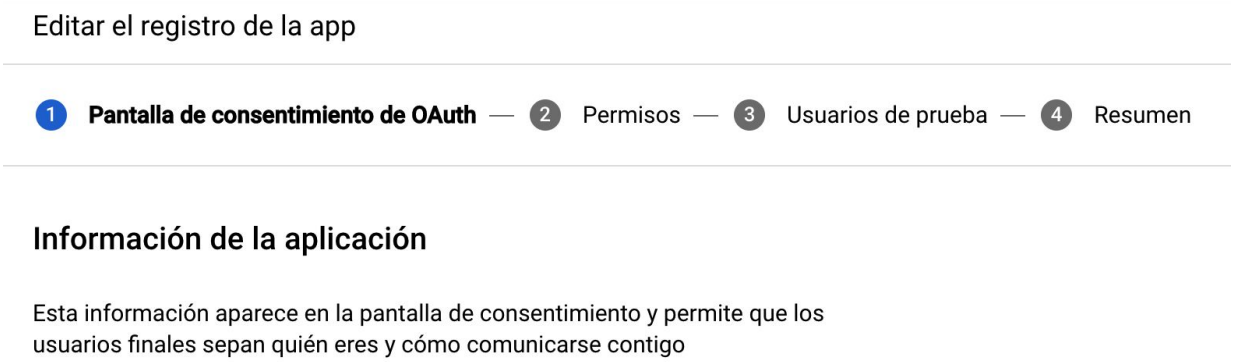
Busca y presiona en credenciales



Selecciona Crear Credencial > ID de cliente de OAuth



Si te pide que llenes la página de consentimiento



Llena solo (Nombre de la aplicación, Correo electrónico de asistencia del usuario y Direcciones de correo electrónico) y dar clic en Guardar y continuar.

Llenar la información para la creación del client OAuth

Un ID de cliente se usa con el fin de identificar una sola app para los servidores de OAuth de Google. Si la app se ejecuta en varias plataformas, cada una necesitará su propio ID de cliente. Consulta Configura OAuth 2.0 para obtener más información.

Tipo de aplicación *

Aplicación web

Más información sobre los tipos de clientes de OAuth

Nombre *

Cliente web 1

El nombre de tu cliente de OAuth 2.0. Este nombre solo se usa para identificar al cliente en la consola y no se mostrará a los usuarios finales.

Los dominios de los URI que agregues a continuación se incorporarán automáticamente a tu pantalla de consentimiento de OAuth como dominios autorizados.

Orígenes autorizados de JavaScript

Para usar con solicitudes de un navegador

+ AGREGAR URI

URI de redireccionamiento autorizados

Para usar con solicitudes de un servidor web

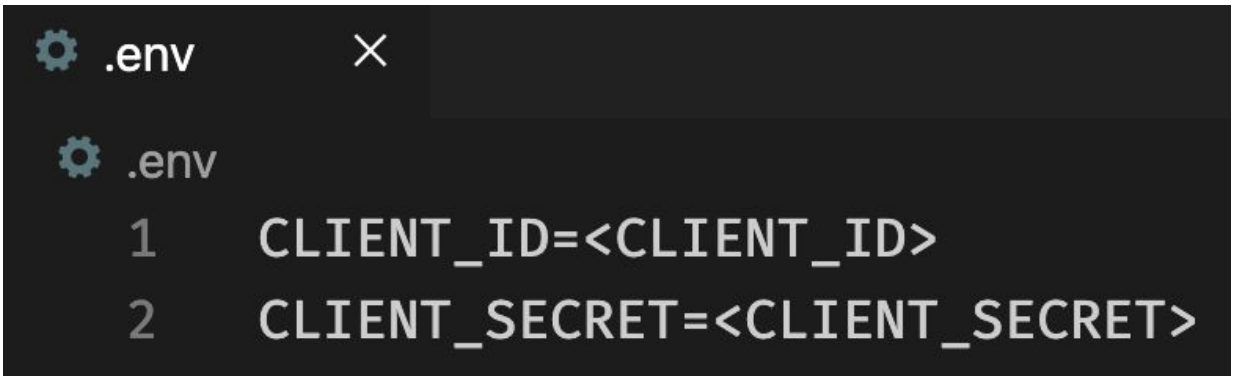
URI

http://localhost:3000/auth/google/callback

http://localhost:3000/auth/google

+ AGREGAR URI

Copiar valores Client ID y Client Secret, y guardarlos en el archivo .env



IMPORTANTE

No debes de hacer track del archivo .env, es decir, debe de estar en tu .gitignore files

Nunca se debe subir a github el client id ni el client secret

3. Configuración de passport

En tu aplicación instala passport y passport-google-oauth-20

```
npm install passport passport-google-oauth20 dotenv cookie-session --save
```

En config/passport.js

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth20').Strategy;
passport.use(
  new GoogleStrategy(
    {
      clientID: process.env.CLIENT_ID,
      clientSecret: process.env.CLIENT_SECRET,
      callbackURL: 'http://localhost:3000/auth/google/callback',
    },
    function (accessToken, refreshToken, profile, done) {
      console.log('working');
    }
  )
);
```

En index.js

```
require('dotenv').config();
require('./config/passport');
```

En auth.js

```
router.get('/google', passport.authenticate('google', { scope: ['profile',
```

```
 'email'] }));
```

Prueba que al hacer login te redirige.

En get /google/redirect imprime en consola el req.query.code y haz redirect a "/"

```
router.get(
  '/google/callback',
  passport.authenticate('google', { failureRedirect: '/login' }),
  function (req, res) {
    // print req.query.code
    // Successful authentication, redirect home.
  }
);
```

4. Callback y almacenamiento

Ahora que hemos recibido ese código para autenticarnos tenemos que solicitar a google el perfil para ello en el mismo redirect usaremos el middleware de passport.

```
router.get(
  '/google/callback',
  passport.authenticate('google', { failureRedirect: '/login' }),
  function (req, res) {
    // print req.query.code
    // Successful authentication, redirect home.
  }
);
```

Lo que debes hacer aquí es guardar el usuario en el archivo, nos interesan 4 atributos id timestamp, googleId: (profile.id), email y imageUrl. Observa el profile y obtén esta información.

Antes de guardar asegúrate que no exista el usuario (no crees 2 usuarios)

1. Buscar profile.id entre los usuarios
2. Si existe solamente regresar done(null, findUser) //null porque no hay errores
3. Si no existe guardar nuevo usuario en el archivo y done(null, newUser)

Prueba que guarda, aunque todavía no se pueda continuar con la práctica

5. Serializar y desearializar

Ahora estaremos usando una sesión que guardará el id del usuario (no el id de google), esto se realiza de la siguiente manera.

En passport.js

```
passport.serializeUser(function (user, done) {
  done(null, user.id);
});

passport.deserializeUser(function (id, done) {
  const user = users.find(id)
  .then(user => done(null,user));
});
```

En index.js

```
const cookieSession = require('cookie-session');
const passport = require('passport');

app.use(cookieSession({
  maxAge: 24 * 60 * 60 * 1000,
  keys: ['clave'] //clave para encriptar
}))

//inicializar passport
app.use(passport.initialize());
app.use(passport.session());
```

Observa que ya se puede leer el user en el request req.user en /profile

6. Flujo completo y logout

Falta completar el flujo para ello será necesaria una ruta para salir /logout

Para ello passport nos facilita con solo poner req.logout(), req.session = null y puedes hacer redirección a la raíz.

El flujo debe ser el siguiente:

1. Entrar a Home, ver login (si no está logueado) o ver (logout si está logueado)
2. Al entrar a login ver el formulario (que no hace nada) y el botón de google
3. Al entrar al boton de google se loguea, manda solo a google/callback y nos redirije a /profile que imprime en consola el perfil
4. En profile se revisa con el middleware que este autenticado (exista user) y de no ser así nos manda a /login (al formulario), si está autenticado imprime en consola el perfil y muestra algo de contenido de “este es el perfil”.

Si el usuario ya existe no lo crea y si no existe lo guarda en la base de datos

7. Dar estilo a las vistas y perfil con imagen

Muestra tu ventana de perfil con imagen y su correo y además modifica el código para que se incluya también el nombre del usuario.

8. Completar con base de datos (vale por 2)

En lugar de guardar en un archivo .json guarda los usuarios usando alguna base de datos (postgres, mysql, mongodb, etc)

9. Entregables

1. Video con el flujo completo (sección 6)
2. Una url con el repo de github, puede ser el mismo github de la entrega pasada

3. Referencias extras

1. Implementación completo de flujo
2. Documentación de passport
3. Video explicación oauth2.0
4. Explicación OAuth2.0