

Tema 9

Gestión de usuarios

Micael Gallego
micael.gallego@gmail.com
[@micael_gallego](https://twitter.com/micael_gallego)

Gestión de usuarios

- Introducción
- Comunicaciones seguras
- Autenticación básica
- Gestión de contraseñas
- Autenticación con tokens
- OAuth2

Introducción

- Las APIs REST que hemos creado hasta ahora son **públicas**
- Pueden usarse por **cualquier usuario/cliente** que conozca la IP o dominio y las rutas
- **No se puede identificar** al usuario/cliente que hace la petición

Introducción

- Habitualmente se **controla el acceso** a las APIs REST
 - Se **limitan las operaciones** que ciertos usuarios pueden realizar (sólo lectura, creación/modificación/borrado)
 - Para **limitar su abuso** se limitan las peticiones de un mismo usuario por minuto

Introducción

Autorización vs Autenticación

Introducción

- Autenticación

- Proceso de **verificar una identidad**
- Confirmar que una persona es quien dice ser
- El usuario **usa algo que conoce** para demostrar su identidad, como un usuario y una contraseña

Introducción

• Autorización

- Proceso de verificar lo que un usuario puede hacer.
- Por ejemplo, un usuario puede añadir canciones en una lista compartida de Spotify, pero no puede eliminar dicha lista.
- La autorización ocurre después de que un usuario se haya autenticado.

Gestión de usuarios

- Introducción
- **Comunicaciones seguras**
- Autenticación básica
- Gestión de contraseñas
- Autenticación con tokens
- OAuth2

Comunicaciones seguras

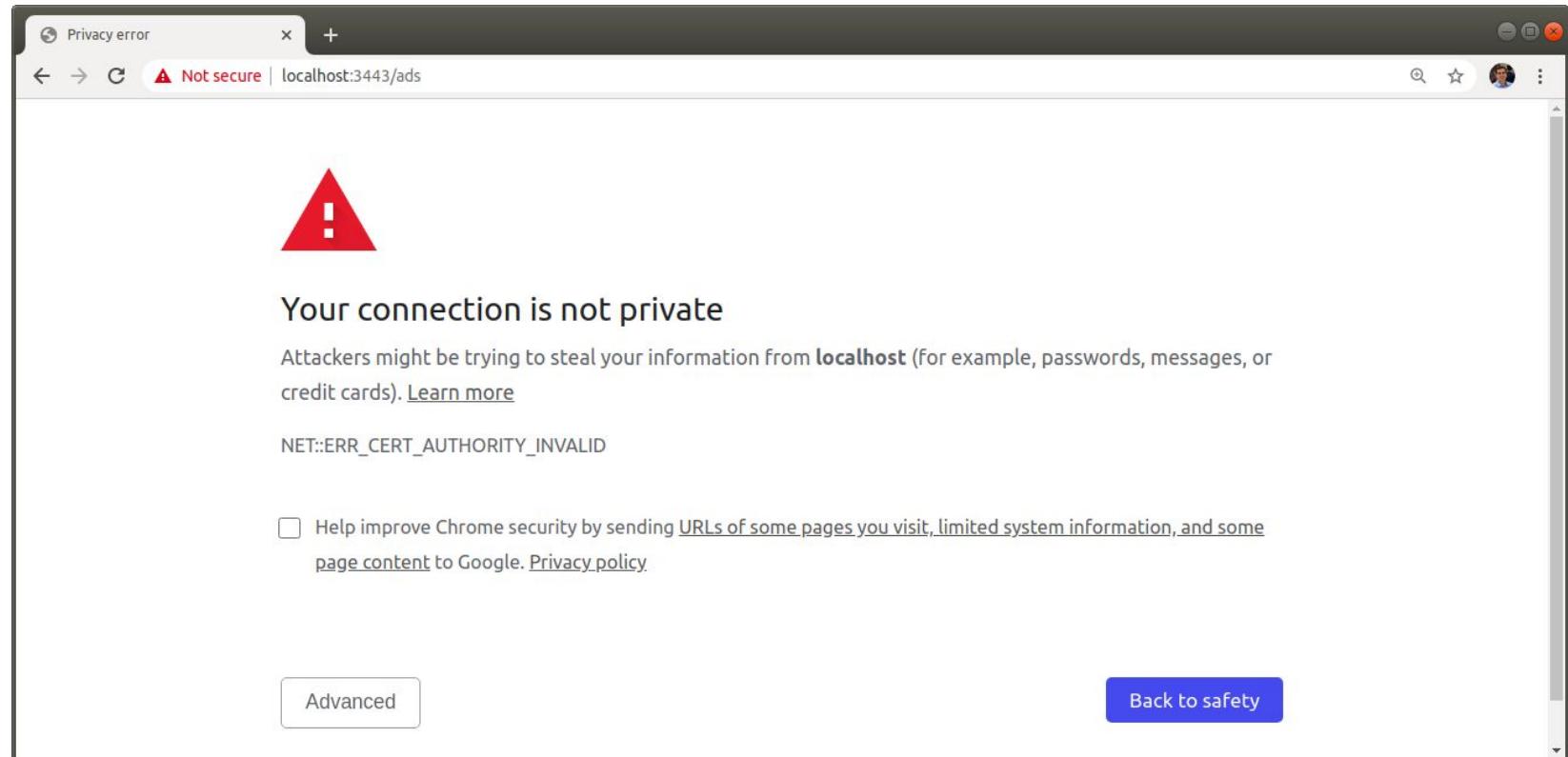
- La forma más sencilla de **autenticar** un usuario en una **API REST** es enviar el nombre y la contraseña en cada petición
- La **comunicación** debe estar **cifrada** para evitar que un observador malicioso pueda ver esas **credenciales**
- Para cifrar las comunicaciones se usa **https**

Comunicaciones seguras

- Una comunicación cifrada **https** necesita certificados SSL
 - Se pueden **generar INSEGUROS** localmente (sólo para desarrollo)
 - Se pueden solicitar **gratuitamente** a Let's Encrypt (es necesario un dominio)
 - Se pueden **comprar** acompañados de servicios adicionales como seguros (Twate, Verisign...)

Comunicaciones seguras

- Certificado inseguro



Comunicaciones seguras

- **Certificado inseguro para desarrollo**

- Se generan dos ficheros, uno con la clave pública (server.cert) y otro con la clave privada (server.key)

```
$ openssl req -nodes -new -x509 \
-keyout server.key -out server.cert
```

- Se indican los valores solicitados (pueden ser ficticios)

Comunicaciones seguras

- Certificados gratuitos con Let's Encrypt

- Empezó a funcionar en 2016
- Antes los certificados eran comerciales
- Desarrollado por el Internet Security Research Group (ISRG)



<https://letsencrypt.org/>

Comunicaciones seguras

- Certificados comerciales



VERISIGN®



Comunicaciones seguras

ejem2

• Configuración https en Express

```
const express = require('express');
var fs = require('fs');
var https = require('https');

const app = express();

app.get(...);

https.createServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.cert')
}, app).listen(3443, () => {
  console.log("Https server started in port 3443");
});
```

Ejercicio 1

- Añade https al ejemplo 6 del tema 8 (API REST de Anuncios en Mongo con Mongoose)

Gestión de usuarios

- Introducción
- Comunicaciones seguras
- Autenticación básica
- Gestión de contraseñas
- Autenticación con tokens
- OAuth2

Autenticación básica

- La forma más sencilla de autenticar a un usuario que usa una API REST es incluir el **login** y la **password** en cada petición
- Para ello se usa el mecanismo conocido como **Autenticación de acceso básico** (*basic access authentication* o *basic auth*) de http

https://en.wikipedia.org/wiki/Basic_access_authentication

Autenticación básica

- Las credenciales (login y password) se envían en la cabecera **Authorization**
- El valor se calcula como:
`'Basic ' +base64(login+':'+password)`
- Ejemplo para login “Aladdin” y password “OpenSesame”

```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11
```

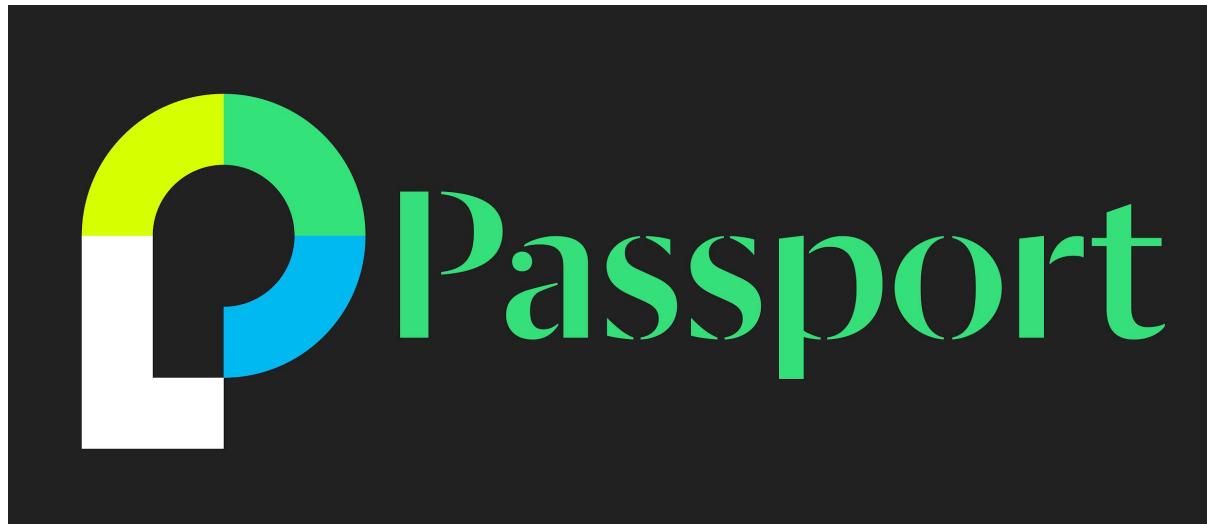
Autenticación básica

- En Postman se especifica el tipo de Autorización “Basic Auth”, el login y la password y él genera la cabecera

The screenshot shows the Postman interface for a GET request to `http://localhost:3000/ads/`. The 'Authorization' tab is selected, indicating 'Basic Auth' is chosen. The 'Username' field contains 'admin' and the 'Password' field contains '****'. A 'Show Password' checkbox is unchecked. A note on the left explains that the authorization header will be generated automatically when the request is sent.

Autenticación básica

- Autenticación de usuarios con Express



Simple, unobtrusive authentication for Node.js

<http://www.passportjs.org/>

Autenticación básica

- **Passport**

- Permite diferentes mecanismos (*strategies*) para autenticación de usuarios en APIs REST y webs MVC implementadas con express
 - Basic Auth,
 - OAuth2
 - OpenID
 - Login con servicios (Twitter, GitHub, Google...)
 - ... (500 estrategias disponible)

Autenticación básica

ejem3

- **Basic Auth con Passport**

- Instalación de passport

```
$ npm install --save passport
```

- Soporte de basic auth

```
$ npm install --save passport-http
```

Autenticación básica

ejem3

- **Basic Auth con Passport**

- Se configura la estrategia basic auth

```
const passport = require('passport');
const BasicStrategy = require('passport-http').BasicStrategy;
...
function verify(username, password, done) {
    if (username == 'admin' && password == 'pass') {
        return done(null, { username, password });
    } else {
        return done(null, false, { message: 'Incorrect username or password' });
    }
}

passport.use(new BasicStrategy(verify));
app.use(passport.initialize());
...
```

En una app real,
los usuarios están
en una base de
datos

Autenticación básica

ejem3

- **Basic Auth con Passport**

- Se define qué rutas están “protegidas”

Cada estrategia tiene su propio identificador

```
app.get('/ads',
  passport.authenticate('basic', { session: false }),
  (req, res) => {

  console.log('Logged user:', req.user);

  var sampleAds = ...

  res.json(sampleAds);
});
```

En las APIs REST es preferible no gestionar sesión http (mediante cookies) en el servidor

En req.user se tiene acceso al objeto user que haya configurado la función “verify”

Ejercicio 2

- Añade Basic Auth al Ejercicio 1 con un usuario admin (password = pass) definido en el código

Gestión de usuarios

- Introducción
- Comunicaciones seguras
- Autenticación básica
- **Gestión de contraseñas**
- Autenticación con tokens
- OAuth2

Gestión de contraseñas

- Las contraseñas de los usuarios es una información muy **sensible**
- Si hay una **brecha de seguridad** y un atacante accede a las **contraseñas**, se podría hacer pasar por cualquier usuario
- Existen técnicas para autenticar usuarios pero **sin guardar las contraseñas** en las BBDD

Gestión de contraseñas

- Las funciones criptográficas de tipo **hash(...)** convierten cualquier información (password, documentos...) en una ristra de números
- Dada esa ristra de números (**el hash o resumen**), es **imposible*** obtener de nuevo la información original

* Imposible sin la potencia de cómputo suficiente para un ataque por fuerza bruta

Gestión de contraseñas

- Estrategia de protección de password
 - **Guardar el hash** de la password en BBDD
 - Un atacante no puede obtener la password conociendo el hash
 - Para autenticar a un usuario, se **calcula el hash de la password introducida** y se **compara** con el hash guardado.

Gestión de contraseñas

- Existen muchas **funciones hash** basadas en diferentes mecanismos matemáticos
- Funciones hash más famosas (algunas ya **vulnerables**)
 - MD5 (<https://en.wikipedia.org/wiki/MD5>)
 - SHA1 (<https://en.wikipedia.org/wiki/SHA-1>)
 - BCrypt (<https://en.wikipedia.org/wiki/Bcrypt>)

Gestión de contraseñas

ejem4

• Uso de BCrypt en Node

- Dependencia

```
$ npm install --save bcrypt
```

```
var bcrypt = require('bcrypt');
```

- Obtener el hash de la password

```
var passwordHash = await bcrypt.hash(password, bcrypt.genSaltSync(8), null);
```

- Comparar password introducida con el hash

```
await bcrypt.compare(password, user.passwordHash);
```

Gestión de contraseñas

ejem4

```
var bcrypt = require('bcrypt');

const users = new Map();

async function createSampleUsers(){ ... }

createSampleUsers();

async function addUser(username, password){
  var passwordHash = await bcrypt.hash(password, bcrypt.genSaltSync(8), null);
  users.set(username, { username, passwordHash });
}

exports.find = async function(username){
  return users.get(username);
}

exports.verifyPassword = async function(user, password){
  return await bcrypt.compare(password, user.passwordHash);
}
```

Ejemplo con los usuarios guardados en memoria.

En una app real, los usuarios están en la base de datos

Gestión de contraseñas

ejem5

• Usuarios en MongoDB con Mongoose

```
async function addUser(username, password) {
    var passwordHash = await bcrypt.hash(password, bcrypt.genSaltSync(8), null);

    var user = await User.findOne({ username }).exec();

    if(!user){
        user = new User({ username, passwordHash });
    } else {
        user.passwordHash = passwordHash;
    }

    await user.save();
}

exports.init = async function() { ... }

exports.find = async function (username) {
    return await User.findOne({ username }).exec();
}

exports.verifyPassword = async function (user, password) {
    return await bcrypt.compare(password, user.passwordHash);
}
```

Ejercicio 3

- Añade los usuarios a la base de datos Mongo en el Ejercicio 2

Gestión de usuarios

- Introducción
- Comunicaciones seguras
- Autenticación básica
- Gestión de contraseñas
- **Autenticación con tokens**
- OAuth2

Autenticación con tokens

- La autenticación **enviando la contraseña en cada petición** no es recomendable
- Si un atacante llegase a **interceptar una única petición**, se podría hacer pasar por el usuario
- Es recomendable usar la contraseña una única vez para obtener un **token** con el que poder hacer las siguientes peticiones
- Ese token suele tener **caducidad**, limitando el impacto en caso de ataque

Autenticación con tokens

- **Paso 1) Se hace una petición con username y contraseña para obtener el token**
 - Se puede usar Basic Auth para autenticar
 - Se genera un token y se devuelve
- **Paso 2) Se hace una petición con el token**
 - Se verifica si el token es “válido”
 - Si lo es, se atiende la petición

Autenticación con tokens

- **Tipos de tokens**
 - Número aleatorio grande difícil de adivinar
 - Fácil de generar
 - Se mantiene una lista de tokens válidos asociados al nombre de cada usuario
 - Es el funcionamiento de las cookies de sesión

Autenticación con tokens

- **Tipos de tokens**
 - **Número aleatorio grande difícil de adivinar**
 - En sistemas distribuidos la gestión de la lista de tokens puede limitar la escalabilidad
 - Lo ideal es que el propio token tuviese todo lo necesario para saber si es válido o no

Autenticación con tokens

- **Tipos de tokens**
 - **JSON firmado con tiempo de expiración**
 - El token puede llevar información como **username**, rol, permisos
 - Como el token está **firmado**, su contenido no se puede alterar (o se detectaría)
 - **Elimina** la necesidad de mantener una lista de tokens válidos y usernames
 - El token se **invalida** cuando caduca

Autenticación con tokens

- JSON Web Tokens



JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

<https://jwt.io/>

Autenticación con tokens

- **JSON Web Tokens**

- Los tokens JWT pueden cifrarse o no
- Pueden usarse en muchos contextos, son muy flexibles
- Un servidor puede generar un token y verificarse en otro servidor
- Los usaremos para autenticar peticiones

<https://jwt.io/>

Autenticación con tokens

ejem6

- Ejemplo
 - /login
 - Endpoint POST para obtener el token
 - Autenticación Basic Auth
 - /ads
 - Endpoint GET para consultar información
 - Autenticación con token JWT
 - Enviado en cabecera Authentication: Bearer <token>

Autenticación con tokens

- **/login**

```
async function verify(username, password, done) { ... }

passport.use(new BasicStrategy(verify));

app.post("/login",
  passport.authenticate('basic', { session: false }),
  (req, res) => {

    const { username } = req.user;

    const opts = { expiresIn: 120 }; //token expires in 2min
    const token = jwt.sign({ username }, SECRET_KEY, opts);

    return res.status(200).json({ message: "Auth Passed", token });

});
```

Se genera el token JWT con el username como payload y se devuelve en la petición



Autenticación con tokens

ejem6

- /ads

```
const jwtOpts = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: SECRET_KEY
}

passport.use(new JwtStrategy(jwtOpts, async (payload, done) => {
  ...
})) );

app.get('/ads',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {

  console.log('Logged user:', req.user);
  var sampleAds = ...
  res.json(sampleAds);
});
```

Se puede realizar alguna verificación adicional o cargar datos extra que no estén en el token



Autenticación con tokens

ejem6

- /ads

```
passport.use(new JwtStrategy(jwtOpts, async (payload, done) => {  
  
    var user = await users.find(payload.username);  
  
    if (user) {  
        return done(null, user);  
    } else {  
        return done(null, false, { message: 'User not found' });  
    }  
  
})) );
```

En el ejemplo se carga el usuario completo de la base de datos

Ejercicio 4

- Añade autenticación con tokens JWT al Ejercicio 3

Gestión de usuarios

- Introducción
- Comunicaciones seguras
- Autenticación básica
- Gestión de contraseñas
- Autenticación con tokens
- OAuth2

OAuth2

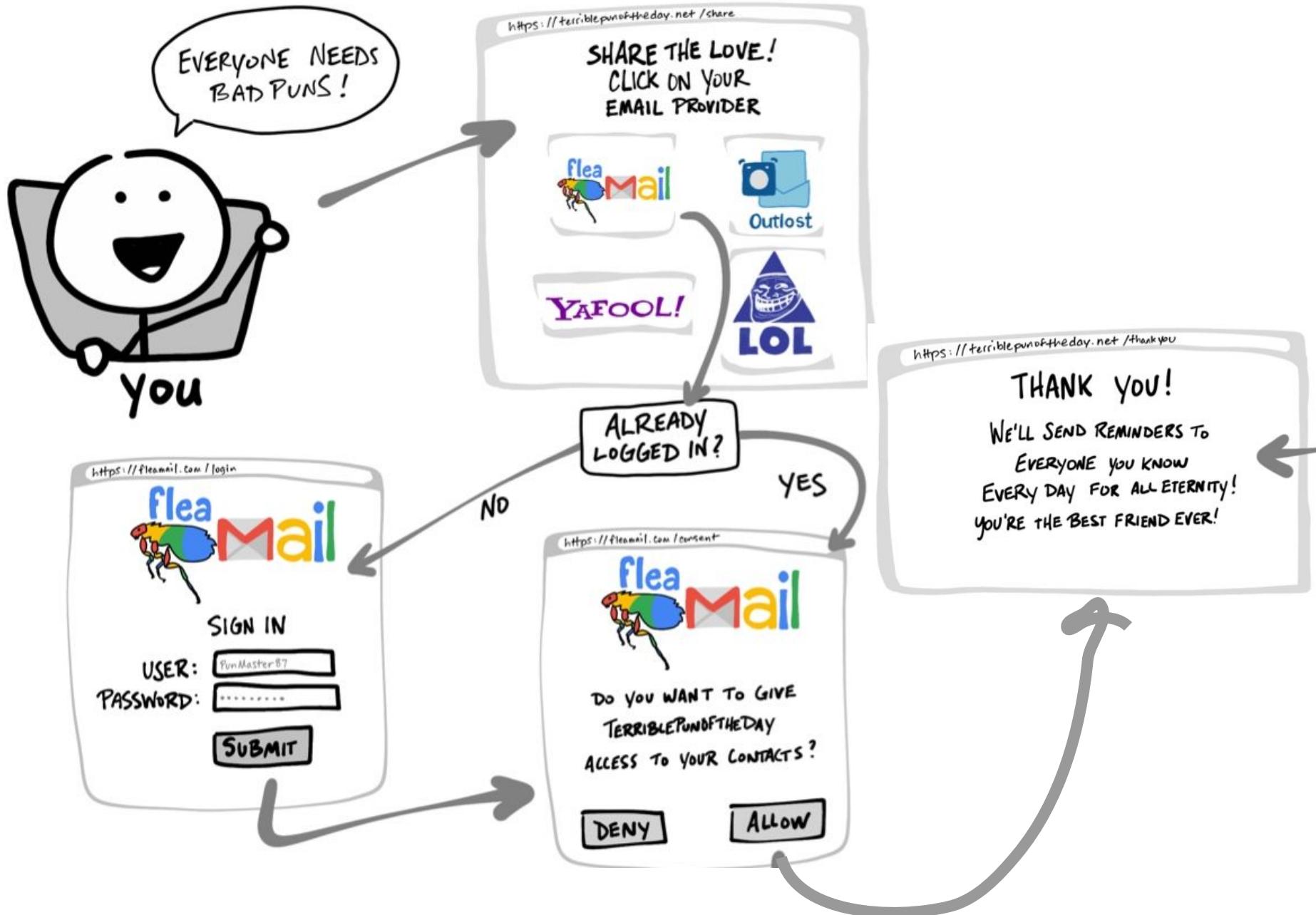
- OAuth2 es un estándar de seguridad que permite a un usuario **conceder permisos** a una aplicación para que acceda a **sus datos** en otra aplicación

<https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc>

OAuth2

- Supongamos que una aplicación “**Terrible pun of the day**” quiere enviar un chiste a los contactos de tu agenda del proveedor de correo

<https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc>



OAuth2

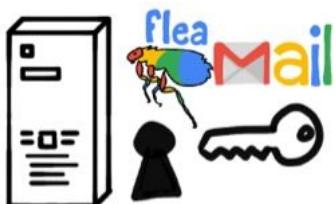
- Hemos visto un flujo (*flow*) de OAuth.
- En concreto el flujo “**authorization code**”
- Existen otros tipos de flujos para otras situaciones
- OAuth define un protocolo de intercambio de **mensajes** con ciertos **contenidos** entre las aplicaciones

OAuth2

• Conceptos



- **Resource owner:** El usuario
- **Client:** La aplicación que quiere los datos del usuario ("Terrible pun of the day")
- **Authorization server:** La aplicación en la que el usuario está registrado (fleamail)



OAuth2

• Conceptos

- **Resource server:** Lugar donde están los datos del usuario que serán usados por el cliente.
- **Redirect URI:** La URL a la que el usuario será redirigido cuando acepte que el cliente use sus datos

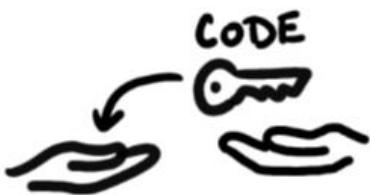


<https://terriblepunoftheday.net/callback>

OAuth2

• Conceptos

- **Response type:** El tipo de información que el cliente espera recibir del Authorization server. El más común es “code” cuando el cliente espera un **Authorization Code**
- **Scope:** Permisos que el cliente quiere

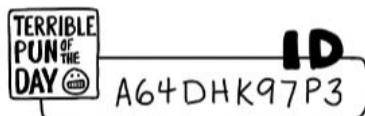
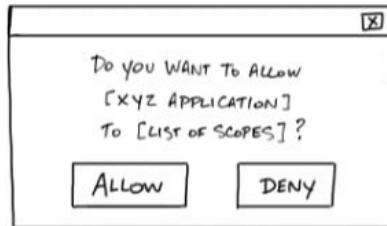


READ CONTACTS
 CREATE CONTACT
 DELETE CONTACT
 READ PROFILE

OAuth2

• Conceptos

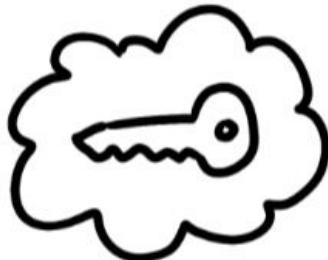
- **Consent:** ¿El usuario (resource owner) da su consentimiento para que el cliente acceda a los datos indicados por el scope?
- **Client ID:** Identificador del cliente en el Authorization server
- **Client Secret:** Password del cliente en el Authorization server



OAuth2

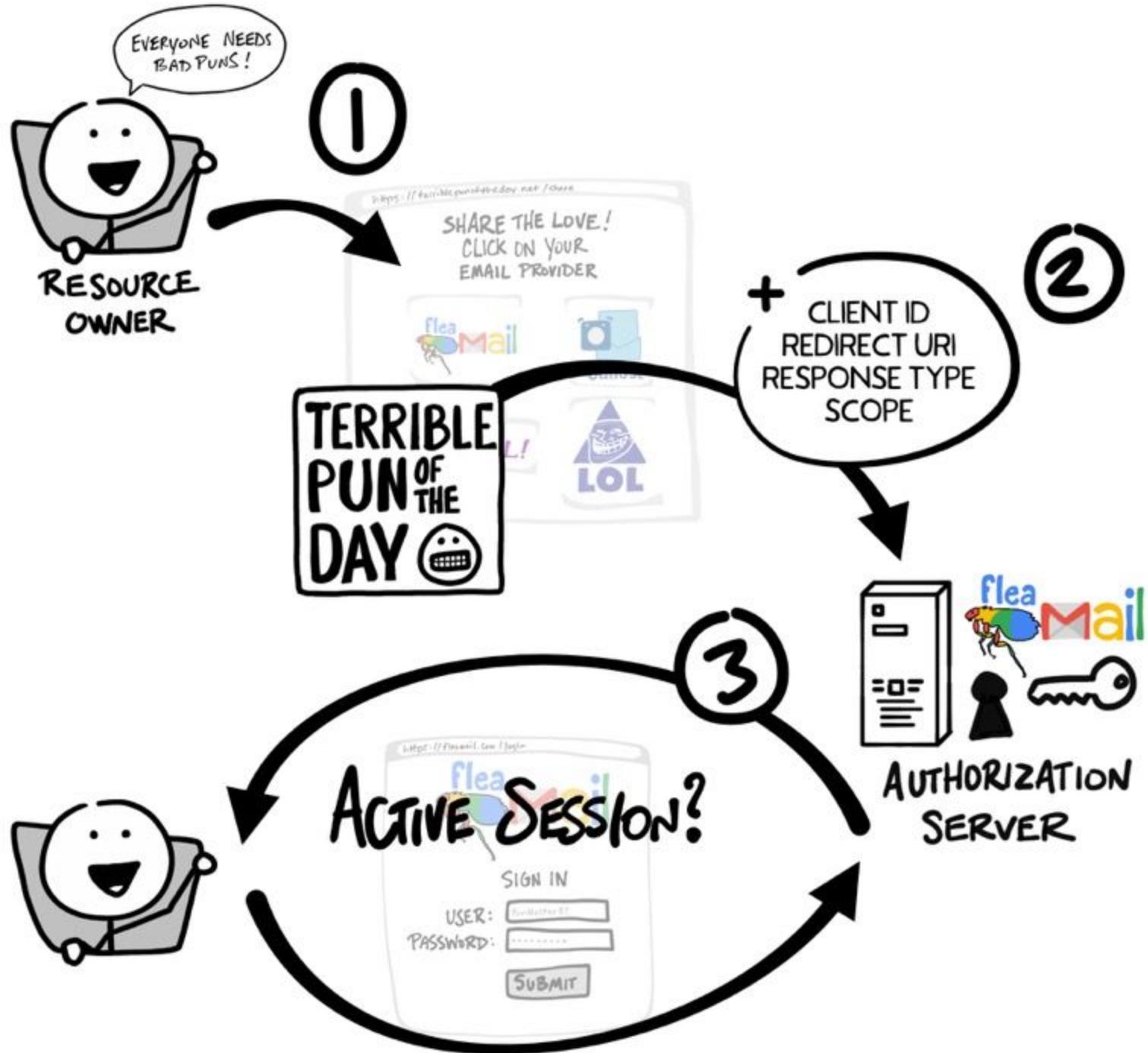
• Conceptos

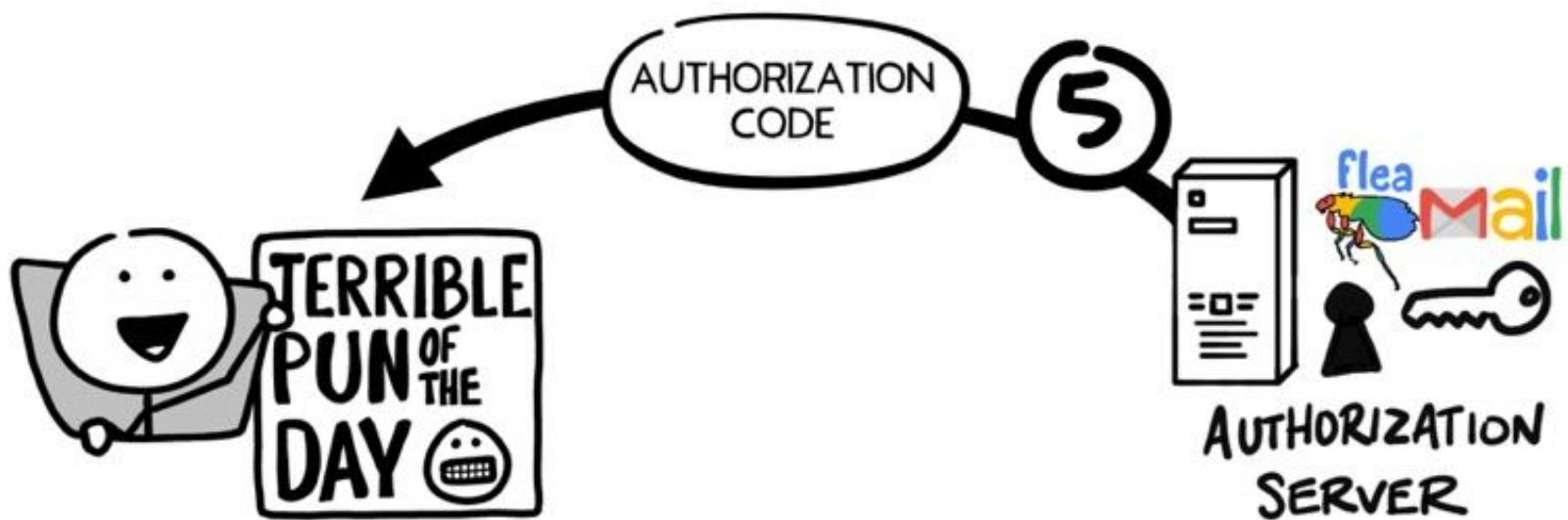
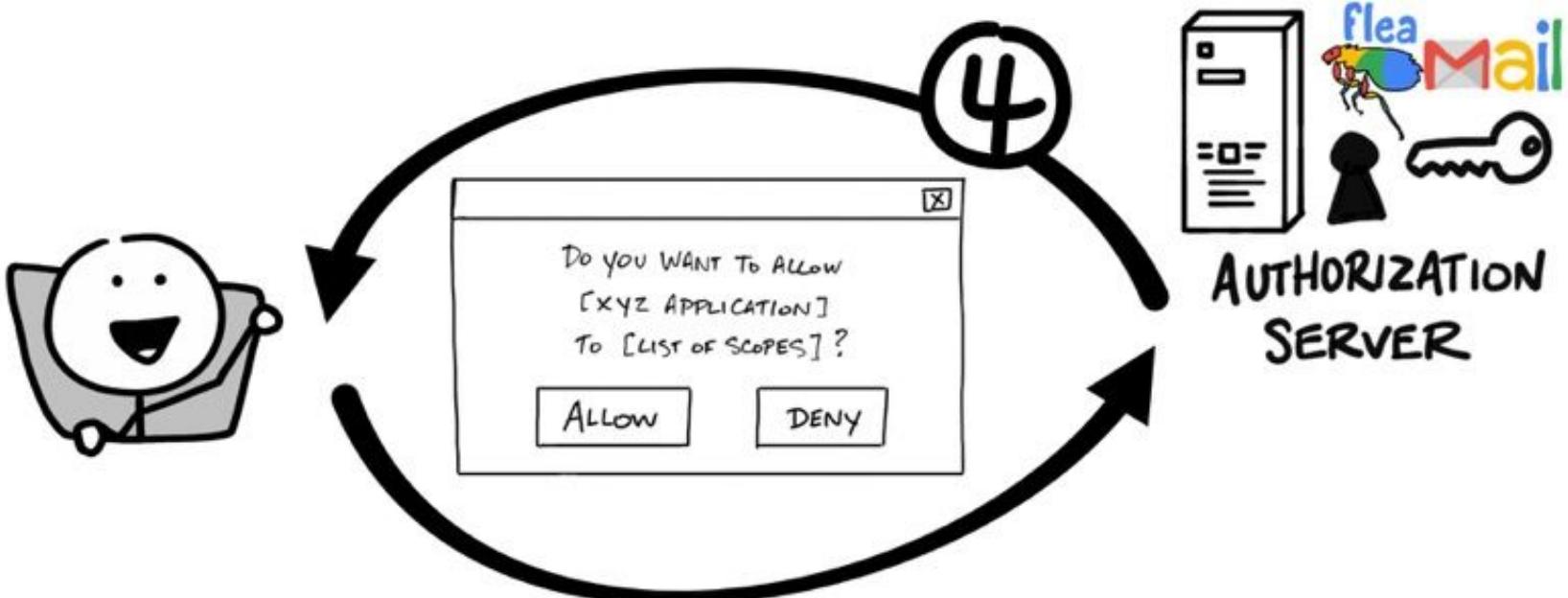
- **Authorization code:** Código temporal que el authorization server envía al cliente para que solicite un token
- **Access token:** Token que el cliente usará para comunicarse con el resource server cuando quiera solicitar los datos del usuario (resource owner)



OAuth2

- El flujo **authorization code** define los mensajes que se tienen que intercambiar los servidores para obtener el permiso del usuario

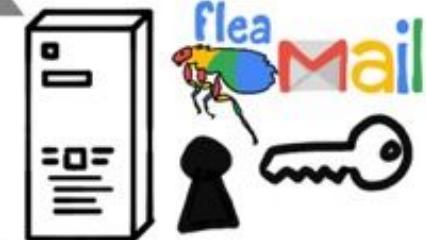






6

AUTHORIZATION CODE
CLIENT ID
CLIENT SECRET



AUTHORIZATION
SERVER

ACCESS TOKEN

7



8

CONTACTS



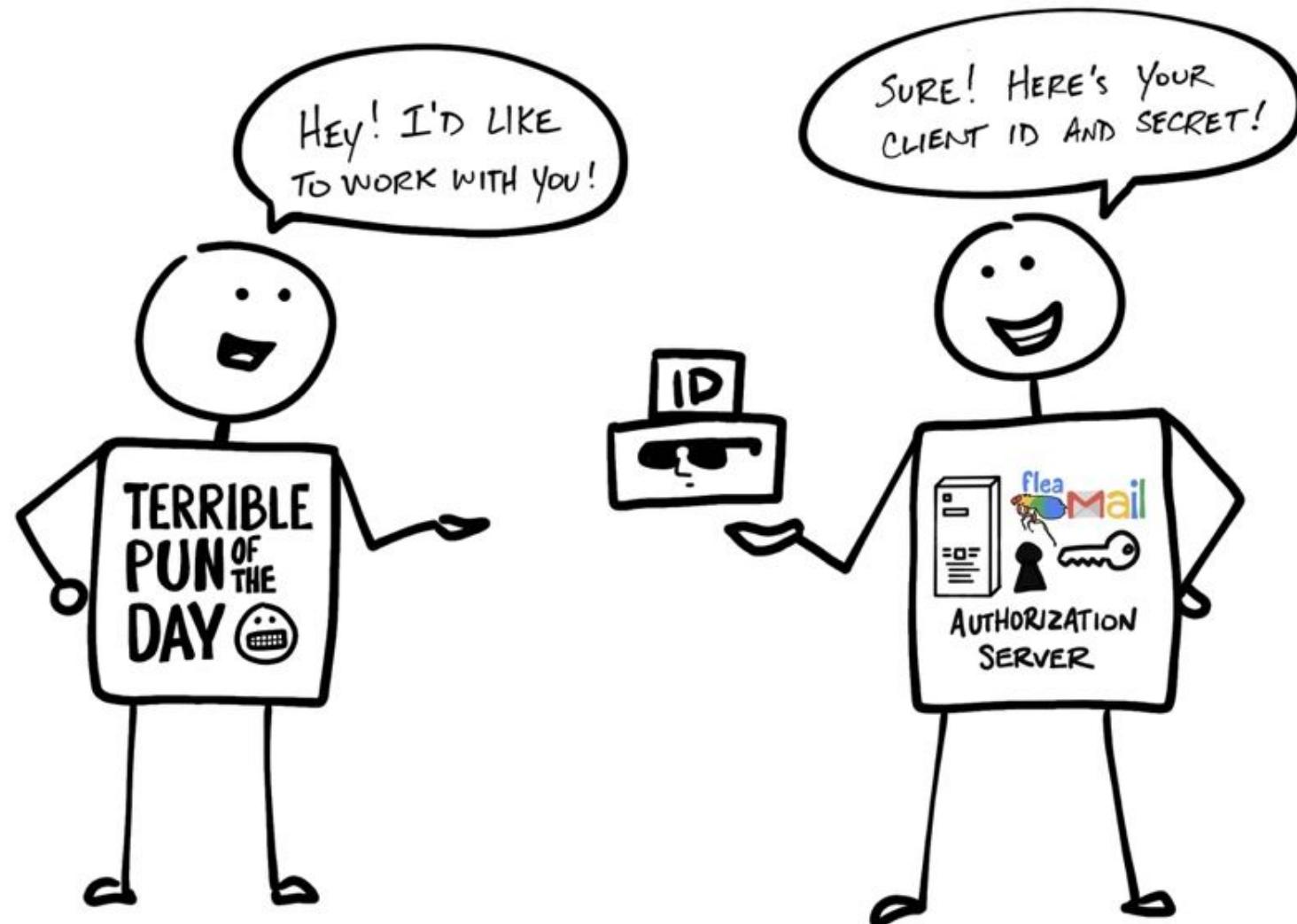
RESOURCE
SERVER

CONTACTS
API

OAuth2

- Para que el **cliente** pueda solicitar acceso a los datos del **usuario (resource owner)** en el **authorization server**, previamente el cliente se ha dado de alta
- El cliente dispone de un Client ID y de un Client Secret para poder **autenticarse**

OAuth2

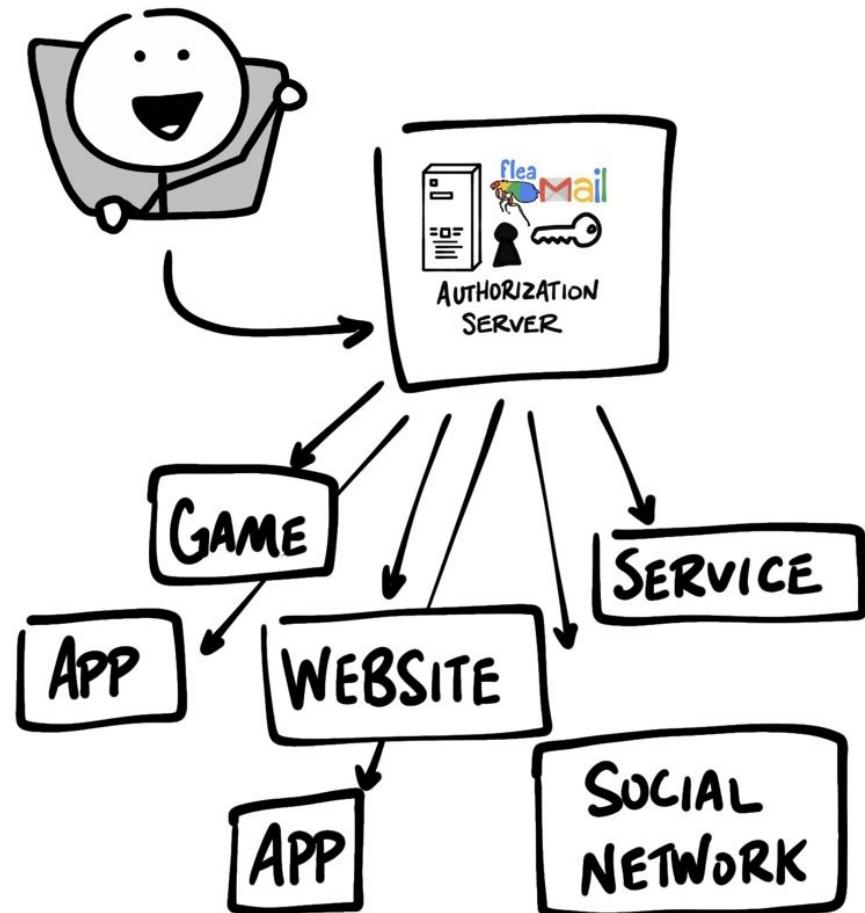


OAuth2

- OpenID Connect (OIDC)
 - Capa sobre OAuth2 que permite **identificar** al usuario (resource owner)
 - Información sobre el usuario se denomina **identidad** (*identity*)
 - Al **authorization server** que soporta OIDC se le llama **identity provider**

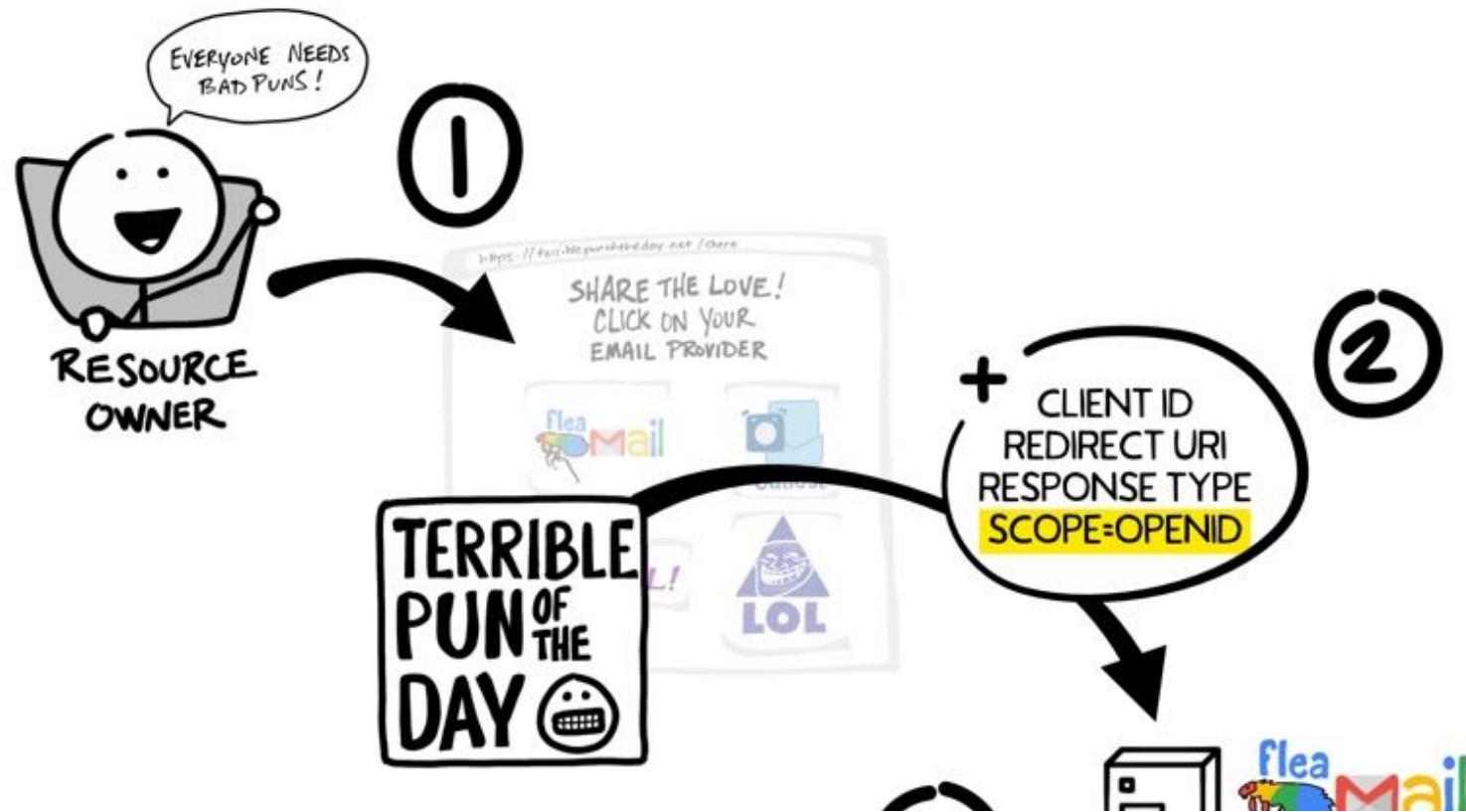
OAuth2

- OpenID Connect (OIDC)
 - Se pueden implementar esquemas de Single Sign On (SSO)



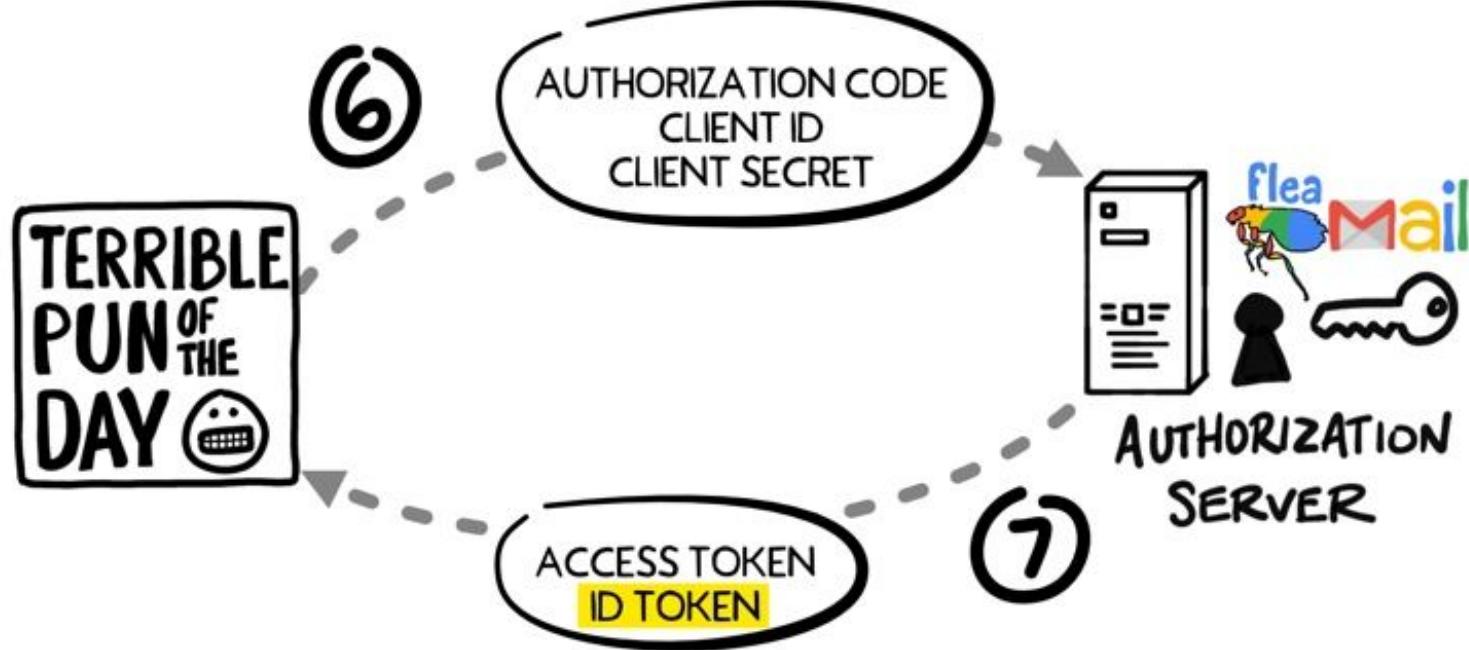
OAuth2

- OpenID Connect (OIDC)



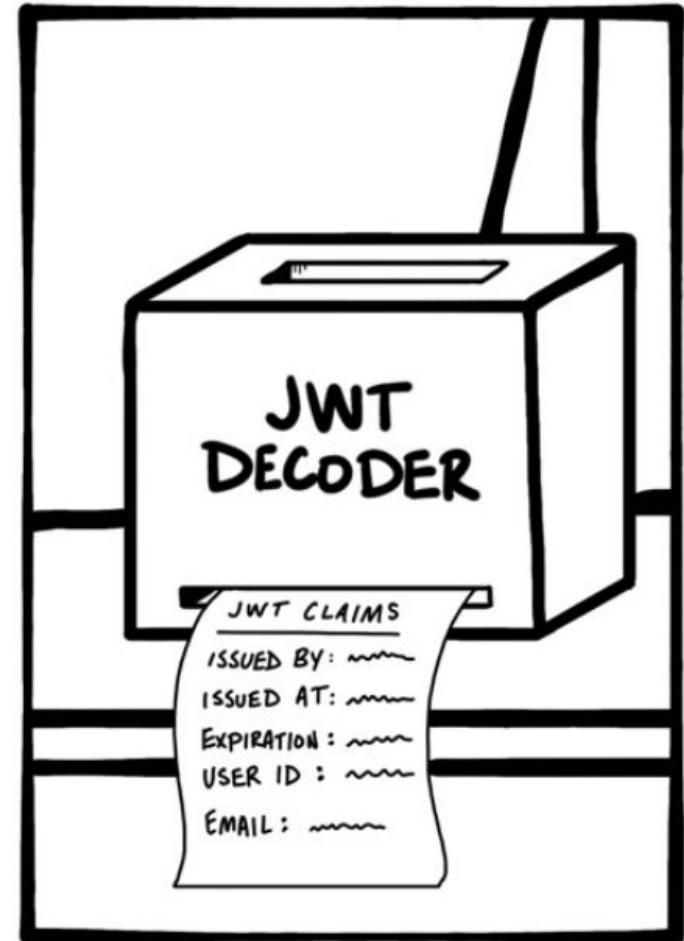
OAuth2

- OpenID Connect (OIDC)



OAuth2

- OpenID Connect (OIDC)
 - ID Token es un JWT con información del usuario (*claims*)



OAuth2

- **Casos de uso de OAuth2 en una app**
 - **Login social** con Facebook, Google, GitHub...
(acceso al email y la foto de perfil)
 - **Acceso a datos del usuario** en Facebook, Google, GitHub...
 - Implementación de **Single Sign On** en varias aplicaciones con un Authorization Server **propio** basado en OAuth2 y OIDC

OAuth2

- Casos de uso de OAuth2 en una app
 - Login con user y password para generación de token
 - Es un flow llamado “Resource Owner Password Credentials Grant”

<http://tools.ietf.org/html/rfc6749#section-1.3.3>

<http://tools.ietf.org/html/rfc6749#section-4.3>

<https://developer.okta.com/blog/2018/06/29/what-is-the-oauth2-password-grant#what-is-an-oauth-2-o-grant-type>

<http://www.passportjs.org/packages/passport-oauth2-client-password/>

<https://github.com/FrankHassanabad/Oauth2orizeRecipes/wiki/Resource-Owner-Password-Credentials>

OAuth2

ejem7

- **Implementar authorization server**

- Existen varias librerías en Node
- **node-oauth2-server** y **oauth2orize** son las más conocidas
- Permiten diferentes flows

<https://github.com/oauthjs/node-oauth2-server>

<https://github.com/jaredhanson/oauth2orize>

OAuth2

ejem7

- Ejemplo con node-oauth2-server

- Configuración del servidor OAuth

```
const Request = OAuth2Server.Request;
const Response = OAuth2Server.Response;

var app = express();

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.oauth = new OAuth2Server({
  model: require('./model.js'),
  accessTokenLifetime: 60 * 60,
  allowBearerTokensInQueryString: true
});
```

Model es un objeto con diversas funciones usadas por OAuth

OAuth2

• Ejemplo con node-oauth2-server

ejem7

- Funciones del model

- Usadas por todos los flows (grant types)

- getAccessToken(token)
- getClient(clientId, clientSecret)
- saveToken(token, client, user)
- getUser(username, password)

OAuth2

• Ejemplo con node-oauth2-server

ejem7

- Funciones del model

- Usadas por client_credentials grant type
 - getUserFromClient(client)
- Usadas por refresh_token grant type
 - getRefreshToken(refreshToken)
 - revokeToken(token)

OAuth2

• Ejemplo con node-oauth2-server

- Generación de token

```
app.all('/oauth/token', async (req, res) => {  
  
  var request = new Request(req);  
  var response = new Response(res);  
  
  try {  
  
    var token = await app.oauth.token(request, response);  
    res.json(token);  
  
  } catch (e) {  
    res.status(e.code || 500).json(e);  
  }  
});
```

ejem7

OAuth2

• Ejemplo con node-oauth2-server

- Autenticación

```
async function authenticateRequest(req, res, next) {  
  
  var request = new Request(req);  
  var response = new Response(res);  
  
  try {  
  
    await app.oauth.authenticate(request, response);  
    next();  
  
  } catch (e) {  
    res.status(e.code || 500).json(e);  
  }  
}  
  
app.get('/ads', authenticateRequest, (req, res) => { ... });
```

ejem7

OAuth2

ejem7

• Credenciales

- **password grant**
 - Client
 - clientId: application
 - clientSecret: secret
 - User
 - username: pedroetb
 - password: password
- **refresh_token grant**
 - There is one client added to server and ready to work:
 - clientId: application
 - clientSecret: secret
- **client_credentials grant**
 - Confidential client added to server
 - clientId: confidentialApplication
 - clientSecret: topSecret

OAuth2

• Obtención del token

ejem7

- POST a <https://localhost:3443/oauth/token>
- Password grant
 - Basic Auth: clientId y clientSecret
 - Body:
 - application/x-www-form-urlencoded
 - grant_type=password&username=pedroetb&password=password

Para otros grant types mirar el README.md del ejemplo

OAuth2

• Obtención del token

- Respuesta

```
{  
    "accessToken": "951d6f603c2ce322c5def00ce58952ed2d096a72",  
    "accessTokenExpiresAt": "2018-11-18T16:18:25.852Z",  
    "refreshToken": "67c8300ad53efa493c2278acf12d92bdb71832f9",  
    "refreshTokenExpiresAt": "2018-12-02T15:18:25.852Z",  
    "client": {  
        "id": "application"  
    },  
    "user": {  
        "id": "pedroetb"  
    }  
}
```

ejem7

OAuth2

ejem7

- **Acceso a la API**

- GET a <https://localhost:3443/ads/>
- Headers
 - Authorization: Bearer + accessToken

Ejercicio 5

- Cambia la autenticación de tokens JWT por OAuth2 en el Ejercicio 4