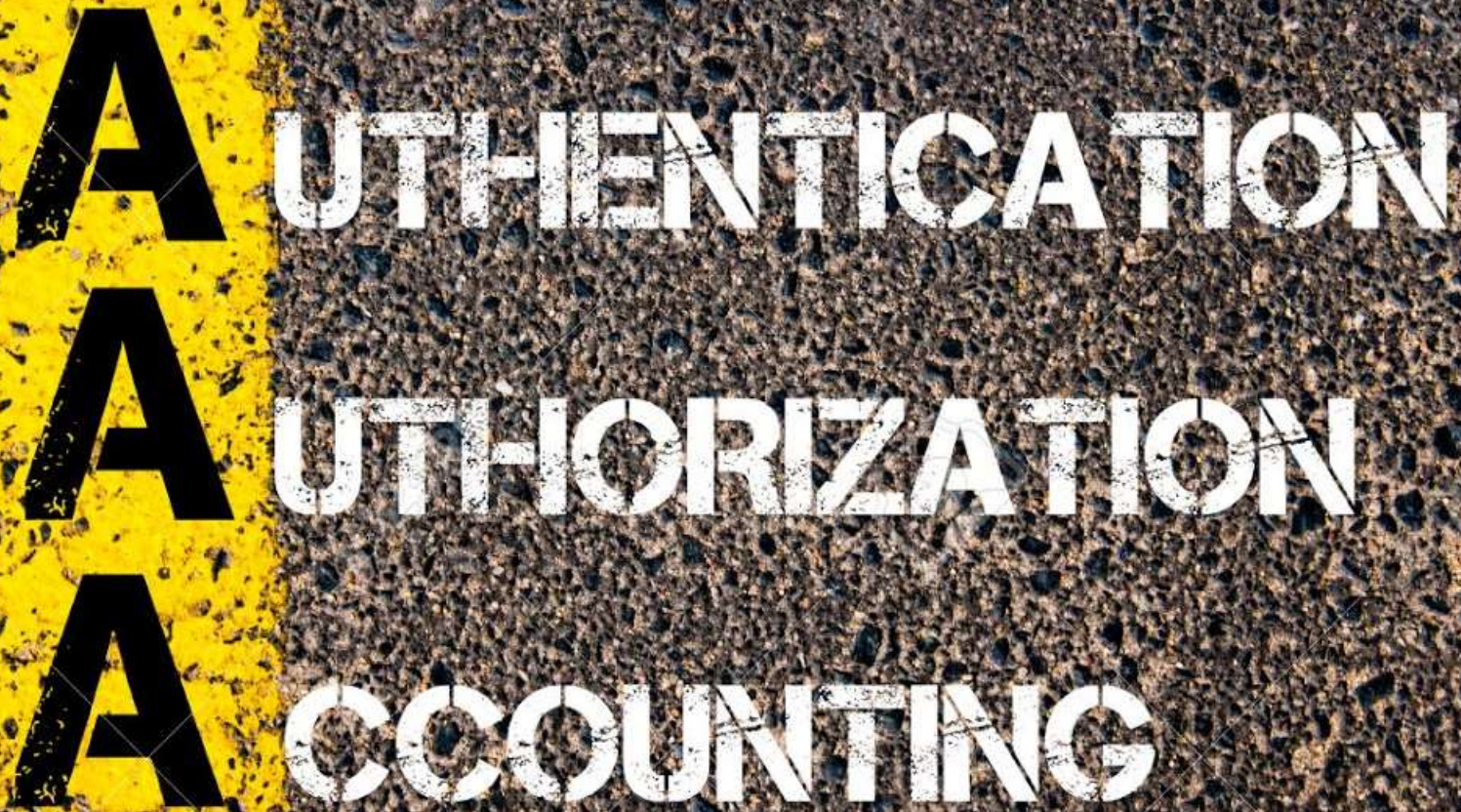


Autenticación





AUTHENTICATION
AUTHORIZATION
ACCOUNTING

Autenticación (Acreditación)

- Es el proceso por el que se comprueba la **identidad de alguien o algo**, para ver si es lo que dice ser.
- Ese "alguien" o "algo" se denomina **principal**.
- La autenticación requiere pruebas de identidad, denominadas **credenciales**.
- Por ejemplo, una aplicación cliente puede presentar una contraseña como sus credenciales. Si la aplicación cliente presenta las credenciales correctas, se asume que es quien dice ser.

Métodos de Autenticación

1. Sistemas basados en algo que el **usuario conoce**. Por Ejemplo, un *password* (Unix) o *passphrase* (PGP).
2. Sistemas basados en algo que el **usuario tiene**. Ejemplo, una tarjeta de identidad, una tarjeta inteligente(*smartcard*), dispositivo usb tipo epass token, Tarjeta de coordenadas, smartcard, un celular o dongle criptográfico.
3. Sistemas basados en una característica física del usuario (algo que el **usuario es**) o un acto involuntario del mismo: Ejemplo, verificación de voz, de escritura, de huellas dactilares, de patrones oculares.
4. Algo que el usuario **hace** (ejemplo, reconocimiento de voz, firma)

Autenticación (o verificación) de 2 pasos

- Que es? Es la combinación de 2 métodos de autenticación. Ej. Algo que tienes + algo que sabes!!!
- Por que? Mayor nivel de seguridad!!!
- En el escenario de que alguien “adivina tu password” podemos verificar que el usuario tenga lo que debes tener (Ej. Celular)
- Contra? Más tedioso para el usuario cada vez que se tiene que autenticar.



Autenticación de usuarios

- El proceso general de autenticación consta de los siguientes pasos:
 1. El usuario solicita acceso a un sistema.
 2. El sistema solicita al usuario que se autentique.
 3. El usuario aporta las credenciales que le identifican y permiten verificar la autenticidad de la identificación.
 4. El sistema valida según sus reglas si las credenciales aportadas son suficientes para dar acceso al usuario o no.

Una vez autenticado el sistema puede aplicar los mecanismos de autorización y/o auditoría oportunos.

Autenticación

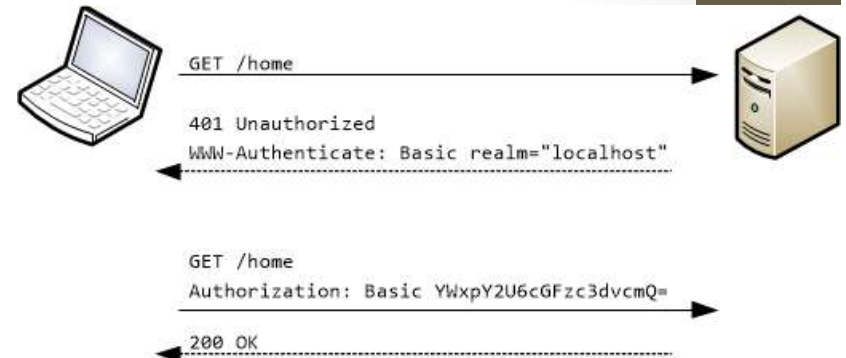
En Aplicaciones Web

1. Basada en uso de HTTP. (Obsoleta)
2. Basada en cookies o archivos. (solo para aplicaciones simples)
3. Con sesiones y usuarios en Base de Datos.(aplicaciones web clásicas – Utiliza estado)
4. Autenticación basada en token (Muy usada en APIs REST – NO Utiliza estado)
5. OAuth 2.0 (Autorización abierta) (apps web y APIs REST)
6. Con verificación de Imágenes (Protege contra bots)

Autenticación Básica

Obsoleta!!

- La más simple.
- Se basa principalmente en un nombre de usuario y una contraseña para identificarte.
- se debe realizar mediante el encabezado HTTP Autorización (Authorization).
- Anticuoado
- Susceptible a ataque Man-In-The-Middle (MiTM)
- Requiere HTTPs



Autenticación de usuarios basada en cookies o archivos

- Permite tener más control a través de la creación propia de un form de Login.
- Cuando un usuario se “loguea” exitosamente, se almacena encriptadamente el nombre del usuario en una cookie.

```
$secret_word = 'odio la espinaca';  
if user_validate($_POST['username'],$_POST['password'])) {  
    setcookie('login',  
    $_POST['username'].'.'.md5($_POST['username'].$secret_word));  
}
```

Obsoleta!!

Autenticación de usuarios con sesiones y BD

- Esquema básico de la BD.

```
CREATE TABLE user (  
    id VARCHAR(10) NOT NULL,  
    password CHAR(255) NOT NULL,  
    PRIMARY KEY (id));
```

Como Guardar el Password?

- Usando la función PASSWORD de mysql (No se recomienda)
- MD4, MD5 o SHA-1 y similares (No se recomienda)
- Usando alguna librería de encriptación que provea métodos actuales de encriptación como crypto.
 - > `npm install crypto`
- Usando la librería bcrypt que dispone algoritmos de hashing lentos y computacionalmente costosos
 - > `npm install bcrypt` (Mejor opción)

Autenticación de usuarios con sesiones y BD

Registro de Usuario

```
const bcrypt = require("bcrypt");
const express = require("express");
const User = require("../models/user");
const router = express.Router();

// registrarse
router.post("/signup", async (req, res) => {
  const body = req.body;
  if (!(body.email && body.password)) {
    return res.status(400).send({ error: "Datos no tienen formato apropiado" });
  }
  // Creando un nuevo usuario
  const user = User.create(body);
  // generar salt para hashear el password
  const salt = await bcrypt.genSalt(10);
  // hasheamos el password con salt anexado
  user.password = await bcrypt.hash(user.password, salt);
  user.save().then((doc) => res.status(201).send(doc));
});
```

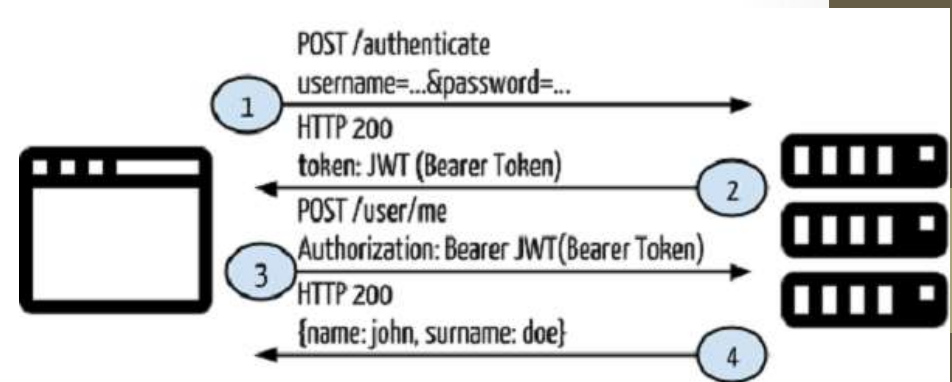
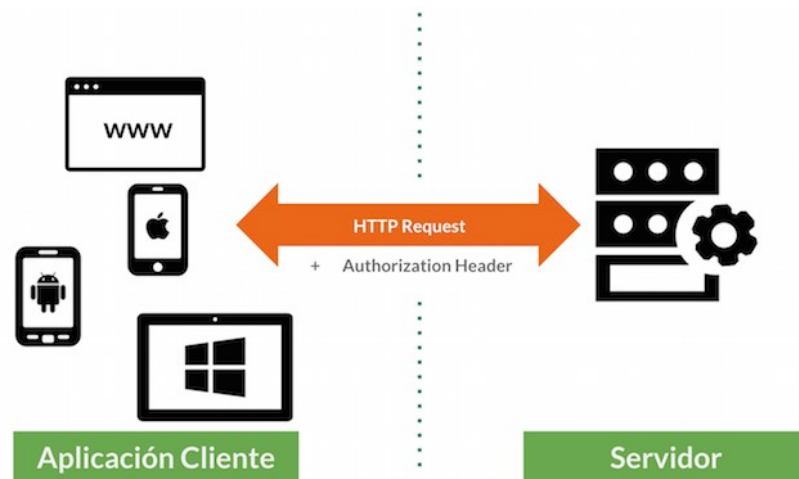
Autenticación de usuarios con sesiones y BD - **Lógin de Usuario**

// login route

```
router.post("/login", async (req, res) => {  
  const body = req.body;  
  const user = await User.findOne({ email: body.email });  
  if (user) {  
    // compara password del usuario con password hasheado en la BD  
    const validPassword = await bcrypt.compare(body.password, user.password);  
    if (validPassword) {  
      res.status(200).json({ message: "Usuario Autenticado" });  
      //configurar la session para no autenticar en cada requerimiento  
    } else {  
      res.status(400).json({ error: "Password Inválido" });  
    }  
  } else {  
    res.status(401).json({ error: "El usuario no existe" });  
  }  
});  
module.exports = router;
```


Autenticación basada en Token

1. El usuario se identifica con sus credenciales.
 2. El servidor valida las credenciales y **genera un token codificado**.
 3. El usuario envía en cada petición el token
 4. El servidor valida el token
- Los tokens por lo general tienen fecha y tiempos de caducidad.
 - Si un atacante consigue el token, le es mas difícil utilizarlo (según reglas de vencimiento)
 - Requiere HTTPS para funcionar en forma segura.



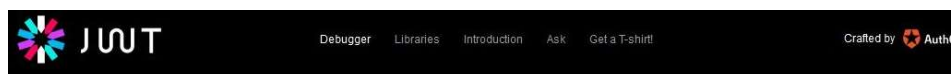
Autenticación por token - JWT

- JWT es un estándar RFC 7519 para transmitir información con la identidad (claims) de un usuario de forma segura entre un cliente/servidor.
- Dicha información puede ser verificada y confiable porque está firmada digitalmente.
- Es una cadena de texto que tiene 3 partes codificadas en Base64, separadas por un punto (header.payload.firma) que generamos y entregamos a los clientes de nuestra App.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWRtaW4iOnRydWV9.TjVA95OrM7E2cBa
b30RMHrHDcEfxjoYZgeFONFh7HgQ

JWT

- Es importante aclarar que la cadena/token esta codificada y lo crea nuestra aplicación
- Nos permite de manera muy fácil inspeccionar su contenido, por ejemplo con jwt.io



Debugger

ALGORITHM	HS256
-----------	-------

Encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjE5OTk0MTUyNjV9.TjVA95OrM7E2eC8ab3
0RMhrHdCEfxjoYZgeFONfhHqQ

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
"alg": "HS256",  
"typ": "JWT"
```

PAYLOAD: DATA

```
"sub": "1234567890"  
"name": "John Doe",  
"admin": true
```

VERIFY SIGNATURE

```
HMACSHA256(  
    base64UrlEncode(header) +  
    base64UrlEncode(payload),  
    secret  
)
```

Signature Verified

Algoritmo a usar y token

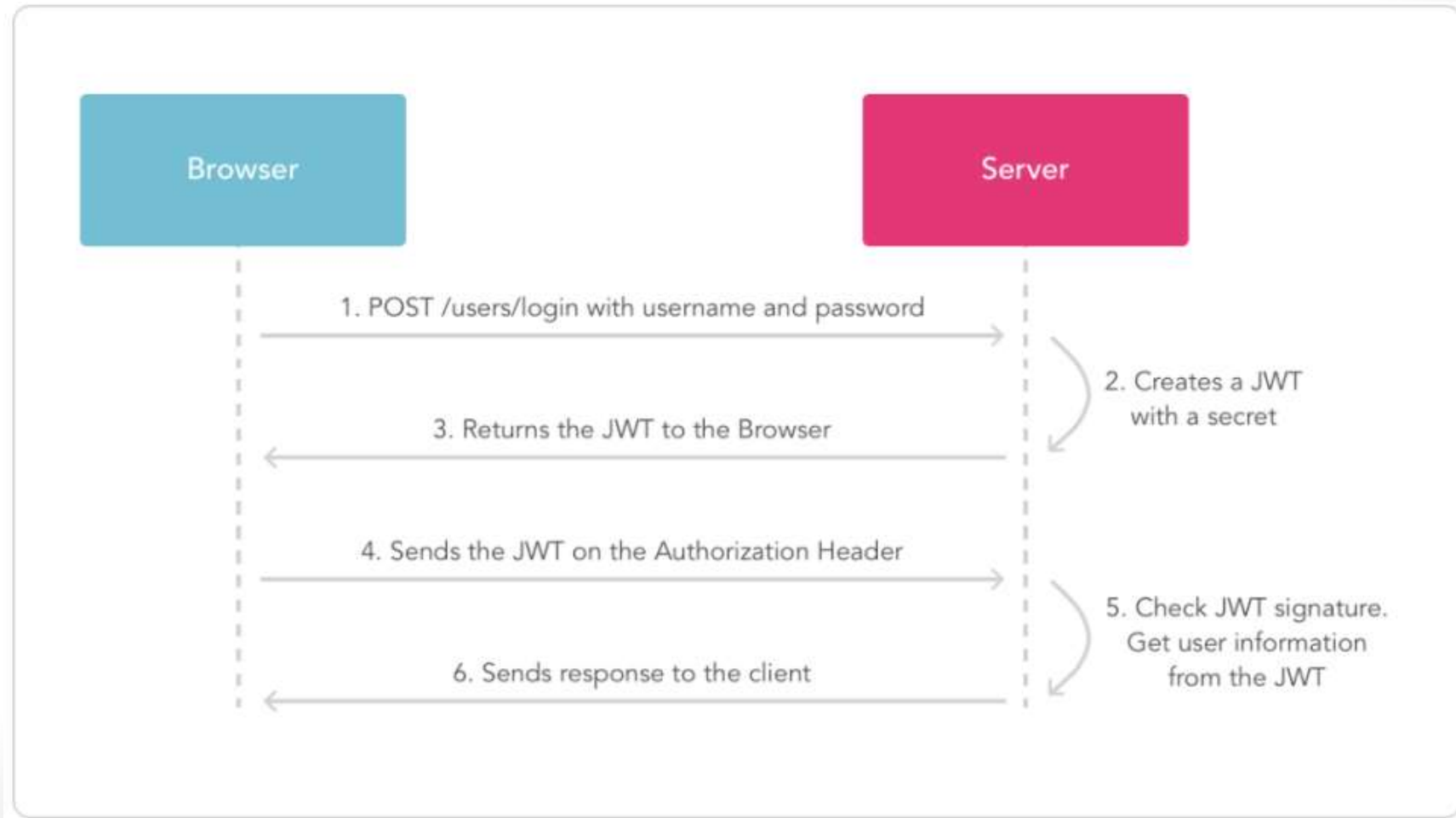
Lo que necesite nuestra app para validar:
User, mail, fechaCreación, fechaExpiración, etc

Firma

Firma

- *la información puede ser verificada y confiable porque está firmada digitalmente", con un "secret-key".*
- Es responsabilidad de nuestra aplicación cuando recibamos un Token en nuestra API, **verificarlo** con nuestro "secret-key", garantizar que la firma es válida para aceptarlo o denegarlo.
- **Se debe usar HTTPS para encriptar todo el tráfico entre los clientes y servidor con un certificado!!**

Ciclo de vida de un Token



Algunas consideraciones sobre JWT

- *"Authorization: Bearer "*, es la forma más común, indicar que existen otras técnicas para hacerlo.
- JWT es muy ligero: podemos codificar gran cantidad de datos sensibles en el payload y pasarlo como una cadena.
- Creamos servicios de autenticación optimizados desacoplados del servidor y tenemos protección contra ataques CSRF.
- Nos ahorramos mantener el estado del usuario en el servidor y lo delegamos al cliente.
- Recordar que *siempre, siempre, siempre debemos usar HTTPS* entre el cliente/servidor para las peticiones.
- Y lo más importante: ¡Nos olvidamos de cookies!

Módulo jsonwebtoken

```
var jwt = require('jsonwebtoken')
var express = require('express')
var bodyParser = require('body-parser')
app.use(express.urlencoded({extended: false}))
app.use(express.json({limit: '10mb'}))
```

```
npm install --save jsonwebtoken
```

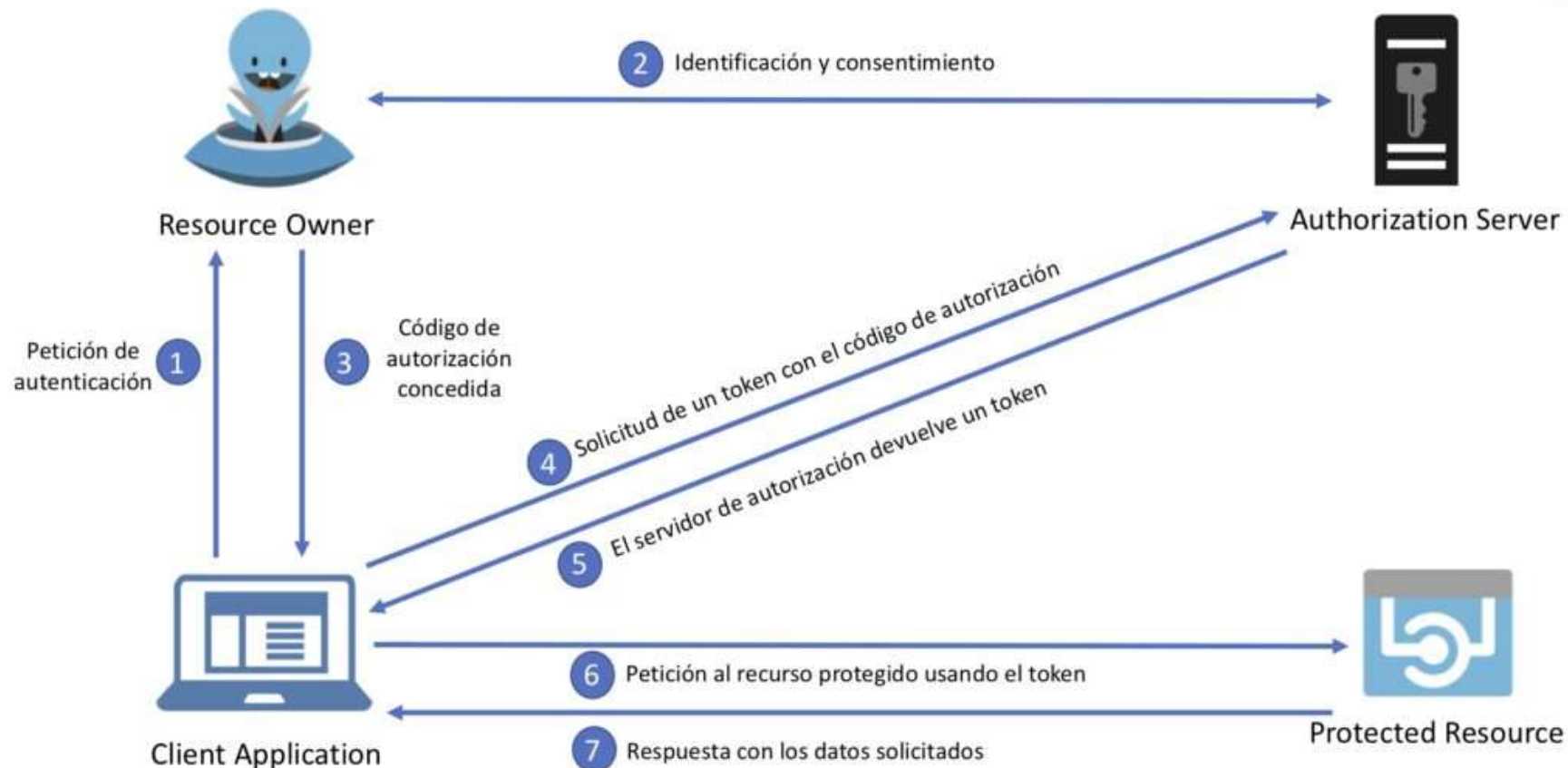
```
app.post('/login', (req, res) => {
  var username = req.body.user
  var password = req.body.password
  {...} // Aquí deberíamos comprobar que exista el usuario y password en la BD
  let token = jwt.sign({username: username, 'mi clave secreta', { expiresIn: '24h' }}); // expira en 24 horas
  // Retorna el token JWT para futuras llamadas
  res.json({
    success: true, message: 'Authentication Exitosa!', token: token
  });
} else {
  res.send(403).json({
    success: false, message: 'Usuario o password incorrectos' });
}
```

Verificar JWT

```
let jwt = require('jsonwebtoken');
const misecretkey = 'mi clave secreta'; //recupera la secret-key
app.get('/autenticacion', (req, res) => {
  let token = req.headers['x-access-token'] || req.headers['authorization'];
  if (token.startsWith('Bearer ')) {
    // Eliminar Bearer desde el string
    token = token.slice(7, token.length);
  }
  if (token) {
    jwt.verify(token, misecretkey, (err, decoded) => {
      if (err) { return res.status(401)({mensaje: 'Token no es válido' }); }
      else { req.decoded = decoded; next(); }
    });
  }
  else { return res.status(401).send({mensaje: 'No se provee Token Auth ' }); }
};
```

Autenticación Abierta (Oauth 2)

- Su propósito es permitir a otros proveedores, servicios o aplicaciones, el acceso a la información sin facilitar directamente las credenciales de los usuarios.



Passport - Oauth 2



Simple, unobtrusive authentication for Node.js

- <http://www.passportjs.org/>
- Simple, middleware no intrusivo para Node.js
- Soporte para mas de 500 estrategias de authentication usando [Local](#), [password](#), [Facebook](#), [Twitter](#), y [more](#).
- Estrategias empaquetadas como módulos separados.

Oauth 2 – Auth con Proveedor Github

1. Ir al sitio que servirá de autenticación.
2. Configurar datos de la aplicación (Callback url)
3. La aplicación nos dara un CLIENT ID y un CLIENT SECRET

```
var GitHubStrategy = require('passport-github').Strategy;
passport.use(new GitHubStrategy({
  clientID: GITHUB_CLIENT_ID,
  clientSecret: GITHUB_CLIENT_SECRET,
  callbackURL: "http://127.0.0.1:3000/auth/github/callback"
},
function(accessToken, refreshToken, profile, cb) {
  User.findOrCreate({ githubId: profile.id }, function (err, user) {
    return cb(err, user);
  });
});
```

- El callback retorna los token de acceso y los datos públicos del perfil del usuario.

Oauth 2 - Autenticando

- Use `passport.authenticate()`, especificando la estrategia 'github', para autenticar las solicitudes.

```
app.get('/auth/github', passport.authenticate('github'));
```

```
app.get('/auth/github/callback',  
  passport.authenticate('github', { failureRedirect: '/login' }),  
  function(req, res) {  
    // Successful authentication, redirect home.  
    res.redirect('/');  
  });
```

Oauth 2 - Autenticando

```
var EstaAutenticado = function (req, res, next) {  
  // Passport agrega este método al objeto request.  
  if (req.isAuthenticated())  
    return next();  
  // si el usuario no esta autenticado entonces lo redirigimos a login  
  res.redirect('/login');  
}
```

Oauth 2 - Autenticando

```
/* Autenticar sobre una única ruta*/  
router.get('/home', EstaAutenticado, function(req, res){  
  res.render('home', { user: req.user });  
});  
/* Logout ----- */  
router.get('/signout', EstaAutenticado , function(req, res) {  
  req.logout(); // agregado por passport  
  res.redirect('/');  
});  
  
/* Autenticar sobre un conjunto de rutas - Router*/  
app.use("/", indexRouter);  
app.use("/medicos", EstaAutenticado, medicoRouter);  
app.use("/pacientes", EstaAutenticado, pacienteRouter);
```