

Proyecto y Grupo



"DOCUMENTO DE SEGURIDAD"

Hito: 1

Fecha entrega: 23-12-2016

Versión: 1

Componentes:

- **Emilio Maestre Hortal**
- **Jose Francisco Moreno Fernández**
- **Pablo López Riquelme**
- **Sergio Pérez Seré**



Contenido

1. JSON WEB TOKEN (JWT).....	2.
2. Seguridad para Usuarios y Claves dentro del código.....	3.
3. Cifrado de contraseñas en la Base de Datos.....	4.
4. CORS (CONTROL DE ACCESO HTTP).....	4.
5. Delegación de almacenamiento de Tarjetas y Pagos en TPV externo.....	5.
6. Evitar Inyección de Código SQL.....	5.
7. Verificación de Usuarios Reales.....	6.
8. Cifrado SSL.....	6.
9. Copias de Seguridad de la Base de Datos.....	7.



1. JSON WEB TOKEN (JWT).

En Appay hemos decidido emplear un token para autenticar al usuario en cada petición que se hace al servidor.

El flujo de control es el siguiente:

1. El usuario provee un **nombre de usuario y contraseña** en el formulario de login y clickea en **Ingresar**.
2. Una vez hecha la petición, validamos el usuario en el back-end mediante una consulta a nuestra base de datos. Si la petición es válida, creamos un token utilizando la información de usuario brindada por la base de datos, y luego retornamos esa información en el encabezado de la respuesta, para así guardar el token en almacenamiento local.

En nuestro caso, hemos utilizado la librería JsonWebToken de nodeJS. Para generar el token guardamos todos los datos del usuario disponibles en la tabla Usuarios de nuestra base de datos. Esta información la ciframos con una clave secreta aplicando el algoritmo HS256 y aplicamos una caducidad de 7 días. El token lo almacenamos en LocalStorage del navegador, después de informarnos de forma profunda hemos comprobado que es la forma idónea de almacenar este token de forma correcta, ya que incluso los creadores de la librería recomiendan guardarla de esa forma. Si alguien accede al ordenador de un usuario con un token activo y roba el token, es un fallo de seguridad del usuario.

Artículo interesante donde trata las distintas opciones de almacenamiento de JWT y los problemas de seguridad que pueda tener cada opción:
<https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage>

3. Se provee la información del token en el encabezado de cada petición para acceder endpoints restringidos de la aplicación.
4. Realizamos el descifrado del token en el servidor mediante nuestra clave secreta y si el token tomado del encabezado de la petición es válido, permitimos al usuario acceder al endpoint especificado, y respondemos con JSON.

También hemos implementado los JWT para la función de restablecer contraseña, en caso de que un usuario no se acuerde de la contraseña al querer acceder a la aplicación, y para la función de confirmar registro de usuario. En ambos casos enviamos un correo electrónico con un enlace que contiene el token (con fecha de caducidad de 24 horas). Al acceder al enlace si el token es válido podrán realizar estas funciones.

2. Seguridad para usuarios y claves dentro del código.

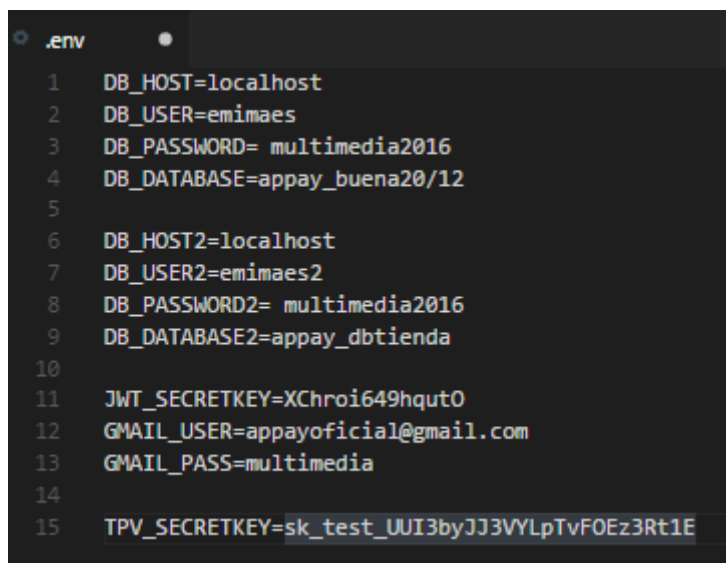
Un gran fallo de seguridad que puede tener un Backend de una aplicación es tener a la vista las claves e información secreta dentro del código, de forma que si un hacker pudiese acceder a nuestro código no podría acceder a esa información. Para solucionar este problema hemos utilizado las variables de entorno de NodeJS.

Las variables de entorno también nos permiten trabajar de forma local de forma que cada desarrollador pueda trabajar con sus propias claves sin tener que modificar el código fuente.

Para la utilización de estas variables de entorno, cada desarrollador tiene un propio archivo '.env' en el que tiene almacenados sus credenciales.

En nuestro caso tenemos almacenadas:

- Los datos de nuestra base de datos
- La clave secreta del JWT
- La clave secreta del TPV
- Los datos de nuestra cuenta de correo electrónico



```
.env
1 DB_HOST=localhost
2 DB_USER=emimaes
3 DB_PASSWORD= multimedia2016
4 DB_DATABASE=appay_buena20/12
5
6 DB_HOST2=localhost
7 DB_USER2=emimaes2
8 DB_PASSWORD2= multimedia2016
9 DB_DATABASE2=appay_dbtienda
10
11 JWT_SECRETKEY=XChroi649hqt0
12 GMAIL_USER=appayoficial@gmail.com
13 GMAIL_PASS=multimedia
14
15 TPV_SECRETKEY=sk_test_UUI3byJJ3VYLpTvFOEz3Rt1E
```

Forma de utilizar las variables de entorno dentro del código fuente:

```
var apiKey = process.env.APIKEY
```

En este caso asignamos la variable apiKey a la variable de entorno APIKEY.

Cabe mencionar que tenemos configurado nuestro GitHub de tal forma que los archivos '.env' no se suban al repositorio. Para el modo de producción almacenamos las variables dentro de la configuración de nuestro servidor (Heroku):



Config Vars

AUTOBUS_URL	http://www.autobus.io:443/user/5820ae21e4		
CLEARDB_DATABASE_URL	mysql://b3f3fb41577c3b:8d4e1afd@us-cdbr-i		
DATABASE_URL	mysql://b3f3fb41577c3b:8d4e1afd@us-cdbr-i		
DB_DATABASE	heroku_3ac2f6300e00435		
DB_HOST	us-cdbr-iron-east-04.cleardb.net		
DB_PASSWORD	8d4e1afd		
DB_USER	b3f3fb41577c3b		

3. Cifrado de contraseñas en la Base de Datos.

Hemos utilizado un cifrado de tipo MD5 para cifrar nuestras contraseñas en la base de datos. De esta forma nos aseguramos de que, en caso de que pudiesen hackearnos y acceder a nuestra base de datos, nunca podrían acceder a las contraseñas de los usuarios:

CP	Telefono	Foto	Contra	Rol	Estado
3680	667081603	pablolopez.jpg	7e4b64eb65e34fdfad79e623c44abd94	3	1
3590	617283260	sergio.jpg	3bffa4ebdf4874e506c2b12405796aa5	3	1
3006	2147483647	jose.jpg	8395e4ee2ae005332879ce4cfda8a6bb	3	1

4. CORS (CONTROL DE ACCESO HTTP).

Los CORS o Cross-Origin Resource Sharing es una potente tecnología que permite al servidor restringir ciertas acciones del cliente en función del origen de la llamada. En concreto CORS es un grupo de headers de respuesta especiales enviados desde el servidor que indican a un navegador si permitir o no que la solicitud pase.

En nuestro caso vamos a utilizar los CORS como medida de seguridad para que solo se pueda acceder a nuestra API desde los orígenes de nuestra aplicación de escritorio y de nuestra aplicación móvil. Para ello hemos utilizado un middleware de Express. Las funciones de middleware nos permiten añadir métodos antes de cada llamada a la API:

```
//CORS, PERMITIMOS ACCESO A LA API SOLO EN ESTAS RUTAS
var whitelist = [
  'http://localhost:3000',
  'http://localhost:4200',
];
var corsOptions = {
  origin: function(origin, callback){
    var originIsWhitelisted = whitelist.indexOf(origin) !== -1;
    callback(null, originIsWhitelisted);
  },
  credentials: true
};
app.use(cors(corsOptions));
```



En nuestro caso, como aún estamos nos tenemos las partes del cliente desarrolladas y subidas a un servidor, tenemos restringidas las llamadas a la API a los siguientes orígenes: 'localhost:3000' y 'localhost:4200'. Sin embargo, cuando tengamos las aplicaciones de cliente finalizadas lo cambiaremos a 'appay.com'...

5. Delegación de almacenamiento de Tarjetas y Pagos en TPV externo.

Por motivos de seguridad, y por la dificultad de gestionar pagos nosotros mismos, hemos delegado los pagos de nuestros usuarios en el servicio externo Stripe.

Stripe nos proporciona una clave pública y una clave secreta. El funcionamiento una vez enviamos la clave pública al servidor junto con la cantidad a pagar y los datos del usuario el servicio realiza el pago y si ha sido correcto se devuelve el token correspondiente. Para realizar las pruebas de desarrollador Stripe ofrece un modo Test con una tarjeta de prueba: 4242 4242 ... con la que podemos comprobar la correcta implementación de la API de su servicio.

Pruebas realizadas en el modo Test de Stripe:

	€100.00 EUR	ch_19IzHIEN5BWvjtPuQx0RK...	emiliomaestre94@gmail.com	2016/11/23 12:51:33	...
	\$40.00 USD	ch_19Iz8GEN5BWvjtPu7Mtm...	emiliomaestre94@gmail.com	2016/11/23 12:41:44	...

[View all payments >](#)

6. Evitar Inyección de Código SQL.

Los ataques de inyección de SQL se están extendiendo más su uso por la facilidad que tiene de ponerlo en marcha cualquier persona, aunque no tenga experiencia en SQL simplemente sabiendo la cadena de texto adecuada.

En Appay hemos evitado la posibilidad de que puedan hackearnos de esta forma añadiendo 'connection.escape(nombre_variable)'. Esta opción que nos provee Express es suficiente para garantizar la seguridad de nuestro backend para esta función:

```
var username =connection.escape(req.body.username);
var password =connection.escape(req.body.password);
```

7. Verificación de Usuarios Reales.

Para verificar que nuestros usuarios son reales y no sean Bots que sobrecarguen nuestra base de datos, hemos implementado un proceso de verificación en el que los usuarios tienen que verificar mediante su correo electrónico su cuenta.

Para enviar los correos electrónicos utilizamos el plugin 'nodemailer' para NodeJS, que nos permite utilizar el transporte tipo SMTP y el servicio gmail para enviar los correos.

```
var smtpTransport = nodemailer.createTransport("SMTP",{
  service: "gmail",
  auth: {
    user: process.env.GMAIL_USER,
    pass: process.env.GMAIL_PASS
  }
});
```




En los correos enviamos un enlace con un token de 24 horas de validez:



8. Cifrado SSL.

Para garantizar la autenticidad del sitio. Poseemos un certificado SSL en el servidor. <https://appayservidor.herokuapp.com>


Security Overview

This page is secure (valid HTTPS).

- Valid Certificate**
 The connection to this site is using a valid, trusted server certificate.
[View certificate](#)
- Secure Connection**
 The connection to this site is encrypted and authenticated using a strong protocol (TLS 1.2), a strong key exchange (ECDHE_RSA with P-256), and strong cipher (AES_128_GCM).
- Secure Resources**
 All resources on this page are served securely.

General Detalles Ruta de certificación

 **Información del certificado**

Este certif. está destinado a los siguientes propósitos:

- Asegura la identidad de un equipo remoto
- Prueba su identidad ante un equipo remoto
- 2.16.840.1.114412.1.1

* Para ver detalles, consulte la declaración de la entidad de ce

Emitido para: *.herokuapp.com






Emitido por: DigiCert SHA2 High Assurance Server CA

Válido desde: 21/01/2014 **hasta:** 19/05/2017

[Declaración del emisor](#)

9. Copias de Seguridad de la Base de Datos.

En Appay tenemos conectados nuestro repositorio de Github (en concreto la rama 'master') a nuestro hosting del backend. De esta forma cada vez que realizamos un commit a ésta rama se actualiza el servidor.

	Merge branch 'master' of https://github.com/appayservidor/Servidor	19 dic. 2016 19:59	EMILIO\LenovoPc	88bf3d4
	Integrado bien todo lo de sergio	19 dic. 2016 19:54	EMILIO\LenovoPc	d9e57fa
	Get para saber si algo esta pagado	19 dic. 2016 18:53	plr1993 <pablolop>	26c881e
	Error_estadisticas	19 dic. 2016 17:33	plr1993 <pablolop>	9cb633a
	Actualizacion estadisticas	19 dic. 2016 17:32	plr1993 <pablolop>	15b02e8

Gracias a esta metodología de trabajo podemos tener copias de seguridad de cada cambio que hagamos en el código fuente y podemos acceder a ellas y restablecer la rama a cualquier punto del desarrollo:

Commits on Dec 17, 2016


-  **Cambio de líneas en index que ha metido el git**
 plr1993 committed 12 days ago

 cd13abb
 
-  **Conmit pablo**
 plr1993 committed 12 days ago

 390e350
 
-  **Actualizacion Pablo**
 plr1993 committed 12 days ago

 89118d5
 

Commits on Dec 14, 2016

-  **Mirad metodo post usuario si funciona que envíe el correo. Enviara al...**
 emiliomaestre94 committed 15 days ago

 17c811b
 
-  **Añadido todos los metodos de password (Falta comprobar put)**
 emiliomaestre94 committed 15 days ago

 0fc5aca
 