
# UNIVERSIDAD DE GRANADA

# Práctica 2: Ejercicio 5

FR

Fundamentos de Redes

**Autores:**
José Santos Salvador
Jaime Millán Gálvez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

# 1. Descripción de la aplicación

Nuestra propuesta de aplicación tiene dos funcionalidades principales. La primera de ellas es el servicio de financiación de proyectos (videojuego, aplicación, película, serie, etc) donde las personas pueden aportar dinero para que se lleve a cabo y obtener alguna recompensa (copia física o digital de proyecto, material adicional del mismo, etc) por ello dependiendo del dinero aportado. Cada proyecto parte de una meta a nivel de presupuesto y si no consigue llegar a esa cantidad de dinero en X tiempo, el proyecto no es financiado y se devuelve el dinero a cada usuario que ha aportado dinero.

La segunda de ellas es un sistema de donaciones a personas que crean contenido, esto puede ser un youtuber, un columnista, fotógrafo, es decir cualquier persona que necesite un "sueldo" para poder generar contenido que luego publica en internet. Pueden llevarse a cabo ambas por parte de un cliente (tener un proyecto a financiar y además tener tu perfil de donaciones como creador de contenido). Dentro de los proyectos y sistema de donaciones para creadores de contenido, existen distintas metas a alcanzar (si se llega a una de ellas, se realiza X acción, por ejemplo si se recauda X dinero en un proyecto de una película, esta se dobla al español). Estas donaciones pueden ser también suscripciones mensuales.

En nuestro servicio los estados dependen de las acciones anteriormente realizadas. Por ejemplo, si eres donante de un usuario en concreto, se te abre un abanico de posibilidades extra como acceder a su contenido, enviarle mensajes, etc.

En nuestra aplicación tendríamos distintas modalidades o funciones dentro de una cuenta registrada. Una cuenta puede tener un proyecto publicado para que se financie, permitiendo a esta cuenta publicar el progreso del proyecto. Un cliente cuando se registra está en "blanco" por así decirlo, en el momento en que financia un proyecto, cambia de estado y se convierte en cliente_financiador, esta nueva modalidad de cliente, le permite ver las actualizaciones de dicho proyecto, obtener las recompensas de financiarlo, incluso hablar con los creadores del proyecto. Dependiendo del proyecto, este cliente puede obtener unas ventajas u otras. Incluso los propios proyectos o perfiles de creadores de contenido, van cambiando de estado según las metas alcanzadas. Obviamente la parte de pagos, donaciones, devolver dinero, financiación, suscripciones, etc, son stateful entre cliente y servidor.

## 2. Diagrama de estados

# 3. Mensajes del sistema

## 3.1. Servidor

| Código | Cuerpo | Descripción |
|---|---|---|
| 101 | ERR + MENSAJE | Mensaje que envía el servidor cuando ocurre un error. MENSAJE dependerá del tipo de error |
| 202 | HELLO | Mensaje que envía el servidor para confirmar la conexión con el cliente |
| 303 | OK + MENSAJE | Mensaje que envía el servidor para confirmar la recepción de la información |
| 404 | BYE | Mensaje que envía el servidor para confirmar el cierre de sesión |
| 505 | RECOMPENSAS_AÑADIDAS | Mensaje que envía el servidor para confirmar que se han añadido las recompensas |
| 606 | FINANCIACIÓN_CONSEGUIDA | Mensaje que envía el servidor al cliente tras financiación el plazo sobre que ha conseguido la financiación |
| 707 | FINANCANCIÓN_NO_CONSEGUIDA | Mensaje que envía el servidor al cliente tras financiación el plazo sobre que no ha conseguido la financiación |
| 808 | PLAZO_FINALIZADO | Mensaje que envía el servidor cuando acaba el plazo fijado para el proyecto |
| 909 | INFORMACION_VALIDADA | Mensaje que envía el servidor cuando valida la información |
| 1001 | META_CONSEGUIDA | Mensaje que envía el servidor cuando la meta del proyecto se ha conseguido para volver al estado de autenticado_proyecto |
| 2002 | CANCELAR_PROYECTO+idProyecto | Mensaje que envía el servidor cuando se ha acabado el plazo y no se ha financiado el proyecto para cancelarlo, junto con el idProyecto |
| 3003 | CONTENIDO_ACTUALIZADO | Mensaje que envía el servidor para confirmar que se ha actualizado el contenido |

### 3.2. Cliente

| Código | Cuerpo | Descripción |
|---|---|---|
| 4000 | Comienzo | Mensaje que envía el usuario para conectarse al servidor |
| 4001 | LOGIN + PASS | Mensaje que envía el usuario con su usuario y su contraseña |
| 4002 | CREAR_CUENTA + PASS | Mensaje que envía el cliente para crear una cuenta |
| 4003 | FINANCIAR_PROYECTO + idProyecto | Mensaje que envía el cliente con la información del proyecto que quiere financiar |
| 4004 | FINANCIAR_CREADOR + idCreador | Mensaje que envía el cliente con la información del proyecto que quiere financiar |
| 4005 | RELLENAR_FORMULARIO + argumentos | Mensaje que envía el cliente con el formulario y los argumentos específicos de cada uno |
| 4006 | INDICAR_METODO | Mensaje que envía el cliente con el método de pago |
| 4007 | INFORMACIÓN_PAGO | Mensaje que envía el cliente con la información de pago |
| 4008 | RECLAMAR_RECOMPENSAS + recompensas | Mensaje que envía el cliente para recibir las recompensas de cierta financiación |
| 4009 | SUSCRIPCIÓN_PATREON + idCreador | Mensaje que envía el cliente para indicar el creador al que se quiere suscribir |
| 4010 | CANCELAR | Mensaje que envía el cliente para cancelar una suscripción o una financiación de un proyecto |
| 4011 | CANCELAR_SUSCRIPCIÓN | Mensaje que envía el cliente para cancelar una suscripción a un creador de contenido junto con el idSuscripción para que el servidor lo pueda cancelar |
| 4012 | CANCELAR_FINANCIACION | Mensaje que envía el cliente para cancelar una financiación a un creador de contenido junto con el idFinanciacion para que el servidor la pueda cancelar |

| | | |
|---|---|---|
| 4013 | CANCELADA_SUSCRIPCIÓN | Mensaje que envía el cliente para finalizar el proceso de cancelación de la suscripción porque ya ha terminado |
| 4014 | CANCELADA_FINANCIACION | Mensaje que envía el cliente para finalizar el proceso de cancelación de la financiación porque ya ha terminado |
| 4015 | AÑADIR_FUNCIONES_PROYECTO | Mensaje que envía un cliente autenticado para poder tener las funciones de proyecto |
| 4016 | INFORMACION_FINANCIACION*idProyecto financiadores | Mensaje que envía el cliente para dar la información sobre la financiación de un proyecto junto con idProyecto y la lista de financiadores |
| 4017 | CANCELAR_PROYECTO+idProyecto | Mensaje que envía el cliente autenticado como proyecto para cancelar el proyecto junto con el idProyecto |
| 4018 | CAMBIAR_PROYECTO | Mensaje que envía el cliente para cambiar las funciones que tiene como autenticado cliente a autenticado proyecto |
| 4019 | logout | Mensaje que envía el cliente para cerrar sesión en el servicio |
| 4020 | CAMBIAR_CREADOR | Mensaje que envía el cliente para cambiar a modo creador |
| 4021 | AÑADIR_FUNCIONES_CREADOR | Mensaje que envía el cliente para crear una cuenta de creador |
| 4022 | OBJETIVO*idCreador* subscriptores financiadores | Mensaje que envía el cliente con el objetivo alcanzado |
| 4023 | CONTENIDO_A_ACTUALIZAR | Mensaje que envía el cliente con el contenido a actualizar |
| 4024 | CANCELAR+idCreador | Mensaje que envía el cliente con el id de la cuenta que quiere cancelar |
| 4025 | CREAR_PROYECTO*idProyecto | Mensaje que envía el cliente con el id del proyecto que quiere crear |

# 4. Evaluación de la aplicación

El correcto funcionamiento de la práctica lo mostramos en el video adjunto del ejercicio 5, lo hemos decidido así porque es más claro que con capturas. En el **video1** se explica la creación del servidor con nodejs, el uso de socket.io, los archivos de javascript usados y el registro de usuarios. En el **video2** se termina de explicar el resto de funcionalidad; crearProyectos, listarProyectos, iniciarSesion y cambiarModo.

## 4.1. Explicación de la implementación

Para la implementación de nuestra aplicación hemos decidido usar javascript y en concreto nodejs. Esto es debido al carácter que tiene nuestra aplicación, principalmente enfocada a una página web. Siendo esta la parte de cliente que la realiza gestionUsuarios.js y el procesamiento del servidor lo lleva a cabo el archivo servidor.js.
La comunicación se realiza mediante socket.io [1] que es una librería para javascript que permite comunicación bidireccional basada en eventos. Para ello la instalamos con npm (que es un gestor de módulos para nodejs).
Para ejecutar el servidor y abrirlo en el puerto 8080, usamos nodejs (mirar código para mayor detalle).

```
var httpServer = http.createServer();

httpServer.listen(8080, () => {
    console.log('Servidor iniciado');
});
```

Después para abrir el socket y permitir la comunicación

```
var io = socketio.listen(httpServer);

io.sockets.on('connection', function(client){

});
```

El cliente (gestionUsuarios.js) se conectaría al servidor con

```
var serviceURL = 'localhost:8080';
var socket = io.connect(serviceURL);
```

Para ello es necesario indicarle la ruta de socket.io en la carpeta node_modules y el archivo javascript que maneja las acciones del cliente (gestionUsuarios.js) y se comunica con el servidor, eso se realiza en los archivos .html

```
<script type="text/javascript" src="/socket.io/socket.io.js"></script>

<script type = "text/javascript">
    var url= '/gestionUsuarios.js';
    var head = document.getElementsByTagName('head')[0];
    var script = document.createElement('script');
    script.type = 'text/javascript';
    script.src = url;
    head.appendChild(script);
    console.log("a adiendo gestionUsuarios");
</script>
```

Desde el cliente para enviar un mensaje al servidor:

```
socket.emit('registro',{nick:nick, password:password});
```

Y el servidor recibe ese mensaje y los datos (nick:nick, password:password)

```
client.on('registro',function(data){
    //acciones
});
```

Toda esta comunicación se hace con socket.io y es en los mensajes donde se realizan los cambios en la página web de cara al usuario y por tanto su cambio de estado (mirar diagrama de estados). El proceso es entonces el siguiente: Primero se inicia la comunicación con socket.io (por parte del cliente o el servidor), después se recibe el mensaje y se envía la respuesta. Esta respuesta provocará el cambio de estado (procesada por el servidor). Es importante añadir que se podrán conectar al servidor, tantos usuarios como pestañas o navegadores se abran. Y el logout será el cierre de las mismas.

### 4.2. Requisitos para la instalación

Para le ejecución del código es necesario instalar nodejs, socket.io, request y npm. Comandos usados por los miembros del grupo para la práctica (dependerá de la distribución linux):

```
sudo apt-get install nodejs
sudo apt-get install npm
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
sudo apt-get install -y nodejs
sudo npm install socket.io --save
sudo npm install request --save
```

## 5. Conclusión y dificultad

Tal y como se ha podido entrever a lo largo de la memoria, hemos elegido estas tecnologías debido a su gran importancia hoy en día y a la versatilidad que nos ofrece, siendo consciente del trabajo que iba a suponer por nuestra parte para aprender por completo de cero. Cualquier aplicación que se precie (y la nuestra en concreto) hará uso de una página web como forma de interactuar con el cliente y es necesario la comunicación bidireccional y en tiempo real entre cliente y servidor (uso de socket.io). El servidor lo lanzamos con el uso de nodejs (entorno de tiempo de ejecución de javascript) y algunas librerías. Y es importante destacar el uso de npm para la gestión de los módulos de nodejs. Con cada una de estas tecnologías nos hemos encontrado con problemas y ha sido precioso hacer uso de la documentación ofrecidas por ellas, sin embargo con este ejercicio hemos conseguido desarrollar nuestra capacidad de adaptación a nuevas tecnologías y hemos podido observar cómo usar otros lenguajes para la comunicación entre cliente y servidor, cambio de estados, front-end y utilización de sockets.

## 6. Referencias

[1]: https://socket.io/
[2]: https://www.npmjs.com/
[3]: https://nodejs.org/en/docs/
[4]: https://www.javascripttutorial.net/es6/javascript-map/