Universidad del País Vasco

Sistemas Web

Memoria Técnica - Back End

Phonos

Nicolás Aguado, Asier Contreras, Martín Horsfield y Abraham Casas

7 de enero de 2024

Índice

1. Introducción

Phonos es una grabadora y reproductora de audio, del estilo de una aplicación de "notas de voz". Con una interfaz sencilla, permite grabar audios desde el propio navegador y contiene una sincronización para guardar las grabaciones en el servidor. Los usuarios podrán registrarse en la aplicación usando distintos proveedores y guardar así sus audios. Además existirá la posibilidad de compartir mediante un enlace cada una de las grabaciones guardadas en la nube.

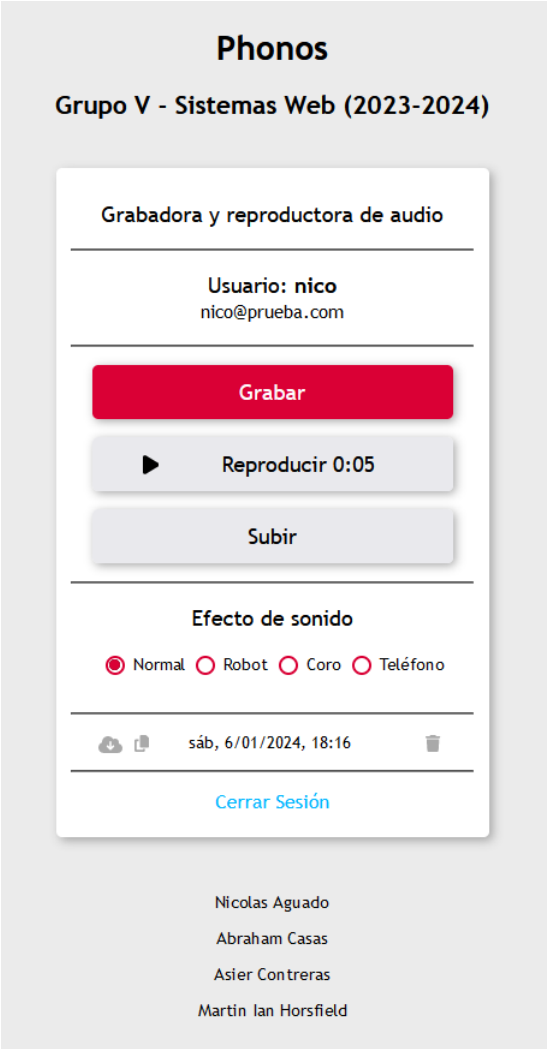

Figura 1: Vista principal de la aplicación

2. Back-End

2.1. API

La API se encargará de realizar todos los métodos de la grabadora de audio (list, upload, delete y play). Para realizar todas las tareas relacionadas con el API, se ha creado otro *router* de express en (`routes/api.js`).

2.1.1. /api/list/

La función /list devuelve los ficheros de audio asociados a cada cuenta de usuario. El router de list tratará la solicitud GET a la ruta '/list'. Cuando se trata la solucitud GET se ejecuta una función asíncrona que toma los parámetros req y res.

La respuesta a la petición GET se realiza con res.send(), que es un método de Express usado para enviar una respuesta al cliente. En este caso se envía un JSON que estará formado por la clave *"files"* y la lista de audios que devolverá el método *handleList*. Como *handleList* es una función asíncrona es necesario usar el `await` para esperar a la solución de la promesa, que en este caso será el resultado de la llamada a la BD.

En el ejemplo de la práctica envía como parámetro un 'id' de usuario. En nuestra BD los usuarios se identifican por el correo electrónico y el tipo de autentificación, por lo que en vez de enviar por parámetro un id, se ha optado por enviar `req`, de esta manera se puede acceder con `req.session` tanto al mail como al tipo de autentificación para hacer la llamada a la BD y solicitar los audios.

2.1.2. /api/upload/

La función /upload es la encargada de subir el audio actual que está guardado en la grabadora. En un principio, el documento de indica como posible guía a utilizar el *middleware* multer. Este middleware trae una serie de inconvenientes (poco control, esquema no flexible). Entonces, se ha decidido no usarlo. Con nuestro modelo, en dos maquinas separadas del estilo aplicacion - base de datos no encaja. Nos interesa guardar los archivos en la propia base de datos, no en el disco donde se ejecute la aplicacion.

Como alternativa, se ha utilizado la codificación en *base64*; con esto lo que se consigue es pasar a una cadena de caracteres el *blob* de audio.Así, lo que se guardará en la base de datos es una cadena de caracteres de tamaño variable (según el tamaño del audio)(base64), un identificador único para cada audio y la fecha en la que se grabó dicho audio.

Para los identificadores únicos de cada audio se ha decidido hacer un resumen hash del string base64. Se ha decidido usar el exelente algoritmo *Cyrb53* [4]. Este algoritmo producirá resúmenes de 32 caracteres, además de que prorcionará una protección excelente ante las colisiones.

2.1.3. /api/delete/

La función /delete es la encargada de manejar las solicitudes delete del cliente. Dicha solicitud en nuestro caso, se utiliza para borrar los audios del usuario que haya iniciado la sesión. Lo primero que hace es verificar si existe una sesión de usuario. En caso de no haber una sesión activa, responde con un código de error 401 (*Unauthorized*). En cambio si la sesión existe, mediante el parámetro id de la URL obtiene el identificador del audio a borrar.

Posteriormente, busca el índice del audio en la propiedad useraudios de la sesión del usuario (req.session.useraudios). Utiliza la función findIndex para encontrar la posición del audio en el array según su ID. Si el audio no se encuentra (índice igual a -1), responde con un código de estado 404 (Not Found) y un mensaje indicando que el audio no pertenece al usuario. Si se encuentra, elimina el audio del array useraudios en la sesión del usuario usando splice.

Tras el borrado exitoso del audio, utiliza db.users.findAndModify para actualizar la información de usuario en la base de datos MongoDB. La actualización se realiza para el documento que cumple con la condición: correo (mail) y tipo de autenticación (authtype) coinciden con los de la sesión actual. Actualiza la propiedad audios del usuario en la base de datos con la nueva información del array useraudios de la sesión del usuario.

Por último, si hay algún error durante el proceso de actualización en la base de datos, responde con un código de estado 500 (Internal Server Error) y un mensaje indicando que hubo un error en la base de datos. Si la actualización es exitosa, responde con un objeto JSON que contiene la lista actualizada de archivos del usuario ("files": ...), donde files es el resultado de una llamada a la función handleList(req).

2.1.4. /api/play/:audioID

La función `api/play/:audioID` es la que obtiene los audios del servidor y se los envía al cliente (en formato base64).

Cuando un usuario genera un enlace compartido de una grabación de voz, éste queda en formato `http://[dominio]/share/:audioID`. Entonces, el servidor detectará esta ruta y renderizará la vista principal en modo *reproductora sin login* (igual que la vista principal, pero sin lista de audios ni

posibilidad de grabar ni iniciar sesion). A su vez la aplicación cuando detecte que se está cargando con un parámetro `share` en la dirección url, lanzará una petición automáticamente a el endpoint del servidor `api/play/:audioID`. Cuando reciba el string en base64, entonces lo decodificará y se le hará disponible al usuario (con los métodos ya implementados en la parte anterior de la práctica).

2.2. Autenticación

Una parte clave de la aplicación se refiere a la autenticación. Se hacen uso de las cuentas de usuario para poder guardar los audios en el servidor. Entonces, habrá una serie de métodos para el registro y login. Para organizar la implementación, se ha creado un *router* de express en `routes/auth.js`.

Esta aplicación tendrá tres tipos de cuentas. Existirán usuarios nativos (registrados mediante la propia aplicación), usuarios creados a través de google y usuarios de github.

Entonces, un aspecto a tener en cuenta es que, al mezclar distintos tipos de inicio de sesión, se ha decidido identificar a cada usuario unívocamente por su **correo electrónico y tipo de autenticación**. De esta manera, el usuario `nico@gmail.com, native` será distinto a `nico@gmail.com, google`.

De cara a que las otras partes de la aplicación puedan trabajar y sepan si está iniciada la sesión (y de qué usuario) se usarán sesiones. El framework que proporcionará esto será *express-sessions* [6]. Entonces, las variables que contendrán la información relativa al usuario serán:

```
req.session.user        - Nombre para mostrar en los títulos
req.session.useraudios  - Audios del usuario
req.session.mail        - Correo electrónico del usuario
req.session.authtype    - Native, google ó github
```

Como aspecto a destacar, para comprobar si la sesión está iniciada o no se comprobará si el parámetro `req.session.user` es nulo o no.

2.2.1. Registro (Nativo)

El registro es relativamente sencillo. Se necesitará proporcionar un correo electrónico, un nombre y una contraseña. El sistema comprobará si el usuario está ya registrado y si no lo está procederá.

2.2.1.1. 2 Pasos

Como elección de diseño se ha dedidido incluir un registro en 2 pasos. Entonces, una vez introducidos todos los datos, se debe recibir un código

de verificación (Figura 2.2.1.1) en el correo electrónico que luego hay que introducir.

Todo el código relativo a la implementación de esta funcionalidad está creado. El problema ha llegado en que enviar correos automatizados (gratis) y que no sean detectados como spam o auto rechazados es muy complejo. El informático Carlos Fenollosa tiene un excelente artículo [5] sobre este asunto. Entonces, para no pagar el coste y porque esto es una aplicación educativa, el código de verificación será **auto-rellenado**.

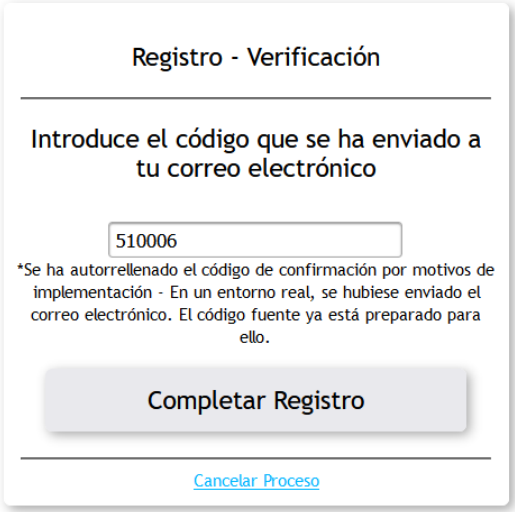

Figura 2: Código de Verificación para Registro en 2 Pasos

2.2.1.2. Hasing de Contraseñas

A la hora de guardar la información del usuario, se tiene que saber su contraseña para luego comprobarla con la que se está introduciendo para iniciar sesión. Ahora, en vez de guardar la contraseña en texto plano, para mejorar la seguridad en caso de compromiso de la base de datos, guardaremos un *hash* en vez de la contraseña en sí. Como algoritmo de cifrado y descifrado, uno de los más populares y el que se ha decidido usar en el proyecto es *BCrypt* [2]. En concreto, la implementación en node del algoritmo, usando el paquete de npm *bcrypt* [3].

2.2.2. Login (Nativo)

El login se realizará usando el correo electrónico proporcionado por el usuario y con su contraseña.

2.2.3. OAuth - Google

Para poder añadir como método de inicio de sesión las cuentas de google para nuestra aplicación, el estándar actual es *OAuth*. Para implementar este modelo harán falta realizar una serie de pasos.

1. Primero, habrá que registrar la aplicación dentro de la consola de administración de google. Habrá una cuenta de google que será *responsable* de la aplicación, y ésta proporcionará algunos datos relativos a ella (permisos que se desean, url principal, nombre, etc...).

2. La cuenta responsable generarán una serie de *secretos* e identificadores para que al llamar a la API de Google ésta sepa qué aplicación es la que se está identificando.

3. Para que una vez se ha iniciado sesión con la cuenta de Google la aplicación reciba de vuelta los datos (el usuario registrado en sí), se le indicarán unas direcciones de *callback* a donde volver enviando la información.

*Nota: Google tiene una política muy restrictiva respecto a sus cuentas de usuario. Para implementar autenticación *OAuth* fuera de su entorno de pruebas, hay que firmar varios contratos legales y proporcionar muchas explicaciones. Es por esto que para esta aplicación educativa nos limitaremos al entorno de pruebas. Esto nos supondrá que existirán una serie de **usuarios aprobados** que podrán iniciar sesión. Entonces, **no** podrá iniciar sesión cualquiera sino sólo los correos electrónicos previamente aprobados.

2.2.4. OAuth - GitHub

La autenticación con *OAuth* dentro de GitHub sigue prácticamente los mismos pasos que la de Google excepto por un detalle.

A la hora de enviar la información del usuario de vuelta, Google no tiene ningún problema en facilitar la dirección de correo electrónico, mientras que Github lo considera *información sensible*. Estos datos sólo son proporcionados después de firmar contratos de confidencialidad y licencias legales con GitHub. De nuevo, como trabajamos con una aplicación simplementa educativa, buscaremos una solución. Entonces, a la hora de guardar el correo electrónico de nuestros usuarios de GitHub en la base de datos, usaremos su dirección (URL) del perfil como alternativa.

Además, GitHub no tiene una política *OAuth* tan estricta como Google y con este método de autenticación **sí** podrá iniciar sesión cualquiera.

2.3. Cleanup

El servidor se puede llenar de audios muy fácil, y tener que borrarlos manualmente puede ser pesado. Para arreglar este problema, se ha propuesto implementar a nivel de aplicación un programa que cada hora borre los audios que sean más de 5 días.

Se pide que se implemente con `setInterval` y es lo que se ha hecho en este proyecto. Aun así, creemos que con una tarea `cron` (o similar) se puede hacer de manera más eficiente porque `setInterval` no es muy exacto pasado mucho tiempo.

3. Deployment

Una vez finalizado el desarrollo, es hora de hacer disponible la aplicación a todos los usuarios. Se ha decidido alojar *la aplicación node.js* en la dirección web `https://sw.nico.eus`. Por motivos de facilitar una re-implementación, en toda esta explicación se ha sustituido la dirección IP real de la máquina por `[ip]`.

3.1. Callback URL

Entonces, lo primero a tener en cuenta será cambiar la dirección de redirección para todas las integraciones *OAuth*. No tiene sentido que les indiquemos a los proveedores de autenticación que redirijan a `localhost`, cuando tenemos la grabadora de voz alojada en internet y no en la máquina local.

Así, donde antes estaba

```
http://127.0.0.1:3450/auth/[proveedor]/callback
```

ahora indicaremos

```
https://sw.nico.eus/auth/[proveedor]/callback
```

3.2. Configuración del Servidor (Node.js)

Como máquina para alojar el proceso *node*, se ha hecho disponible un servidor virtual con 4 vCPUs y 4GB de RAM con ubicación en la ciudad alemana de Núremberg.

Para que esté el subdominio accesible en internet, se han añadido los registros correspondientes en el servidor DNS que gestiona `nico.eus`.

```
sw.nico.eus         A           [ip]
sw.nico.eus         CAA         0 issue "letsencrypt.org"
```

Además, se ha configurado nginx como *reverse-proxy* para poder acceder fácilmente a la URL y no tener que insertar `http://[ip]:8450` para acceder a la grabadora de voz. Esto también nos permitirá configurar un certificado TLS-SSL para garantizar la seguridad de las comunicaciones mediante el proveedor gratuito *let's-encrypt*. Entonces, después de aplicar las correcciones de seguridad necesarias y los ajustes adecuados, la configuración de nginx ha quedado de la siguiente manera:

/etc/nginx/conf.d/domains/sw.nico.eus.ssl.conf

```
server {
        listen       [ip]:443 ssl;
        server_name sw.nico.eus ;
        error_log   /var/log/apache2/domains/sw.nico.
           eus.error.log error;

        ssl_certificate      /home/nico/conf/web/sw.nico
           .eus/ssl/sw.nico.eus.pem;
        ssl_certificate_key /home/nico/conf/web/sw.nico
           .eus/ssl/sw.nico.eus.key;
        ssl_stapling        on;
        ssl_stapling_verify on;

        # TLS 1.3 0-RTT anti-replay
        if ($anti_replay = 307) { return 307 https://
           $host$request_uri; }
        if ($anti_replay = 425) { return 425; }

        location ~ /\.(?!well-known\/|file) {
                deny all;
                return 404;
        }

        location / {
                proxy_pass http://127.0.0.1:3450;
                proxy_ssl_server_name on;
        }

        location @fallback {
                proxy_pass https://[ip]:8443;
        }

        location /error/ {
                alias /home/nico/web/sw.nico.eus/
                   document_errors/;
        }

        proxy_hide_header Upgrade;
```

```
}
```

Por último, para ejecutar la aplicación, se ha clonado el repositorio disponible en github y mediante la utilidad *pm2* [1] se ha creado un proceso para alojarla.

```
root@serv:~/Phonos-main# pm2 start bin/www
[PM2] Applying action restartProcessId on app [www](ids: [ 0 ])
[PM2] [www](0) OK
[PM2] Process successfully started
```

```
| id | name   | mode  | rep   | status  | cpu | memory   |
-------------------------------------------------------------
| 0  | www    | fork  | 15    | online  | 0%  | 10.2mb   |
```

3.3. Configuración del Servidor (MongoDB)

La configuración del servidor Mongo es fácil inicialmente. En una máquina tenemos que instalar MongoDB, y hacerla accesible a la aplicación. Sin embargo, hemos querido hacerlo más interesante, y hemos decidido que el servidor MongoDB esté en una máquina distinta al servidor web.

El servicio de MongoDB está situado en `martinh.info` en el puerto `27017`. El servidor está configurado para que automáticamente inicie Mongo en el startup.

Hubo problemas consiguiendo que funcionase MongoDB en el servidor `martinh.info` debido a la virtualización del procesador del servidor, la versión de Ubuntu que se utiliza y la versión de MongoDB. La solución fue instalar una versión antigua de MongoDB con una solución proporcionada en el siguiente enlance: askubuntu

Cuando se solucionaba un problema, surgía otro como este. Finalmente, MongoDB funcionaba y se configuró el cortafuegos del servidor para que aceptasen paquetes en ese puerto.

Ahora el servicio está abierto a internet y por lo tanto, abierto a vulnerabilidades. MongoDB ofrece la opción de crear usuarios de autenticación para poder acceder a la base de datos y está activo en nuestro servidor.

Este es el mensaje del estado del servicio:

```
 mongod.service - MongoDB Database Server
  Loaded: loaded (/lib/systemd/system/mongod.service;
  enabled; vendor preset: enabled)
  Active: active (running) since
  Sat 2023-12-30 15:58:00 CET; 1 week 0 days ago
    Docs: https://docs.mongodb.org/manual
Main PID: 742 (mongod)
  Memory: 107.5M
     CPU: 1h 8min 17.400s
  CGroup: /system.slice/mongod.service
          742 /usr/bin/mongod --config /etc/mongod.conf

Dec 30 15:58:00 martinh.info systemd[1]:
Started MongoDB Database Server.
```

Para acceder al servidor MongoDB, desde nuesta apliación, usamos el siguiente *string*.

```
mongodb://phonosuser:*********@martinh.info:27017/phonos
?authSource=admin
```

- **mongodb:** indica que estamos buscando un servicio de MongoDB.
- **phonosuser** es el nombre de usuario que vamos a usar
- ******** en los asteriscos debería de estar la contraseña del usuario.
- **@martinh.info** es el nombre del servidor donde está el servicio de MongoDB.
- **:27017** es el puerto donde está escuchando MongoDB.
- **/phonos** es el nombre de la base de datos, en nuestro caso, se llama `phonos`
- **?authSource=admin** es un aviso para MongoDB. Se le está diciendo que busque el usuario en la base de datos `admin` y no en la que busca por defecto, la proporcionada (`phonos`).

Preferiblemente se deberían de haber usado variables de entorno para guardar la contraseña sin que sea visible en *plaintext* pero no guardamos nada importante en la base de datos y además el usuario no tiene acceso de administrador.

Referencias

[1] Battle-Hardened Node.js Applications. https://pm2.io/, [Online; accessed 30-December-2023]

[2] bcrypt - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Bcrypt, [Online; accessed 30-December-2023]

[3] bcrypt npm package. https://www.npmjs.com/package/bcrypt, [Online; accessed 30-December-2023]

[4] bryc - cyrb53 algorithm in javascript. https://github.com/bryc/code/blob/master/jshash/experimental/cyrb53.js, [Online; accessed 6-January-2023]

[5] Carlos Fenollosa - After self-hosting my email for twenty-three years I have thrown in the towel. The oligopoly has won. https://cfenollosa.com/blog/after-self-hosting-my-email-for-twenty-three-years-i-have-thrown-in-the-towe
html, [Online; accessed 30-December-2023]

[6] express-session npm package. https://www.npmjs.com/package/express-session, [Online; accessed 30-December-2023]