UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

CREACIÓN DE UN SECUENCIADOR WEB CON LA WEB AUDIO API

Jairo Fraga Álvarez

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN

INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

"CREACIÓN DE UN SECUENCIADOR WEB CON LA WEB AUDIO API"

Autor: **Jairo Fraga Álvarez**

Director: **Carlos Delgado Hita**

Tribunal:

Presidente: .....

Vocal 1º: ..... Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

RESUMEN

Los avances experimentados durante la última década en las tecnologías web han permitido llevar a cabo desde un cliente web tareas que hasta entonces estaban ligadas necesariamente a la instalación de paquetes de software en la máquina en la que se querían utilizar. Esto ha facilitado el desarrollo de aplicaciones web en nuevos campos de trabajo, con las ventajas que ofrece el realizarlo sobre un entorno web, como pueden ser el alto nivel de compatibilidad alcanzado entre distintos tipos de dispositivo y sistemas operativos, o el acceso instantáneo a los recursos desde distintos dispositivos sin necesidad de realizar instalaciones previas.

En este contexto, la aparición de la Web Audio API ha abierto nuevos horizontes a la hora de interactuar con audio en un entorno web, haciendo posible llevar a cabo tareas de procesamiento avanzado con una latencia mínima.

En el mundo del audio, un secuenciador es una herramienta utilizada para componer patrones rítmicos o melódicos mediante la reproducción y alteración de piezas de audio digitales. En este proyecto se han tratado de explotar las capacidades de la Web Audio API para crear un secuenciador de audio en un entorno web. Para ello, se ha procedido a la creación de una librería específica sobre Angular, *framework* de JavaScript con una filosofía de trabajo basada en la modularidad.

Palabras clave: Web Audio API, MIDI, Angular, secuenciador web, música

ABSTRACT

The advances experienced over the last decade in web technologies have allowed us to carry out tasks from a web client that until then were necessarily linked to the installation of software packages on the machine on which they wanted to be used. This has facilitated the development of web applications in new fields of work with the advantages offered by the web environment, such as the high level of compatibility achieved between different types of device and operating systems or instant access to resources from different devices without the need for prior installations.

In this context, the emergence of the Web Audio API has opened new horizons when interacting with audio in a web environment, making it possible to carry out advanced processing tasks with minimal latency.

In the audio world, a sequencer is a tool used to compose rhythmic or melodic patterns by playing and altering digital audio pieces. This project has attempted to exploit the capabilities of the Web Audio API to create an audio sequencer in a web environment. To do this, we have proceeded to create a specific library on Angular, a JavaScript framework with a work philosophy based on modularity.

Keywords: Web Audio API, MIDI, Angular, web sequencer, music

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Este trabajo trata sobre el proceso de creación de un secuenciador de audio en el entorno web. Para ello, en el proceso se ha ahondado en las capacidades que ofrecen las tecnologías de entorno cliente y, en especial, JavaScript.

La parte principal de este proyecto se ha basado en la realización de una aplicación Angular que extienda las capacidades de la Web Audio API de JavaScript, y facilite el uso de esta a la hora de construir aplicaciones basadas en la interactuación con audio.

Aunque se trata de un proyecto cuyo desarrollo está claramente enfocado al entorno de cliente web, se ha considerado oportuno el desarrollo de una modelo de datos y una aplicación de entorno servidor, que complementen los trabajos realizados sobre la aplicación de cliente web, de forma que el proyecto constituya una aplicación web completa y totalmente funcional.

En la siguiente sección se expondrán las principales motivaciones y los objetivos que se han tratado de conseguir con este trabajo.

En la sección tres se presenta cada una de las tecnologías principales con las que se ha realizado este proyecto y se justifica la elección de cada una de estas herramientas.

Seguidamente, se explicarán las razones que han motivado el desarrollo de una aplicación externa en forma de librería, de modo que pueda ser utilizada tanto en este proyecto, como en proyectos futuros.

La sección cinco, titulada *Desarrollo*, expone la forma en la que se han estructurado cada una de las áreas del proyecto, así como las soluciones propuestas ante problemas concretos a los que se ha enfrentado el desarrollador.

En la sección seis se exponen los resultados obtenidos a través de imágenes de la aplicación final, así como indicaciones sobre el funcionamiento de esta.

Finalmente, se presentan las conclusiones extraídas de la realización de este trabajo, así como un análisis de posibles trabajos futuros que podrían ser llevados a cabo sobre la aplicación.

2. MOTIVACIONES Y OBJETIVOS

Las motivaciones para la realización de este proyecto nacen como respuesta a la carencia de una herramienta que permita desarrollar tareas de composición musical mediante secuenciación con funcionalidades avanzadas, dentro de un entorno web.

Un secuenciador musical, o secuenciador de audio, es un aparato o aplicación de software que permite realizar acciones tales como grabar, editar o reproducir música, típicamente, controlado de forma externa a través de interfaces CV/Gate o MIDI [1]. Una de las principales características de un secuenciador musical es su capacidad para estructurar patrones de audio, de modo que formen una pieza musical.

Aunque el concepto de secuenciador de audio surgió a través de la creación de máquinas especialmente construidas para tal fin, pronto la evolución en las capacidades de los ordenadores de uso personal condujo al desarrollo de paquetes de software que trataban de replicar el funcionamiento de los secuenciadores hardware, de modo que pudieran ser usados en un ordenador personal [2]. Sin embargo, pese a que estas soluciones se han venido desarrollando desde hace décadas, ninguna de ellas se ofrece en un entorno web, con las ventajas que ello aportaría. Entre estas ventajas podemos destacar:

- Compatibilidad entre distintos equipos y sistemas operativos.
- Acceso sin necesidad de instalaciones previas.
- Posibilidad de trabajar en un mismo proyecto desde distintos equipos.

Si bien es cierto que existen proyectos como BandLab (https://www.bandlab.com) o Soundation (https://soundation.com), este tipo de herramientas se basan en la adaptación a la web de los software DAW (Digital Audio Workstation). Este tipo de herramientas abarca un entorno de trabajo mucho más amplio que el de secuenciación de patrones y está dedicado a la creación de obras musicales completas, desde su grabación, hasta la exportación de la obra final, pasando por todas las labores intermedias, como son la estructuración de los fragmentos, la mezcla, etc.

El único ejemplo encontrado de implementación del funcionamiento de un secuenciador en el entorno web ha sido el del proyecto iO-808 (https://io808.com), que trata de replicar el funcionamiento de uno de los secuenciadores de hardware más icónicos, el Roland TR-808. Si bien el trabajo realizado en dicho proyecto es más que satisfactorio, este parte de unos requisitos funcionales muy limitados, propios de la época a la que pertenece el aparato original, que fue lanzado en 1980. Así, este tipo de máquinas sólo podía utilizar sonidos sintetizados que trataban de emular sonidos de instrumentos de percusión y contaban con rigideces tales como un máximo de 4 posiciones por compás.

Figura 1. Roland TR-808 y captura de la aplicación iO-808

Fue el desarrollo de los *sampler* (muestreadores) con capacidades de secuenciación lo que permitió empezar a trabajar con audio real. Este tipo de aparatos permitían escoger y personalizar los sonidos a reproducir, además de permitir un número de posiciones disponibles dentro de una secuencia, mucho más elevado, ofreciendo alguno de los primeros modelos de este tipo hasta 192 posiciones por compás. Sobre este tipo de secuenciadores avanzados, resulta interesante el proyecto SP950 (https://wavetracing.com/products/sp950web), que trata de replicar el comportamiento del secuenciador E-mu SP 1200. Sin embargo, dicho proyecto se encuentra en fase de desarrollo, y en el momento de creación de nuestra aplicación, solo permite la reproducción de sonidos en tiempo real, careciendo de la habilidad de programar patrones de reproducción, funcionalidad principal de un secuenciador.

Figura 2. E-mu SP 1200 y captura de la aplicación SP950

El proyecto del presente trabajo tiene como objetivo el crear una aplicación web con las funcionalidades esperadas de un secuenciador de audio avanzado, entre ellas:

- Seleccionar las fuentes de audio que se desean utilizar.
- Modificar de forma independiente para cada fuente de audio parámetros tales como volumen o velocidad de reproducción.
- Visualizar en tiempo real la representación gráfica de las muestras siendo reproducidas.
- Grabar y reproducir secuencias de audio con una precisión de 196 posiciones por compás.
- Cuantización: autocorregir posición de las notas tocadas con distintos rangos de precisión.

Como punto de partida para determinar el diseño y funcionamiento de la aplicación, se tomará como referencia la serie MPC de la marca Akai, un estándar en el mundo de los secuenciadores avanzados de hardware, la cual ha sido adaptada de forma reiterada en diversos productos, tanto de hardware como de software.

Figura 3. MPC 2000XL

Además de lo expuesto anteriormente, el sistema contará con las funcionalidades añadidas que ofrece una aplicación web, como son la capacidad de persistir en base de datos la información sobre las secuencias y de guardar en servidor los sonidos utilizados, para poder todo ello ser recuperado por el usuario en sesiones posteriores.

Por último, se considera interesante incluir en esta aplicación el soporte para el protocolo MIDI, de forma que se pueda interactuar con la aplicación a través de controladores externos.

3. TECNOLOGÍAS UTILIZADAS

En este apartado se describen las principales tecnologías utilizadas en el desarrollo de la aplicación, así como las razones que han motivado a escoger las mismas frente a otras opciones disponibles.

3.1. Tecnologías de Entorno Cliente

3.1.1. Angular

Figura 4. Logotipo de Angular

Angular es un *framework* de desarrollo JavaScript de código abierto, desarrollado y mantenido por Google, para la creación de aplicaciones de página única, o SPA (*Single Page Application*).

Antes de continuar, conviene mencionar que se suele establecer una distinción entre las versiones 1.x.x, y las versiones 2.0.0 o mayores. Así, mientras la primera versión estaba basada en el patrón de arquitectura MVC (Modelo-Vista-Controlador), la versión 2 introdujo grandes cambios en la filosofía del *framework* y este pasó a centrarse en el concepto de *componente*, que se explicará más adelante. De este modo el proyecto pasó a ser denominado AngularJS en sus versiones anteriores a la versión 2 y siendo Angular el nombre utilizado para todas las versiones a partir de la versión 2. Por tanto, cuando hablemos de Angular a lo largo del texto, nos estaremos refiriendo siempre a las versiones 2 y superiores.

Una de las características de Angular es que se trata de un entorno "agnóstico de plataforma", ya que permite desarrollar aplicaciones compatibles con distintos entornos, por lo que una aplicación desarrollada en Angular podrá ser utilizada con diferentes tecnologías de servidor y sistemas de gestión de bases de datos.

Angular utiliza el lenguaje de programación TypeScript, que es un superconjunto de JavaScript, al que extiende introduciendo el paradigma de la programación orientada a objetos, con conceptos tales como la herencia, la composición, las clases o las interfaces, y además permite la detección de errores en tiempo de compilación en lugar de en tiempo de ejecución. Cabe destacar que, al tratarse de un superconjunto de JavaScript, todo código JavaScript es válido en TypeScript, y el código escrito en Typescript es transpilado a JavaScript.

Angular se basa en el concepto de módulos y componentes, de forma que se pueden crear unidades de código independientes que pueden ser reutilizables en distintas partes de un proyecto o incluso en proyectos distintos. De entre los numerosos tipos de recurso utilizados en la arquitectura de los proyectos de Angular, cabe destacar tres de especial relevancia [3]:

- **Módulos**: Un módulo está formado por un conjunto de componentes, servicios, directivas, pipes y demás recursos disponibles en Angular. Representa la mayor unidad de medida para un bloque de código dentro de un proyecto de Angular. En la práctica, solo las aplicaciones de gran tamaño suelen dividirse en distintos módulos. Si bien es cierto, que casi toda aplicación creada en Angular importará varios módulos externos al proyecto.
- **Componentes**: Los componentes son bloques de código reutilizable y están formados normalmente por tres tipos de fichero: un fichero *.ts*, donde se define la lógica del componente; un fichero *.html*, que contiene el marcado de la vista; y un fichero *.css*/*.scss*, que define los estilos. Dependiendo de la complejidad de la aplicación, un componente puede utilizarse para representar varias pantallas, una sola pantalla, una utilidad que se usa en distintas pantallas o incluso utilizarse solo para aplicar un proceso de lógica desde el cliente web.
- **Servicios**: Los servicios son clases creadas para centralizar la lógica de tareas determinadas o para poder compartir una utilidad en distintos componentes de la aplicación, como puede ser la de comunicación entre el front-end y el back-end. Una de las cualidades de los servicios es que pueden ser usados siguiendo el patrón *singleton*, es decir, son instanciados una única vez y esa instancia es compartida por toda la aplicación, por lo que también resultan de utilidad para guardar variables globales de la aplicación.

Las razones por las que se ha escogido esta tecnología para desarrollar la parte del cliente web de la aplicación son principalmente el carácter modular comentado anteriormente, que hace que la aplicación sea fácilmente escalable, y la abstracción que se permite de la capa de datos y de la lógica de negocio, de modo que, si en algún momento se decidieran cambiar estos, la aplicación de cliente podría seguir funcionando sin necesidad de ajustes algunos.

También resultan interesantes las ventajas que suponen el uso del lenguaje TypeScript, como son la utilización de tipos y el uso de clases. Por último, cabe destacar que, siendo uno de los *frameworks* de desarrollo front-end más utilizados, cuenta con una amplia comunidad de usuarios que desarrollan utilidades específicas él y la información disponible sobre problemas o desarrollos concretos es somera.

En concreto se ha utilizado la versión 9, última versión publicada al inicio del desarrollo de este proyecto.

3.1.2. Web Audio API

La Web Audio API es un estándar de la W3C que permite controlar y procesar audio en la Web a través de una API de alto nivel de JavaScript.

El primer borrador de la especificación aparece en la W3C en 2011 [4] y su creación nace como respuesta a las limitaciones que ofrecía la utilización del elemento <*audio*> de HTML5, que era la forma de interaccionar con audio en la web hasta entonces. Entre las principales limitaciones del elemento de HTML5 se encuentran [5]:

- Falta de precisión en los controles de *timing*.
- Límite muy bajo del número de sonidos reproducidos a la vez.
- Carencia de un sistema fiable para precargar sonidos en memoria.
- Imposibilidad de la aplicación de efectos en tiempo real.
- Imposibilidad de analizar sonidos.

El funcionamiento de la Web Audio API se basa en la utilización de un paradigma de grafos, creando nodos que se conectan entre sí para formar una ruta de audio, del mismo modo que se realizaría en un equipo de audio en el mundo real.

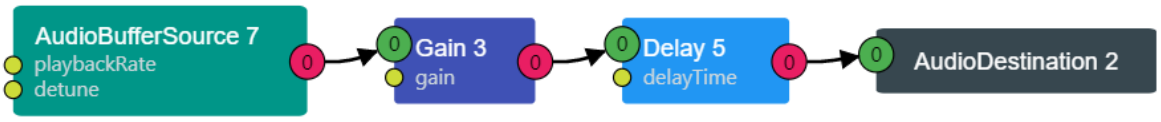

Figura 5. Ejemplo de cadena de audio

La figura 5 muestra la representación gráfica de cuatro nodos en la Web Audio API a través de la extensión Web Audio Inspector del navegador Chrome. El primer nodo representa la fuente de audio (en este caso un buffer), el segundo nodo representa la ganancia o amplitud de onda (volumen) aplicada sobre el audio, el tercer nodo representa el procesamiento de la señal digital (en este caso, aplicando un efecto de *delay*) y el último nodo representa la salida del audio (por defecto, los altavoces o auriculares del dispositivo).

Dado que la funcionalidad principal de nuestra aplicación se basa en la manipulación de audio en el entorno Web, el uso de la Web Audio API o de una herramienta construida sobre la misma resulta imprescindible para el desarrollo de esta.

En la sección siguiente, *Descripción de Necesidades*, se comentan con más detalle algunas consideraciones adicionales sobre aspectos que han tenido que ser tomados en cuenta a fin de utilizar la Web Audio API en conjunción con el *framework* Angular.

3.1.3. Web MIDI API

MIDI (*Musical Instrument Digial Interface*) es un protocolo de comunicación nacido para facilitar la comunicación y sincronización entre instrumentos musicales electrónicos, ordenadores y controladores [6]. De esta manera, un dispositivo (maestro) puede mandar a otro (esclavo) mensajes estandarizados para realizar acciones tales como tocar una nota, fijar un volumen, cambiar en tempo, etc. Asimismo, se puede utilizar un dispositivo para sincronizar el tempo de varios dispositivos.

La Web MIDI API es una especificación de la W3C que permite enviar y recibir mensajes MIDI desde el cliente web. Cabe destacar que, aunque el propósito inicial de esta tecnología fue el control de aparatos musicales, puede ser utilizada para realizar todo tipo de acciones. Un aspecto a tener en cuenta es que, pese a estar reconocida como un estándar de la W3C desde 2015, aun no todos los navegadores principales del mercado incluyen soporte para la Web Audio API.

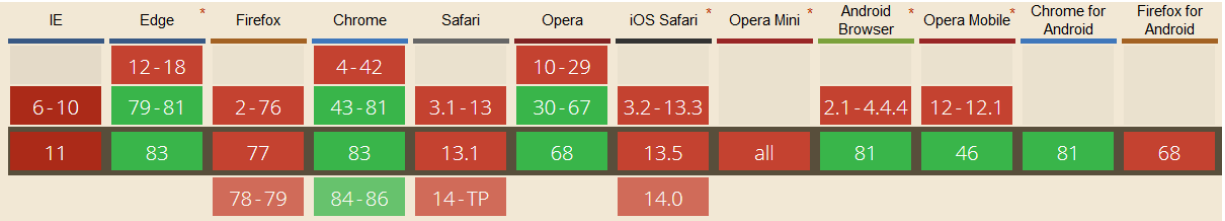

Figura 6. Soporte de la Web Audio API de los principales navegadores
(fuente: https://caniuse.com)

3.1.4. Bootstrap

Figura 7. Logotipo de Bootstrap

Bootstrap es un *framework* CSS de código abierto desarrollado por Twitter que ofrece facilidades para crear páginas web adaptables a distintos tipos de dispositivo.

Su filosofía está basada en el uso de clases sobre elementos del HTML que aplican unos estilos prefijados. De este modo, se facilita el proceso de desarrollo, ya que no resulta necesario aplicar estilos específicos a cada grupo de elementos. Además, se garantiza la compatibilidad entre distintos navegadores y la consistencia entre distintos proyectos. Entre otras cualidades, Bootstrap ofrece la capacidad de simplificar el uso de las utilidades Flexbox y Grid de CSS mediante la aplicación de clases específicas.

Se ha escogido utilizar Boostrap 4, última versión estable en el momento de iniciar este proyecto, dado que su uso facilita de forma importante las labores de diseño y maquetación, especialmente en lo referente a la "responsividad", o adaptabilidad a distintos tamaños de pantalla.

3.2. Tecnologías de Entorno Servidor

3.2.1. Spring Boot

Figura 8. Logotipo de Spring Boot

Spring Boot es un *framework* de desarrollo Java construido sobre el Spring. Se trata de un *framework* "opinionado", lo cual quiere decir que parte de unas asunciones sobre las técnicas o procedimientos que se han de llevar a cabo en la forma de desarrollar aplicaciones en Spring. De este modo, hace que sea mucho más fácil comenzar a trabajar en un nuevo proyecto, ya que ofrece la capacidad de construir arquetipos a través de la inclusión de dependencias y configuraciones específicas, de una forma rápida e intuitiva, y ajustándose a las necesidades del sistema.

Las principales funcionalidades ofrecidas por Spring Boot son [7]:

- Creación de aplicaciones Spring autónomas.
- Tomcat, Jetty o Undertow integrados (sin necesidad de desplegar ficheros WAR).
- Provee un inicializador "opinionado" de dependencias para simplificar la creación de configuraciones.
- Configura de forma automática librerías de terceros para Spring siempre que es posible.
- Provee utilidades como métricas, análisis y configuraciones avanzadas para entornos de producción.
- No requiere de generación de código ni configuraciones XML.

Pese a establecer unas pautas iniciales de configuración, sigue siendo un *framework* muy flexible y configurable, que se puede adaptar y extender en el caso de que se amplíen o modifiquen los requerimientos de la aplicación.

En este proyecto hemos hecho uso de Spring Boot en su versión 2.3 para la creación de un servicio RESTful, con el que se comunica el cliente web creado en Angular. Las principales razones para escoger esta tecnología han sido que, además de ser una herramienta muy poderosa, dispone de configuraciones específicas para distintos tipos de proyecto, así como tutoriales detallados para implementar estos [8].

3.2.2. PostgreSQL

Figura 9. Logotipo de PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional y basado en objetos, de código abierto, que extiende el lenguaje SQL [9].

Alguna de las características principales de PostgreSQL son el tener una gran cantidad de funcionalidades de forma nativa y el cumplir con los estándares SQL.

Además de las características antes mencionadas, una de las razones por las que se ha escogido PostgreSQL frente a otro tipo de sistemas de bases de datos relacionales ha sido el buen soporte que tiene para los datos en formato JSON. Este tipo de datos ha sido utilizado en la aplicación para guardar configuraciones y secuencias de los usuarios y, tras analizar la posibilidad de utilizar un sistema de base de datos no relacional, se concluyó que dada la atomicidad que presenta cada bloque de configuraciones o secuencias, lo idóneo sería la utilización de una base de datos relacional con buen soporte para datos de tipo JSON.

4. DESCRIPCIÓN DE NECESIDADES

Una vez escogidas y analizadas las principales tecnologías a utilizar para el desarrollo de la aplicación se ha considerado necesario el uso de algún tipo de librería o *framework* para trabajar con la Web Audio API en el entorno de Angular. Estos son los motivos principales que han conducido a tal conclusión:

- Pese a ser considerada una herramienta de alto nivel, la Web Audio API aún representa un elevado nivel de abstracción a la hora de interactuar con elementos de audio. A modo de ejemplo, una acción tan habitual como es reproducir un sonido requiere ejecutar una serie de acciones previas como crear un nuevo nodo fuente, cargarlo con los datos de audio, establecer parámetros como el volumen y conectar el nodo. Esto es así porque la forma en la que trabaja la Web Audio API es creando y destruyendo nuevos nodos cada vez que se quiere interactuar con ellos, siendo este proceso optimizado a través del recolector de basura de JavaScript.
- Se permite hacer ciertas asunciones sobre tareas concretas. Por ejemplo, resulta interesante utilizar una herramienta que asuma que si el usuario intenta reproducir un sonido que está actualmente en reproducción, este sea parado antes de volver a empezar a sonar, de forma que esto se controle de manera automática, sin necesidad de realizar tareas adicionales por parte del desarrollador.
- Esta opción es acorde a la filosofía de Angular, que intenta desacoplar tareas independientes entre sí.
- El trabajar con una herramienta específica de Angular aportaa las ventajas del tipado y el uso de clases en TypeScript. Por ejemplo, se permite utilizar distintos tipos de datos admitidos para crear un recurso de audio (a través de URL, de datos en bruto, de otro objeto de audio...) o crear un objeto específico que encapsule los datos referentes a un sonido (fuente de audio, volumen, velocidad de reproducción, etc.).
- Permite la reutilización en futuros proyectos. Las funcionalidades concretas pueden ser fácilmente implementadas en otro proyecto de Angular procediendo a la instalación del paquete en el proyecto.

Habiendo puesto de manifiesto la conveniencia de utilizar algún tipo de *framework* o librería sobre Angular que ayude a interactuar con la Web Audio API, se pasó a realizar un estudio sobre las herramientas disponibles en el mercado.

Tras recabar información sobre distintos recursos y descartar la mayoría de ellos por no ajustarse a las necesidades requeridas, se decidió profundizar en el estudio de tres herramientas en concreto: Tone.js, ng-web-apis y HowlerJS:

Aunque las herramientas analizadas resultaron ser productos de alta calidad, se detectaron ciertas carencias respecto alguno de los requisitos buscados. Se exponen continuación algunas de las conclusiones extraídas del estudio de la documentación y de las pruebas realizadas con cada una de estas herramientas:

- **Tone.js**: Esta librería es una de las de las más utilizadas a la hora de trabajar con la Web Audio API. Sin embargo, su funcionalidad principal es la de facilitar la transformación de del audio y las utilidades de síntesis, no ajustándose esto a los requisitos de nuestra

aplicación. Además, al ser una librería de JavaScript nativo, carece de las ventajas que ofrece TypeScript descritas anteriormente.

- **ng-web-apis**: Esta ha sido la única solución de calidad encontrada para trabajar con la Web Audio API en Angular. Es un proyecto que incluye soporte para distintas APIs en el entorno de Angular e incluye un subproyecto dedicado a la Web Audio API. Pero tras revisar la documentación y el código, pronto se llegó a la conclusión de que no aporta nada allá del soporte de tipado y el uso de los elementos de la Web Audio API a través de directivas de atributo, haciendo necesario tratar de forma manual tareas tales como la carga de una fuente de audio o la reproducción de sonidos.
- **HowlerJS**: Esta solución fue desarrollada para servir de motor de audio en el desarrollo de videojuegos en el entorno Web y fue la más prometedora de las analizadas. De hecho, llegó a probarse su integración en el proyecto, siendo su funcionamiento satisfactorio en las tareas más simples, pero careciendo de soporte para realizar tareas avanzadas, como pudieran ser la adición de efectos o la manipulación de las fuentes de audio.

Por otro lado, cabe mencionar que ninguno de los recursos analizados ofrece soporte directo para crear un renderizado las amplitudes de onda de los recursos de audio. Peso a que esto puede ser conseguido a través de otras herramientas, como la librería wavesufer.js, resultaría ideal utilizar una solución que incluyese dicha funcionalidad en el contexto de uso de la Web Audio API.

Figura 10. Ejemplo de renderizado de amplitud de onda
(fuente: https://wavesurfer-js.org)

Por todo ello, se considera necesario la creación de una librería propia. Esto permitirá satisfacer los requisitos de nuestra aplicación de una forma más precisa y realizar los ajustes oportunos ante la aparición de nuevos requisitos. En esta librería, además de facilitarse el trabajo con la Web Audio API, se dará soporte al renderizado de amplitudes de onda de los recursos de audio, lo cual se conseguirá a través del análisis de los datos de la fuente de audio y su posterior representación gráfica mediante el uso de la API Canvas de HTML.

5. DESARROLLO

En este apartado se trata de presentar una visión global de cómo se ha estructurado el desarrollo de la aplicación y de aportar información más detallada acerca de la implementación de ciertas funcionalidades cuya mención se considera relevante.

5.1. Casos de uso

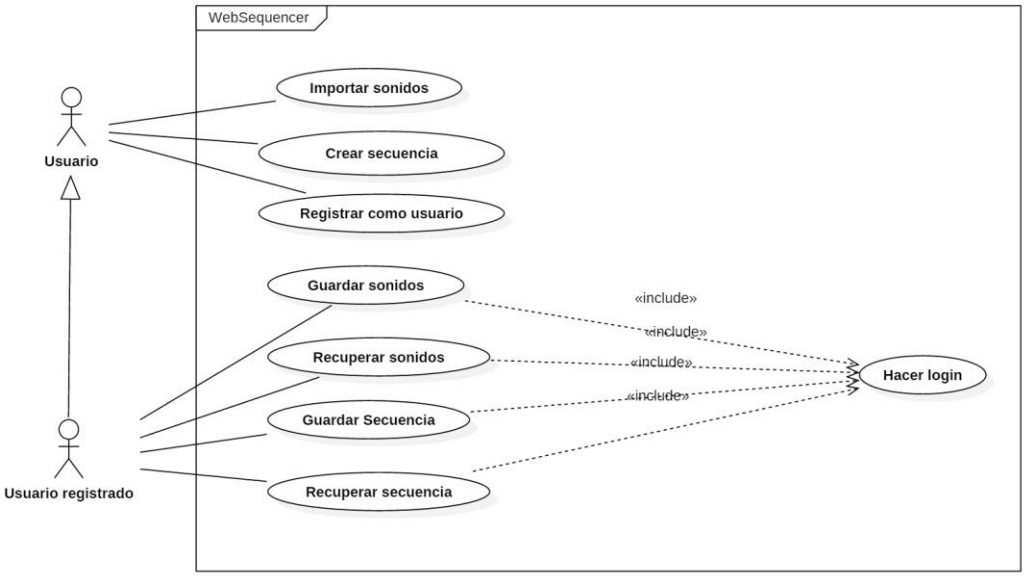

Figura 11. Diagrama de casos de uso

Como puede observarse en el diagrama de la figura 11, la aplicación presenta una estructura de casos de uso bastante sencilla. Un usuario no registrado podrá hacer uso de la aplicación, de modo que se le permite añadir sonidos al secuenciador y crear secuencias sin limitación alguna. Sin embargo, estos datos se perderán al cerrar el navegador o refrescar la pantalla. Para poder persistir los datos y recuperarlos posteriormente el usuario habrá de estar registrado y autenticado.

5.2. Modelo de datos

El modelo de datos está formado por tres tablas:

- **roles**: Esta tabla contiene el listado de roles disponibles, que en este caso son el de *administrador* y *usuario*. Aunque la aplicación aun no contempla acciones separadas para cada uno de los roles, el modelo de datos ha sido realizado teniendo en cuenta la escalabilidad de la aplicación (ver sección *Conclusiones y Trabajos Futuros*).
- **sequences**: Contiene la información de una secuencia, como son el usuario propietario, el nombre de la secuencia y el objeto JSON con los datos en bruto a interpretar por el cliente.

- **users**: Esta tabla contiene los datos de los usuarios registrados en la aplicación, entre ellos se encuentran el nombre, contraseña y rol del usuario. Esta tabla tiene una relación n:1 con la tabla *roles* y una relación 1:n con la tabla *sequences*.

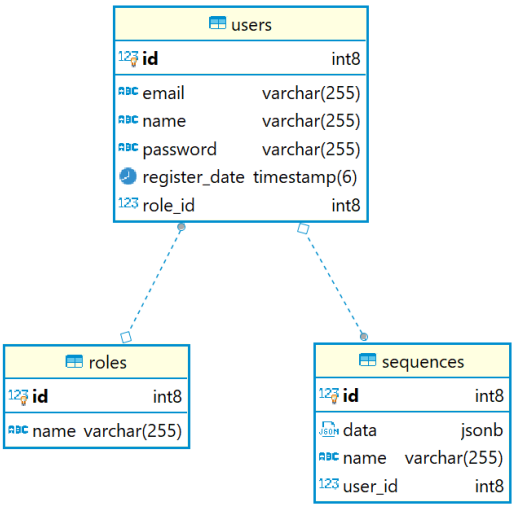

Figura 12. Modelo de datos

5.3. Librería para la Web Audio API

Se analizan en esta sección los aspectos más relevantes sobre el desarrollo de la librería de Angular creada sobre la Web Audio API, a la cual se ha dado el nombre de *ngx-audio-mngr*.

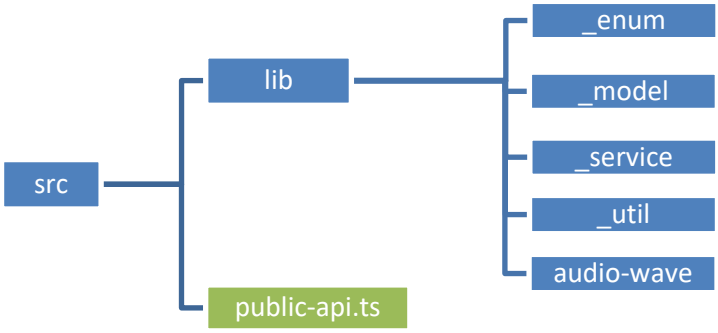

Figura 13. Estructura de la librería ngx-audio-mngr

La estructura del proyecto no dista demasiado de la de cualquier proyecto Angular, aunque tiene algunas particularidades propias de un proyecto de librería. Una de ellas es la presencia del fichero

public-api.ts en el directorio *src*. En este fichero se declaran aquellos recursos que han de ser exportados para poder ser consumidos desde el proyecto que haga uso de la librería.

La lógica del proyecto se encuentra en el directorio *lib*. Pasamos a continuación a describir los elementos que nos encontramos en este directorio:

- **_enum**: Contiene todos los enumerados utilizados en la aplicación.
- **_model**: Contiene el modelo creado para representar una fuente de audio con sus respectivos parámetros.
- **_service**: Contiene el servicio que centraliza toda la gestión de la librería.
- **_util**: Directorio de utilidades.
- **audio-wave:** Contiene el único componente del proyecto, encargado de crear la visualización de un fichero de audio.

La librería ha sido diseñada de tal manera que el servicio **AudioMngrService** centraliza las acciones principales. Este servicio representa el contexto en el que viven los distintos objetos de audio. Es el encargado de crear y eliminar los objetos AudioMngrSound y controla los valores globales comunes a todos los sonidos, como puede ser el volumen maestro.

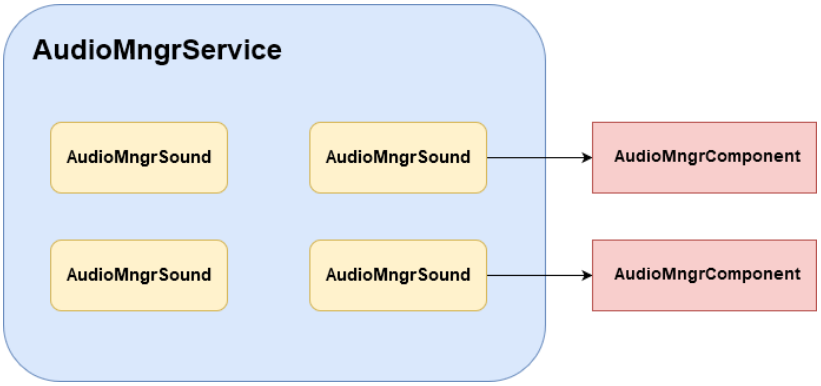

Figura 14. Diagrama librería ngx-audio-mngr

Cada una de las instancias de la clase **AudioMngrSound** representa un objeto sonido, esto es, la fuente de audio y sus parámetros, tales como velocidad de reproducción, volumen, etc. De este modo se cubre una de las principales limitaciones de la Web Audio API, que es la volatilidad de los recursos de audio. Esto es así porque la Web Audio API obliga a crear un nuevo nodo de audio cada vez que se desea reproducir un sonido, lo que conlleva asignar de nuevo la fuente de audio y cada uno de los parámetros individuales de la reproducción. Creando un objeto que encapsule todos estos datos y utilice los mismos de forma automática a la hora de reproducir un sonido facilita de forma considerable la interacción con la Web Audio API.

En caso de querer visualizar la amplitud de onda de los sonidos que están siendo reproducidos y su progreso en tiempo real, se puede utilizar un componente **AudioMngrComponent**, que recibirá, además de ciertos parámetros de configuración, un objeto AudioMngrSound, el cuál será la fuente de

renderizado. Además, el componente permite cambiar la posición actual del audio haciendo clic sobre la posición deseada en la pantalla.

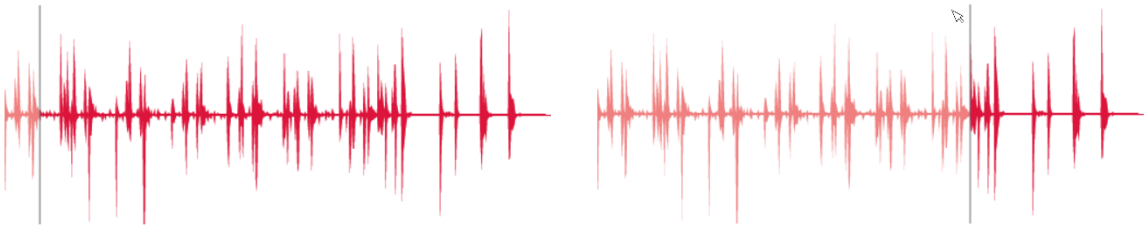

Figura 15. Avance de audio con clic en AudioMngrComponent

En el anexo final se incluye la documentación de la librería, en la que se detalla el uso de cada una de las partes de esta.

5.4. Front-end

Se expone a continuación cómo se ha estructurado el proyecto Angular, que encapsula el frontal de la aplicación.

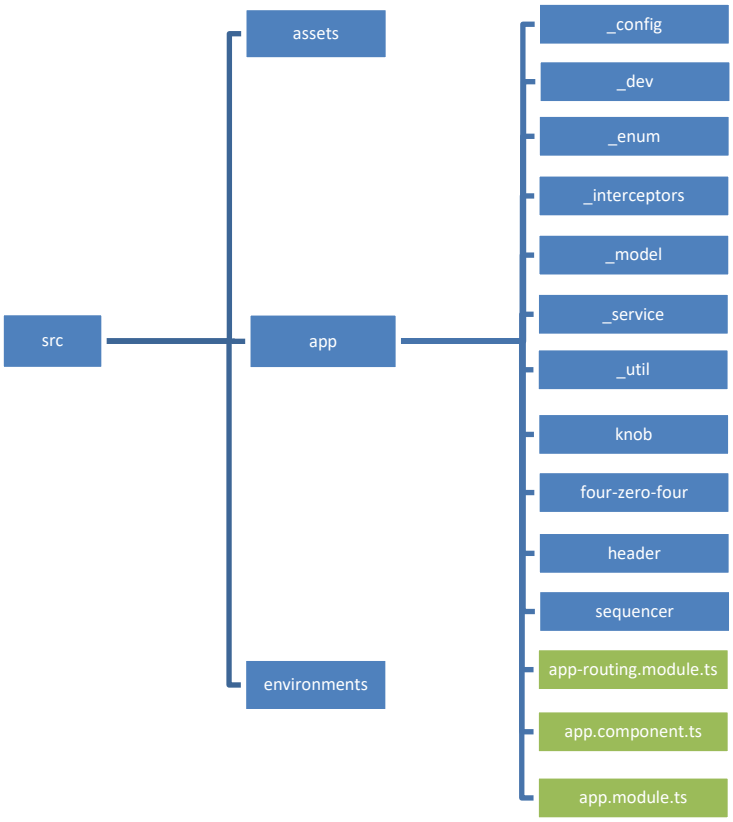

Figura 16. Estructura del front-end

El directorio raíz *src* contiene tres directorios principales:

- **assets**: En él se incluyen recursos tales como hojas de estilo, ficheros de internacionalización, fuentes de texto, imágenes, etc.
- **app**: En este directorio se encuentra el grueso del proyecto Angular. Incluye, entre otras cosas, los componentes. Por ello, más adelante, desglosaremos su contenido.
- **environments**: Este directorio contiene los ficheros de configuración para distintos entornos. En este caso se cuenta con una configuración para el entorno de desarrollo y otra para el entorno de producción.

Todas las cadenas de texto de la aplicación se encuentran en el directorio *assets/i18n*, de forma que existe un fichero JSON por cada idioma soportado, lo cual facilita las labores de internacionalización. De esta manera, al inicializarse la aplicación se consulta el idioma por defecto configurado en el navegador y, en caso de contar con soporte para dicho idioma, éste será el que use la aplicación para mostrar los textos. Si el idioma del navegador no está en la lista de idiomas soportados por la aplicación, los textos se mostrarán en el idioma por defecto, en este caso, el inglés. Actualmente la aplicación cuenta con soporte para inglés y castellano.

Tal y como se ha comentado, el directorio *app* alberga la parte más importante del proyecto Angular, por lo que se describen a continuación sus contenidos:

- **Componentes**: En este caso, contamos con un directorio para cada uno de los cuatro componentes: *header*, que incluye la cabecera de la aplicación y las ventanas modales de registro y login; *sequencer*, que contiene la vista y la lógica del secuenciador, parte principal de la aplicación; *knob*, para los controles rotatorios de parámetros; y *four-zero-four*, que incluye la página a la que se redirige al usuario en caso de no encontrarse el recurso solicitado.
- **_config**: Contiene clases y constantes de configuración que son utilizadas en distintas partes de la aplicación.
- **_dev**: En este directorio se encuentran recursos que resultan de utilidad para realizar pruebas en un entorno de desarrollo.
- **_enum**: Contiene los enumerados utilizados en la aplicación.
- **_interceptors**: Aquí están contenidos los interceptores, que son un tipo de servicio utilizado para interceptar peticiones y respuestas HTTP, de modo que puedan ser modificadas. En este caso se utiliza un interceptor para incluir en la cabecera de las peticiones el token de autenticación, en caso de que se encuentre guardado en el *SessionStorage*.
- **_model**: Alberga los modelos de los distintos objetos utilizados en el cliente.
- **_service**: Contiene los servicios de la aplicación, encargados de comunicarse con la API REST, mantener variables globales y controlar el grueso de la lógica del secuenciador.
- **_util**: Incluye utilidades para el uso en los componentes y servicios de la aplicación. Aquí se incluyen los ficheros de mapeo de distintos controladores MIDI disponibles en el mercado.

- **app-routing.module.ts:** Es el fichero utilizado para definir el enrutamiento interno de la aplicación.
- **app.component.ts:** Se trata del componente raíz de la aplicación, del cual parten todos los demás componentes.
- **app.module.ts:** En este fichero se definen los distintos módulos, componentes y servicios utilizados en la aplicación. En este caso la aplicación cuenta con un solo módulo.

El desarrollo de la aplicación para el front-end se ha simplificado de gran manera gracias al uso de la librería *ngx-audio-mngr* previamente creada. Esto ha permitido desacoplar la lógica propia de la aplicación, de las tareas de más bajo nivel relacionadas con el tratamiento y procesamiento de audio.

Uno de los mayores retos en el desarrollo del secuenciador web ha sido el conseguir un timing lo suficientemente preciso.

El propósito principal de un secuenciador musical es el poder programar la reproducción de sonidos dentro de un patrón determinado, que sean ejecutados en tiempo real y respondan de forma rápida a las acciones ejecutadas por el usuario, como pueden ser modificar el tempo (velocidad de reproducción del patrón), introducir nuevos sonidos, editar parámetros de sonido, etc.

Sin embargo, una de las limitaciones encontradas en el funcionamiento de la Web Audio API es la rigidez en la programación de eventos futuros. Esto se debe a que la Web Audio API no hace uso del reloj JavaScript, sino que accede al reloj del hardware, el cual indica el número de segundos transcurridos desde que el contexto de audio fue creado, con una precisión de hasta 15 dígitos decimales, lo cual garantiza un timing casi perfecto. El problema surge cuando se necesita actuar de forma rápida a cambios en la programación de eventos, como ocurre en el caso de un secuenciador. El reloj utilizado por la Web Audio API no permite la creación de *callbacks*, como sucede en el uso de la función *setTimeout()* de JavaScript.

Podría parecer que el problema planteado puede resolverse fácilmente con la utilización de la función de JavaScript *setTimeout()* para programar eventos futuros cada vez que el usuario realice una nueva acción. Sin embargo, la razón fundamental por la que la Web Audio API utiliza el reloj interno de hardware es la escasa estabilidad del reloj de JavaScript. Esto se debe a que el reloj JavaScript corre en el hilo principal de ejecución, lo que puede provocar de forma habitual retrasos mayores a los 50 milisegundos, además de ser interrumpido cuando la ventana en el navegador pasa a segundo plano.

La solución al problema planteado se encontró partiendo de la estrategia propuesta en el artículo "*A Tale of Two Clocks*" [10], referenciado de forma consistente por desarrolladores que utilizan la Web Audio API en sus proyectos. La estrategia propuesta en este artículo consiste en combinar la flexibilidad que aporta el reloj de JavaScript mediante la utilización de *callbacks*, con la precisión del reloj utilizado por la Web Audio API.

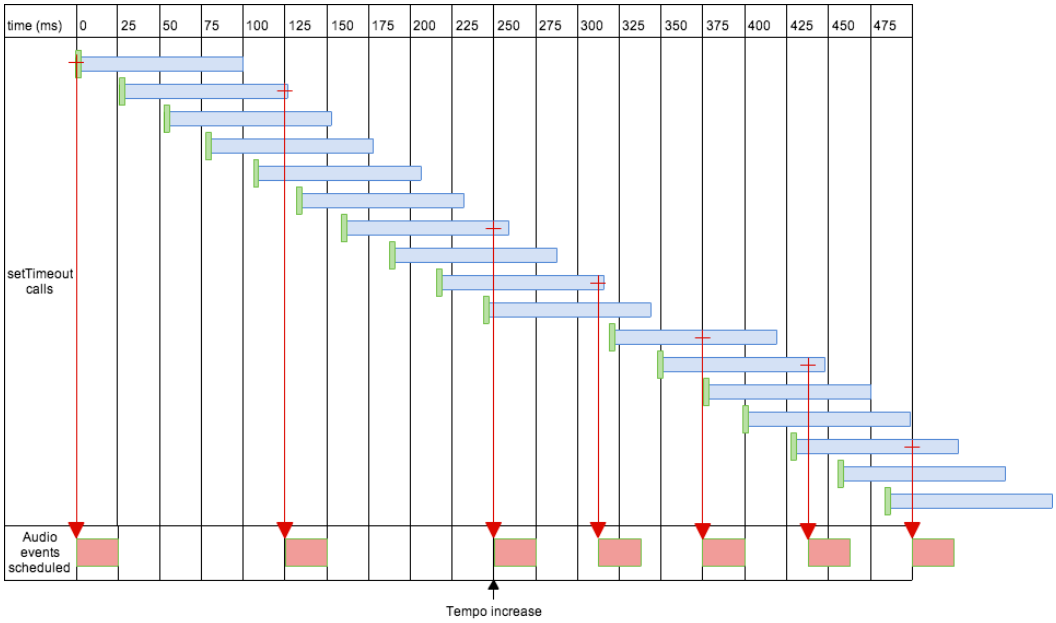

Figura 17. Combinación de setTimeout con reloj de audio
(fuente: https://www.html5rocks.com/en/tutorials/audio/scheduling)

El ejemplo representado en la figura 17 utiliza un *setTimeout* con un intervalo de 25 milisegundos con un periodo de programación en el futuro de 100 milisegundos. Lo que permite esta estrategia es comprobar cada 25 milisegundos (aproximadamente) si se ha de producir algún cambio en la programación de eventos (cambio del tempo, reproducir nuevo sonido, etc.) desde el momento de ejecución de la función hasta 100 milisegundos en el futuro. En caso positivo, un nuevo evento será programado con la precisión del reloj de la Web Audio API.

El utilizar un periodo de tiempo de programación en el futuro (100ms) superior al intervalo de cada *setTiemout* (25ms) previene las desviaciones que pueda presentar el reloj JavaScript. En este caso, el sistema estaría cubriendo la posibilidad de un retraso de hasta 75ms. Por ejemplo, en la figura se observa que en la posición 270ms se debería de haber producido una nueva llamada por el *setTimeout,* pero esta se produjo en la posición 320ms, lo que supone un retraso de 50ms.

Estos valores propuestos han de ser ajustados según los requerimientos de la aplicación, de forma que se encuentre un equilibrio entre la estabilidad del sistema y el retraso en la respuesta a nuevos cambios introducidos por el usuario.

5.5. Back-end

El back-end de la aplicación ha sido desarrollado con Spring Boot. Se trata de un servicio RESTful con el que se comunicará la aplicación cliente de Angular para realizar las tareas de registro, autenticación, y guardado y recuperación de secuencias y sonidos.

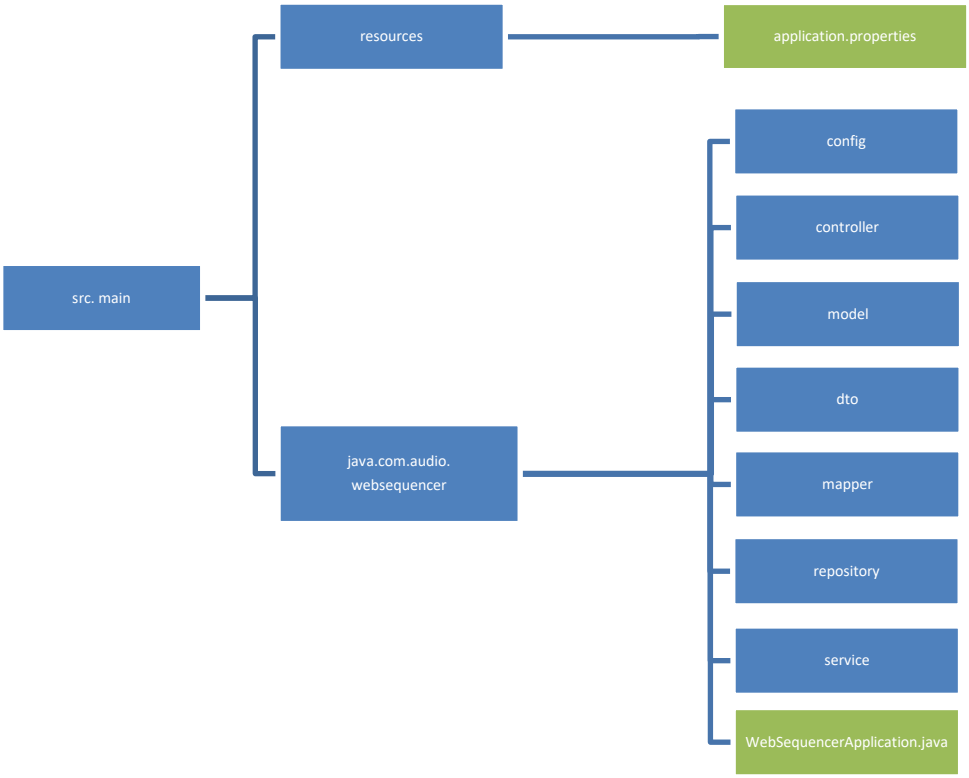

Figura 18. Estructura del back-end

El paquete raíz contiene dos paquetes principales, *resources* y *java.com.audio.websequencer*. Este último contiene el código Java de la aplicación y a su vez está estructurado en distintos paquetes según funcionalidad. Pasamos por tanto a describir el contenido de los principales paquetes y ficheros de la aplicación:

- **resources**: El contenido más significativo de este paquete es el fichero *application.properties*, que contiene parámetros sobre la configuración de Hibernate, la conexión con la base de datos y el guardado de ficheros en el servidor.
- **config**: Contiene configuraciones sobre la autenticación a través de JWT y los permisos necesarios para acceder a cada uno de los recursos de la API.
- **controller**: Contiene las clases controladoras que atienden las peticiones y respuestas del servicio web, así como las redirecciones internas de la aplicación.
- **model**: En este paquete se encuentran las entidades correspondientes a las distintas tablas de la base de datos.
- **dto**: Contiene los respectivos DTOs de cada entidad, utilizados en los traspasos de información desde y hacia el cliente web.
- **mapper**: En él se incluyen las interfaces y clases utilizadas para la transformación de entidades en DTO y viceversa.
- **repository**: Contiene interfaces que heredan de las clases de persistencia de JPA, las cuáles son utilizadas por los servicios para realizar consultas a la base de datos.
- **service**: Contiene clases e interfaces que ejecutan la lógica de negocio.

- **WebSequencerApplication.java**: Es la clase principal del proyecto, en la cual se declaran la autoconfiguración y el escaneo de componentes, y que referencia la configuración del fichero *applicaion.properties*, todo ello por medio de la anotación *@SpringBootApplication*.

La autenticación de usuarios se realiza mediante el uso de JWT (JSON Web Tokens), estándar utilizado para transmitir información de forma segura en formato JSON. De este modo, el usuario envía su nombre y contraseña al servidor, donde se genera un token que contiene los datos de acceso, y se firma mediante un algoritmo con una clave secreta que reside en el servidor. El servidor devuelve el token generado al cliente web, el cuál guardará ese token para incluirlo en la cabecera de futuras peticiones.

Los ficheros de audio subidos por los usuarios son almacenados en el directorio indicado en el fichero *application.properties*. Cuando un usuario suba ficheros de audio, se creará, en caso de no existir, un subdirectorio con el *id* del usuario, y dentro un directorio por cada programa, o grupo de sonidos.

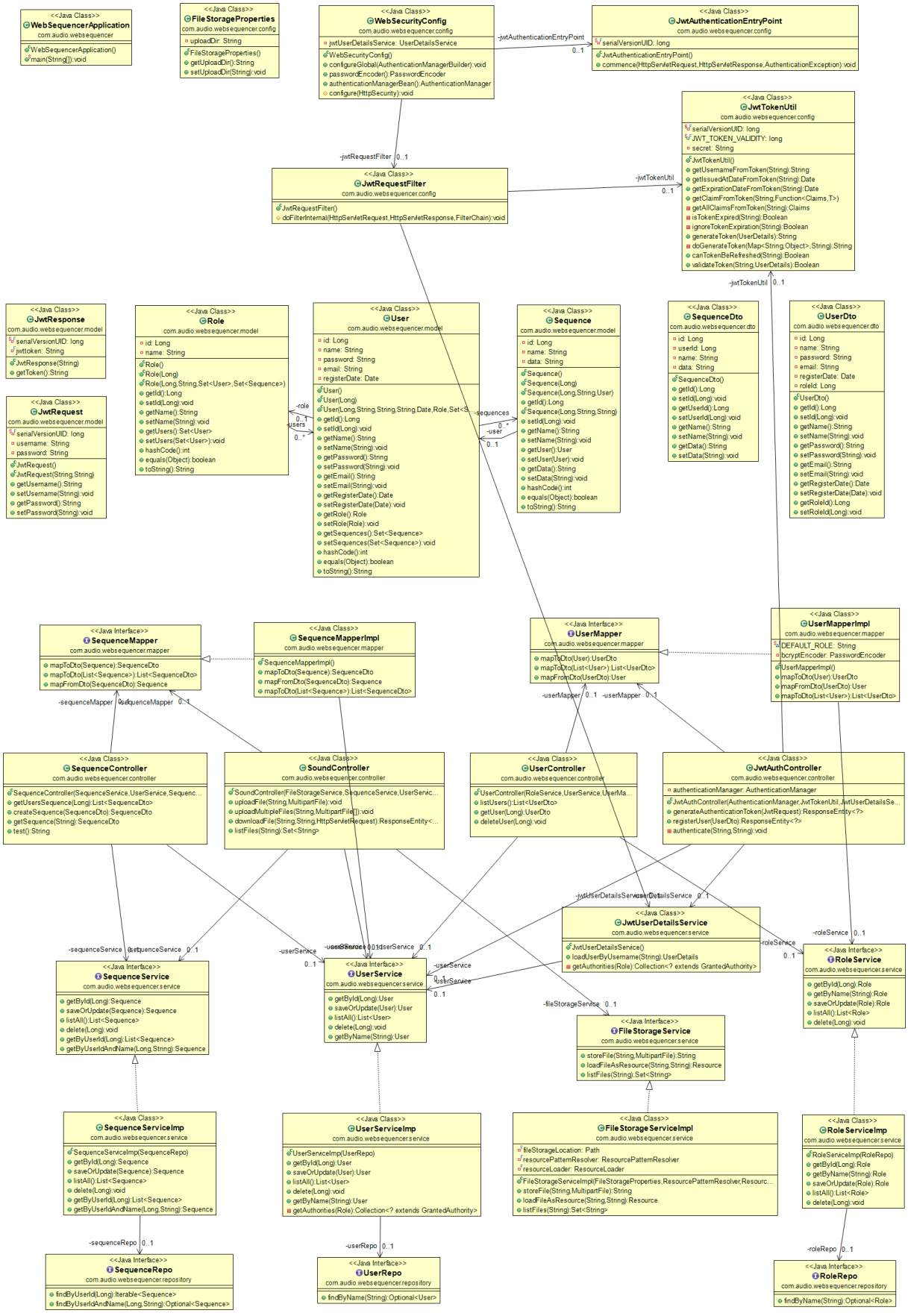

Figura 19. Diagrama de clases

6. RESULTADOS

Esta sección ofrece una visión sobre los resultados obtenidos a través de un recorrido sobre la interfaz con la que el usuario final interactuará y de las distintas funcionalidades de la aplicación.

Figura 20. Logotipo de Sequentia

La aplicación se llama *Sequentia*, que significa "secuencia" en latín, y cuenta con una interfaz formada por una única pantalla, en la cual se encuentra un encabezado, con las opciones de registro, login y logout; y el secuenciador, que ocupa la mayor parte de la pantalla.

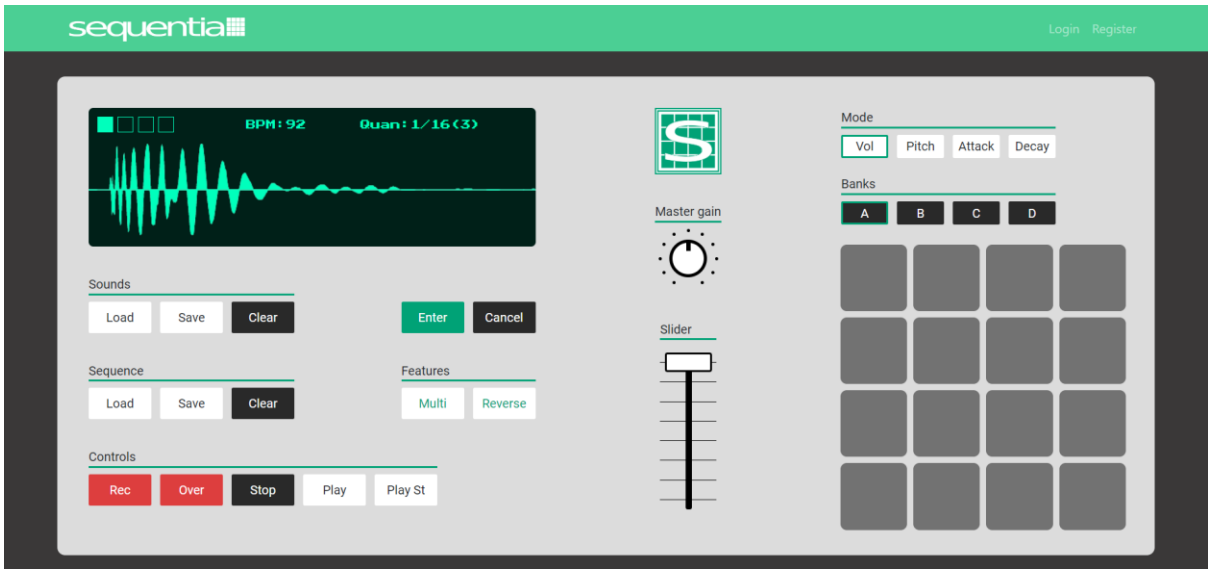

Figura 21. Vista en dispositivo de escritorio

El secuenciador creado trata de replicar el diseño y funcionamiento típicos de un secuenciador de hardware. Así, está formado por un display informativo, varios botones, un control deslizante y un control rotatorio.

La aplicación está pensada para poder ser utilizada en distintos tipos de dispositivo, permitiendo diferentes modos de interactuación con los elementos del secuenciador según el dispositivo en se ejecute, y adaptándose a distintos tamaños y formatos de pantalla.

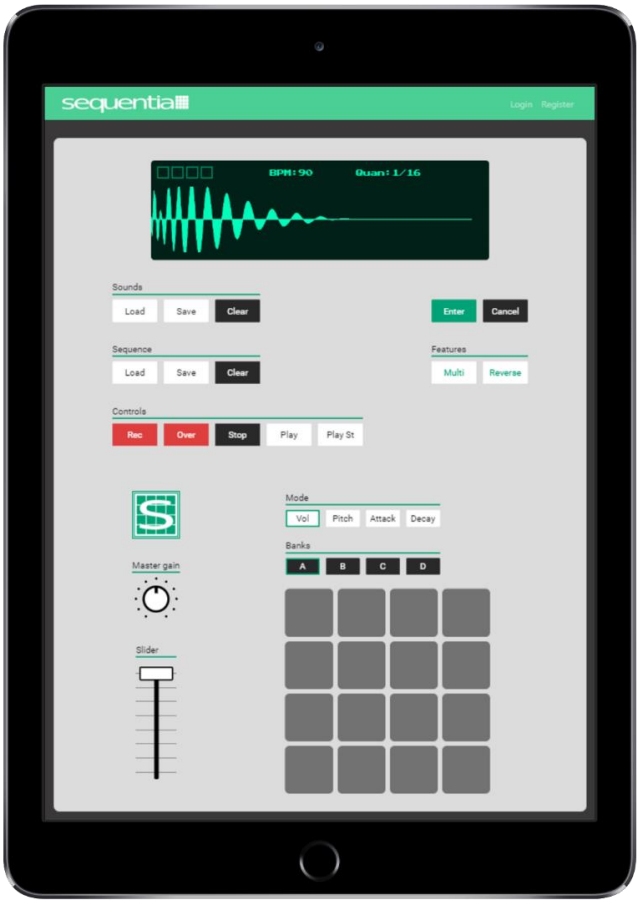

Figura 22. Tableta en modo retrato

Figura 23. Móvil en modo retrato

6.1. Menú superior

En el menú del encabezado se encuentran las opciones de registro e inicio de sesión.

A través del formulario de **registro** se puede crear una nueva cuenta en la aplicación, lo que permitirá al usuario guardar sus secuencias y sonidos.

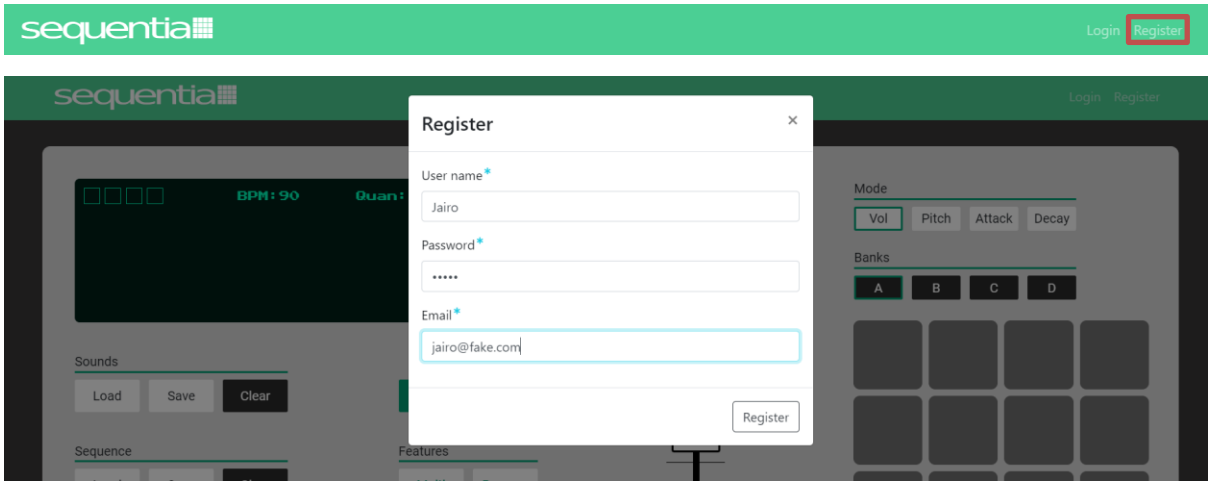

Figura 24. Formulario de registro

Si el usuario ya ha creado una cuenta, puede **iniciar sesión** a través del formulario de login.

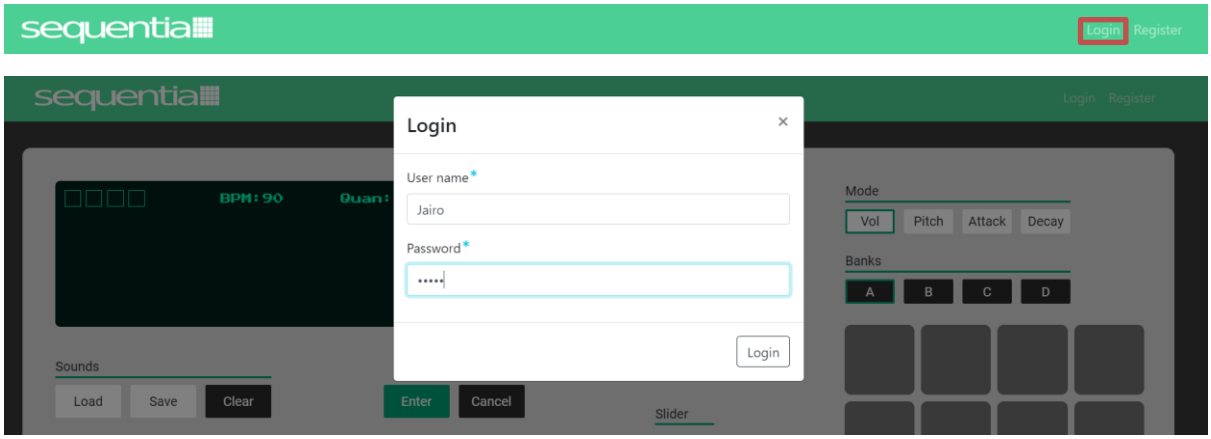

Figura 25. Formulario de inicio de sesión

Una vez el usuario tiene la sesión iniciada, aparecerá en el menú una nueva opción para **cerrar la sesión**.

Figura 26. Opción de menú para cierre de sesión

6.2. Secuenciador

6.2.1. Cargar y reproducir sonidos

Los sonidos son reproducidos a través de los pads. El secuenciador cuenta con cuatro bancos de 16 pads, pudiendo por tanto usarse en un mismo proyecto hasta 64 sonidos.

El display mostrará la onda del sonido que se esté reproduciendo, así como su progreso en tiempo real.

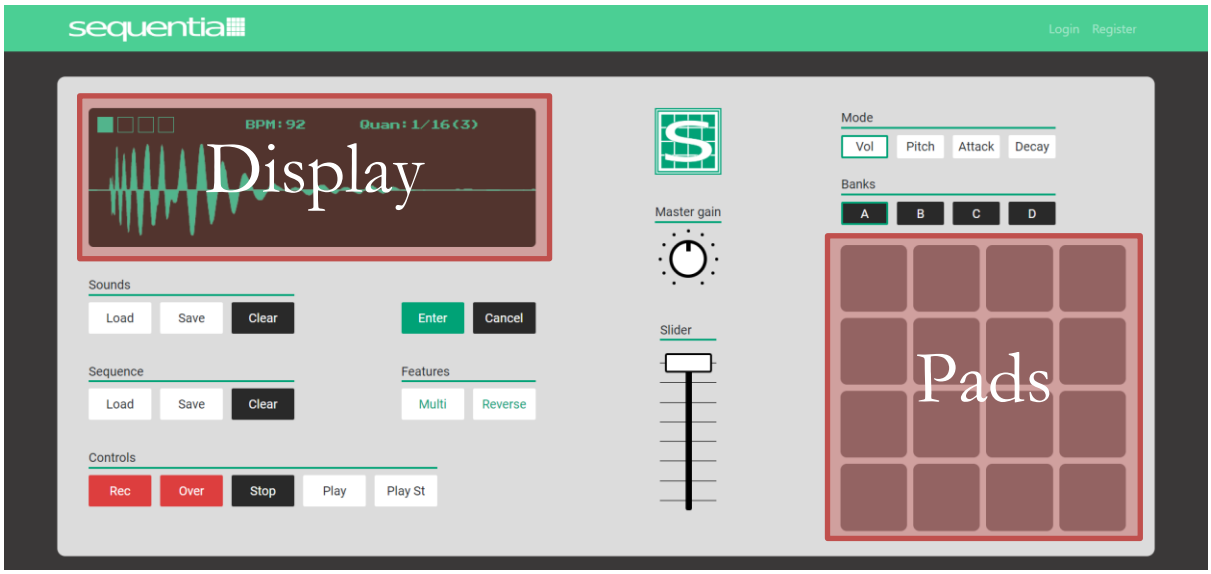

Figura 27. Área de display y pads

Si acabamos de iniciar un proyecto, no tendremos sonidos cargados. Para **cargar un nuevo sonido** de forma local basta con arrastrar un nuevo fichero de audio y soltarlo en el pad al que lo queramos asignar.

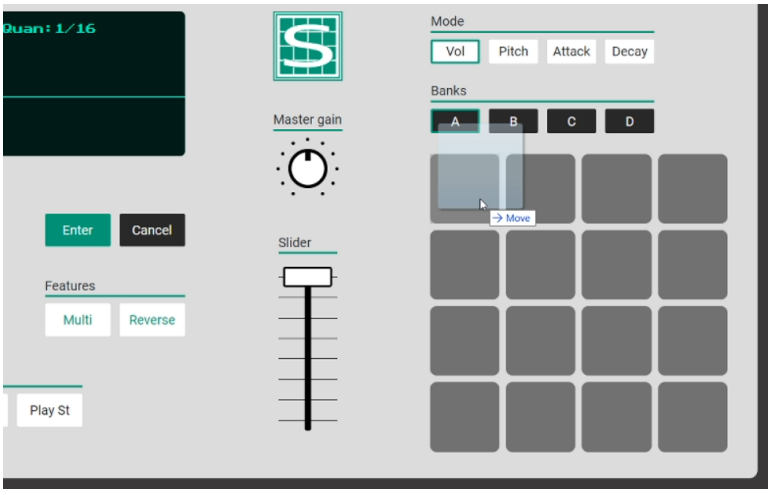

Figura 28. Carga de sonido

Para sustituir el sonido por uno nuevo bastará con arrastrar un nuevo fichero al pad que queramos. Si en algún momento queremos eliminar todos los sonidos, esto se puede realizar a través del botón *Clear* de la sección *Sounds*.

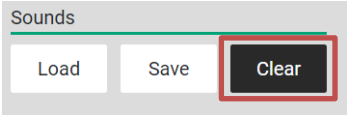

Figura 29. Eliminación de sonidos

Una vez tenemos sonidos cargados, podemos **reproducirlos** de distintas maneras:

- La primera de ellas es haciendo **click sobre el pad** o tocándolo si estamos trabajando con una pantalla táctil.
- Si se está trabajando con un **teclado**, se pueden utilizar las teclas configuradas por defecto para controlar cada uno de los pads.

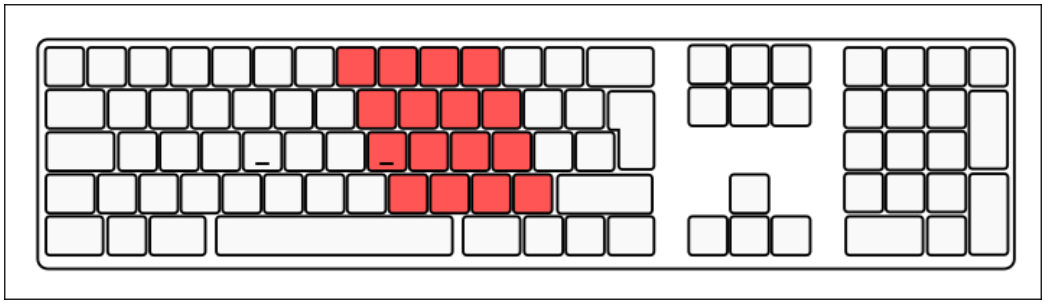

Figura 30. Distribución de teclas para control de pads

- Por último, se puede utilizar un **controlador MIDI** para controlar los pads, siempre que este sea reconocido por el navegador que se está utilizando. Para ello debemos de usar un navegador compatible con Web MIDI. Actualmente todos los navegadores basados en *Chromium* son compatibles en sus últimas versiones.

Además, el display permite cambiar la posición del sonido, es decir, avanzar en el tiempo o volver hacia a atrás en la reproducción del sonido. Para ello solo es necesario hacer clic o tocar el punto al que queremos desplazar el avance del sonido.

Figura 31. Avance en sonido con clic en el display

Para **cambiar el banco** que queremos usar, utilizaremos el selector de bancos, que se encuentra encima de los pads. En todo momento el banco en uso aparecerá resaltado por un borde de color verde.

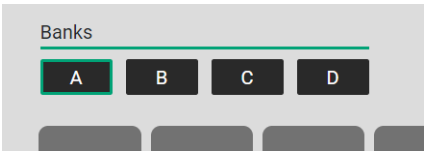

Figura 32. Selector de bancos

6.2.2. Ajustar el volumen global

El volumen del máster se ajusta a través del control rotatorio *Master gain*. Si se gira en el sentido de las agujas del reloj, el volumen del sonido global aumentará y si se gira en el sentido contrario disminuirá.

Para manipular este control por medio del ratón, tenemos que colocar el puntero sobre el control y utilizar una de estas dos opciones:

- Mover la rueda del ratón
- Hacer clic y desplazar el ratón hacia arriba y hacia abajo

En dispositivos de pantalla táctil se manipulará el control tocando sobre la superficie de este y arrastrando hacia arriba o abajo.

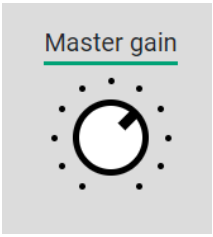

Figura 33. Control rotatorio de ganancia

6.2.3. Manipular sonidos

Para manipular sonidos de forma individual usaremos el control deslizante y el selector de modo.

El control deslizante modifica el parámetro que se encuentre seleccionado en el selector *Mode* sobre el sonido que esté activo, esto es, el correspondiente al último pad tocado, cuya onda se podrá visualizar en el display.

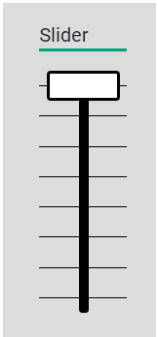

Figura 34. Control deslizante

Figura 35. Selector de modos

Veamos por medio de un ejemplo los pasos a seguir para modificar un sonido:

1) El usuario pulsa el primer pad del banco A, con lo que este queda seleccionado.
2) El display muestra la onda del sonido seleccionado, en este caso el primer sonido del banco A.

3) El usuario selecciona el parámetro que quiere manipular en el sonido seleccionado, en este caso, el pitch.
4) El usuario modifica el pitch desplazando el control deslizante y el sonido se altera en tiempo real.

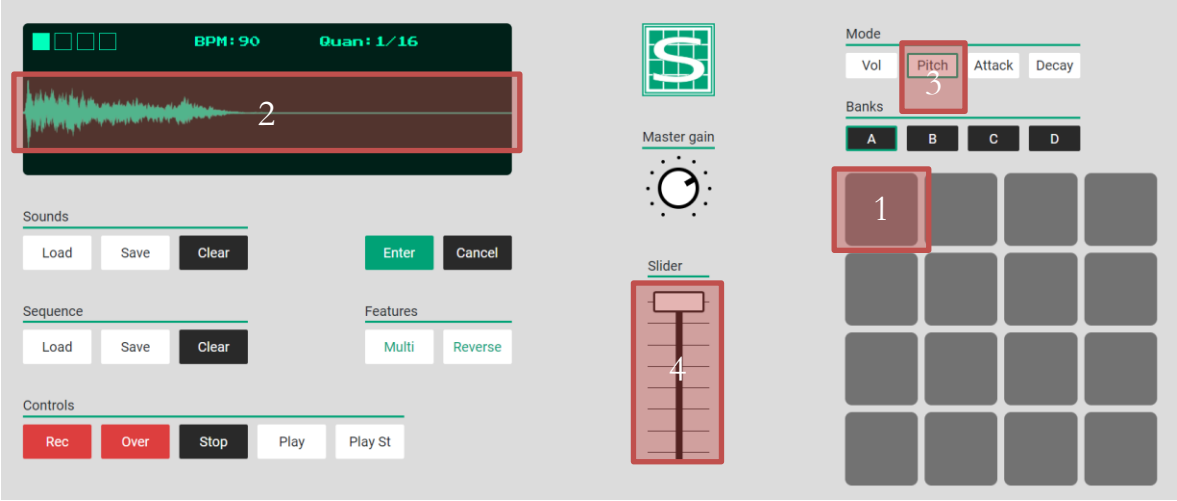

Figura 36. Pasos para modificar un sonido

En equipos de escritorio el control deslizante puede ser manipulado de varias maneras:

- Posicionando el puntero del ratón sobre el control y moviendo la rueda del ratón.
- Haciendo clic en la posición en la que queremos colocar el control.
- Utilizando las teclas de desplazamiento vertical del teclado.

En equipos de pantalla táctil podremos tocar la posición en la que queremos colocar el control.

Estos son los cuatro parámetros que podemos manipular para cada sonido:

- **Vol**: Modifica el volumen del sonido. A diferencia del control *Master gain*, que actúa sobre todo el proyecto, este parámetro modificará el volumen de un sonido de forma individual.
- **Pitch**: Modifica la velocidad de reproducción del sonido, afectando por tanto al tono de este. Cuanto más bajo sea el pitch, más grave será el tono del sonido.
- **Attack**: Modifica el tiempo trascurrido desde que la amplitud del sonido pasa de 0 (silencio) al pico máximo (volumen máximo) al empezar a sonar el sonido.
- **Decay**: Modifica el tiempo trascurrido desde que la amplitud del sonido pasa del pico máximo (volumen máximo) a 0 (silencio) al final del sonido.

Para cambiar el parámetro sobre el que queremos actuar, podemos hacer clic sobre los botones de selección (*Mode*) o utilizar las teclas de desplazamiento horizontal del teclado.

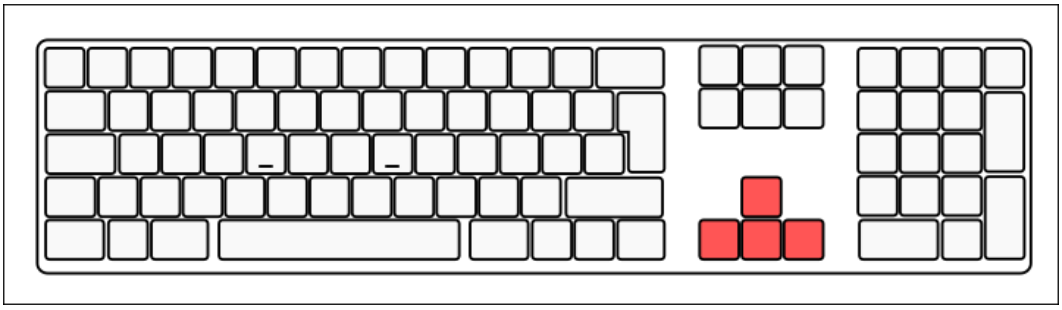

Figura 37. Teclas para control de deslizante y cambio de modo

Controles avanzados

Existen dos controles adicionales, agrupados en *Features*, que se pueden utilizar para manipular sonidos. Estos controles funcionan de forma independiente al control deslizante, es decir, al pulsarlos son aplicados directamente sobre el sonido activo.

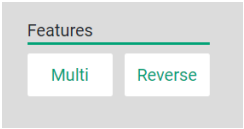

Figura 38. Controles avanzados

El control ***Multi*** distribuye el sonido activo entre los 16 pads, asignando un valor de pitch distinto para cada pad. De esta manera, sitúa el sonido con su pitch original en el pad número siete, y por cada pad por arriba o por debajo modifica el pitch en un semitono. Esto permite al usuario utilizar una única fuente de sonido para crear melodías, como si estuviese utilizando un teclado.

La siguiente figura muestra el número de semitonos en que el sonido es alterado para cada pad. También se puede observar cómo mientras la función *Multi* está habilitada, el botón aparecerá con un borde de color verde.

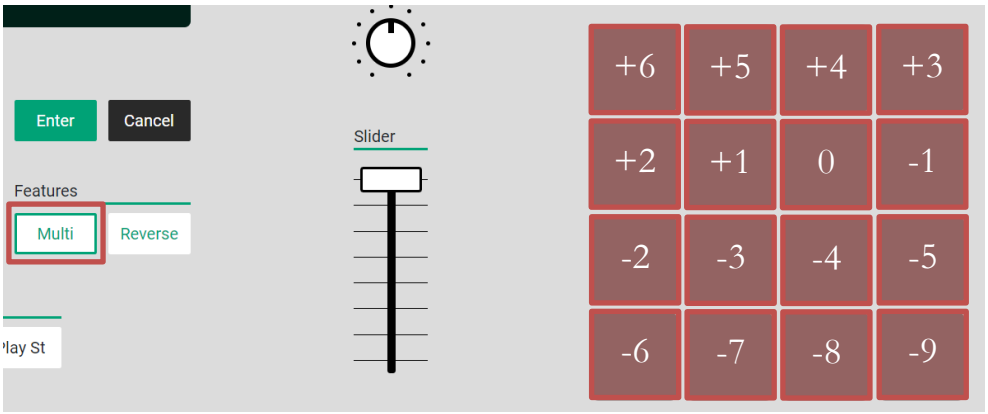

Figura 39. Distribución de sonido con la función Multi

Para deshabilitar la opción basta con volver a pulsar el botón *Multi* y cada pad volverá a estar ocupado por el sonido que tenía asignado.

Por su parte, el control ***Reverse*** invierte la onda de audio, de forma que el sonido pasará a reproducirse de forma inversa a como lo hacía originalmente.

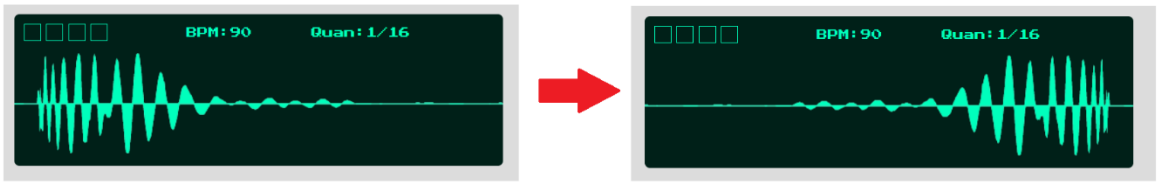

Figura 40. Representación de un sonido antes y después de aplicar el control Reverse

6.2.4. Crear secuencias

En la parte superior del display encontramos tres apartados relacionados con la funcionalidad de secuenciación.

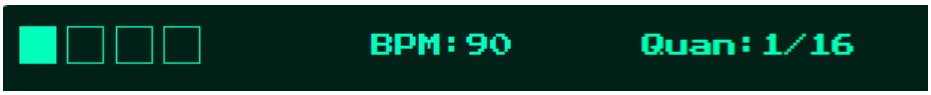

Figura 41. Parte superior del display

A la izquierda hay un bloque con cuatro **cuadrados**. Cada uno de estos cuadrados representa un beat (golpe o nota) dentro de cada compás. Cuando la secuencia esté en modo reproducción o en modo grabación, se mostrará en tiempo real el avance de cada nuevo beat rellenando el contenido del cuadrado correspondiente a la posición del beat actual.

BPM representa los beats por minuto, es decir, el tempo de la secuencia. Este parámetro puede ser modificado en cualquier momento posicionando el cursor del ratón sobre el mismo y moviendo la rueda del ratón, o con el gesto de arrastre en dispositivos táctiles.

Quan representa la notación a la que las notas serán cuantizadas. Por ejemplo, un valor de 1/8 indica que dentro de un beat habrá ocho posiciones disponibles en los que el sonido puede ser colocado. Si un sonido es tocado fuera de una de esas ocho posiciones, se le asignará la posición válida más cercana. Los valores disponibles son OFF, 1/8, 1/8(3), 1/16, 1/16(3), 1/32, 1/32(3). *OFF* significa que los sonidos no se cuantizarán y la notación *(3)* indica que se trata de *triplets*. Del mismo modo que ocurre con *BPM*, este parámetro puede ser modificado posicionando el cursor del ratón sobre el mismo y moviendo la rueda del ratón, o con el gesto de arrastre en dispositivos táctiles.

Configurados los parámetros de la secuencia, podemos grabar, parar y reproducir ésta a través de los botones agrupados en *Controls*. Veamos para qué sirve cada uno de estos controles:

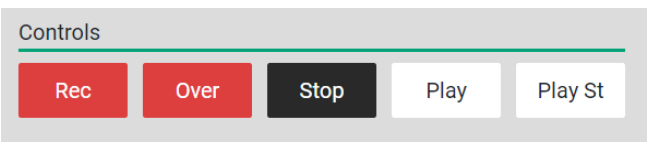

Figura 42. Controles de secuencia

- **Rec**: Inicia la grabación de la secuencia, sobrescribiendo todo lo que hubiese grabado en la secuencia actual. La secuencia será repetida de forma continua hasta que sea parada. Después de la primera reproducción, se dejará de sobrescribir lo que haya grabado y se añadirán los sonidos tocados sobre los ya existentes. Cuando se esté en modo grabación, además de los sonidos de la secuencia, se escuchará un metrónomo indicando el tempo de la secuencia.
- **Over**: Funciona igual que *Rec*, con la diferencia de que con esta opción en ningún momento se eliminarán los sonidos ya existentes en la secuencia.
- **Stop**: Detiene la grabación o la reproducción en curso.
- **Play**: Reproduce la secuencia actual desde la posición en que haya sido parada.
- **Play St**: Reproduce la secuencia actual desde el principio.

Para eliminar todos los sonidos de la secuencia actual utilizaresmos el botón *Clear* de la sección *Sequence*.

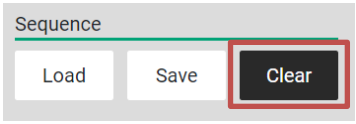

Figura 43. Eliminación de secuencia

6.2.5. Guardar y cargar programas de sonidos

Estas opciones solo están disponibles para usuarios registrados.

Para **guardar los sonidos** actuales del proyecto pulsaremos el botón *Save* bajo el grupo *Sounds*, indicaremos un nombre con el teclado y pulsaremos el botón *Enter*. Si se quiere abortar la operación, esto se realizar a través del botón *Cancel*.

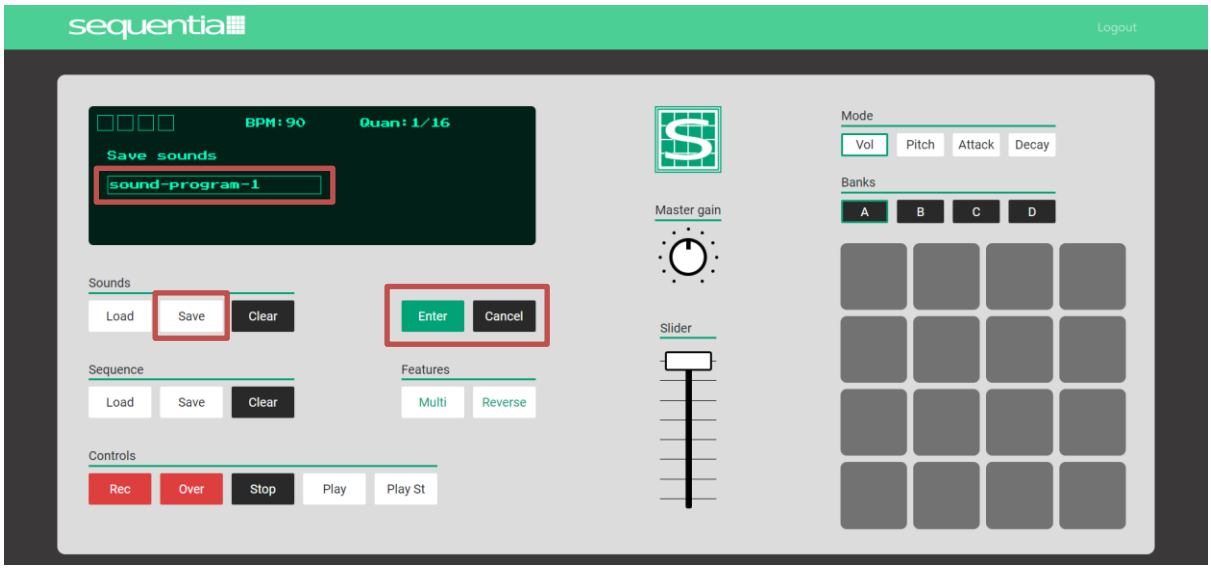

Figura 44. Guardado programa de sonidos

Para **recuperar sonidos** guardados previamente pulsaremos el botón *Load* bajo el grupo *Sounds*, escogeremos el nombre del programa de sonidos que queremos recuperar y pulsaremos el botón *Enter*. Los sonidos serán cargados y se eliminarán los que hubiese en el proyecto actual. Si se quiere abortar la operación, esto se realizar a través del botón *Cancel*.

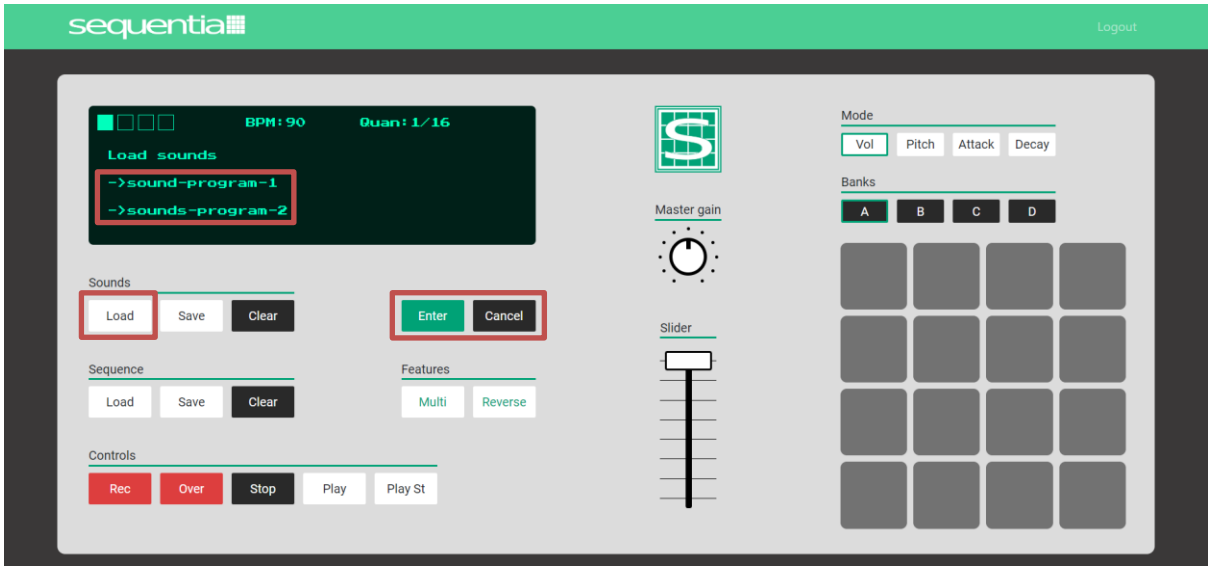

Figura 45. Carga de programa de sonidos

6.2.6. Guardar y cargar secuencias

Estas opciones solo están disponibles para usuarios registrados.

Para **guardar la secuencia actual** del proyecto pulsaremos el botón *Save* bajo el grupo *Sequence*, indicaremos un nombre con el teclado y pulsaremos el botón *Enter*. Si se quiere abortar la operación, esto se realizar a través del botón *Cancel*.

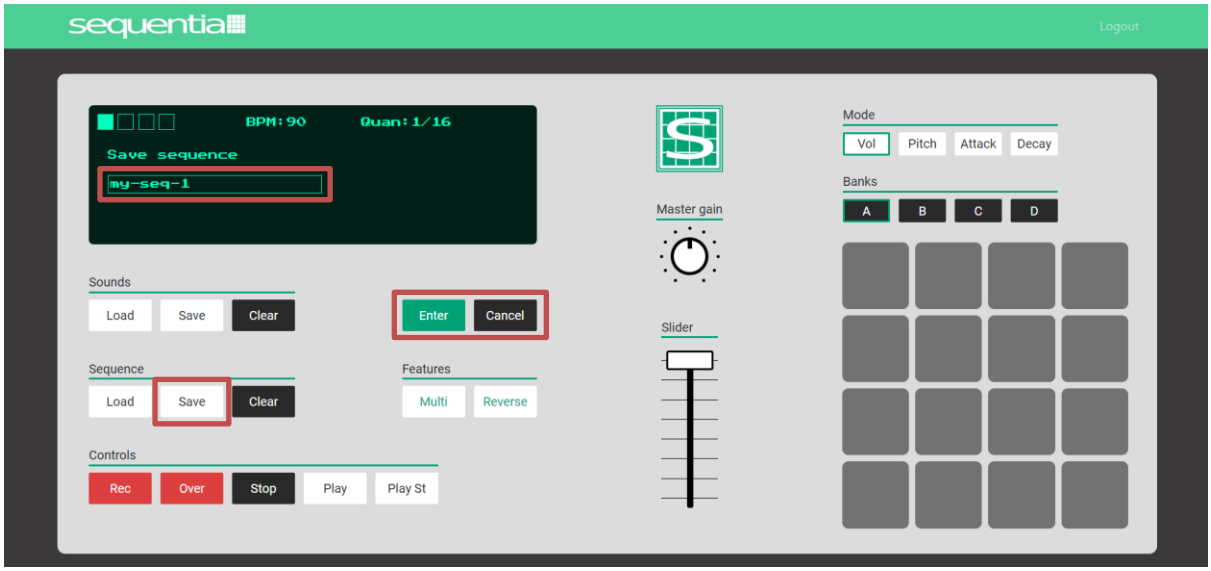

Figura 46. Guardado de secuencia

Para **recuperar una secuencia** guardada previamente pulsaremos el botón *Load* bajo el grupo *Sequence*, escogeremos el nombre de la secuencia que queremos recuperar y pulsaremos el botón *Enter*. La secuencia será cargada y se eliminará la secuencia que hubiese en el proyecto actual. Si se quiere abortar la operación, esto se realizar a través del botón *Cancel*.

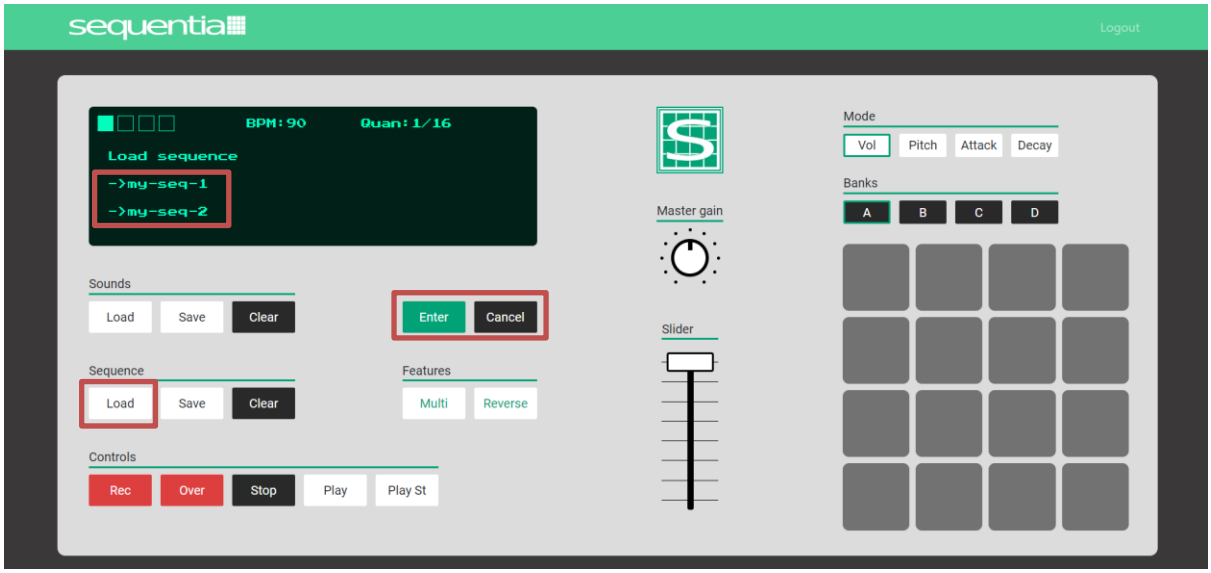

Figura 47. Carga de secuencia

6.3. Página no encontrada

Esta será la pantalla a la que el usuario será redirigido cuando se produzca un error 404, por recurso no encontrado.

Figura 48. Pantalla de error 404

7. CONCLUSIONES Y TRABAJOS FUTUROS

Conclusiones

La realización de este proyecto ha supuesto un gran desafío, en cuanto a que la naturaleza de este no encaja con el esquema de ninguna aplicación web desarrollada hasta la fecha. Cuando se desarrolla, por ejemplo, una aplicación dedicada a la gestión de activos, mediante el uso de tablas y formularios, la red proporciona una basta cantidad de recursos, información y ejemplos que ayudan a satisfacer los requisitos demandados y a resolver los problemas surgidos en el proceso. Sin embargo, en este proyecto a menudo nos hemos enfrentado a problemas cuya solución sólo ha podido ser encontrada a base de la experimentación y el estudio profundo de las tecnologías con las que se trabajaban.

Una de las motivaciones de este trabajo era el demostrar que se podía trasladar a la web un concepto específico, hasta ahora limitado a las aplicaciones de escritorio. Con una premisa fundamental, como era la de mantener una latencia mínima, el desarrollo de una aplicación centrada en el entorno cliente se hacía imprescindible.

Una de las nociones más importantes extraídas del trabajo ha sido el gran poder y potencial que tienen las tecnologías del entorno cliente en la web, permitiendo el desarrollo de aplicaciones dedicadas para distintos ámbitos de trabajo, con tanto o más poder que las aplicaciones de escritorio.

La mayor parte del desarrollo se ha realizado sin la necesidad de conexión a un servidor, resultando aun así la aplicación perfectamente funcional. Esto nos ha hecho plantearnos el potencial que tienen las aplicaciones de entorno cliente limitadas a un entorno local. Así, la concepción de una aplicación web cuyas funcionalidades básicas puedan utilizarse sin la necesidad de conexión a internet, pero a su vez disponga de funcionalidades adicionales a través de la conexión a la red, como pueden ser el guardado de datos en un servidor, el acceso a recursos adicionales, la capacidad de recibir actualizaciones de forma automática, etc., resulta una idea interesante, en cuanto que aunaría las ventajas de las tradicionales aplicaciones de escritorio (acceso sin necesidad de conectarse a internet), con aquellas que ofrece una aplicación web (compatibilidad entre distintos sistemas y dispositivos).

Trabajos futuros

Pese a que el proyecto final es un producto totalmente funcional, este tiene margen para mejoras y ampliaciones. Se enumeran a continuación algunas de las líneas de desarrollo que se consideran a la hora de extender el presente proyecto:

- Adición de más opciones para el tratamiento de las fuentes de audio, como puedan ser la aplicación de efectos espaciales o la habilidad de recortar los ficheros.
- Grabación de muestras de audio para ser utilizadas como sonidos en la aplicación.
- Composición de temas formados por distintas secuencias.
- Desarrollo de un apartado dentro del sitio web, que permita realizar la gestión de usuarios, secuencias y sonidos por parte de usuarios administradores.
- Creación de un tutorial interactivo que ayude a los nuevos usuarios a familiarizarse con el funcionamiento de la aplicación.

- Posibilitar el mapeo por parte del usuario de las teclas del teclado y de los controladores MIDI, de modo que pueda personalizar los controles del secuenciador.
- Mejora del sistema de registro, de modo que se solicite verificación a través del correo electrónico.
- Implantación en la aplicación de un modelo de negocio *freemium*, de modo que se ponga a disposición de los usuarios la opción de acceder a servicios extra a cambio del pago de una cuota mensual. Esto servicios adicionales incluirían el acceso a mayor capacidad de almacenamiento y una nueva opción de exportación del audio final resultante de la secuencia.

8. BIBLIOGRAFÍA Y REFERENCIAS

Bibliografía

Callicoder. *Spring Boot File Upload / Download Rest API Example*. https://www.callicoder.com/spring-boot-file-upload-download-rest-api-example. (Consultado el 8 de julio 2020)

HTML5 *Canvas Tutorials*. https://www.html5canvastutorials.com/tutorials. (Consultado el 8 de julio 2020)

Mozilla. *Documentación de la Web Audio API*. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API. (Consultado el 8 de julio 2020)

Palmer, Todd. *Creating a Library in Angular 6 series*. https://indepth.dev/creating-a-library-in-angular-6-using-angular-cli-and-ng-packagr. (Consultado el 8 de julio 2020)

Smus, Boris. *Getting Started with Web Audio API*. https://www.html5rocks.com/en/tutorials/webaudio/intro. (Consultado el 8 de julio 2020)

Tech Geek Net. *Spring Boot Security + JWT*. https://www.techgeeknext.com/spring/spring-boot-security-token-authentication-jwt. (Consultado el 8 de julio 2020)

W3C. *Documentación de la Web Audio API*. https://www.w3.org/TR/webaudio. (Consultado el 8 de julio 2020)

Referencias

[1] Pejrolo, Andrea (2011). *Creative Sequencing Techniques for Music Production: A Practical Guide to Pro Tools, Logic, Digital Performer, and Cubase* (p. 48). Taylor & Francis.

[2] Russ, Martin (2009). *Sound Synthesis and Sampling* (pp. 379-381). Routledge.

[3] Angular. *Documentación oficial*. https://angular.io/docs. (Consultado el 8 de julio 2020)

[4] W3C. *Primer borrador Web Audio API*. https://www.w3.org/TR/2011/WD-webaudio-20111215. (Consultado el 8 de julio 2020)

[5] Smus, Boris (2013). *Web Audio API* (pp. 1, 2). O'Reilly.

[6] Hispasonic. *El protocolo MIDI*. https://www.hispasonic.com/reportajes/protocolo-midi/13. (Consultado el 8 de julio 2020)

[7] Spring Boot. *Documentación oficial*. https://spring.io/projects/spring-boot. (Consultado el 8 de julio 2020)

[8] Spring. *Building a RESTful Web Service*. https://spring.io/guides/gs/rest-service. (Consultado el 8 de julio 2020)

[9] PostgreSQL. *Documentación oficial*. https://www.postgresql.org/about. (Consultado el 8 de julio 2020)

[10] Wilson, Chris. *A tale of two clocks*. https://www.html5rocks.com/en/tutorials/audio/scheduling (Consultado el 8 de julio 2020)

APÉNDICE. DOCUMENTACIÓN DE LA LIBRERÍA NGX-AUDIO-MNGR

ngx-audio-mngr

Figura 49. Logotipo de la librería ngx-audio-mngr

ngx-audio-mngr es una librería creada para facilitar el uso de la Web Audio API en proyectos de Angular. Permite realizar las tareas de tratamiento de audio más comunes de una forma más simplificada, manteniendo la flexibilidad original ofrecida por la Web Audio API, ya que es posible acceder a alguna de sus interfaces.

Además, esta librería incorpora la capacidad de visualizar amplitudes de onda de las fuentes de audio consumidas a través de un componente, que ofrece distintos parámetros de configuración.

Instalación

```
npm install @ngx-audio-mngr -save
```

Uso

1. **Importar AudioMngrModule en el NgModule raíz de la aplicación:**

```
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AudioMngrModule } from '@jfraga/ngx-audio-mngr';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ AudioMngrModule ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. **Inyectar y utilizar AudioMngrService en un componente:**

```
import { AudioMngrService} from "ngx-audio-mngr";

/** Declara nuevo objeto de sonido */
mySound: AudioMngrSound;

class AppComponent implements OnInit {
   /** Inyecta AudioMngrService en el constructor */
  constructor(private audioMngr: AudioMngrService) {}

  ngOnInit() {
    /** Inicializa y carga sonido */
    this.mySound.initSound('https://freesound.org/488795.mp3');
  }

  play() {
    /** Reproduce sonido */
    this.audioMngr.play(this.mySound);
  }
}
```

3. **Utilizar componente de visualización en plantilla:**

```
<ngx-audio-mngr
   [waveId]="1"
   [height]="200"
   [width]="600"
   [sound]="mySound"
   [waveColor]="'#00ffbc'"
   [progressColor]="'#c9fff1'"
   [lineColor]="'white'">
</ngx-audio-mngr>
```

API

AudioMngrService

Propiedades

audioContext

- **Tipo**: *AudioContext*
- **Descripción**: Interfaz *AudioContext* de la Web Audio API. Puede ser utilizada para tareas avanzadas, como crear sonidos personalizados aplicando procesamientos de sonido a un sonidos ya existente a través del método *createBuffer()*, o para acceder al reloj de hardware a través de la propiedad *currentTime*. Para más información sobre esta interfaz visitar https://developer.mozilla.org/en-US/docs/Web/API/AudioContext.

---

masterGain

- **Tipo**: *number*
- **Descripción**: Valor de la ganancia (volumen) entre 0 y 2. El valor 0 silencia la salida de audio y el valor 1 representa la amplitud original.
- **Valor por defecto**: 1

---

masterMuted

- **Tipo**: *boolean*
- **Descripción**: Si su valor es *true*, no sonará ningún sonido.
- **Valor por defecto**: *false*

Métodos

cloneSound

- **Descripción:** Crea una copia de un objeto *AudioMngrSound* existente.

- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
sound	No	AudioMngrSound	Objeto AudioMngrSound a ser clonado
name	Sí	string	Nombre del nuevo objeto

- **Tipo devuelto:** *AudioMngrSound*

---

initSound

- **Descripción:** Crea un objeto AudioMngrSound.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
source	Sí	File | string | AudioBuffer	Fuente de audio. Si es provisto, la fuente será cargada tras la creación del objeto Admite diferentes tipos: - File: Fichero de audio - string: URL de la fuente de audio - AudioBuffer: Objeto AudioBuffer nativo
name	Sí	string	Nombre del nuevo objeto

- **Tipo devuelto:** *AudioMngrSound*

---

muteMaster

- **Descripción:** Silencia el máster.
- **Tipo devuelto:** *void*

---

pause

- **Descripción:** Pausa un sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
sound	No	AudioMngrSound	Sonido que será pausado

- **Tipo devuelto:** *void*

---

pauseAll

- **Descripción:** Pausa todos los sonidos que están siendo reproducidos.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
sound	No	AudioMngrSound	Sonido que será pausado

- **Tipo devuelto:** *void*

---

play

- **Descripción:** Reproduce un sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
sound	No	AudioMngrSound	Sonido que será reproducido
offset	Sí	number	Posición de inicio del sonido (en segundos) El valor por defecto es 0
delay	Sí	number	Número de segundos que pasarán hasta que el sonido sea reproducido El valor por defecto es 0

- **Tipo devuelto:** *void*

---

resumeAll

- **Descripción:** Reanuda todos los sonidos que se encuentren pausados.
- **Tipo devuelto:** *void*

---

setAudioTimeout

- **Descripción:** Se comporta como el timeout nativo de JavasScript, pero haciendo uso del reloj del hardware. Ejecutará la función que se pase como parámetro transcurrido el intervalo de tiempo indicado.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
fn	No	Function	Función a ejecutar

delay	No	number	Número de segundos que pasarán hasta que la función sea ejecutada

- **Tipo devuelto:** *void*

---

setMasterGain

- **Descripción:** Establece la ganancia (volumen) del máster.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
gain	No	number	Nuevo valor para la ganancia Se admiten valores entre 0 y 2 0 silenciará la salida de audio y 1 representa la amplitud original El valor por defecto es 1

- **Tipo devuelto:** *void*

---

stop

- **Descripción:** Para la reproducción de un sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
sound	No	AudioMngrSound	Sonido que será parado
delay	Sí	number	Número de segundos que pasarán hasta que el sonido sea parado El valor por defecto es 0

- **Tipo devuelto:** *void*

---

stopAll

- **Descripción:** Para la reproducción de todos los sonidos.
- **Tipo devuelto:** *void*

---

unmuteMaster

- **Descripción:** Deshabilita el silencio para el máster.
- **Tipo devuelto:** *void*

AudioMngrSound

Eventos

endedListener

- **Descripción**: Este evento es lanzado cuando un sonido deja de reproducido, bien porque sea parado, o porque se complete la reproducción del sonido.

---

loadedListener

- **Descripción**: Este evento es lanzado cuando la fuente de audio acaba de ser cargada en un objeto AudioMngrSound.

Propiedades

audio

- **Tipo**: *AudioBuffer*
- **Descripción**: Objeto AudioBuffer que contiene los datos en bruto del sonido.

---

fadeIn

- **Tipo**: *number*

- **Descripción**: Número de segundos que han de transcurrir hasta que se alcance la amplitud máxima del sonido desde que este empieza a ser reproducido (attack).
- **Valor por defecto**: 0

---

fadeOut

- **Tipo**: *number*
- **Descripción**: Número de segundos que han de transcurrir desde la amplitud máxima del sonido hasta el final de la reproducción (decay).
- **Valor por defecto**: 0

---

gain

- **Tipo**: *number*
- **Descripción**: Valor de la ganancia (volumen) de un sonido. Acepta valores de 0 a 2. El valor 0 silenciará el sonido y 1 representa la amplitud original.
- **Valor por defecto**: 1

---

loaded

- **Tipo**: *boolean*
- **Descripción**: Si su valor es *true*, indica que la fuente de sonido ha terminado de ser cargada.
- **Valor por defecto**: *false*

---

muted

- **Tipo**: *boolean*
- **Descripción**: Si su valor es *true*, el sonido estará silenciado.
- **Valor por defecto**: *false*

---

name

- **Tipo**: *string*
- **Descripción**: Nombre del sonido.

---

position

- **Tipo**: *number*
- **Descripción**: Posición actual del sonido, en segundos.

---

semitones

- **Tipo**: *number*
- **Descripción**: Número de semitonos arriba o abajo respecto a la fuente original del sonido. En la práctica, realiza la misma función que el parámetro *speed*, pero utilizando como unidad de medida el semitono. Permite valores entre -24 (1/4 de la velocidad original) y 24 (cuatro veces la velocidad original). Si su valor es distinto a 0, el valor del parámetro *speed* no será tenido en cuenta.
- **Valor por defecto**: 0

---

speed

- **Tipo**: *number*
- **Descripción**: Velocidad de reproducción de un sonido. Permite valores entre 0 (el sonido será parado) y 4 (cuatro veces la velocidad original). Si el valor del parámetro *semitones* es distinto a 0, el valor de *speed* no será tenido en cuenta.
- **Valor por defecto**: 1

---

state

- **Tipo**: *State*
- **Descripción**: Estado actual del sonido. Los valores disponibles son: NONE, LOADING, PLAYING, STOPPED, PAUSED y SCHEDULED.
- **Valor por defecto**: *NONE*

Métodos

loadSound

- **Descripción:** Carga un nuevo sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
source	No	File | string | AudioBuffer	Fuente de audio que será cargada en el sonido. Admite diferentes tipos: - File: Fichero de audio - string: URL de la fuente de audio - AudioBuffer: Objeto AudioBuffer nativo

- **Tipo devuelto:** *void*

---

mute

- **Descripción:** Silencia el sonido.
- **Tipo devuelto:** *void*

---

reverse

- **Descripción:** Invierte la fuente de audio del sonido. El sonido será reproducido de atrás hacia adelante.
- **Tipo devuelto:** *void*

---

setFadeIn

- **Descripción:** Establece tiempo de fade-in para el sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
fadeIn	No	number	Valor del fadeIn en segundos. Se admiten valores comprendidos entre 0 y la duración del sonido. El valor por defecto es 0.

- **Tipo devuelto:** *void*

---

setFadeOut

- **Descripción:** Establece tiempo de fade-out para el sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
fadeOut	No	number	Valor del fadeOut en segundos. Se admiten valores comprendidos entre 0 y la duración del sonido. El valor por defecto es 0.

- **Tipo devuelto:** *void*

---

setGain

- **Descripción:** Establece la ganancia del sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
gain	No	number	Valor de la ganancia. Se admiten valores comprendidos entre 0 y 2. El valor 0 silencia el sonido y 1 representa la amplitud original. El valor por defecto es 1.

- **Tipo devuelto:** *void*

---

setSemitones

- **Descripción:** Establece el número de semitonos arriba o abajo para sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
semitones	No	number	Número de semitonos arriba o abajo respecto al valor original de la fuente de audio. Se admiten valores comprendidos entre -24 (1/4 de la velocidad original) y 24 (cuatro veces la velocidad original). El valor por defecto es 0.

- **Tipo devuelto:** *void*

---

setSpeed

- **Descripción:** Establece la velocidad de reproducción del sonido.
- **Parámetros:**

Nombre	Opcional	Tipo	Descripción
speed	No	number	Velocidad de reproducción del sonido. Se admiten valores comprendidos entre 0 (el sonido se parará) y 4 (cuatro veces la velocidad original). El valor por defecto es 1.

- **Tipo devuelto:** *void*

---

unmute

- **Descripción:** Deshabilita el silencio para el sonido.
- **Tipo devuelto:** *void*

AudioMngrComponent

Inputs

channel

- **Tipo**: *Channel*
- **Descripción**: Canal de la fuente de audio que será renderizado. Permite las siguientes opciones de enumerado:
  - Channel.LEFT: Sólo el canal izquierdo será renderizado
  - Channel.RIGTH: Sólo el canal derecho será renderizado
  - Channel.BOTH: Ambos canales serán renderizados
  - Channel.MERGED: Los canales izquierdo y derecho serán combinados en un nuevo canal, el cuál será renderizado.

  Si la fuente de audio solo tiene un canal, el valor de *channel* será ignorado.
- **Valor por defecto**: *Channel.LEFT*

---

height

- **Tipo**: *number*

- **Descripción**: Altura del contenedor de la onda, en píxeles.

---

lineColor

- **Tipo**: *string*
- **Descripción**: Valor CSS para el color de la línea de posición.
- **Valor por defecto**: "dimGray"

---

progressColor

- **Tipo**: *string*
- **Descripción**: Valor CSS para el área de la onda que ha sido reproducido.
- **Valor por defecto**: "lightCoral"

---

responsive

- **Tipo**: *boolean*
- **Descripción**: Si su valor es *true*, el contenedor de la onda se adaptará a los cambios en el ancho.
- **Valor por defecto**: *true*

---

sound

- **Tipo**: *AudioMngrSound*
- **Descripción**: Sonido a ser renderizado.

---

updateSource

- **Tipo**: *Observable*
- **Descripción**: Sirve para recibir un evento que fuerce a renderizar de nuevo la fuente de audio.

---

waveColor

- **Tipo**: *string*
- **Descripción**: Valor CSS para el área de la onda que no ha sido reproducido.
- **Valor por defecto**: "crimson"

---

waveId

- **Tipo**: *string*
- **Descripción**: Identificador único para la onda.

---

width

- **Tipo**: *number*
- **Descripción**: Ancho del contenedor de la onda, en píxeles.
