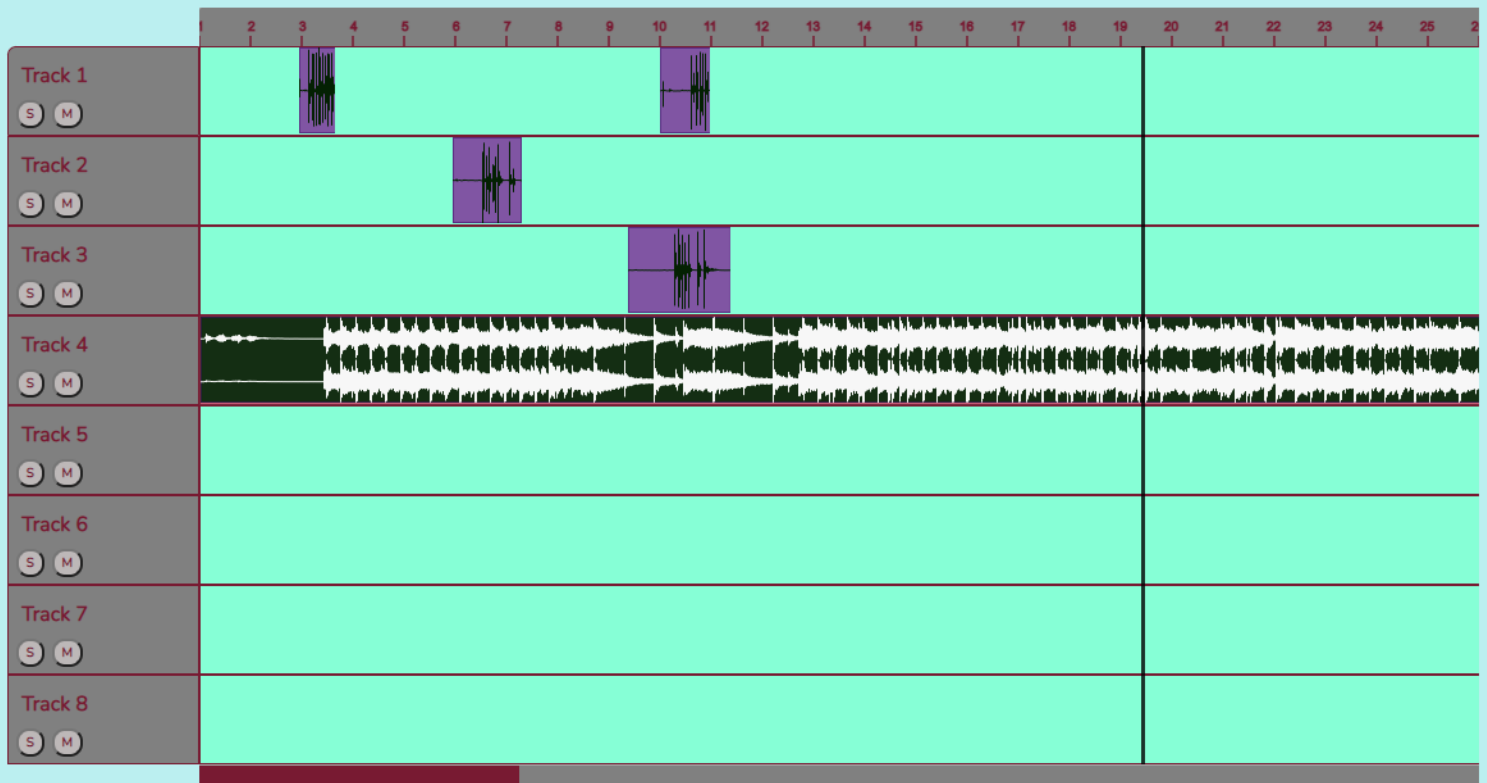


Daw Cloud –

Desarrollo de un editor de audio online



Víctor López Gómez

1. Índice

1. Índice	2
2. Introducción	3
3. Palabras clave	4
4. Módulos formativos aplicados en el trabajo	5
5. Objetivos del proyecto	6
6. Herramientas y lenguajes utilizados	7
7. Fases del proyecto:	
- Modelo de datos	9
- Back end Laravel	11
- Casos de uso	15
- Diagrama de clases	16
- Web Audio API	18
- MediaStream Recording API	19
- Canvas API	20
- Interacción cliente-servidor. AJAX	22
- Laravel Mix	23
8. Conclusiones y mejoras del proyecto	25
9. Bibliografía	27
10. Anexos	
- Manual de despliegue de la aplicación en un servidor	28

2. Introducción

El presente es un documento donde se recogen las premisas, desarrollo, observaciones y conclusiones relativas al proyecto de investigación experimental que acompaña.

La motivación principal del mismo parte de la siguiente idea: “¿existen aplicaciones en entorno web que estén orientadas al proceso y edición de audio?”. Tras una búsqueda exhaustiva, se puede confirmar que no existen *softwares* de nivel profesional. Esto es debido a la dificultad que plantea la sincronización precisa de diferentes ficheros de audio, teniendo en cuenta que toda petición http tiene que tardar una cantidad de tiempo.

No obstante, y gracias al desarrollo continuo de nuevas tecnologías; con el advenimiento de las *single page applications* y la evolución de las *web API* que los navegadores web incluyen en su código, este proyecto pretende comprobar si es posible, pues, desarrollar una aplicación que pueda ser funcional, más el factor añadido de que gracias a internet, sería teóricamente posible que diversos usuarios compartan un mismo proyecto a distancia.

Siendo definitorio por sí mismo, el título de la aplicación es *daw Cloud*. Es importante remarcar que muchas de las funcionalidades necesarias para ser un producto acabado no han sido aún implementadas a fecha de la entrega del proyecto, y se comentarán en detalle más adelante en este mismo documento.

3. Palabras clave

- Laravel: *framework* de código abierto para desarrollar aplicaciones y servicios web con PHP. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti".
- DAW: Siglas de *Digital Audio Workstation*, o estación de trabajo de audio digital. Son utilizadas actualmente en casi toda la Producción discográfica a nivel mundial y para la posproducción de sonido para cine, televisión y videojuegos.
- *Panning*: Distribución de una señal de sonido (ya sea en pares monoaurales o estereofónicos) en un nuevo campo de sonido estéreo o multicanal determinado por un ajuste de control de panorámica. Un ejemplo clásico es emitir un sonido de manera que a la persona que lo escucha le parece que se está produciendo más a su derecha que a su izquierda o viceversa.
- BPM: Siglas de *Beats Per Minute*, ritmos por minuto, medida utilizada para el control de la velocidad de un ritmo musical.
- Frecuencia de muestreo: número de muestras por unidad de tiempo que se toman de una señal continua (sonido analógico) para producir una señal discreta (almacenamiento digital). El estándar de un CD es, por ejemplo, de 44100 muestras por segundo.
- Cuantificación: teniendo en cuenta la frecuencia de muestreo, se trata del conjunto de valores de amplitud de una onda de audio en el mundo digital. A tipos de datos numéricos más pesados, más fiel será la conversión A/D (el estándar de un CD es de 16 bits, JavaScript trabaja nativamente con floats de 64 bits).

4. Módulos formativos aplicados en el trabajo

- Bases de datos

Uso de la base de datos MySQL, creación de modelo E-R y modelo relacional.

- Programación

Uso de diversos lenguajes de programación.

- Desarrollo web en entorno cliente

Uso de JavaScript, AJAX, etc.

- Desarrollo web en entorno servidor

Uso del patrón MVC mediante Laravel.

- Lenguajes de marcas

Uso de HTML5.

- Entornos de desarrollo

Uso de GIT, diagramas UML.

- Despliegue de aplicaciones web

Despliegue de servidor de aplicaciones y software necesario para el funcionamiento de la aplicación, creación de un manual de despliegue de la aplicación en un servidor.

- Inglés técnico

Uso generalizado del idioma en la aplicación y la documentación.

5. Objetivos del proyecto

daw Cloud, a nivel de proyecto de investigación experimental, tiene como objetivo el desarrollo de una estación de edición de audio online, con las siguientes características y funcionalidades:

- Reproducir archivos de audio y observar sus formas de onda.
- Grabar audio y secuenciarlo en el tiempo, dados un tempo y métrica.
- Ajustar bpm
- Ajustar métrica de compás
- Hacer zoom en el proyecto para edición de precisión.
- Cargar ficheros de audio propios del usuario.
- Ajustar los niveles de ganancia de cada pista, además de volumen master.
- Seleccionar la reproducción de una pista concreta, silenciando el resto (solo).
- Silenciar la reproducción de una pista concreta (mute).
- Ajustar el panorama estéreo de cada pista.
- Sistema de usuarios seguro: registro, *log in* y *log out*.
- Guardar y cargar proyectos asociados a una cuenta de usuario.
- Eliminar proyectos.

Varios objetivos propuestos en la idea inicial no han sido implementados a fecha de la entrega de este proyecto, y serán tratados en el apartado 8. *Conclusiones y mejoras del proyecto*.

6. Herramientas y lenguajes utilizados

A continuación, se listan todas las herramientas y software utilizados en el proyecto:

- Laravel: *framework* para desarrollo de aplicaciones web basado en el patrón MVC y PHP como lenguaje.
- Composer: software de distribución de paquetes, necesario para instalar Laravel y características del mismo.
- Node.js/NPM: Laravel utiliza este sistema, basado en Node.js, para instalar distintos nodos o librerías.
- Laragon: software de despliegue de aplicaciones. Gracias a este se puede configurar el servidor web, los lenguajes de programación que implementa, así como la base de datos escogida, activándolo todo con tan solo un click. Dentro del mismo se despliegan:
- Apache 2.4: servidor web escogido para el despliegue de la aplicación.
- MySQL 8: base de datos relacional
- Visual Code Studio: software IDE, muy conocido, con multitud de funcionalidades y ayudas para el desarrollo informático.
- Visual Paradigm: generador de diagramas UML online gratuito.
- Bootstrap: librería de procesamiento de CSS. Está disponible opcionalmente con laravel.
- AudioBuffer to Wav: pequeño nodo/librería de JavaScript que añade automáticamente la cabecera necesaria a un array con información de audio para que pueda ser codificado como wav.
- GIT/GitHub: sistema de control de versiones, utilizado ampliamente a lo largo del desarrollo del proyecto.

Estos son los lenguajes que han sido utilizados en el proyecto:

- PHP 8: lenguaje orientado a la programación del lado servidor. Utilizado en controladores, creación de tablas y rutas de la aplicación.
- Blade: lenguaje para desarrollar vistas. Parte de PHP.
- Artisan: Interfaz de consola de comandos de Laravel.
- HTML5: HTML, CSS, JavaScript ES6
- JSON: utilizado para almacenar todos los datos de un proyecto.
- jQuery: utilizado muy mínimamente en peticiones Ajax POST para su incorporación en el proyecto.
- Sass – scss: preprocesador CSS, utilizado por Bootstrap, que permite utilizar variables.

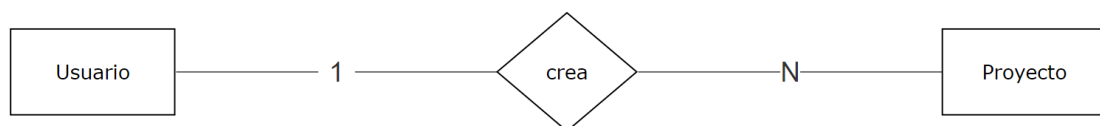
7. Fases del proyecto

Es muy importante indicar que, por el planteamiento y objetivos de este proyecto, la inmensa parte de la aplicación ha sido desarrollada en JavaScript en lado cliente, ya que, por la sincronización necesaria, no sería viable un desarrollo desde servidor, además de la gran carga que tendría a nivel computacional si se diera el caso de que haya muchos usuarios conectados simultáneamente. Por ello, la necesidad de disponer de una base de datos se limita a únicamente a los usuarios y los datos asociados a un proyecto, que JavaScript genera.

- Modelo de datos

El *framework* Laravel permite escoger entre diversos tipos de bases de datos. Dado que el modelo necesario para el funcionamiento de este proyecto es bastante sencillo, ha sido utilizado MySQL, generando así un modelo relacional de datos.

Dado que aún no se ha implementado la posibilidad de que varios usuarios compartan proyectos, el modelo dispone de dos entidades con relación 1n (uno a muchos).



Gracias a Laravel, es posible crear, sobre una base de datos ya existente, todas las tablas de manera automática, mediante la funcionalidad *migrate*. Con un único comando de artisan, se genera el fichero php desde el que se dará forma a cada tabla:

```
php artisan make:migration create_projects_table
```

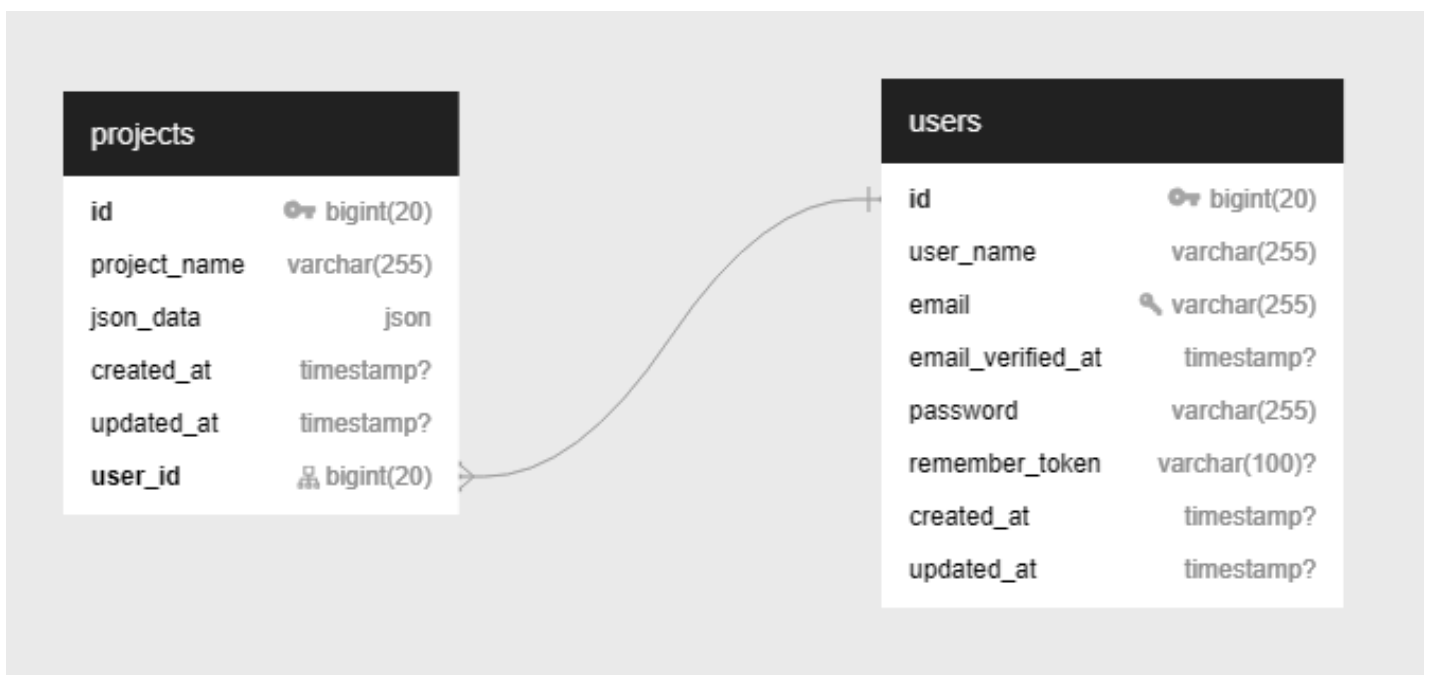
```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('user_name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

Una vez preparadas las instrucciones, se ejecuta el siguiente comando, que crea la tabla en la base de datos:

```
php artisan migrate
```

En este caso, la tabla usuarios dispone de las columnas “remember_token” y “email_verified_at”, relativos a la función de recordar un usuario en un dispositivo y la fecha de confirmación de email a través de correo, respectivamente; estas no han sido implementadas en la aplicación a día de hoy, no obstante, no son requeridas para el correcto funcionamiento de la misma.

El modelo relacional resultante quedaría del siguiente modo:



Es interesante observar cómo se hace uso de una columna con tipo de dato JSON, con el fin de almacenar los datos del proyecto. Al no haber necesidad de que un controlador del servidor gestione esa información, es “empacada” y enviada a JavaScript para su proceso en el entorno *front end*.

- Back end Laravel

Una vez dispuesta la base de datos, este apartado tratará del back end desarrollado. Como se ha indicado anteriormente, Laravel sigue el patrón MVC, e incorpora multitud de herramientas y ayudas para facilitar la programación de una aplicación web. Es de interés indicar que el *framework* dispone de dos modos de interacción con la capa de datos:

- *Eloquent*: el desarrollador crea un modelo asociado a cada tabla, cuyas funciones serán llamadas en los controladores.
- *Query Builder*: no es necesario generar un modelo específico, sino que es posible hacer *queries* directamente desde el controlador.

Dada la simplicidad del *back end* del proyecto, se ha optado por el uso de *Query Builder*; este no disminuye las funcionalidades de validación de datos y autorización de Laravel.

Se han desarrollado dos controladores, uno para la gestión de usuarios, y otro para la gestión de proyectos.

Se pone como ejemplo el proceso de validación para el registro de usuarios, mediante la clase *Validator*:

```
$validator = Validator::make($request->all(), [
    'email' => ['required', 'unique:users,email', 'max:255'],
    'user_name' => ['required', 'max:255'],
    'password' => ['required', Password::min(8)]
]);

if ($validator->fails()) {
    return back()
        ->withErrors($validator)
        ->withInput();
}
```

Si los datos introducidos por el usuario no cumplen las normas establecidas, se redirige a la misma página, imprimiendo el/los errores.



Mencionar también el uso de la clase Password para generar condiciones referentes a la contraseña, en este caso se solicita que tenga al menos 8 caracteres.

Una vez la validación es correcta, se inserta el usuario en la base de datos, retornando su id, que permite iniciar sesión desde la misma función:

```
$user_id = DB::table('users')->insertGetId([
    'user_name' => $user_name,
    'email' => $email,
    'password' => Hash::make($password),
    'created_at' => now(),
    'updated_at' => now()
]);

Auth::loginUsingId($user_id);
```

La contraseña es cifrada mediante la clase Hash de laravel.

 id	user_name	 email	email_verified_at	password	remember_token	created_at	updated_at
21	sry	xdfygh@ho.co	(NULL)	\$2y\$10\$hOyYj/hg3/RpdCJmlMvcj.ViuM4AVMjMte8/t4LIAi...	(NULL)	2021-06-06 09:53:32	2021
22	set	arg@ho.co	(NULL)	\$2y\$10\$yhLFRewsGBFIHPBDOMqMyuTioVIw6EsaEJ3dpy...	(NULL)	2021-06-06 09:54:15	2021

Usuarios en la base de datos.

En lo referente al inicio de sesión, se ha hecho uso de la clase *Auth*, la cual automatiza el proceso de almacenamiento y duración de la sesión.

Previamente debe de modificarse el fichero de configuración de Auth, para que, en lugar de realizar las comprobaciones mediante el modelo, las haga directamente en la base de datos:

```
'providers' => [
    //'users' => [
        //'driver' => 'eloquent',
        //'model' => App\Models\User::class,
    //],

    'users' => [
        'driver' => 'database',
        'table' => 'users',
    ]
],
```

La función de inicio de sesión realiza la búsqueda en la base de datos, almacena el usuario en la sesión y redirige hacia la aplicación gracias a un código muy sencillo:

```
public function login(Request $request)
{
    $credentials = $request->validate([
        'email' => ['required', 'email'],
        'password' => ['required']
    ]);

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect('app');
    }

    return back()->withErrors([
        'email' => 'Something is wrong with the provided info...'
    ]);
}
```

La clase auth automatiza la comparación de la contraseña cifrada con la escrita por el usuario.

En cuanto a los proyectos, el controlador guarda los elementos de audio como ficheros .wav en la carpeta de almacenamiento del servidor:

```
public function saveSound(Request $request)
{
    $sound = $request->file('audio-blob');
    $id = $request->input('recording-id');
    $projectname = $request->input('project-name');
    Storage::makeDirectory('public/projects/' . $projectname);
    $filename = 'public/projects/' . $projectname . '/'
        . $projectname . '_' . $id . '.wav';


    if (Storage::exists($filename)) {
        Storage::delete($filename);
    }
    Storage::put($filename, file_get_contents($sound));
}
```

Del mismo modo, deposita la información del proyecto en una única columna JSON, interaccionando con JavaScript mediante AJAX:

```
public function saveProject(Request $request)
{
    $projectname = $request->input('project-name');
    $project = $request->input('project');

    $search = DB::table('projects')
        ->where('project_name', $projectname)
        ->where('user_id', Auth::user()->id)
        ->first();

    if (!$search) {
        DB::table('projects')->insert([
            'project_name' => $projectname,
            'json_data' => json_encode($project),
            'user_id' => Auth::user()->id,
            'created_at' => now(),
            'updated_at' => now()
        ]);
    } else {
        DB::table('projects')
            ->where('project_name', $projectname)
            ->where('user_id', Auth::user()->id)
            ->update([
                'json_data' => json_encode($project),
                'updated_at' => now()
            ]);
    }
}
```

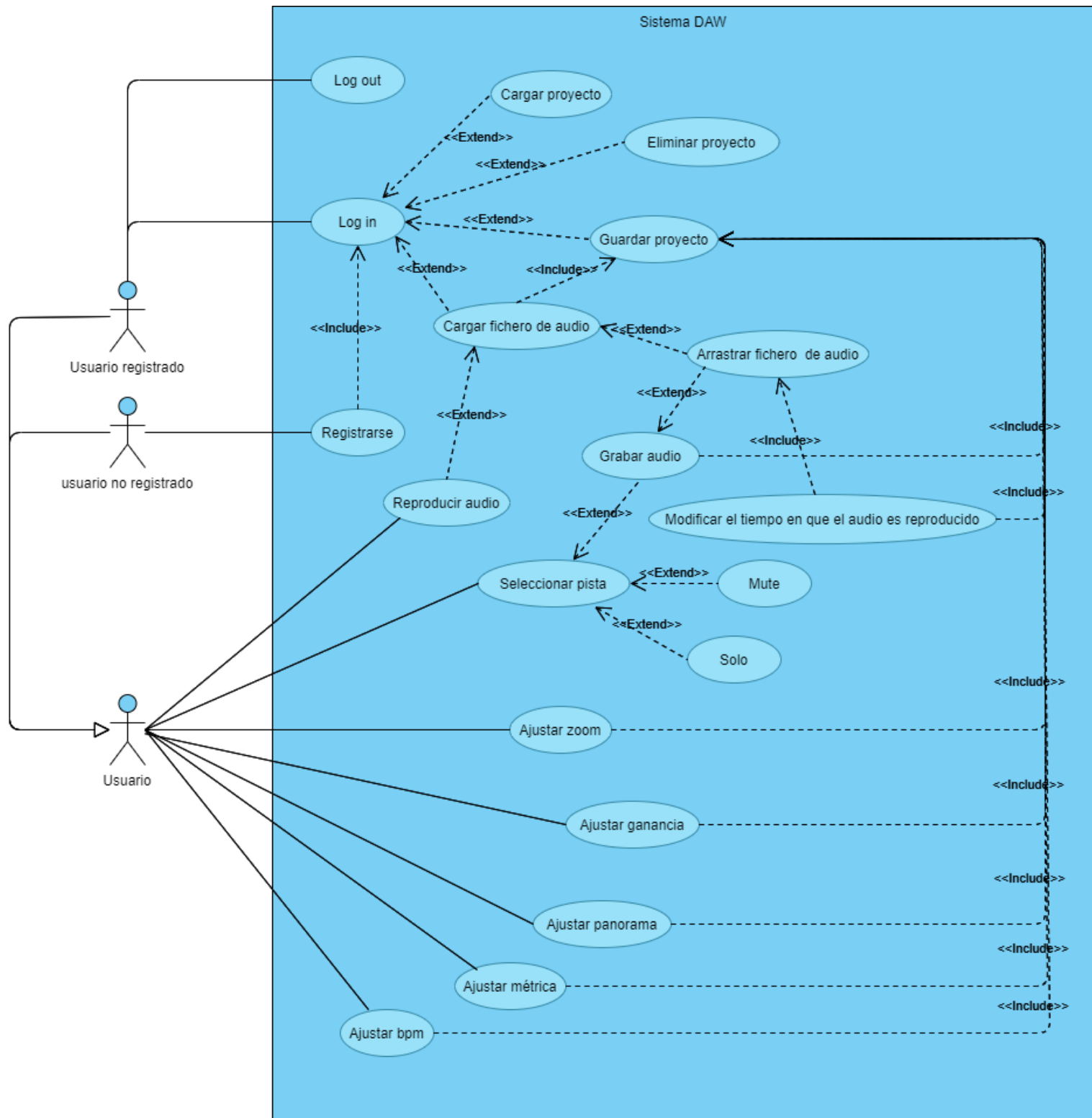
 id	project_name	json_data
27	project 1	"{"timeSpace":{"space":363,"zoom":0.05,"bpm":1,"compas":2},"recordings":[{"id":"recording_0","tracknumber":
28	prueba 1	"{"timeSpace":{"space":580,"zoom":0.05,"bpm":1,"compas":2},"recordings":[{"id":"recording_0","tracknumber":
29	prueba 2	"{"timeSpace":{"space":399,"zoom":0.05,"bpm":1,"compas":2},"recordings":[{"id":"recording_0","tracknumber":

Muestra de diversos proyectos dentro de la base de datos

Una vez documentado el *back end*, las siguientes secciones tratarán de lo que podría decirse como la “aplicación” en sí, desarrollada en el *front end*:

- Casos de uso

Bajo el actor usuario, existen dos actores en función de si se han registrado en la aplicación o no:



- Diagrama de clases

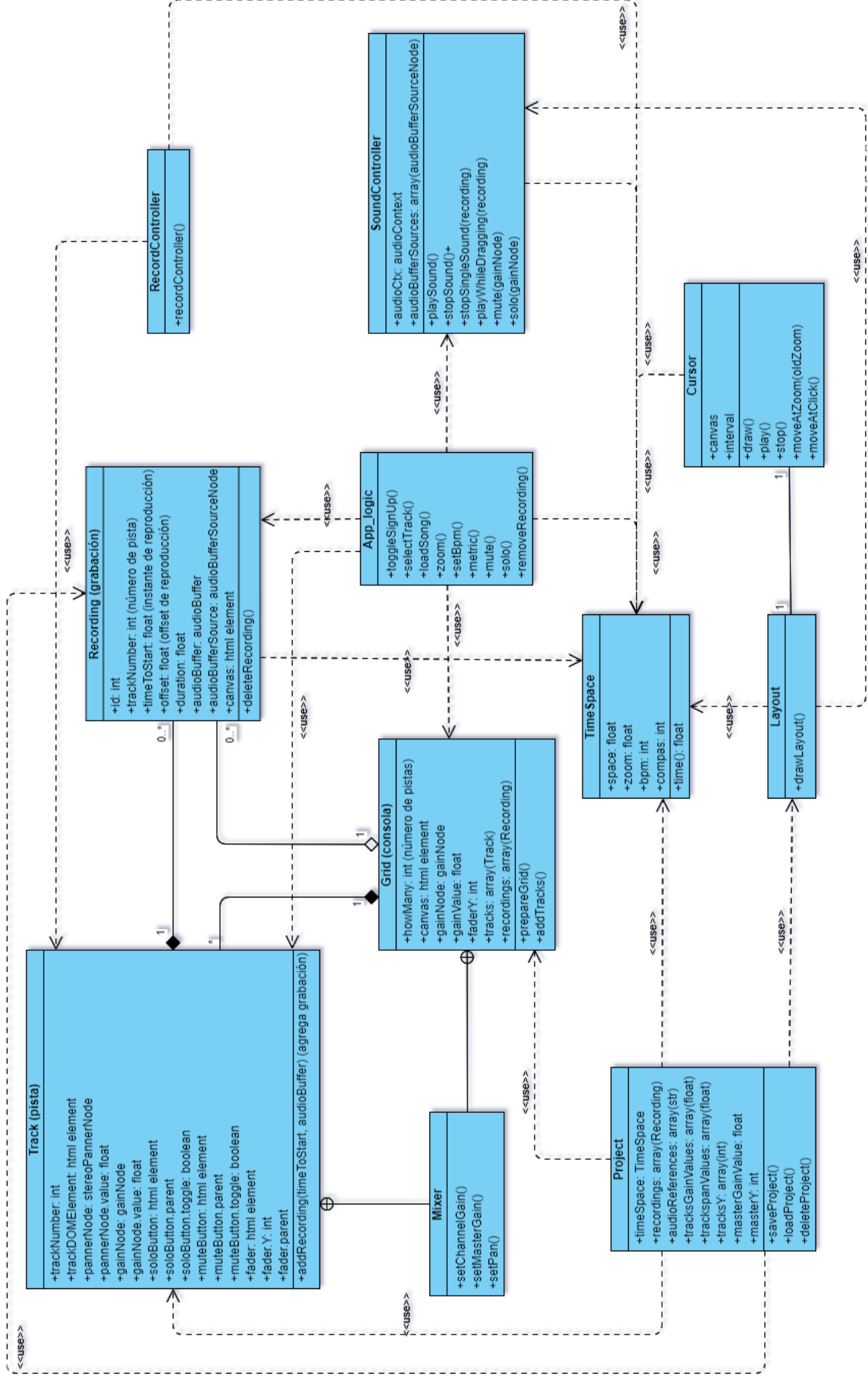
Dado que el grueso de la aplicación está desarrollado en Javascript para poder ser ejecutada por el navegador del propio usuario, la aplicación de la Programación Orientada a Objetos no es la usual que se esperaría con otros lenguajes.

No obstante, gracias a las últimas actualizaciones de JavaScript (ES6 en adelante), se ha podido emular un modelo de objetos similar al que se realizaría con una aplicación de escritorio. Concretamente, gracias a la posibilidad de segmentar el código en módulos y diferentes ficheros (Export/Import), y la creación de clases para instanciar objetos con propiedades y métodos específicos.

Existen muchas funciones asociadas a eventos, que no forman parte de ninguna clase como métodos estáticos. (este sería el caso de todas las funciones del fichero *app_logic.js*). Por otro lado, ninguna propiedad ha sido declarada privada. Muy probablemente el código podría reorganizarse para tomar partido de estas nuevas características que posee JavaScript y así también modularizar el programa en lo máximo posible.

Indicar también que la aplicación sigue la siguiente lógica: si el objeto es instanciado varias veces, con datos distintos (por ejemplo, las pistas o las grabaciones), ha sido definida una clase. Sin embargo, existen objetos, como el cursor o la abstracción donde se recogen las variables de configuración (*timeSpace*), que tan solo existen una vez en el curso de uso de la aplicación, que han sido creados sin una clase asociada.

Para simplicidad y mejor entendimiento de la estructura de la aplicación, el siguiente diagrama tiene en cuenta las anteriores observaciones, entendiendo como clase ficheros que a nivel informático pueden no tratarse de un objeto o clase, ya que no es necesario para su correcto funcionamiento, pero sí para la comprensión humana del código:



- Web Audio API

El proyecto desarrollado hace uso de Web Audio API, una especificación implementada por la mayoría de navegadores modernos.

Bajo la interfaz *AudioContext*, encargada de instanciar el resto de funcionalidades, la API emula el funcionamiento de una cadena de audio al uso, con nodos que pueden ser conectados en serie o en paralelo, finalizando el camino en el “destino”, o los altavoces del equipo.

En este proyecto se ha hecho uso de las siguientes interfaces:

- *AudioBuffer*: almacena información de audio en la memoria. Podría definirse como el análogo de un cassette o un CD.

- *AudioBufferSourceNode*: controla la reproducción del audio que se le “introduce”. Siguiendo la analogía anterior, sería el reproductor de cassette o CD.

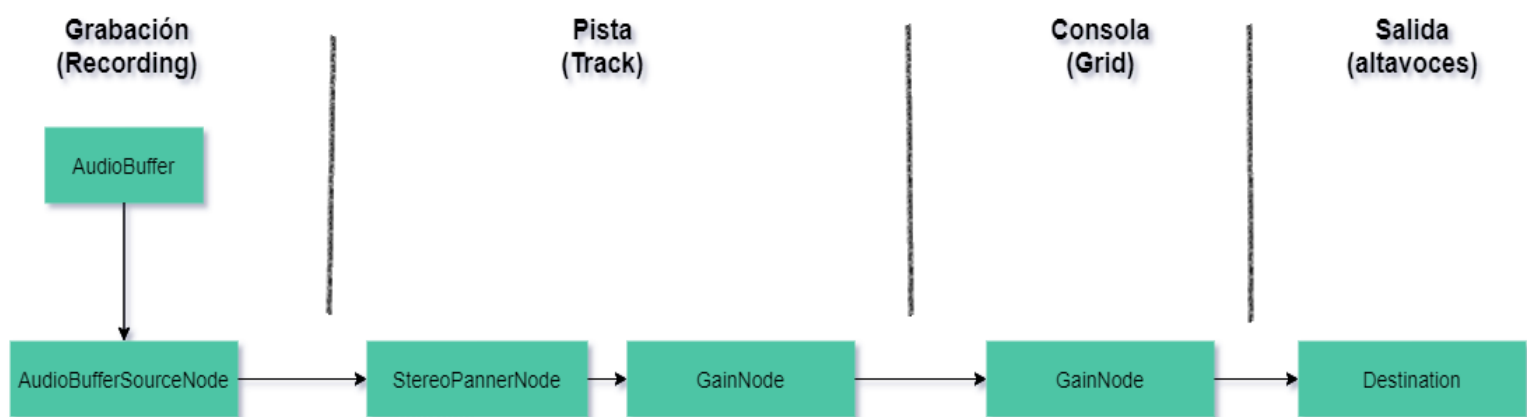
Esta interfaz tiene la peculiaridad de que, una vez se ordena la parada de la reproducción, o el audio llega a su fin, el elemento finaliza su ciclo de vida. Es por ello que el controlador de sonido desarrollado genera una interfaz por cada *AudioBuffer* cada vez que el usuario clicca en el botón de reproducir:

```
playSound() {
  for (var h = 0; h < grid.recordings.length; h++) {
    const source = audioCtx.createBufferSource();
    source.buffer = grid.recordings[h].audioBuffer;
    source.connect(grid.tracks[grid.recordings[h].tracknumber]
      .pannerNode);
    var start = Math.max((grid.recordings[h].timeToStart
      - timeSpace.time() + audioCtx.currentTime), 0);
    var offset = Math.max((timeSpace.time()
      - grid.recordings[h].timeToStart)
      + grid.recordings[h].offset, 0);
    var duration;
    if (grid.recordings[h].duration) {
      duration = grid.recordings[h].duration;
    }
    source.start(start, offset, duration);
    this.audioBufferSources.push(source);
    grid.recordings[h].audioBufferSource = source;
  }
}
```

- *GainNode*: interfaz encargada de controlar la ganancia de una fuente de audio. Cada pista tiene un nodo de ganancia, además de existir un control master, siguiendo el diseño de las mesas de mezcla analógicas. Es utilizada también para las funciones de *solo* y *mute* en la aplicación.

- *StereoPannerNode*: permite controlar el panorama de una fuente de audio, situándola a la izquierda o la derecha en un sistema estéreo.

La disposición de la cadena desarrollada puede resumirse en el siguiente diagrama:



- **MediaStream Recording API**

Además de la anterior especificación para controlar el audio, la aplicación desarrollada hace uso de *MediaStream Recording API*, la cual permite grabar audio a través del dispositivo de entrada que el usuario dispone en su ordenador.

La implementación realizada es sencilla: el controlador de grabaciones diseñado primero se asegura de que el usuario disponga de un dispositivo de entrada (y mediante una *promise* de JavaScript, que se trate de un micrófono).

El controlador genera un objeto de la interfaz *MediaRecorder()*, el cual, al clicar en el botón de grabar, empieza a almacenar la información de audio, hasta que se clicca en el botón de stop. En ese momento, el array generado “chunks” es convertido en un blob, este en un *arrayBuffer*, un tipo de array de JavaScript que recoge datos en código binario, y este, a su vez, en un *AudioBuffer*, mediante la interfaz *AudioContext* instanciada en el proyecto:

```
mediaRecorder.onstop = function recordSound() {
  const blob = new Blob(chunks, { 'type': 'audio/ogg; codecs=opus' });
  chunks = [];
  blob.arrayBuffer().then(arrayBuffer => {
    audioCtx.decodeAudioData(arrayBuffer, (audioBuffer) => {
      var track = document.querySelector('[data-selected]').id;
      grid.tracks[track].addRecord(startTime, audioBuffer);
    });
  });
}
```

- Canvas API

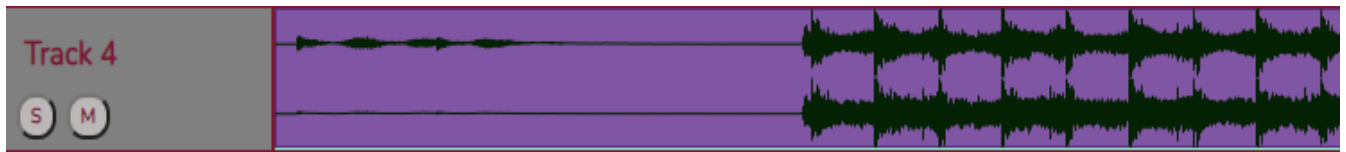
Canvas es una Web API que, a partir de la etiqueta <canvas> presente en html, permite realizar dibujos de gran complejidad. Ha sido utilizada para el dibujo dinámico de la parrilla de compases, modificable en función del compás escogido, del BPM, o del zoom aplicado, así como en el dibujo de las formas de onda de los audios.

Es remarcable el método utilizado para lograr esto último: extrayendo la información del AudioBuffer en cuestión, se compara con la longitud del dibujo en función de la duración del audio y el nivel de zoom, y a partir de los valores de amplitud que posee el buffer, se dibuja un rectángulo en cada píxel, la altura del cual depende del valor extraído.

Asimismo, la función también capacita el dibujo tanto de ficheros estéreo como mono. Este es el código correspondiente al dibujo de un audio mono:

```
else if (recording.audioBuffer.numberOfChannels === 1) { // si es mono..
  var data = recording.audioBuffer.getChannelData(0);
  var step = Math.ceil(data.length / width);
  var amp = height / 2;
  for (var i = 0; i < width; i++) {
    var min = 1.0;
    var max = -1.0;
    for (var j = 0; j < step; j++) {
      var datum = data[(i * step) + j];
      if (datum < min)
        min = datum;
      if (datum > max)
        max = datum;
    }
    canvasCtx.fillStyle = '#022100';
    canvasCtx.fillRect(i, (1 + min) * amp, 1, Math.max(1, (max - min) * amp));
  }
}
```

Forma de onda audio estéreo:



Forma de onda audio mono:



Forma de onda al ser clicada:



Relacionado con este campo, es importante mencionar que, para relacionar píxeles y el transcurso del tiempo en el audio, se ha creado un objeto, *timeSpace*, donde se almacenan las variables que relacionan todo el proyecto. El usuario modifica estas variables clicando en el cursor de tiempo, cambiando el bpm, moviendo el nivel de zoom, etc. El tiempo es calculado en función de estos:

```
export var timeSpace = {  
  space: 0,  
  time: function () {  
    var time = this.space * this.zoom;  
    return time;  
  },  
  zoom: 0.0500,  
  bpm: 1,  
  compas: 2  
};
```

- Interacción Cliente/Servidor. AJAX

Otro apartado remarcable en el desarrollo realizado en JavaScript trata del código aplicado para guardar, recuperar y eliminar proyectos creados por el usuario. La metodología empleada consiste en el uso de la clase Proyecto, la cual recoge como propiedades toda la información que el usuario ha ido generando, en el estado en que se encuentra en el momento de guardar el proyecto:

```
class Project {  
  constructor(timeSpace, recordings, tracksGainValues, trackspanValues,  
    tracksY, masterGainValue, masterY) {  
    this.timeSpace = timeSpace;  
    this.recordings = recordings;  
    this.audioReferences = [];  
    this.tracksGainValues = tracksGainValues;  
    this.trackspanValues = trackspanValues;  
    this.tracksY = tracksY;  
    this.masterGainValue = masterGainValue;  
    this.masterY = masterY;  
  }  
}
```

Por un lado, el objeto generado contendrá todas las variables del proyecto, como el *panning*, la ganancia, los datos correspondientes a cada grabación, etc. Asimismo, se genera un índice de los nombres de los audios, que son exportados a fichero .wav en el servidor.

Para lograr esto último, se ha hecho uso de una pequeña librería que introduce automáticamente la cabecera propia del formato .wav en el AudioBuffer. El resultado se convierte en un blob:

```
for (let i = 0; i < grid.recordings.length; i++) {  
  var blob = new window.Blob([new DataView(  
    toWav(grid.recordings[i].audioBuffer)]), {  
    type: 'audio/wav'  
  });  
}
```

Se genera un objeto FormData para enviar información al servidor como si se tratase de un formulario:

```
var formdata = new FormData();
formdata.append('audio-blob', blob);
formdata.append('recording-id', grid.recordings[i].id);
formdata.append('project-name', projectName);
```

Se genera una petición POST por AJAX por cada fichero de audio del proyecto, y una última petición para enviar el objeto Project generado, que es convertido en un string, para que el controlador en el servidor lo convierta en JSON, y lo incorpore en la base de datos:

```
var projectForm = new FormData();
projectForm.append('project-name', projectName);
projectForm.append('project', JSON.stringify(project));
$.ajax({
  headers: {
    'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
  },
  type: 'POST',
  url: 'saveproject',
  data: projectForm,
  processData: false,
  contentType: false,
  success: function (data) {
    console.log('Project saved successfully');
  }
});
```

- Laravel Mix

Como última mención referente a la estructura de la aplicación, Laravel incorpora Mix, una API orientada a compilar todos los ficheros del front end (css, sass, JavaScript...) en un único fichero para cada tipo de dato. Esta función es muy interesante para aplicaciones como la desarrollada, que despliega multitud de ficheros .js para establecer un modelo de objetos entendible para un programador. Los navegadores actuales, por su modo de operar, harían llamadas continuas a cada fichero, ralentizando la aplicación, y en este caso concreto, imposibilitando su funcionamiento modular.

Para muchas aplicaciones web, esto tiene vital importancia en cuanto el SEO del sitio.

Para compilar todos los *assets* correctamente, se debe configurar el fichero “webpack.mix.js”:

```
mix.js('resources/js/app_core.js', 'public/js')
    .js('resources/js/home.js', 'public/js')
    .sass('resources/sass/app.scss', 'public/css')
    .sourceMaps()
    .disableSuccessNotifications();
```

El fichero principal de Javascript en el proyecto hace uso del sistema de módulos CommonJs, así como del nuevo sistema modular que se ha integrado en JavaScript nativamente:

```
require('./bootstrap');
require('./app_logic');
require('./components/timelayout');
require('./project');
require('./components/mixer');

import { grid } from './components/generalgrid';
import drawLayout from './ui/ui_layout';
import { cursor } from './components/cursor';
import SoundController from './audio/soundcontroller';
import recordcontroller from './audio/recordcontroller';
```

Con este commando, Mix compila todos los ficheros, y “traduce” el fichero sass en css plano:

```
npm run dev
```

Es posible minimizar los ficheros públicos más aún para los entornos ya en producción:

```
npm run prod
```


8. Conclusiones y mejoras del proyecto

Este ha sido un proyecto bastante arduo de realizar, ya que, en el momento de iniciarlo, no disponía de conocimiento acerca del framework Laravel, ni de las web API utilizadas. Tenía certeza de que el uso que sería necesario darles implicaría una lógica compleja, donde todos los elementos interaccionan entre sí.

Considero que finalmente ha sido posible llegar a los objetivos principales que se plantearon en el inicio, y por el camino he recogido muchos conocimientos sobre multitud de materias y campos del desarrollo web.

Como observaciones, es interesante confirmar que durante el desarrollo me he percatado que el modelo diseñado dispone de limitaciones, causadas por la desincronización entre distintas variables. Un ejemplo importante de esto es que la función `setInterval()` de JavaScript, utilizada en el proyecto para el movimiento del cursor durante la reproducción de audio, tiene un valor mínimo de 10 milisegundos, no siendo suficiente llegado cierto nivel de zoom. Esto sería solucionable creando algoritmos que requieren de matemática más compleja.

Mi conocimiento profesional en audio me ha permitido agilizar ciertos aspectos, como es el caso del control de ganancia mediante fader: el oído humano tiene un comportamiento “logarítmico”, y eso se ha de aplicar a los valores dados:

```
gainValue = Math.log10(1 / ((Y + 5) / 260));  
// "Y" siendo el valor height en píxeles del elemento html
```

Si bien la estructura raíz de la aplicación ya ha quedado desarrollada, existen multitud de funciones que no han sido desarrolladas en el momento de la entrega del proyecto, tales como:

- Función metrónomo
- Función loop
- Efectos de sonido (eco, reverberación, chorus...)
- Distintos usuarios pueden compartir un mismo proyecto.
- Permitir bloquear el movimiento de cursor y audios por compases.
- Recortar y modificar la longitud de los audios

- Exportar la mezcla restante en formato audio y que el usuario pueda descargar el fichero.

A día de hoy, la función para exportar el proyecto como fichero de audio, algo realmente importante como objetivo dentro de una aplicación DAW, está en desarrollo; uniendo los datos de todos los buffers, teniendo en cuenta la frecuencia de muestreo.

En el aspecto visual han quedado muchos detalles por mejorar, complejizar la interfaz para que disponga de más datos visuales y sea más intuitiva:

- Marcador de compases mucho más detallado en función del zoom
- Faders de ganancia con indicadores numéricos, así como barras de nivel a tiempo real.
- El fondo de las pistas presente colores distintos a cada compás

No se ha realizado un estudio de UX ya que, el usuario de este tipo de programas tiene unas necesidades muy específicas, bien desarrolladas por los grandes programas de edición de audio: Pro Tools, Cubase, Ableton...

Referente a este aspecto, durante el transcurso del proyecto se trazó una usabilidad muy similar a la de estos programas, incorporando “shortcuts” de teclado para la interacción del usuario, funcionalidad finalmente postergada y aplicable en un futuro.

Por último, considero que la interfaz puede verse sencilla a día de hoy, y espero que con la implementación de las funcionalidades arriba mentadas vaya tomando una forma más “profesional”.

Tras la investigación llevada a cabo con este proyecto, he aprendido el funcionamiento del framework Laravel con bastante detalle, siendo este un genial método para desarrollar una aplicación web, ya que facilita muchas operaciones de backend, y proporciona aspectos de seguridad y autorización avanzados, de un modo fácil de aplicar.

9. Bibliografía

Documentación de Laravel:

<https://laravel.com/docs/8.x>

Documentación de PHP:

<https://www.w3schools.com/php/default.asp>

<https://www.php.net/manual/en/index.php>

Documentación de las web API empleadas:

<https://www.w3.org/TR/webaudio/>

https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API

https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

Librería *audiobuffer to wav*:

<https://www.npmjs.com/package/audiobuffer-to-wav>

Documentación general de JavaScript:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

<https://www.w3schools.com/js/default.asp>

10. Anexos

Manual de despliegue de la aplicación

A continuación, se detalla el software necesario, junto a las recomendaciones o elecciones que han sido escogidas en este proyecto:

- Enlace al repositorio de GitHub del proyecto:

<https://github.com/itt-victor/dawcloud>

- Servidor web: Apache

<https://www.apachelounge.com/download/VS16/binaries/httpd-2.4.48-win64-VS16.zip>

- Base de datos: MySQL

<https://downloads.mysql.com/archives/get/p/23/file/mysql-8.0.23-winx64.zip>

- PHP

<https://windows.php.net/download#php-8.0>

- Node.js

<https://nodejs.org/dist/v14.17.0/node-v14.17.0-win-x64.zip>

- Composer

<https://getcomposer.org/download/>

- Laragon

<https://github.com/leokhoa/laragon/releases/download/5.0.0/laragon-wamp.exe>

En primer lugar, y como recomendación por su facilidad de uso, debe configurarse Laragon para que llame a iniciarse Apache y MySQL, cargando PHP y Node.js en el mencionado servidor web. Es tan sencillo como introducir los directorios de las 4 aplicaciones en el directorio “bin” de Laragon.

Acto seguido, se deben de habilitar las siguientes extensiones de PHP, fácilmente a través del menú de opciones de Laragon:

Curl, fileinfo, gd, mbstring, mysqli, pdo_mysql

Desde un terminal de comandos, se hace “cd” al directorio donde se encuentra el proyecto, y se ejecutan los siguientes comandos:

```
composer install
#se instalan las dependencias requeridas por el proyecto
npm install
#se instalan los módulos requeridos
cp .env.example .env
#se genera un nuevo fichero env donde se almacena la configuración del proyecto
php artisan key:generate
#se genera una nueva clave de encriptación
```

A continuación, se crea una nueva base de datos vacía en MySQL, y se conecta al proyecto, editando los campos DB_HOST, DB_PORT, DB_DATABASE, DB_USERNAME, y DB_PASSWORD en el fichero .env

El proyecto entregado está configurado con los siguientes datos:

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=dawcloud

DB_USERNAME=root

DB_PASSWORD=Abcedf.1

Por último, se crean las tablas con el siguiente comando:

```
php artisan migrate
```