
**Universidad de Valladolid**

**ESCUELA DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**
**MENCIÓN EN INGENIERIA DE SOFTWARE**

---

# Un DJ Virtual como aplicación web

---

**Alumno/a: Carlos Lobo Mata**

**Tutor/es/as: Yania Crespo González-Carvajal**

*A mis padres, que siempre me han apoyado.*

# Agradecimientos

A mi familia, por apoyarme desde el principio, por difícil que se lo pusiera a veces.

A mi tutora Yania, por todo el trabajo que realiza a diario para ayudar a sus alumnos a aprender.

A mis amigos y compañeros de carrera, que me acompañaron durante el viaje, me enseñaron y me impulsaron a aprender muchas cosas, sin las que ahora no estaría donde estoy.

A mi novia Cristina, que me motiva a dar lo mejor de mi día a día.

**Gracias a todos**

# Resumen

El objetivo de este proyecto es desarrollar una Aplicación Web de mezcla musical, que simula una interfaz de mesa de mezclas. Está pensado para personas que se están iniciando en el mundo de la mezcla musical.

El proyecto se ha desarrollado utilizando el framework Angular, Typescript, HTML y CSS siguiendo una metodología ágil y aplicando los principios de SCRUM.

El software desarrollado se ha llamado Web Virtual DJ y se ha publicado con Licencia Publica General de GNU, versión 3 (GPLv3).

# Abstract

The purpose of this project is to develop a music mixing WebApp that mimics a physical mixing table. It is designed for amateurs in music mixing.

The project has been developed using the Angular framework, Typescript, HTML and CSS, following agile methodology and applying the SCRUM principles.

The developed software has been named Web Virtual DJ and has been published under the GNU General Public License, version 3 (GPLv3).

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción y objetivos

## 1.1. Introducción

En los últimos años ha aumentado la popularidad de la música electrónica. Con este aumento se ha visto la aparición de múltiples aplicaciones de DJ virtual. Este es el tipo de aplicación que se pretende desarrollar en este trabajo. La plataforma elegida es la Web pues garantiza portabilidad y accesibilidad. Por tanto, se pretende desarrollar la aplicación como una WebApp.

## 1.2. Motivación

La idea de este proyecto surgió mientras se intentaba buscar un reproductor más complejo que los presentes habitualmente en Spotify o Youtube. Tras una búsqueda rápida se vio que los que estaban disponibles requerían una compra y descarga, tenían funcionalidades muy pequeñas o directamente no funcionaban.

## 1.3. Estudio de las alternativas actuales

Se ha realizado una búsqueda más exhaustiva para conocer no sólo la existencia sino también los puntos fuertes y débiles de las aplicaciones similares. En los siguientes apartados se presenta el resultado de forma resumida resaltando los pros y contras de cada una de las herramientas revisadas[1].

[1]Se utilizan términos técnicos para los que se ha incluido una descripción en el Glosario (ver página 113)

### 1.3.1. Virtual DJ

Virtual DJ [3] es el competidor más completo y conocido.

Principales **pros**:

- Muchos niveles de personalización (pitch, volumen de graves, agudos y medios, efectos, loops...).
- Ayuda a la mezcla (automixer, indicador de beats per minute, ajustador de beats per minute).
- Conexión de hardware de DJ.
- Múltiples decks (2 a 99).
- Reproductor de vídeo.
- Karaoke.
- Retransmisión.
- Listas de reproducción.

Aún así, tiene **contras**:

- No permite la carga de música o vídeos desde servicios de streaming.
- Alto coste (19$ al mes) si se desea usar fuera del hogar o altas limitaciones en la versión gratuita.
- Descarga necesaria, no es una webapp.
- Disponibilidad sólo en Windows y Mac.

La interfaz de usuario de Virtual DJ se puede consultar en la Figura 1.1

### 1.3.2. You DJ

You DJ [4] es el competidor más parecido en concepto.

Los **pros**:

- Webapp: permite su uso en todos los sistemas operativos (no compatible con interfaz táctil).
- Carga desde webs de streaming o vídeo (con muchas limitaciones).


Figura 1.1: Interfaz de usuario de Virtual DJ [3]

- Grabación de la mezcla.
- Muchos niveles de personalización (pitch, volumen de graves, agudos y medios, efectos, loops...).
- Versión para escritorio.
- Listas de reproducción.
- Conexión a hardware MIDI.
- Ayuda a la mezcla (limitada).

Aún así, los **contras** pesan bastante más:

- No permite aplicar efectos sobre las canciones cargadas desde Youtube (al ser un vídeo).
- La carga de música desde Soundcloud no funciona.
- No permite cargar canciones locales excepto con la versión de escritorio.
- Los efectos son muy limitados por defecto y para desbloquear más requiere de un pago mensual. Al no funcionar sobre la única plataforma que carga música (Youtube) pierde su sentido.
- Para descargar la versión de escritorio se requiere estar suscrito.

La interfaz de usuario se puede consultar en la Figura 1.2, apreciamos que es bastante más básica que la de Virtual DJ.

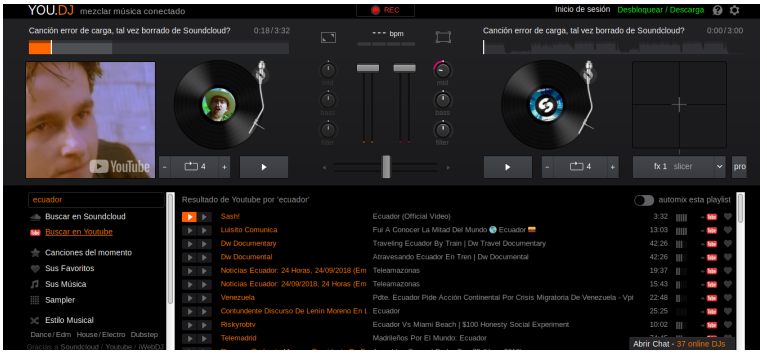


Figura 1.2: Interfaz de usuario de You.dj [4]

### 1.3.3. youtube-dj

Youtube-dj [5] es el último competidor evaluado y el más básico.

Entre sus **pros** se encuentran:

- Totalmente gratuita.
- Carga desde Youtube.
- Atajos de teclado para facilidad de uso.
- Webapp (no compatible con interfaz táctil).

Aunque tiene notorios inconvenientes.

**Contras**:

- Poca personalización (volumen).
- Sin versión de escritorio.
- Sin carga de canciones locales.

La interfaz de usuario se puede consultar en la Figura 1.3, es la más simple de las tres.

## 1.4. Objetivos

### 1.4.1. Objetivos de desarrollo

Vistas las características que ofrece la competencia se procede a elaborar una lista de características que debe tener la aplicación a desarrollar como parte de este TFG. Siendo


Figura 1.3: Interfaz de usuario de Youtube-dj [5]

el objetivo principal el desarrollo de una aplicación de DJ virtual, las características que no deben faltar son:

- Múltiples decks (2, 3 o 4).
- Nivel medio de personalización (volumen general y específico de graves, agudos y medios, loops, pitch, efectos).
- Carga de música local y desde plataformas musicales (en un principio Youtube).
- Webapp.
- Compatibilidad táctil al menos para resolución alta.

Otras características a considerar con prioridad baja en caso de poder ajustarse en el tiempo previsto para el desarrollo:

- Versión para escritorio.
- Efectos personalizables.
- Gestión de usuarios.
- Listas de reproducción.

Características deseables pero que por su complejidad no será posible implementar:

- Compatibilidad con hardware de DJ

### 1.4.2. Objetivos personales

Mis objetivos personales a la hora de empezar con este proyecto:

- Mejorar mi entendimiento de la planificación de un proyecto desde cero con metodologías ágiles (en el marco de SCRUM).
- Reforzar mis conocimientos de desarrollo *Frontend* con las tecnologías más actuales (Angular 6, Typescript, HTML 5 y CSS 3), así como aprender nuevas librerías para la representación y reproducción de audio.
- Familiarizarme con todos los pasos a seguir durante el desarrollo de un software (especificaciones del producto, diseño de interfaz de usuario, desarrollo, testing y despliegue).

## 1.5. Nicho de mercado y modelo de negocio

Habiendo estudiado los competidores se aprecia que los posibles nichos de mercado que la aplicación a desarrollar puede cubrir son:

- Bares, o negocios similares, que quieren reproducir música local a la vez que de plataformas musicales sin un coste elevado.
- Personas que quieren organizar una fiesta y pinchar música pero no quieren descargarse una suite completa de DJ en su ordenador.

Por tanto el modelo de negocio propuesto para este producto de software sería:

- Publicidad no intrusiva.
- Subscripción de bajo coste con la consiguiente eliminación de la publicidad.

## 1.6. Estructura de la memoria

En lo que sigue este documento se estructura de la siguiente forma:

- En el Capítulo 2 se presenta el proceso de desarrollo utilizado, así como la forma de aplicarlo a este proyecto. También se presenta la planificación y calendarización del proyecto.
- En el Capítulo 3 se presentan las distintas tecnologías usadas a lo largo de la planificación, gestión, desarrollo y testing del proyecto.
- En el Capítulo 4 se presenta el plan de riesgos y el cálculo del presupuesto.

- En el Capítulo 5 se presenta el seguimiento del proyecto a lo largo del tiempo.
- En el Capítulo 6 se explican las decisiones de diseño tomadas y se presentan los diagramas de componentes, de clases y la organización de código.
- En el Capítulo 7 se presentan los pasos seguidos durante la implementación, en orden cronológico.
- En el Capítulo 8 se explica el testing, tanto programático como con usuarios.
- Finalmente en el Capítulo 9 se presentan las conclusiones así como las líneas de trabajo futuras.

Para finalizar se añaden tres anexos:

- En el Anexo A se presentan las instrucciones para preparar el entorno de desarrollo
- En el Anexo B se presentan las instrucciones para desplegar la aplicación
- En el Anexo C se presentan los contenidos del CD-ROM

# Capítulo 2

# Proceso de desarrollo y planificación

## 2.1. Scrum

### 2.1.1. ¿Qué es Scrum?

Scrum [6] es un marco de trabajo simple que promueve la colaboración en los equipos para lograr desarrollar productos complejos. Es un proceso de gestión que reduce la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes. En sí Scrum no es una metodología, sino un marco para el desarrollo ágil y la gestión de proyectos.

Scrum se caracteriza [62] por:

- Aportar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- La calidad del resultado se basa principalmente en el conocimiento técnico de las personas en equipos auto organizados, antes que en la calidad de los procesos empleados.
- Solapamiento de las diferentes fases de desarrollo.
- Seguir los pasos del desarrollo ágil: Desde el concepto o visión general de la necesidad del cliente hasta la construcción del producto de forma incremental a través de iteraciones. Estas iteraciones (En Scrum se llaman sprints) se repiten de forma continua hasta que el cliente da por cerrada la evolución del producto.
- El tamaño del sprint en Scrum es fijo a lo largo de todo el desarrollo.

Los artefactos del marco Scrum son [11]:

- Pila del producto o *Product Backlog*: Lista de requisitos del usuario en forma de historias de usuario (*user stories*). Se crea a partir de la visión inicial pero puede ir evolucionando durante el desarrollo.
- Pila del sprint o *Sprint Backlog*: Lista de historias de usuario que debe realizar el equipo durante el sprint para generar el incremento previsto.
- Incremento: Parte del producto desarrollada en un *sprint* en condiciones de ser usada.

### 2.1.2. Eventos de Scrum

Los eventos de Scrum se utilizan para reducir la necesidad de reuniones no definidas en Scrum, así como establecer una cadencia que permita mejorar la comunicación y colaboración del equipo. Todos los eventos tienen una duración y una periodicidad marcadas. Los eventos de Scrum son [8, 9]:

- *Sprint* o Iteración: Intervalo de tiempo con duración fija, de entre 1 y 4 semanas, donde se desarrolla el incremento de un producto, potencialmente entregable. La periodicidad está dada por la duración del sprint y mientras dure el proyecto siempre hay un sprint activo.
- Reunión de planificación de Sprint o *Sprint Planning*: Reunión que se realiza una vez por sprint. La duración de esta reunión puede variar dependiendo del número de personas involucradas, siendo de entre dos y cuatro horas en condiciones normales (aunque puede llegar en condiciones especiales a durar una jornada de trabajo completa). Trata de responder a dos preguntas "¿Qué va a ser entregado en el incremento resultante del próximo Sprint?" y "¿Cómo se va a realizar el trabajo seleccionado?". Es decir, elegir qué elementos forman parte del siguiente Sprint Backlog.
- El Scrum diario o *Daily Scrum*: Es una reunión de 15-30 minutos cuyo objetivo es que el equipo de desarrollo sincronice actividades, y cree un plan para las próximas 24-48 horas. Se realiza cada día (o cada dos días en caso de no haber tanta disponibilidad).
- Revisión del Sprint o *Sprint Review*: Se lleva a cabo al final de cada sprint con el objetivo de inspeccionar el incremento y adaptar, si es necesario, el Product Backlog.
- Retrospectiva del sprint o Sprint Retrospective: Se lleva a cabo al final de cada sprint tras el Sprint Review y es la oportunidad para el Equipo Scrum de analizar qué ha salido bien y qué ha salido mal en el último sprint y crear un plan de mejoras para ejecutar durante el siguiente sprint.

### 2.1.3. Participantes de Scrum

Los participantes de Scrum o roles [10] son:

- *Facilitador o Scrum master*: Persona que lidera el equipo guiándolo para que cumpla las reglas y procesos del marco de trabajo, así como actúa de facilitador durante las reuniones para lograr que sean productivas. También ayuda al equipo a salir de situaciones de bloqueo y enseña al equipo a auto gestionarse [61].

- *Product owner*: Representante de los accionistas y/o clientes que usan el software. Se focaliza en la parte de negocio. Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias de usuario a incorporar en el Product Backlog y las reprioriza de forma regular.

- Equipo: Grupo de profesionales con los conocimientos técnicos necesarios para el desarrollo del producto. Desarrollan las historias de usuario elegidas para cada Sprint.

- Interesados: Resto de implicados. Sólo pueden asesorar y observar.

En la Figura 2.1 se muestra un resumen gráfico de todo lo explicado anteriormente.

### 2.1.4. Requisitos de Scrum

Para poder poner en práctica Scrum se deben cumplir ciertos requisitos [7]:

- Cultura de empresa: La cultura de la empresa proveedora del proyecto debe estar alineada con la filosofía de una gestión ágil de proyectos. Por ello debe fomentar el trabajo en equipo y la colaboración, los equipos auto-gestionados, la creatividad del equipo y la transparencia y mejora continua.

- Compromiso del cliente: Scrum exige al cliente una alta implicación y una dedicación regular. Debe disponer siempre de una visión de alto nivel del producto y debe tener reflejadas sus expectativas en forma de lista de requisitos priorizada (actualizada cada sprint).

- Compromiso de la dirección: se harán muy evidentes los obstáculos, ya existentes y por venir, que impiden el correcto desarrollo de los proyectos y será necesario tomar decisiones, realizar cambios organizativos, alinear personas y proporcionar recursos para hacer la transición.

- Compromiso del equipo: Scrum se basa en el compromiso conjunto y la colaboración entre los miembros del equipo. La transparencia entre todos es fundamental para poder inspeccionar la situación real del proyecto y así poder hacer las mejores adaptaciones que permitan conseguir el objetivo común.

- Relación entre proveedor y cliente: Debe estar basada en el principio ganar - ganar, en vez de estar basado en un contrato férreo de alcance, tiempo y coste. Para obtener los mejores resultados debe asumirse que habrá cambios con respecto al documento original y puede que algunos puntos pierdan su sentido. Por tanto debe existir transparencia en la ejecución del proyecto para facilitar esta relación.

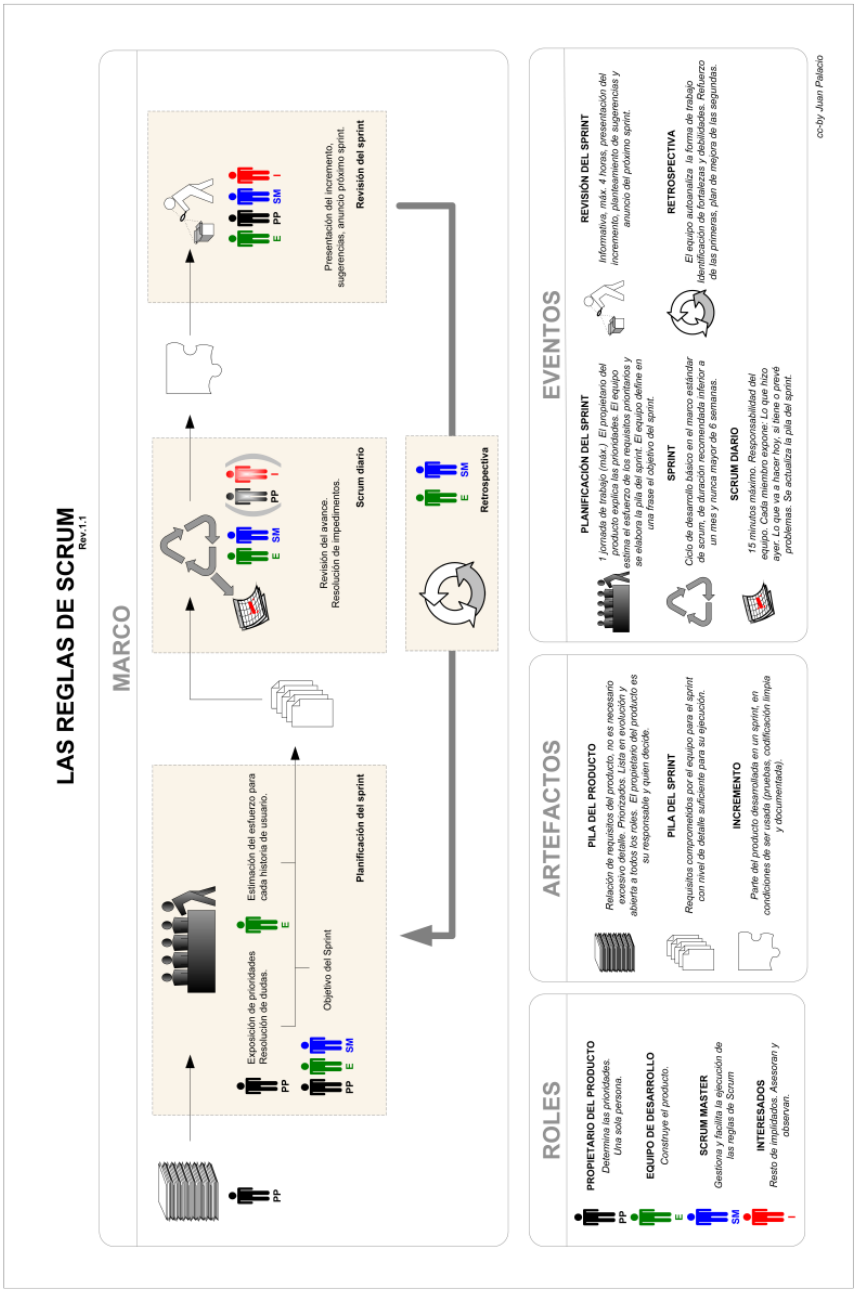


Figura 2.1: Resumen gráfico de Scrum [11]

- Facilidad para realizar cambios en el proyecto: Se debe poder incorporar requisitos de manera incremental en el proyecto sin un coste prohibitivo para el cliente.
- Tamaño del equipo: El tamaño ideal estará entre 5 y 9 personas. Es posible realizar Scrum con menos o más pero puede haber problemas derivados (con menos cualquier ausencia de una persona en el proyecto puede comprometer el desarrollo del sprint y con más la complejidad para comunicarse y colaborar aumenta).
- Localización geográfica: Es muy recomendable que todos los miembros del equipo compartan localización geográfica pues esto facilita la comunicación.
- Dedicación a tiempo completo: Es importante que los miembros del equipo se dediquen a tiempo completo para evitar dañar su productividad y facilitar la gestión de recursos humanos.
- Estabilidad del equipo: El equipo debe ser estable durante el proyecto, sus miembros deben cambiar lo mínimo posible, para poder aprovechar el esfuerzo que les ha costado construir sus relaciones interpersonales, engranarse y establecer su organización del trabajo.

## 2.2. Aplicación de Scrum en este proyecto

Para este proyecto se ha decidido aplicar Scrum. Es necesario realizar una adaptación de Scrum al contexto del proyecto. La forma de aplicarlo se describe a continuación.

Dada la limitación de personal en este proyecto, el alumno debe cumplir varios roles. Será el product owner y el equipo de desarrollo (gestiona las historias de usuario y las implementa). La tutora se encargará de ser Scrum master y de ser un agente interesado (guía al alumno para cumplir la metodología y asesora al alumno en otros aspectos).

Se ha establecido la duración del sprint en 2 semanas con "dailies" los viernes (por el tamaño del equipo y por disponibilidad) y uno de cada dos viernes marcará el final de un sprint y el inicio del siguiente. Por tanto, será este día en el que se realiza la revisión, retrospectiva del sprint finalizado y la planificación del siguiente sprint. La tecnología usada para la aplicación de Scrum es GitLab y será discutida en el próximo capítulo. El product backlog, sprint backlog e incremento de cada sprint serán discutidos en los capítulos dedicados a la planificación y seguimiento.

### 2.2.1. Planificación

El trabajo de fin de grado del Grado en Ingeniería Informática (mención en Ingeniería del Software) tiene una valoración en créditos total de 12 ECTS [1]. En vista de que cada crédito equivale a 25-30 horas de trabajo [2] este trabajo debería estar dimensionado en 300-360 horas. En este caso se tomará como referencia 25 horas pues es la cantidad que se usa en las asignaturas de la Escuela de Ingeniería Informática de Valladolid.

Se planificará un proyecto de un total de 300 horas de dedicación a este trabajo.

Habiendo decidido usar metodologías ágiles se divide la carga de trabajo en varias iteraciones o sprints. Tener iteraciones de aproximadamente 30 horas parece lo más adecuado. Por tanto, se tendrán un total de 10 sprints. En vista de que dispongo de unas 15 horas por semana lo mejor será que cada sprint dure 2 semanas. Esto resulta en una duración de la planificación de 20 semanas. Lo anterior se resume en la Tabla 2.1.

| Sprint | Fecha de comienzo | Fecha de fin | Observaciones |
|---|---|---|---|
| 1 | 06/10/2018 | 19/10/2018 | |
| 2 | 20/10/2018 | 02/11/2018 | |
| 3 | 03/11/2018 | 16/11/2018 | |
| 4 | 17/11/2018 | 30/11/2018 | |
| 5 | 01/12/2018 | 14/12/2018 | 14-21 fuera, añadida semana extra |
| 6 | 22/12/2018 | 04/01/2019 | |
| 7 | 05/01/2019 | 18/01/2019 | |
| 8 | 19/01/2019 | 01/02/2019 | |
| 9 | 02/02/2019 | 15/02/2019 | |
| 10 | 16/02/2019 | 01/03/2019 | |

Tabla 2.1: Planificación inicial por fechas

Como se puede apreciar en la Tabla 2.1 la fecha estimada de finalización sería el 01/03/2019. Durante la semana del 14 al 21 de diciembre me será imposible trabajar. Por tanto, el siguiente sprint empieza una semana después.

### 2.2.2. Product Backlog o User stories

Las historias de usuario que se han especificado son las que se muestran en la Tabla 2.2.

### 2.2.3. Tareas

Las tareas que se han realizado en cada sprint se ven en el capítulo de seguimiento (Capítulo 5).

| Número | Título | Descripción |
|---|---|---|
| 001 | Interfaz gráfica de usuario | Como usuario quiero ver una interfaz gráfica de usuario de DJ virtual |
| 002 | Cargar música en local | Como usuario quiero poder cargar mi música desde mi ordenador a mi navegador |
| 003 | Reproducir música | Como usuario quiero poder reproducir y pausar música |
| 004 | Modificar el volumen de la música | Como usuario quiero poder modular el volumen de la música que estoy reproduciendo. |
| 005 | Modificar volumen de distintas frecuencias | Como usuario quiero poder modificar el volumen de distintas frecuencias (agudos, medios y graves) |
| 006 | Más de un deck | Como usuario quiero poder tener más de una canción cargada y lista para reproducir a la vez |
| 007 | Cargar música desde servicios de internet | Como usuario quiero poder cargar música desde servicios de internet |
| 008 | Cambio de pitch | Como usuario quiero poder cambiar el pitch de la música que está sonando |
| 009 | Agregar efectos | Como usuario quiero poder reproducir efectos mientras suena la música |
| 010 | Loops | Como usuario quiero poder hacer loops con la música |
| 011 | CUE | Como usuario quiero poder cargar CUEs |
| 012 | Mostrar onda | Como usuario quiero poder ver la onda de la música que estoy escuchando |

Tabla 2.2: User stories

# Capítulo 3

# Tecnologías utilizadas

## 3.1. GitLab

GitLab [15] es una aplicación cuyo objetivo es ser la única aplicación que debes usar durante el ciclo completo de desarrollo y despliegue. Para conseguir esto, GitLab integra herramientas de planificación, creación, integración continua, empaquetado, despliegue, configuración, monitorización y seguridad.

### 3.1.1. Planificación

Gitlab integra herramientas [16] de planificación como un tablero Kanban y tablero de *issues*, control del tiempo usado, gestor de *issues*, gestión de hitos, *burndown charts*, fechas de entregas para las *issues*, etiquetas en las *issues*, tableros en función de etiquetas, etc...

Disponer de estas herramientas es más que suficiente para el desarrollo que se quiere realizar. El gestor de *issues* permite crear las historias de usuario y las tareas y marcarlas con etiquetas. Los tableros son muy útiles para el seguimiento de las *user stories* y tareas desde su creación hasta su finalización. Finalmente la gestión de hitos es perfecta para simbolizar los sprints. Con todas estas herramientas puedo realizar toda la gestión del proyecto, desde su planificación.

### 3.1.2. Control de versiones

Gitlab permite [17] crear repositorios privados gratis sin limitación de miembros. Ofrece gráficos de commits y herramientas de reporte, documentación del proyecto basado en Wiki o alojamiento de páginas estáticas (para crear tu propia documentación estática), merge

requests, discusión en los merge requests, mergeo rápido, squash y merge, resolución de conflictos, gestión de ramas, ramas protegidas, IDE web, snippets, cherry-pick, tags protegidos, etc...

De las características disponibles para este proyecto basta con unas pocas. Las más interesantes son la posibilidad de alojar páginas estáticas, el uso de ramas y merge requests, necesarios para la aplicación de GitFlow (ver apartado 3.1.6). Otras no pueden ser aprovechadas al estar pensadas para grupos más grandes de desarrollo.

### 3.1.3. Integración continua y entrega continua

GitLab ofrece [18] escalado horizontal de trabajadores de integración continua (CI) y entrega continua (CD), debugueo de los contenedores con una terminal integrada, control de calidad de código, gráficos de pipelines, path personalizable para los ficheros de CI y CD y todo con una configuración mínima necesaria.

En caso de preparar el proyecto para CI y CD estas herramientas son muy útiles.

### 3.1.4. Otras herramientas

Las demás herramientas que ofrece, aunque interesantes, se salen del alcance de este proyecto.

### 3.1.5. Resumen y aplicación

En resumen, GitLab facilita la planificación y gestión del código fuente y será usado como `origin` de nuestro repositorio.

### 3.1.6. GitFlow

Para las labores de control de versiones se ha usado GitFlow:

GitFlow es un marco para la gestión de repositorios Git. Trata de evitar que cada desarrollador cree ramas con nombres dispares, poco explicativos, que salen de unas ramas y se juntan con otras. Tener un repositorio así supone un coste de organización cada vez mayor, hasta el punto de ser imposible de gestionar. Por eso Vincent Driessen [19] ideó una forma de gestionar un repositorio, con control de qué ramas permiten ramificación, cuáles no, y qué nombre se debe dar a cada una (o al menos qué prefijo debe tener cada una). Con el tiempo a este marco de trabajo se le acabó conociendo como GitFlow.

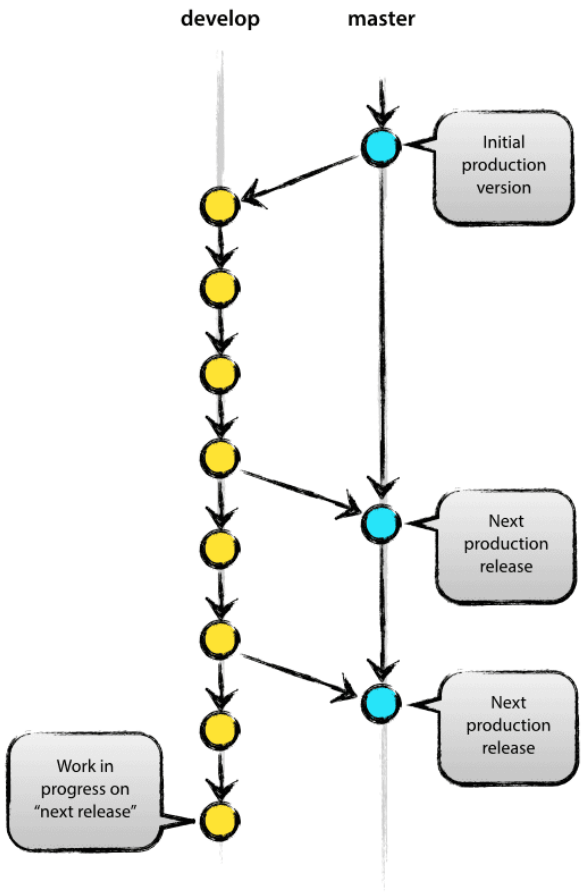


Figura 3.1: Ramas principales [19]

### Ramas principales

Las dos ramas principales (Figura 3.1) tienen un tiempo de vida ilimitado y son `master` y `develop`:

- `origin/master`[1]: es una rama bastante común en todos los repositorios git. En GitFlow representa una rama con código listo para despliegue en producción.
- `origin/develop`: representa una rama totalmente funcional con los últimos cambios realizados en el desarrollo. Se le puede considerar una rama de integración. Las versiones nightly automáticas se construyen en base a esta rama.

[1]Las ramas precedidas de `origin` representan ramas que están en el repositorio remoto (el repositorio de GitLab en nuestro caso).

Cuando hay una versión en `develop` lista para ser lanzada se mergea `develop` en `master` y se crea una nueva rama `release`.

### Ramas de soporte

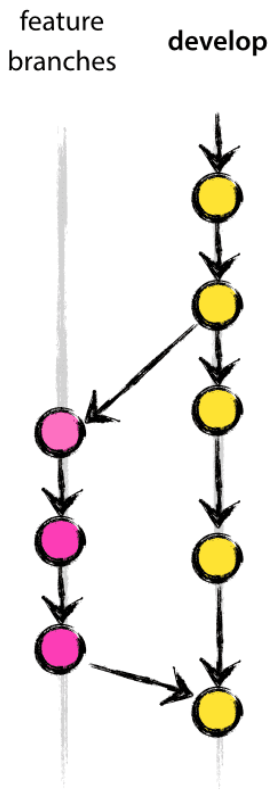


Figura 3.2: Ramas `feature` [19]

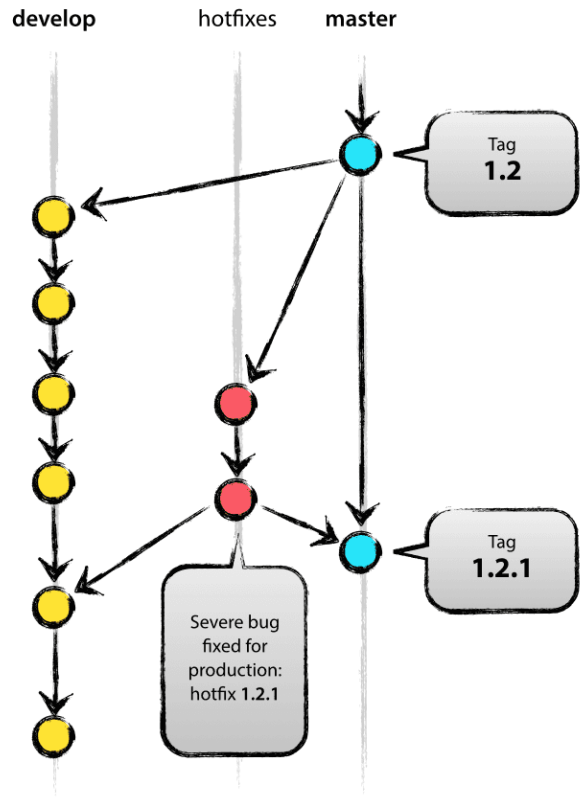


Figura 3.3: Ramas `hotfix` [19]

Las ramas de soporte se usan para ayudar al desarrollo paralelo de los distintos miembros. Estas ramas tienen un tiempo de vida limitado, es decir, tarde o temprano serán eliminadas. Los tres tipos de ramas son: `feature`, `release` y `hotfix`. Cada una de estas ramas tienen un propósito y unas estrictas reglas que deben seguir en cuanto rama de origen y destino. Por supuesto son ramas normales, desde el punto de vista de Git no tienen nada de especial.

- `feature` (Figura 3.2): Se puede originar en `develop` y debe tener como destino `develop`. La convención para darle nombre es cualquier cosa menos `master`, `develop`, `release*` o `hotfix*`, aunque es buena practica nombrarla `feat*` o `feature*`, para dejar claro que es una rama `feature`. Se usan para desarrollar nuevas funcionalidades. Su tiempo de vida es hasta que la nueva funcionalidad se incorpora a `develop` o hasta que se

descarta. Por lo general solo existen en el repositorio de desarrollador y no en `origin` hasta que hay que juntar las ramas.

- `release`: Se puede originar en `develop` y debe tener como destino `develop` y `master`. La convención para darle nombre es `release*` donde el asterisco se sustituye por la versión que se va a lanzar. Desde que se crea hasta que se junta con otras ramas se le pueden aplicar correcciones de bugs, pero no nuevas funcionalidades. Al realizar el lanzamiento se deben reflejar los cambios en `master` pero también en `develop`, por si hubiera bugs corregidos.

- `hotfix` (Figura 3.3): Se puede originar en `master` y debe tener como destino `master` y `develop`. La convención para darle nombre es `hotfix*`. Aunque parecidas a las ramas `release` en el sentido de que están pensadas para preparar una nueva versión de producción solo se usan para corregir bugs localizados en `master`. Por supuesto si ese bug también afecta a `develop` también se deben reflejar los cambios en esta rama.

#### Resumen

En resumen, GitFlow facilita la gestión del repositorio. Aunque para algunos casos más complejos es posible modificar o añadir nuevos tipos de ramas.

Dado que voy a usar un gestor de repositorios con soporte para Git, se puede aplicar esta metodología. Para esto no se necesita ninguna tecnología a mayores, pues solo especifica cómo gestionar el repositorio, pero éste sigue siendo un repositorio Git normal o `origin` en GitLab.

## 3.2. HTML5

HTML5 (HyperText Markup Language, versión 5) [21] es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. Tiene algunas características nuevas pero para el uso general es HTML. HTML [22] es un lenguaje de marcado que se usa para construir las páginas web. Define una estructura básica y un código para la definición de contenido de una página web.

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), éste no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

### 3.2.1. Marcado HTML

HTML se escribe en forma de etiquetas rodeadas por corchetes angulares (<,>,/). Describe la estructura del documento, y hasta cierto punto la apariencia, aunque con CSS y Javascript se puede modificar esa apariencia.

Como en XML en HTML tenemos elementos, y estos tienen atributos y contenido.

El elemento y sus atributos se indican en la etiqueta. Hay dos tipos de elementos:

- De marcado estructural: por ejemplo <h2>Titulo<h2/> establece "Titulo" como encabezado de segundo nivel.
- De marcado presentacional: por ejemplo <b>negrita<b/> indica que los navegadores web visuales deben mostrar el texto en negrita (bold).

Los atributos suelen ser pares nombre-valor, separados por un signo igual y escritos en la etiqueta de comienzo de un elemento, después del nombre del elemento. Hay veces que solo tener el nombre del atributo es suficiente y da igual que valor se le asigne pues solo con su presencia ya afecta al elemento.

El contenido es cualquier texto o contenido marcado que esté dentro de las etiquetas.

### 3.2.2. Resumen y aplicación

Si presentamos una analogía entre una página web y un ser humano, simplificada por supuesto, si un ser humano tiene un esqueleto, unos músculos y la piel, se podría decir que HTML es como el esqueleto de una página web; marca la estructura de la página web. Se usará por tanto para marcar la estructura de las vistas que se diseñen.

## 3.3. CSS3

CSS3 es la ultima revisión de CSS o Cascading Style Sheets [23]. CSS es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML.

CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de éste, características tales como las capas o layouts, los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características presentacionales, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento.

La especificación CSS describe un esquema prioritario para determinar qué reglas de estilo se aplican si más de una regla coincide para un elemento en particular. Estas reglas son aplicadas con un sistema llamado de cascada, de modo que las prioridades son calculadas y asignadas a las reglas, así que los resultados son predecibles.

CSS tiene una sintaxis simple, y usa un conjunto de palabras clave en inglés para especificar los nombres de varias propiedades de estilo.

Una hoja de estilos consiste en una serie de reglas. Cada regla, o conjunto de reglas consisten en uno o más selectores, y un bloque de declaración.

El CSS se puede aplicar al HTML directamente o con hojas de estilos, siendo esto lo recomendado al ser posible cambiar los estilos sin necesidad de modificar el código fuente de la página.

### 3.3.1. Selectores

Los selectores identifican a qué etiquetas se aplican los estilos señalados en la regla.

Los selectores pueden aplicarse a:

- Todos los elementos de un tipo, como los párrafos <p>. Se representa en el CSS por el elemento seguido de los estilos a aplicar:

```
p{
    color: #fff;
    font-size: 16px;
}
```

- Elementos seguidos de un atributo, en particular:
  - id: identificador, un identificador único para la etiqueta. Se representa en HTML con un atributo id:

    ```
    <label id="title">...</label>
    ```

    Se representa en el CSS por el id precedido por #:

    ```
    #title{
        color: #000;
        font-size: 20px;
    }
    ```

  - class: clase, un identificador para anotar múltiples elementos. Se representa en HTML con un atributo class:

    ```
    <label class="title">...</label>
    ```

Se representa en el CSS por el id precedido por un punto:

```
.title{
    color: #000;
    font-size: 20px;
}
```

### 3.3.2. Orden de importancia

Como se pueden aplicar distintos estilos de distinta forma hay un esquema de prioridades. De mayor a menor prioridad este esquema quedaría como se muestra en la tabla 3.1:

| Prioridad | Tipo de origen de CSS | Descripción |
|---|---|---|
| 1 | Importancia | La anotación `!important` sobreescribe la prioridad anterior |
| 2 | Inline | Un estilo que se aplica directamente sobre el HTML, por medio del atributo `style` |
| 3 | Media Type (tipo de dispositivo) | Un estilo que se aplica solo en caso de tener un tipo de dispositivo especifico (móvil,tablet, sobremesa, etc...) |
| 4 | Definido por el usuario | La mayoría de los navegadores tienen esta característica de accesibilidad: un estilo CSS definido por el usuario |
| 5 | Especificidad del selector | Un selector contextual específico (`#heading p`) sobreescribe una definición general (`p`) |
| 6 | Orden de las reglas | La última regla especificada tiene una mayor prioridad |
| 7 | Herencia | Si una propiedad no está especificada, es heredada del elemento padre |
| 8 | Definición de propiedad CSS en el documento HTML | Una regla CSS común (selector `*`) sobreescribe el valor del navegador |
| 9 | Predeterminado del navegador | La prioridad más baja: estos valores son determinados por las especificaciones iniciales de la W3C |

Tabla 3.1: Esquemas de prioridad CSS [23]

### 3.3.3. Resumen y aplicación

Siguiendo con la analogía presentada en HTML, si éste era el esqueleto, CSS sería la piel. Es la parte que se puede apreciar. Por tanto se usará en conjunto con HTML para diseñar

cómo está estructurada y cómo se ven las vistas que se diseñen.

## 3.4. Javascript

JavaScript (JS) [24] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Está orientado a objetos y basado en prototipos, es imperativo, débilmente tipado y dinámico.

Se utiliza principalmente para desarrollar aplicaciones de cliente (client-side); implementado como parte de un navegador web permite mejoras en la interfaz de usuario y páginas web dinámicas. También existe una forma de JavaScript del lado del servidor.

JavaScript tiene una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.

### 3.4.1. Características

Las siguientes características son comunes a todas las implementaciones que se ajustan al estándar ECMAScript:

- Imperativo y estructurado: JavaScript es compatible con gran parte de la estructura de programación de C. Una salvedad a destacar: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas, sin embargo en las implementaciones originales de JavaScript cuando se declaraba una variable como `var` el ámbito de las variables era el de la función en la cual eran declaradas. Esto ha cambiado con la versión de ECMAScript 2015, ya que añade compatibilidad con block scoping por medio de la palabra clave `let`.
- De tipado dinámico: Como en la mayoría de lenguajes de scripting, el tipo esta asociado al valor, no a la variable.
- Objetual: JavaScript está formado casi en su totalidad por objetos. Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos
- Con evaluación en tiempo de ejecución: Aunque desaconsejado, se pueden evaluar expresiones de tipo cadena en tiempo de ejecución. Esto es ineficiente e inseguro.
- Con funciones de primera clase: Es decir, las funciones se pueden usar como variables y pueden ser pasadas a otras funciones.
- Prototípico: Se usan prototipos y no clases para el uso de herencia. Se puede simular el comportamiento de clases con éstos.
- Otras: Funciones variádicas, funciones como métodos, arrays y objetos (arrays asociativos en otros idiomas) pueden ser creados con una sintaxis abreviada, expresiones regulares similares a Perl.

### 3.4.2. Resumen y aplicación

Para finalizar la analogía, ya teníamos un esqueleto y una piel, pero nos faltaban los músculos. Javascript es el musculo de la página web. Es Javascript quien permite a la página moverse, por así decirlo. Permite al usuario interactuar con la página y con otros servicios que no están presentes en el navegador. Por tanto usaremos Javascript para crear el modelo y los controladores necesarios para tener una página funcional.

Aun así para facilitarnos las cosas no usaremos Javascript directamente. Usaremos un framework y un superset de Javascript. Estos permiten añadir funcionalidad a Javascript a la par que facilitan las tareas de programación y debugueo.

## 3.5. TypeScript

El framework elegido es Angular, que permite el uso de JavaScript o TypeScript [25].

TypeScript es un superset sintáctico estricto de JavaScript que añade tipado estático. Está diseñado para grandes aplicaciones y traspila a JavaScript. Un programa JavaScript es un programa TypeScript válido.

TypeScript soporta los ficheros de definición de estructuras de datos y sus tipos.

Las principales características que añade a JavaScript son:

- Anotaciones de tipo y comprobación en tiempo de compilación
- Inferencia de tipos
- Interfaces
- Tipos enumerados
- Genéricos
- Espacios de nombres
- Tuplas

### 3.5.1. Resumen y aplicación

Typescript facilita el trabajo al programador al añadir un tipado estático, el cual permite detectar errores en tiempo de compilación, haciendo mas escalable el desarrollo de aplicaciones que traten con diferentes tipos de datos y favoreciendo la mantenibilidad.

Ya que el framework elegido soporta Typescript y que éste aporta notorios beneficios, se usará en este proyecto.

## 3.6. Angular

Angular [26] es un framework ideado para la creación de aplicaciones web con HTML, CSS y TypeScript. Angular mejora estas tecnologías y facilita su integración. Los elementos básicos presentes en Angular son los módulos, componentes y servicios:

- Módulos: Un módulo es una clase precedida por el decorador `@NgModule`. Los módulos permiten agrupar componentes, servicios y otros elementos de funcionalidades similares. Al crear un módulo, más tarde se puede importar este módulo en otro y usarlo indistintamente, lo que permite seguir los principios de bajo acoplamiento y alta cohesión. Se puede elegir que los módulos se carguen con lazy-loading.
- Componentes: Un componente es una clase precedida por el decorador `@Component`. A esta clase se asocia un template HTML y una hoja de estilos. La funcionalidad de la clase TypeScript mejora el template para dotarlo de dinamismo mediante data-binding de eventos y data-binding de propiedades.
- Servicios (e inyección de dependencias): Para la carga de datos y la lógica que se comparte entre distintos componentes, se crea una clase de tipo servicio. Se le precede del decorador `@Injectable`. Esto indica que puede ser inyectado en un componente como una dependencia, permitiendo que los componentes se mantengan ligeros al delegar tareas pesadas en los servicios.

Otros elementos más avanzados son:

- Routing: Existe un tipo especial de módulos conocidos como módulos de routing. Son usados para cargar módulos en función de la ruta a la que se accede desde el navegador y son los que permiten el lazy-loading.
- Directivas: aplican una transformación sobre un elemento del template, por ejemplo mostrándolo solo si se cumple una condición, o creando un elemento por cada elemento de un array.
- Tuberías: realizan una transformación sobre una variable TypeScript pero sólo en el template HTML, por ejemplo si se aplica la tubería JSON a un objeto, sería lo mismo que aplicarle la función `JSON.stringify`.

La relación entre los elementos básicos de Angular se puede consultar en la Figura 3.4.

### 3.6.1. Módulos

Como se ha explicado, los módulos son clases precedidas del decorador `@NgModule`. Este decorador acepta un objeto de meta-datos. Los meta-datos pueden ser:

- `declarations`: qué componentes, directivas y tuberías pertenecen a este módulo.

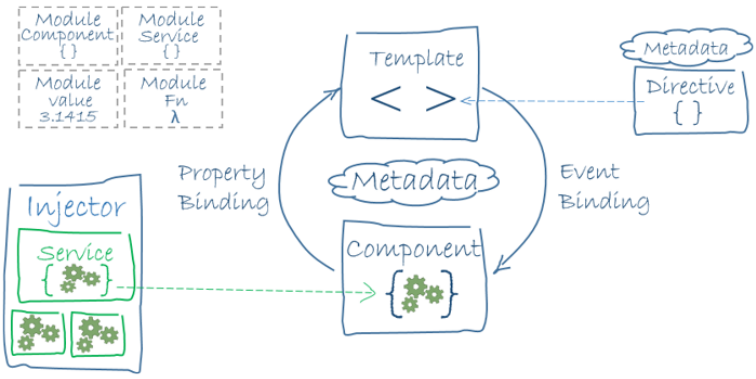


Figura 3.4: Relación entre los elementos básicos de Angular [30]

- `exports`: el conjunto de declaraciones que serán visibles y usables cuando se importe este módulo en otros módulos.
- `imports`: módulos de los cuales se necesita importar clases.
- `providers`: servicios que usa este módulo, o que son inyectados en este módulo.
- `bootstrap`: la vista principal de la aplicación. Solo el módulo raíz debe tener esta propiedad

Pese a que el gestor de módulos de Angular es diferente y no tiene relación con el gestor de módulos de JavaScript, ambos se pueden usar de forma complementaria para cargar módulos Angular y módulos JavaScript de forma conjunta en la misma aplicación.

### 3.6.2. Componentes

Los componentes son clases que acompañadas de el template HTML y una hoja de estilos propia (opcional) conforman la vista y el controlador asociado. A este conjunto en Angular se le llama vista. Las clases componente van siempre precedidas del decorador `@Component`. Este decorador acepta un objeto de meta-datos. Estos meta-datos pueden ser:

- `selector`: Un selector CSS que indica a Angular que debe insertar ese componente cuando encuentre esa etiqueta en un template HTML. Por ejemplo en un componente:

```
@Component{
...
selector: 'app-hero-list',
...
}
```

Y en un template se usaría:

```
<app-hero-list></app-hero-list>
```

- `templateUrl` o `template`: la ruta relativa del template HTML o el template inline (no recomendado).
- `providers`: en caso de necesitar acceder a algún servicio se añade aquí.
- `styleUrls` o `style`: de forma opcional se le puede pasar la ruta relativa a una hoja de estilos que únicamente se aplicará al template de este componente. O pasarle los estilos directamente (no recomendado).

El template HTML es como un archivo HTML normal pero con algunos añadidos. En este template se pueden usar directivas y tuberías y se puede usar binding de datos.

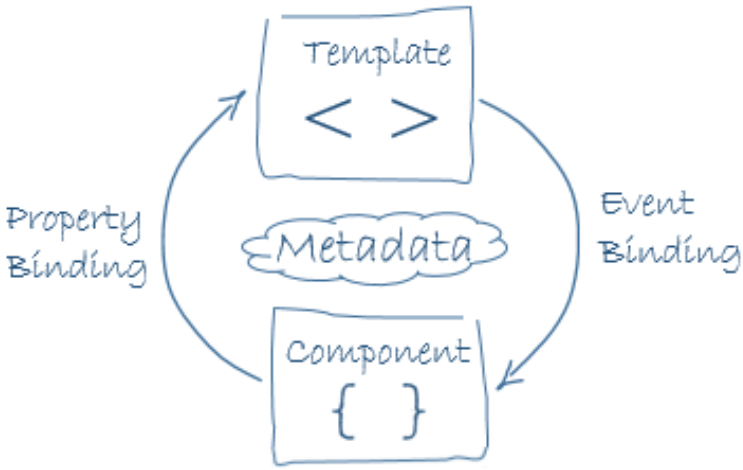


Figura 3.5: Binding de datos de los componentes [31]

Hay cuatro tipos de binding (Figura 3.5 y Figura 3.6)y pueden ser unidireccionales o bidireccionales:

- Binding con llaves: Este es el tipo mas básico de binding. Su función es mostrar una variable del componente en el template. Por ejemplo:

  En el componente:

  ```
  numero = 5;
  ```

  En el template:

  ```
  El numero guardado es: {{numero}}
  ```

  El resultado:

```
El numero guardado es: 5
```

Por supuesto una modificación de la variable en el componente actualizaría el valor en el template, resultando en la muestra dinámica de datos.

- Binding con atributos: para tener un atributo dinámico en Angular se podría hacer mediante binding con llaves pero esto puede dar problemas de rendimiento. La forma correcta de hacerlo es con un binding de atributos. Por ejemplo:

```
<input type="text" [id]="id_dinamico"/>
```

Esto aplicaría la string de la variable id_dinamico como atributo id al input.

- Binding de eventos: permite asociar eventos típicos de JavaScript a funciones en la clase. Por ejemplo:

```
<button (click)="clicked()">Pulsame</button>
```

Esto asociaría el evento de hacer click sobre el botón con la función clicked de nuestro componente.

- Binding con ngModel: Hasta ahora todos los bindings eran unidireccionales. ngModel en cambio es bidireccional. Asocia una variable a un campo dinámico del template. En caso de que el campo cambie en el template se actualiza el valor de la variable, pero también al contrario, si se actualiza la variable el campo varía. Por ejemplo:

```
<input type="text" [(ngModel)]="campo_de_texto"/>
```

Si ahora se escribiera algo en el campo de texto la variable en el componente se modificaría, y si de manera programática se modificara la variable el campo de texto se actualizaría con este valor.

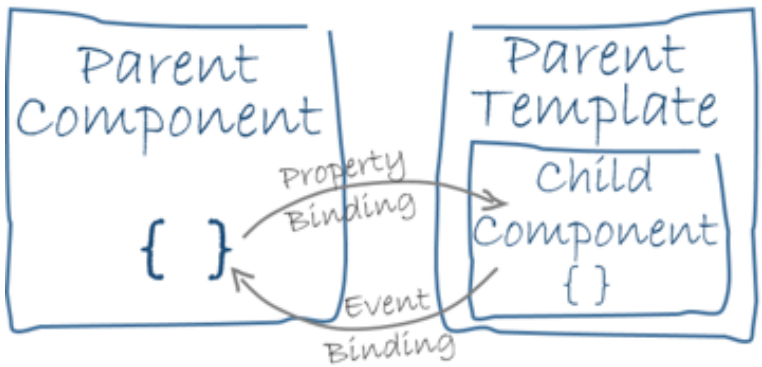


Figura 3.6: Binding de datos entre componentes [31]

Esto aporta dinamismo a nuestra página de forma fácil. También se puede usar el mismo principio para pasar datos entre componentes padres e hijos.

### 3.6.3. Tuberías

Las tuberías se usan en el template de Angular para aplicar transformaciones a las variables que se quieren mostrar. Por ejemplo, si se quiere mostrar un objeto en el template los primero a intentar sería:

```
{{myObject}}
```

Pero al ir a ver que ha mostrado la página web se vería:

```
[object Object]
```

Esto es porque se está mostrando una referencia al objeto. Se podría transformar este objeto en una string y mostrar el objeto transformado, pero Angular facilita las cosas con una tubería por defecto, la tubería sería `json`:

```
{{myObject | json}}
```

Y ahora se mostraría correctamente:

```
{ nombre: 'Carlos', apellido: 'Lobo' }
```

Pero ésta sólo es una de las muchas tuberías disponibles. Otras tuberías permiten por ejemplo transformación de palabras de singular a plural, visualización de precios en monedas distintas o transformación de texto a mayúsculas. Incluso se pueden crear tuberías propias. Todo esto las transforma en un elemento bastante potente dentro del template.

### 3.6.4. Directivas

Las directivas son el último elemento usado dentro del template. Su uso varía dependiendo del tipo. Hay dos tipos, directivas estructurales y directivas de atributo.

- Estructurales: son usadas para añadir, eliminar o cambiar elementos del DOM[2]. Por ejemplo:

  ```
  <img src="pepito.jpg" *ngIf="mostrar">
  ```

  Añadir esta directiva a la imagen conseguirá que solo se muestre la imagen si la variable `mostrar` evalúa a `true`. Pero si se cambia el valor también se cambia el estado de visualización. Esto permite mostrar y ocultar elementos bajo demanda.

  Otras directivas clásicas de este tipo son `*ngFor` que permite mostrar un elemento tantas veces como elementos haya en un array. o `*ngSwitch` que muestra un elemento u otro dependiendo del valor de una variable.

[2]Interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos XML. Es la representación del template que ven los navegadores tras ser cargado. Modificarlo implica cambiar cómo se ve la página en el navegador.

- De atributo: Estas alteran la apariencia o comportamiento de un elemento existente. El ejemplo mas simple es la directiva `ngModel` explicada mas arriba.

  Otro ejemplo seria `ngClass`. Esta directiva permite aplicar clases de forma dinámica, por ejemplo:

```
<p [ngClass]="{'style': active}">Texto</p>
```

  Este ejemplo mostraría un párrafo al que se le aplica la clase `style` si la variable `active` evalúa a true.

### 3.6.5. Servicios

Un servicio es un valor, función o característica que necesita nuestra aplicación. Habitualmente es una clase con un propósito muy específico. Los servicios son usados para encapsular la lógica necesaria en los componentes, pues éstos sólo deben usarse para la parte visual de nuestra aplicación. Los componentes delegan en los servicios tareas como la persistencia de datos, validación de datos de usuario o logs de errores. Los servicios pueden hacer uso de la inyección de dependencias para evitar sobrecargar la aplicación.

### 3.6.6. Inyección de dependencias

La inyección de dependencias es una característica poco visual pero muy importante en Angular. La inyección de dependencias se encarga de controlar que no más de una instancia de una clase sea cargada mientras no sea necesario. Por ejemplo, si dos componentes acceden a los mismos servicios y no hubiera control, se podrían crear dos instancias de un mismo servicio. Angular se encarga, mediante la inyección de dependencias, de implementar el patrón Singleton, es decir, siempre que un componente le pide acceso a un servicio, Angular le da la referencia de la instancia que ya se había creado de este servicio.

Para esto hace uso del inyector y de los proveedores:

- Los proveedores son los servicios que se proveen de datos. Para poder usarse en un componente deben estar especificados en su decorador o en el decorador de su módulo.

```
@Component{
...
providers: [
    BackendService,
    Logger
],
...
}
```

- El inyector se encarga de inyectar los servicios. Para ello requiere que toda clase de tipo servicio sea precedida de un decorador `@Injectable`. Así será identificada como inyectable y podrá ser inyectada.

```
@Injectable({
    providedIn: 'root',
})
```

### 3.6.7. Compilación antes de tiempo

Ahead of time compiling [40] o compilación antes de tiempo es uno de los dos tipos de compilación que se puede realizar en Angular (siendo el otro tipo just in time compiling o compilación justo a tiempo). La compilación antes de tiempo permite preparar los recursos de nuestra aplicación para no necesitar de un servidor para ser totalmente interactivos con el usuario. Al elegir construir una aplicación con compilación antes de tiempo se consiguen múltiples ventajas:

- Renderizado más rápido: Ni el navegador ni el servidor deben compilar la aplicación para poder usarla. Ya estará listo para usar tras descargarse del servidor.
- Menos peticiones asíncronas: El compilador junta HTML, CSS y JS en un solo archivo, así que solo se requiere de esa petición para que funcione.
- Menor tamaño de descarga: No se debe descargar el compilador para usar en el navegador.
- Detección temprana de errores: Algunos errores del template se detectan durante la compilación.

## 3.7. GitLab Runner

GitLab Runner [59] es un proyecto Open Source de GitLab que permite ejecutar trabajos de integración continua en remoto o en local. Esto permite probar los cambios sin tener que esperar a que GitLab te asigne un worker. Por esto se ha usado para probar cambios en el pipeline de CI/CD en local.

## 3.8. Paquetes npm

En esta sección se describen algunos paquetes que pueden facilitar las tareas de programación y testing en este proyecto.

### 3.8.1. Jest

Jest [33] es una plataforma de testing para Javascript. Requiere una configuración mínima para su uso y eso facilita bastante la integración con GitLab y otras tecnologías.

Pese a que Angular viene por defecto con la plataforma de testing Karma he elegido Jest por su simplicidad a la hora de hacer integración continua y por su velocidad, dado que Karma crea una instancia de un navegador mientras que Jest usa un DOM virtual para emular el comportamiento de un navegador (jsdom). Esto tiene como consecuencia que el entorno de integración no necesite un navegador instalado y se ahorra tiempo.

### 3.8.2. karma

Karma [55] es una biblioteca usada para el testing unitario, como se mencionó en el apartado anterior. Tras trabajar con los test unitarios con Jest se descubrió que había problemas a la hora de testear cuando la aplicación web utiliza Canvas o WebAudio, ambas APIs nuevas de HTML 5. Esto resultó en la imposibilidad de realizar tests unitarios con Jest al carecer jsdom de estas APIs. Por tanto, finalmente se decidió usar karma con Firefox para el testing unitario.

Karma funciona sobre navegadores reales en modo headless, así permite probar todas las funcionalidades como si se estuviera navegando de forma normal. Viene integrado por defecto en Angular por tanto es muy fácil hacerlo funcionar. Al igual que Jest es compatible con integración continua.

### 3.8.3. Prettier

Prettier [34] es, como sus propios autores indican, un “formateador de código cabezota”. Formatea el código en función de las reglas que se le especifiquen. Esto facilita producir código legible. Se usa en conjunción con husky para automatizar el formateo del código.

### 3.8.4. husky

husky [35] es una utilidad que permite definir precommit hooks y prepush hooks. Estos son comandos que se ejecutan antes de realizar un commit o un push para que si hay algún error no se permita que se realice la acción.

### 3.8.5. surge

surge [36] es un servicio de publicación de páginas estáticas gratuito. Esto nos permite publicar de forma fácil y rápida la WebApp, con un comando de la terminal, lo que lo hace perfecto para el despliegue continuo.

### 3.8.6. CompoDoc

CompoDoc [37] es, en palabras de sus autores, "la herramienta de documentación que le falta a Angular". Es una herramienta que permite generar documentación parecida a JavaDoc. Esto facilita el uso de nuestra aplicación a otros desarrolladores futuros, así como a cualquier contribuidor que quiera mejorarla más adelante. Genera documentación estática que puede ser servida fácilmente en cualquier página de servidor estático, o incluso en GitLab.

### 3.8.7. Wavesurfer

Durante el desarrollo se investigaron varias tecnologías que permitieran el tratamiento de archivos de audio en una WebApp. El tratamiento se hace con la Web Audio API [41]. Esta provee un sistema poderoso y versátil para controlar audio en la Web. Aun así, se buscaba una interfaz entre esta API y nuestra aplicación que facilitara el desarrollo, para no tener que desarrollar una interfaz propia. Entre las opciones más interesantes estaban Wavesurfer [42] y howler.js [43]. La opción elegida finalmente ha sido Wavesurfer.

Se eligió Wavesurfer pues a simple vista parecía bastante sencillo, al tener casi todas las funciones necesarias para la reproducción de audio como métodos y al permitir el uso de filtros de la Web Audio API fácilmente. También incluye la visualización de la música como onda de frecuencias, lo que la posiciona por encima de howler.js.

### 3.8.8. tuna

tuna [45] es una librería de efectos de audio para la API web audio. Proporciona una forma fácil de crear y conectar efectos a los nodos de Web Audio y ofrece múltiples efectos básicos creados de antemano, fácilmente configurables.

### 3.8.9. Music tempo

Music tempo [50] es una biblioteca que permite calcular el tempo de una canción (o beats per minute) y un array de beats. Music tempo utiliza el algoritmo Beatroot [52].

### 3.8.10. ngx-web-worker

ngx-web-worker [53] es una biblioteca usada para permitir ejecutar web workers en segundo plano en Angular. Esto permite ejecutar grandes cargas de trabajo sin afectar al rendimiento general de la web-app. Es una implementación de los Web Workers de JavaScript [54] en Angular.

### 3.8.11. ngx-translate

ngx-translate [58] es un servicio de traducción para Angular. Permite preparar fácilmente ficheros de traducciones en formato json y realizar traducciones dinámicas. Su uso es bastante sencillo, pues añade una tubería para mostrar datos. Una vez preparado el servicio sólo hay que cambiar el texto por una ruta en el json del texto que se quiere traducir y añadir la tubería. Por ejemplo siendo por una parte `en.json`:

```
{
    "SETTINGS": {
        "LANGUAGE": "Language",
        ...
    },
    ...
}
```

Y, por otra parte, `es.json`:

```
{
    "SETTINGS": {
        "LANGUAGE": "Idioma",
        ...
    },
    ...
}
```

El siguiente fragmento de código:

```
<p>{{SETTINGS.LANGUAGE | translate}}</p>
```

Durante la ejecución, dependiendo del idioma seleccionado en cada instante, se convertiría en:

```
<p>Language</p>
```

o en:

```
<p>Idioma</p>
```

## 3.9. PlantUML

PlantUML [57] es una herramienta de código abierto que permite a los usuarios crear diagramas UML desde un lenguaje de texto sin formato. También permite otros diagramas no-UML. Los diagramas son definidos usando un lenguaje simple e intuitivo. Usa Graphviz para la generación de gráficos.

## 3.10. Visual Paradigm

Pese a que PlantUML es una gran herramienta para la creación de diagramas de clases, su funcionalidad no es lo suficientemente madura para ser usada en diagramas de componentes. Por tanto, también se ha usado Visual Paradigm [64] para este tipo de diagramas.

# Capítulo 4

# Plan de riesgos y presupuestos

## 4.1. Plan de riesgos

Cuando se realiza la planificación de un proyecto de software (o un proyecto de cualquier otra índole) es importante un plan de riesgos [39]. Para elaborar un plan de riesgos debemos:

- Identificar los posibles riesgos así como su tipo. Algunos tipos de riesgo son: técnicos, de planificación, de cliente, de contrato, climáticos, financieros, políticos, del entorno o personales.
- Evaluar esos riesgos: De cada uno de esos riesgos se debe identificar la posibilidad de ocurrencia y su impacto.
- Elaborar un plan de mitigación y un plan de contingencia para gestionar estos riesgos. El plan de mitigación se refiere al conjunto de acciones que se toman por adelantado o acciones proactivas. En cambio el plan de contingencia se refiere al conjunto respuestas o acciones reactivas si realmente ocurre el riesgo.

### 4.1.1. Riesgos de este proyecto

Los riesgos encontrados son los riesgos habituales de la mayoría de proyectos de software, pero al contar con un personal tan limitado (1 desarrollador) estos riesgos pueden tener un impacto mucho mayor de lo habitual.

Las Tablas de la 4.1 a la 4.8 especifican los riesgos identificados, su probabilidad de ocurrencia, impacto y planes de mitigación y contingencia.

| | |
|---|---|
| Riesgo 1 | Enfermedad en un miembro del equipo |
| Tipo | Personal |
| Probabilidad | Baja |
| Impacto | Muy Alto |
| Descripción | Un miembro del equipo cae enfermo |
| Plan de mitigación | Se ha previsto holgura entre la fecha estimada de finalización y la fecha máxima de entrega y se ha dado holgura al presupuesto |
| Plan de contingencia | Posponer las tareas afectadas al siguiente sprint y en caso de necesidad añadir un sprint más |

Tabla 4.1: Riesgo de enfermedad

| | |
|---|---|
| Riesgo 2 | Falta de formación |
| Tipo | Técnico |
| Probabilidad | Baja |
| Impacto | Medio |
| Descripción | El equipo no está suficientemente formado |
| Plan de mitigación | Se han reservado parte de las horas del proyecto a formación del equipo |
| Plan de contingencia | Pedir ayuda a alguien con mayores conocimientos |

Tabla 4.2: Riesgo de falta de formación

| | |
|---|---|
| Riesgo 3 | Requisitos ambiguos |
| Tipo | Contractual |
| Probabilidad | Baja |
| Impacto | Medio |
| Descripción | Los requisitos del proyecto son ambiguos |
| Plan de mitigación | Dar mucha importancia al proceso de elaboración de las user stories |
| Plan de contingencia | Revisar las user stories después de cada sprint y valorar si siguen siendo claras (especialmente si se han modificado o añadido user stories nuevas) |

Tabla 4.3: Riesgo de requisitos ambiguos

| | |
|---|---|
| Riesgo 4 | Requisitos cambiantes |
| Tipo | Contractual |
| Probabilidad | Baja |
| Impacto | Alto |
| Descripción | Algún requisito del proyecto cambia |
| Plan de mitigación | Usar planificación ágil, que lidia bien con requisitos cambiantes |
| Plan de contingencia | Aceptar los cambios |

Tabla 4.4: Riesgo de requisitos cambiantes

| | |
|---|---|
| Riesgo 5 | Retraso en la planificación |
| Tipo | De planificación |
| Probabilidad | Baja |
| Impacto | Medio |
| Descripción | Debido a una mala planificación se necesitan más horas de trabajo |
| Plan de mitigación | Se ha previsto holgura entre la fecha estimada de finalización y la fecha máxima de entrega y se ha dado holgura en el presupuesto |
| Plan de contingencia | Añadir un sprint más en caso de necesidad |

Tabla 4.5: Riesgo de retraso en la planificación

| | |
|---|---|
| Riesgo 6 | Problemas de hardware |
| Tipo | Técnicos |
| Probabilidad | Baja |
| Impacto | Alto |
| Descripción | Un imprevisto en el hardware usado puede resultar pérdida del trabajo realizado, así como retrasos de planificación. |
| Plan de mitigación | Frecuencia alta de actualización del repositorio de software para evitar la perdida de datos. |
| Plan de contingencia | Reparar el hardware implicado y en caso necesario añadir un sprint más. |

Tabla 4.6: Riesgo de problemas de hardware

| | |
|---|---|
| Riesgo 7 | Problemas con T&C de YouTube |
| Tipo | Legales |
| Probabilidad | Media |
| Impacto | Alto |
| Descripción | Puede que no sea posible usar YouTube como servicio para la obtención de música |
| Plan de mitigación | Estudio de la viabilidad del uso de YouTube como servicio de obtención de música |
| Plan de contingencia | Buscar una plataforma alternativa o descartar la funcionalidad. |

Tabla 4.7: Riesgo de problemas con T&C de YouTube

| | |
|---|---|
| Riesgo 8 | Falta de tiempo |
| Tipo | Personal |
| Probabilidad | Baja |
| Impacto | Muy Alto |
| Descripción | Un miembro del equipo no tiene tiempo para trabajar |
| Plan de mitigación | Se ha previsto holgura entre la fecha estimada de finalización y la fecha máxima de entrega. |
| Plan de contingencia | Posponer el final del sprint en el que se está trabajando |

Tabla 4.8: Riesgo de falta de tiempo

## 4.2. Cálculo de presupuesto

Tras consultar el Boletín Oficial del Estado [27] se ha encontrado que el salario base total para un Analista programador; Diseñador páginas web es de 22.993,74. Esto significa que para 160 horas al mes trabajadas a lo largo de 12 meses el sueldo bruto por hora es de 11,976 euros.

Una empresa paga por un trabajador un extra del 30 % a la seguridad social [28]. Si además asumimos algún beneficio social de aproximadamente un 20 % extra, una empresa tendría un coste total de 18 euros la hora de un Analista.

Esto significa que para un trabajo de 300 horas el coste aproximado sería de 5400 euros.

Se añaden al presupuesto 300 euros para gastos de iconografía e imagen en previsión de que deban crearse imágenes para la interfaz de usuario.

En cuanto a la amortización de hardware, se trabaja con un HP EliteBook 840 G3 valorado en 1.129 euros. Con la tabla de amortizaciones en mente [29], el coste de usar este hardware sería de 1.129 * 25 % anual * 6/12 años de trabajo estimado, es decir 141,125 euros.

Se presupuesta además la licencia de GitLab bronze [32] que ayudaría en algunos aspectos del DevOps y la revisión de código, con un coste de 4 dólares al mes o 3,45 euros al mes (16/10/2018) durante 6 meses. Esto costaría 20,7 euros.

El precio final estimado ascendería a 5861,825 euros. Añadiremos un 20 % al presupuesto para contar con un colchón de presupuesto con el que afrontar posibles riesgos. El total resultante es de 7034,19 euros (Tabla 4.9).

| Asunto | Coste |
|---|---|
| 300 horas de trabajo | 5400 € |
| Iconografía | 300 € |
| Amortización | 141,125 € |
| GitLab bronze | 20,7 € |
| Suma total | 5861,825 € |
| Total normalizado | 7034,19 € |

Tabla 4.9: Estimación de presupuesto

### 4.2.1. Costes finales

Tras terminar todos los sprints, se han invertido un total de 323 horas como se puede ver en la Tabla 4.10.

Finalmente no se contrató ningún servicio de iconografía ni se contrató la licencia GitLab bronze, dado que se utiliza la instancia desplegada en la Escuela con funcionalidad pro

| Sprint | Horas invertidas |
|---|---|
| 1 | 33 |
| 2 | 26 |
| 3 | 22 |
| 4 | 31,5 |
| 5 | 25 |
| 6 | 35 |
| 7 | 30 |
| 8 | 34 |
| 9 | 40,5 |
| 10 | 27 |
| 11 | 19 |
| Total | 323 |

Tabla 4.10: Horas finales invertidas

solicitada por ser institución académica. Por tanto, el presupuesto final quedaría como se ve en la Tabla 4.11

| Asunto | Coste |
|---|---|
| 323 horas de trabajo | 5814 € |
| Iconografía | 0 € |
| Amortización | 141,125 € |
| GitLab bronze | 0 € |
| Suma total | 5955,125 € |

Tabla 4.11: Costes finales

Por tanto se ha calculado correctamente el presupuesto, pues el coste final queda entre el total estimado y el total normalizado.

# Capítulo 5

# Seguimiento del proyecto

Durante el seguimiento del proyecto se han ido eligiendo tareas asociadas a las user stories, pero también otras tareas necesarias para la creación del producto final, que en este caso es el producto de software y toda la documentación asociada a su creación.

## 5.1. Seguimiento sprint a sprint

Se realiza un seguimiento de los sprints, tareas asociadas a cada uno y tiempo empleado en horas hombre (HH). Se documentará utilizando unas tablas con el identificador y tipo de tarea. Como tipos de tareas se utilizará: chore, us, qa, bug. Las tareas chore son necesarias para el desarrollo del proyecto pero no están vinculadas a una user story, las tareas us (user story) se acompañarán del id de la user story asociada, los tipos de tarea qa (quality assurance) y bug se corresponden con tareas de prueba y mejora de la calidad del código así como de corrección de errores, respectivamente. En las tablas de seguimiento de cada sprint se incluye las HH correspondientes al tiempo invertido en la tarea y el estado de la tarea al finalizar el sprint.

### 5.1.1. Sprint 1

Durante el primer sprint se ha documentado la introducción, las metodologías y parte de las tecnologías y el planning. También se han especificado las historias de usuario iniciales y para poder preparar el entorno de desarrollo se ha investigado el CI y CD de GitLab, testing con Jest, limpieza de código con prettier y hooks precommit con husky:

También se estima un gasto de 2 horas para labores de planificación en GitLab: creación de issues, creación de tareas, gestión del tablero ágil, gestión de milestones.

Esto significa un tiempo invertido para este sprint de aproximadamente 33 horas.

| Tarea | Tipo | Descripción | HH | Estado |
|---|---|---|---|---|
| T - 001 | Chore | Historias de usuario | 1h | Completado |
| T - 002 | Chore | Documentar planificación, calendarización, riesgos y presupuesto | 3h | En progreso |
| T - 003 | Chore | Documentar la introducción y objetivos | 3h | Completado |
| T - 004 | Chore | Documentar las metodologías y patrones | 8h | Completado |
| T - 005 | Chore | Documentar las tecnologías | 8h | En progreso |
| T - 006 | Chore | Preparar el entorno de desarrollo | 8h | En progreso |

Tabla 5.1: Tareas del sprint 1

Para el sprint backlog del siguiente sprint se han pasado las 3 tareas incompletas del sprint 1 y se han añadido tareas asociadas a la user story 1.

### 5.1.2. Sprint 2

Durante este sprint se ha preparado el entorno de desarrollo con las tecnologías previamente investigadas y se han documentado estas tecnologías. También se ha empezado a desarrollar la aplicación. Se ha comenzado por la interfaz de usuario, esta es nuestra primera historia de usuario. Se ha desglosado en 2 tareas. Primero, realizar bocetos de la interfaz de usuario y segundo realizar la interfaz como tal, con HTML y CSS.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 002 | Chore | Documentar planificación, calendarización, riesgos y presupuesto | 3h | En progreso |
| T - 005 | Chore | Documentar las tecnologías | 1h | En progreso |
| T - 006 | Chore | Preparar el entorno de desarrollo | 2h | Completado |
| T - 007 | US001 | Bocetos de la interfaz gráfica de usuario | 3h | Completado |
| T - 008 | US001 | Interfaz con HTML y CSS | 12h | En progreso |
| T - 009 | US001 | Separación de la interfaz de usuario en componentes. | 3h | En progreso |

Tabla 5.2: Tareas del sprint 2

También se han invertido 2 horas en crear un glosario de términos. En total se han invertido aproximadamente 26 horas en este sprint y se han completado 2 tareas. Las tareas 2 y 5 están bastante avanzadas pero no se pueden completar aún pues puede haber modificaciones. Éstas, junto con las tareas 8 y 9, se pasan al siguiente sprint. No se añaden nuevas tareas pues las tareas de la interfaz de usuario son las más grandes. Se deberían haber separado en varias tareas más pequeñas.

### 5.1.3. Sprint 3

Durante este sprint se ha seguido desarrollando la interfaz de usuario. También se ha documentado todo este desarrollo.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 002 | Chore | Documentar planificación, calendarización, riesgos y presupuesto | 1h | Completado |
| T - 005 | Chore | Documentar las tecnologías | 3h | En progreso |
| T - 008 | US001 | Interfaz con HTML y CSS | 11h | Completado |
| T - 009 | US001 | Separación de la interfaz de usuario en componentes. | 4h | Completado |
| T - 010 | Chore | Documentar la implementación. | 3h | En progreso |

Tabla 5.3: Tareas del sprint 3

En total se han trabajado 22 horas. Se han terminado múltiples tareas. La documentación de las tecnologías aún no se puede finalizar pues probablemente se usará algún paquete de npm para la reproducción de audio. Se han pasado las tareas no terminadas al siguiente sprint.

### 5.1.4. Sprint 4

Durante este sprint se ha decidido empezar a trabajar en la lógica, primero en cargar la música desde el ordenador y segundo en poder reproducir música. Estas son la user stories 2 y 3. Se ha dividido en 3 tareas:

- una tarea de investigación y búsqueda de una biblioteca de reproducción de audio (se detalla en el capítulo de tecnologías)
- una tarea correspondiente a la carga de música local desde el ordenador
- una tarea correspondiente a la integración de la biblioteca elegida en nuestro proyecto

También se ha documentado este desarrollo.

Durante el trascurso del sprint se han añadido tres tareas más. Una consistente en crear un servicio que se encarga de la reproducción de música, que facilita acceder a distintas funciones del reproductor desde distintos componentes. Otra consistente en la gestión de volumen básica, correspondiente a la user story 4, que al tener el servicio ha sido bastante rápida. La última consistente en investigar cómo aplicar efectos de audio a la música que se está reproduciendo.

En total se han dedicado 31.5 horas en las tareas explicadas, más 1 hora extra para tareas de gestión del proyecto (gestión de tareas, user stories, tablero Kanban, documentación de esta sección).

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 0.5h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 2h | En progreso |
| T - 011 | US003 | Investigar bibliotecas de tratamiento de audio. | 8h | Completada |
| T - 012 | US002 | Carga de música local. | 5h | En progreso |
| T - 013 | US003 | Conexión con la librería elegida. | 3h | En progreso |
| T - 014 | US003 | Servicios de reproductor de música. | 4h | En progreso |
| T - 015 | US004 | Control de volumen. | 1h | En progreso |
| T - 016 | US009 | Investigar bibliotecas de efectos de audio. | 8h | En progreso |

Tabla 5.4: Tareas del sprint 4

### 5.1.5. Sprint 5

Durante este sprint se ha decidido trabajar en los efectos de audio y el volumen avanzado (que en términos de implementación es un efecto). Para ello se han creado las dos tareas correspondientes.

| Tarea | Descripción | Tiempo invertido | Estado | |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 0h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 0h | En progreso |
| T - 017 | US005 | Control de volumen avanzado. | 8h | En progreso |
| T - 018 | US009 | Implementación de efectos de audio por defecto. | 16h | Completado |

Tabla 5.5: Tareas del sprint 5

En total se han dedicado 24 horas en las tareas explicadas, más 1 hora extra para tareas de gestión del proyecto. Por desgracia durante este sprint nos ha afectado el riesgo de enfermedad 4.1, lo que ha resultado en una pérdida de 5 horas con respecto a la planificación. Se aplica el plan de contingencia, posponiendo algunas tareas (sobre todo de documentación) para el siguiente sprint.

### 5.1.6. Sprint 6

Durante este sprint primero se han usado unas 4 horas para terminar tareas no terminadas pertenecientes al anterior sprint. Después se ha decidido trabajar en la customización de los efectos de audio. Para ello se han creado las tareas correspondientes.

Se ha dedicado 1 hora para la documentación de tecnologías así como 3 horas para la documentación de la implementación del sprint anterior. Con esto se consigue recuperar

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 1h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 5h | En progreso |
| T - 020 | Bug | Menús maximizables | 2h | Completado |
| T - 019 | US009 | Implementación de customización de efectos de audio | 25h | Completado |

Tabla 5.6: Tareas del sprint 6

el ritmo de trabajo esperado. Tras eso, se han dedicado 29 horas a las tareas 10, 19 y 20 asignadas a este sprint así como 2 horas para tareas de gestión del proyecto. Por lo que se han dedicado un total de 35 horas este sprint y se han cerrado 2 tareas.

### 5.1.7. Sprint 7

Durante este sprint se ha investigado la viabilidad de la descarga de música desde Youtube o Soundcloud, desde el punto de vista legal y técnico.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 0h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 4h | En progreso |
| T - 21 | US007 | Investigación sobre descarga directa de música y pruebas de concepto | 21h | Completado |
| T - 22 | US008 | Implementación del pitch | 2h | Completado |

Tabla 5.7: Tareas del sprint 7

Durante este sprint se han dedicado 21 horas a la investigación sobre descarga de música desde plataformas externas. También se han dedicado 2 horas a implementar el pitch (US 008) y finalmente 4 horas a documentación. Adicionalmente, se utilizó una hora para tareas de gestión del proyecto. Se cerraron dos tareas este sprint y se trabajó un total de 30 horas.

### 5.1.8. Sprint 8

Durante este sprint se han terminado las funcionalidades necesarias (loops y CUEs) además de hacer algunos cambios para mejorar la calidad en uso del software, como son, un sistema de ayuda, la localización del software, y otros cambios menores.

Durante este sprint nos ha vuelto a afectar el riesgo de enfermedad, resultando en la perdida de casi una semana entera. Por tanto se ha decidido posponer el fin del sprint una semana. Esto resulta en aplazar el fin del sprint 8 desde el 1 de febrero al 8. Aun así esto permitió dedicar un poco mas de tiempo al sprint al final, resultando en la realización de más tareas de las originalmente planeadas. En total se han usado 32 horas para la realización de

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 1h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 3h | En progreso |
| T - 023 | US011 | Implementación del CUE | 5h | Completado |
| T - 024 | Bug | QoL changes | 4h | Completado |
| T - 025 | US010 | Implementar los loops | 9h | Completado |
| T - 026 | Chore | Sistema de ayuda | 3h | Completado |
| T - 027 | Chore | Localización | 7h | Completado |

Tabla 5.8: Tareas del sprint 8

tareas, así como 2 horas para tareas de gestión, para un total de 34 horas trabajadas este sprint.

### 5.1.9. Sprint 9

Durante este sprint se ha comenzado el testing programático y el testing con usuarios, así como la resolución de bugs encontrados durante ambos procesos, mejora de la calidad de código e implementación de mejoras sugeridas por los usuarios.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 1h | En progreso |
| T - 010 | Chore | Documentar la implementación. | 1h | Completado |
| T - 035 | Chore | Documentar el testing. | 1h | En progreso |
| T - 028 | QA | Unit testing | 15h | En progreso |
| T - 029 | QA | Planificación de pruebas con usuarios | 2h | Completado |
| T - 030 | QA | Pruebas con usuarios | 4.5h | En progreso |
| T - 031 | Bug | Resolución de bugs | 5.5h | En progreso |
| T - 033 | Bug | Quality Issues | 4h | En progreso |
| T - 036 | Bug | Mejoras | 4.5h | En progreso |

Tabla 5.9: Tareas del sprint 9

Durante este sprint nos ha afectado el riesgo de falta de tiempo, resultando en la necesidad de posponer el sprint una semana. Esto resulta en aplazar el fin del sprint 9 desde el 22 de febrero al 1 de marzo. Se ha usado un total de 38.5 horas para tareas y aproximadamente 2 horas para tareas de gestión, resultando en un total de 40.5 horas trabajadas este sprint.

### 5.1.10. Sprint 10

Durante este sprint se ha seguido con las pruebas de usuarios y la resolución de bugs y mejoras. También se ha documentado el modelo arquitectónico.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 005 | Chore | Documentar las tecnologías | 1h | Completado |
| T - 035 | Chore | Documentar el testing. | 0h | En progreso |
| T - 032 | Chore | Diagramas de arquitectura de la aplicación y organización del código | 15h | En progreso |
| T - 028 | QA | Unit testing | 0h | Completado |
| T - 030 | QA | Pruebas con usuarios | 2,5h | Completado |
| T - 031 | Bug | Resolución de bugs | 0h | Completado |
| T - 033 | Bug | Quality Issues | 3h | Completado |
| T - 036 | Bug | Mejoras | 4,5h | Completado |

Tabla 5.10: Tareas del sprint 10

Se ha vuelto a producir el riesgo de falta de tiempo, resultando en la necesidad de posponer el sprint una semana. Esto resulta en aplazar el fin del sprint 10 desde el 15 al 22 de marzo. Se ha usado un total de 25 horas para tareas y aproximadamente 2 horas para tareas de gestión, resultando en un total de 27 horas trabajadas este sprint.

Este sprint marcaría el final programado originalmente después de 10 sprints. Aunque se han completado todas las User Stories. Se han cerrado todas las tareas relacionadas con mejoras, quality issues, tests programáticos o con usuarios y se ha publicado en master la última versión del código. Aún así, quedan tareas de documentación por completar. Por tanto debemos añadir un sprint a mayores.

### 5.1.11. Sprint 11 (adicional)

Durante este sprint se ha seguido documentando el modelo arquitectónico basado en componentes.

| Tarea | Tipo | Descripción | Tiempo invertido | Estado |
|---|---|---|---|---|
| T - 035 | Chore | Documentar el testing. | 2h | Completado |
| T - 038 | Chore | Redacción de las conclusiones. | 2h | Completado |
| T - 037 | Chore | Revisar documentación completa. | 3h | Completado |
| T - 032 | Chore | Diagramas de arquitectura de la aplicación | 9h | Completada |
| T - 039 | Chore | Añadir licencia al proyecto | 1.5h | Completada |
| T - 40 | Chore | Resumen y Abstract | 1.5h | Completada |

Tabla 5.11: Tareas del sprint 11

En total se han usado 19 horas en este sprint adicional.

# Capítulo 6

# Diseño

## 6.1. Modelo Vista Controlador

Modelo Vista Controlador [12] (MVC) es un estilo de arquitectura software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes.

Se trata de un modelo maduro que ha demostrado su validez a lo largo de los años. En él distinguimos:

- Modelo, que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

### 6.1.1. Modelo

El modelo es responsable de:

- Acceder a la capa de almacenamiento de datos, pero a ser posible siendo independiente del sistema de almacenamiento.
- Definir las reglas de negocio (la funcionalidad del sistema).
- Si estamos ante un MVC activo

a) Llevar un registro de las vistas y controladores.
b) notificar a las vistas y/o controladores los cambios de datos que pueda producir un agente externo.

### 6.1.2. Controlador

El controlador es responsable de:

- Recibir los eventos de entrada
- Lanzar la respuesta a los eventos. Esto puede suponer peticiones al modelo para solicitar datos o a las vistas para actualizarlas.

### 6.1.3. Vista

La vista es responsable de:

- Mostrar al usuario datos del modelo.
- Tener un registro de su controlador asociado.
- Dar acceso a los métodos necesarios para actualizarse.

### 6.1.4. Flujo habitual

El flujo que sigue el control generalmente en una aplicación de MVC a un servicio web es el representado en la Figura 6.1:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o de un callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo. El modelo no debe tener conocimiento directo sobre la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

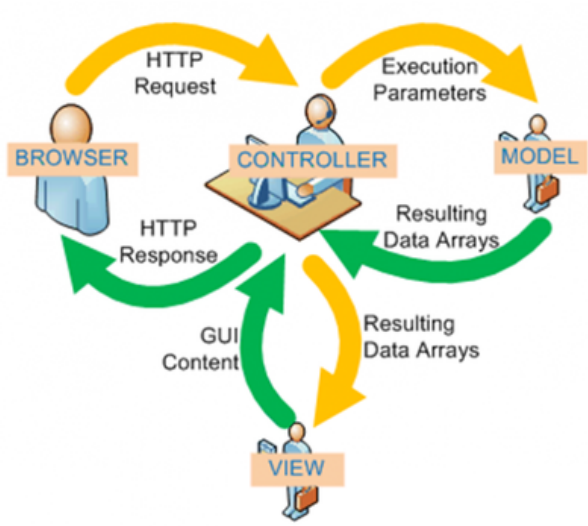


Figura 6.1: Flujo MVC aplicado a un servicio web [12]

### 6.1.5. Aplicación de MVC en este proyecto

Como se ha explicado anteriormente, se ha usado el framework Angular. Por defecto Angular separa la vista del controlador. Además es buena practica separar las acciones que tienen que ver con consulta de datos en clases llamadas servicios. Por lo tanto, con Angular se facilita bastante la aplicación de MVC.

## 6.2. Desarrollo basado en componentes

La complejidad de los sistemas computacionales actuales nos ha llevado a buscar la reutilización del software existente. El desarrollo de software basado en componentes permite reutilizar piezas de código elaborado previamente que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión.

### 6.2.1. Beneficios del desarrollo basado en componentes

En esencia, un componente es una pieza de código elaborado previamente que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea.

El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes. El uso de

este paradigma posee algunas ventajas:

- Reutilización de software
- Simplifica las pruebas: parte de las pruebas se hacen con el componente por separado, facilitando las pruebas generales del producto.
- Simplifica el mantenimiento del sistema: Cuando existe débil acoplamiento entre componentes el desarrollador puede añadir y quitar componentes sin afectar a otras partes del sistema.
- Mayor calidad: Al ser cada componente encargado de una tarea, es mas fácil crear un componente simple, bien documentado y de calidad.

Por supuesto también cuenta con inconvenientes:

- Documentar: la documentación requiere más trabajo pues debe tener una documentación cuidada y fácil de entender, para facilitar su uso.
- Más carga de trabajo al principio: Cuanto más general (o configurable) se quiere hacer un componente más trabajo de desarrollo y pruebas requiere. Se requiere mayor carga de trabajo a la hora de crear el componente pero menos a la hora de usarlo. Hay que estudiar para qué casos merece la pena.

También se puede optar por usar componentes desarrollados por otros. Tiene sus pros:

- Ciclos de desarrollo más cortos: Se usan directamente piezas con una funcionalidad que podría llevar semanas o meses implementar.
- Componente opaco: Si está bien documentado no es necesario saber cómo es su funcionamiento interno, solo es necesario conocer sus interfaces.

Pero a cambio hay contras:

- Dificultad para extender esos componentes directamente.

En general, suele ser beneficioso usar componentes al aplicar los principios [14] de bajo acoplamiento y alta cohesión (glosario: cohesión).

### 6.2.2. Componentes en Angular

Angular está pensado con el desarrollo basado en componentes en mente.

En la Figura 6.2 se puede ver cómo un componente Angular está formado por una clase Typescript y un template HTML.

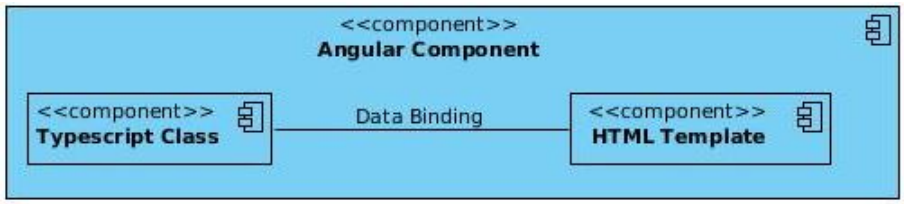


Figura 6.2: Diagrama de componentes: Componente básico

La interacción entre componentes padre e hijo se puede ver en la Figura 6.3. En este diagrama se ve que el componente hijo es parte del template del padre. Esto permite al componente padre comunicarse con un hijo de forma fácil.

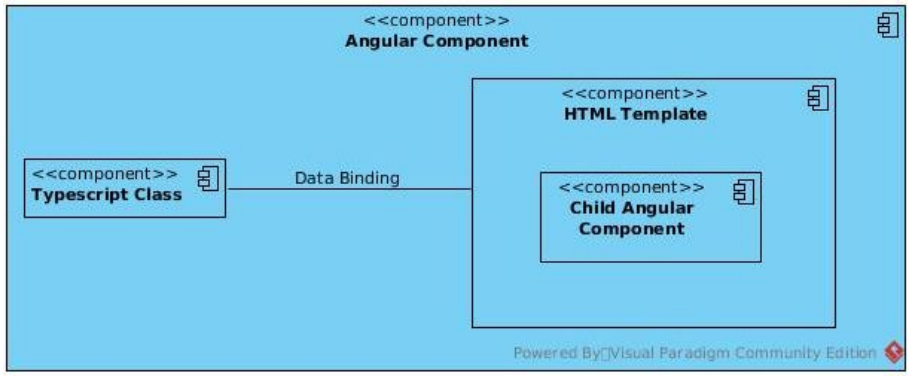


Figura 6.3: Diagrama de componentes: Componentes padre e hijo

En cuanto a los servicios, el inyector de servicios actúa como interfaz para el componente y permite usar los métodos fácilmente. Esto se aprecia en la Figura 6.4.

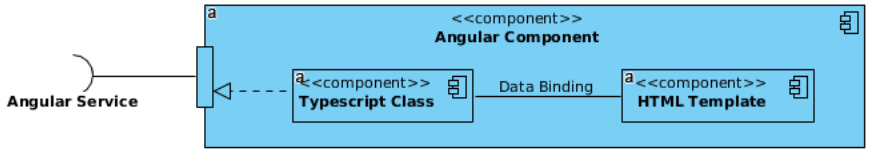


Figura 6.4: Diagrama de componentes: Componente con servicio

También se han usado directivas de Angular. Estas directivas actúan como un componente reutilizable dentro de los templates, que añade funcionalidades al template. Esto se puede apreciar en la Figura 6.5.

Los servicios y directivas de Angular también son componentes programáticos, pero no componentes Angular. Es decir, son piezas reutilizables de software pero no son los componentes como los entiende Angular.

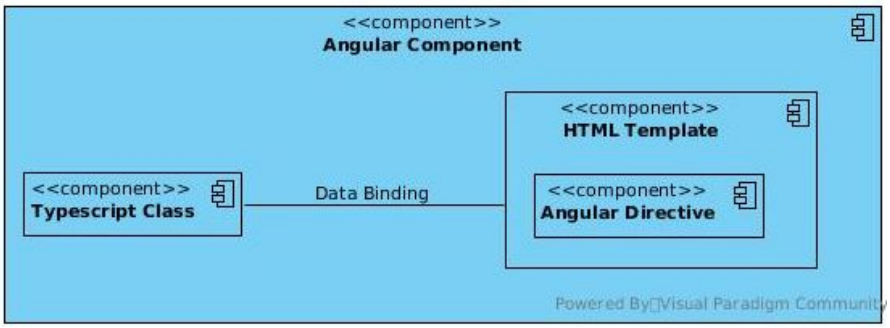


Figura 6.5: Diagrama de componentes: Componente con directiva

### 6.2.3. Componentes creados

Los componentes creados se pueden dividir en componentes y servicios Angular. Esto se ve en los diagramas con el estereotipo.

Los componentes Angular que se han creado son:

- App: Encargado de contener la aplicacion completa (Figura 6.6).
- AppLayout: Encargado de contener la interfaz de usuario (Figura 6.12).
- AppDeck: Encargado de contener las platinas (Figura 6.8).
- AppVolume: Encargado de contener los controles de volumen y ecualizadores (Figura 6.15).
- AppTabs: Encargado de contener el menú de pestañas (Figura 6.14).
- AppMusicList: Encargado de contener la lista de música (Figura 6.16).
- AppEffectsSelector: Encargado de contener la interfaz de selección de efectos (Figura 6.10).
- AppEffectsCreator: Encargado de contener la interfaz de creación de efectos (Figura 6.9).
- AppSettings: Encargado de contener la interfaz de ajustes (Figura 6.13).
- AppHelp: Encargado de contener la interfaz de ayuda (Figura 6.11).
- AppAbout: Encargado de contener la interfaz de acerca de (Figura 6.7).
- SliderController: Encargado de contener el control de tipo range modificado (Figura 6.18).
- RouletteController: Encargado de contener el control de tipo ruleta (Figura 6.17).

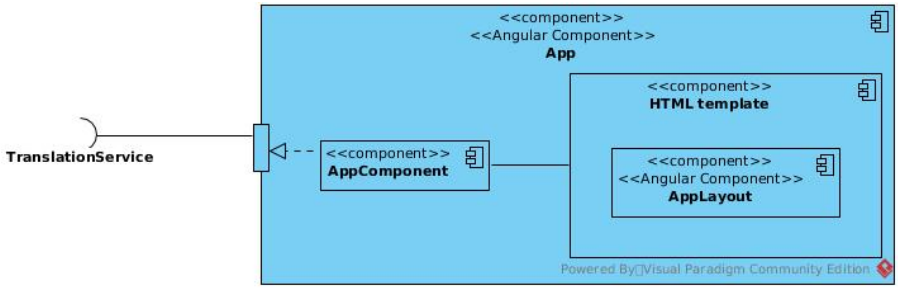


Figura 6.6: Componente App

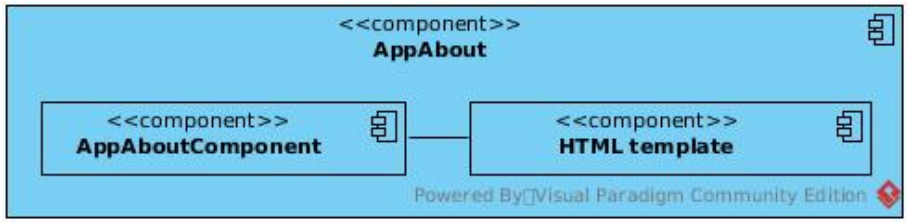


Figura 6.7: Componente AppAbout

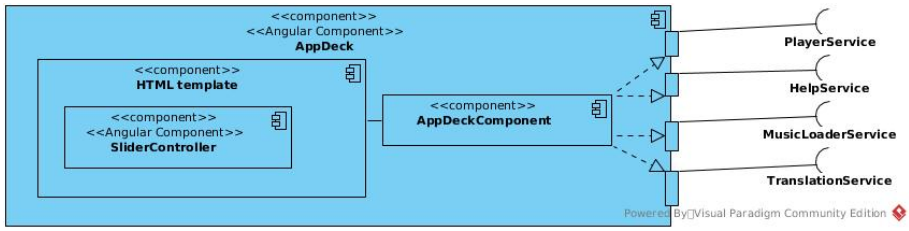


Figura 6.8: Componente AppDeck

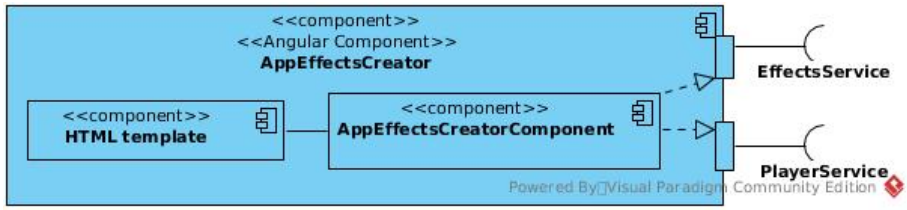


Figura 6.9: Componente AppEffectsCreator

Los servicios Angular presentes son:

- PlayerService: Se encarga de gestionar la reproducción de música, volumen, efectos, CUEs y otras funcionalidades. Es un recubrimiento alrededor de la biblioteca npm Wavesurfer. Representado en la Figura 6.19.

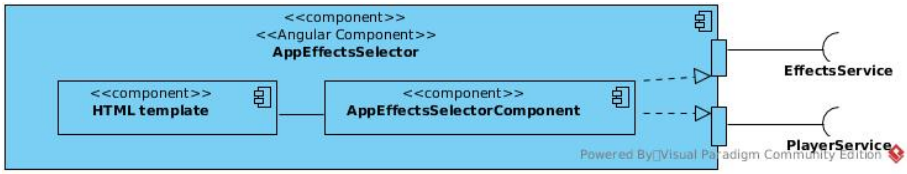


Figura 6.10: Componente AppEffectsSelector

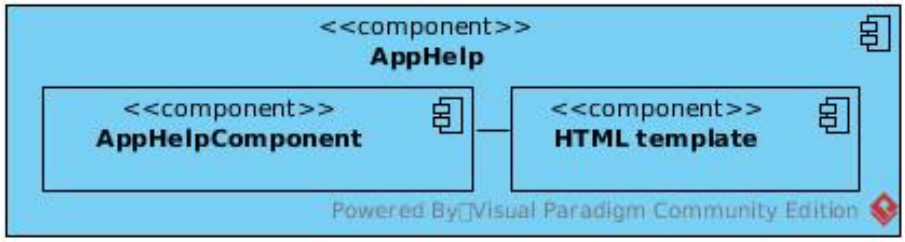


Figura 6.11: Componente AppHelp

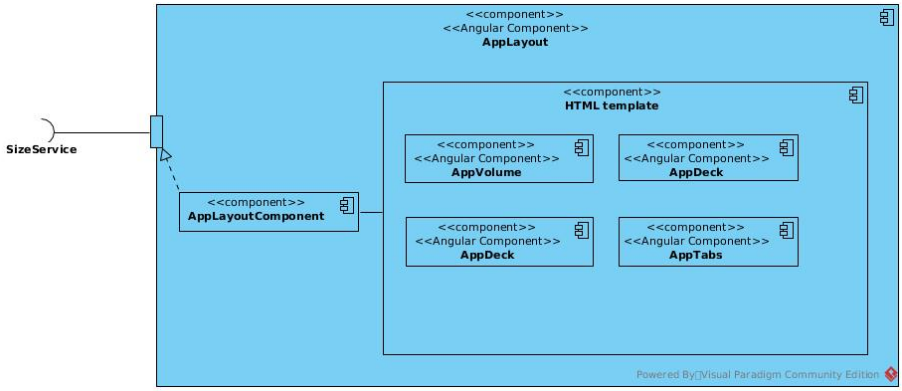


Figura 6.12: Componente AppLayout

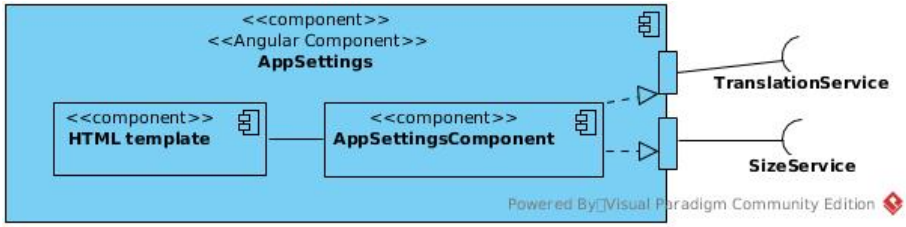


Figura 6.13: Componente AppSettings

- MusicLoaderService: Se encarga de gestionar la carga de música, tanto del ordenador al programa como de la lista de música a una platina. Representado en la Figura 6.19.

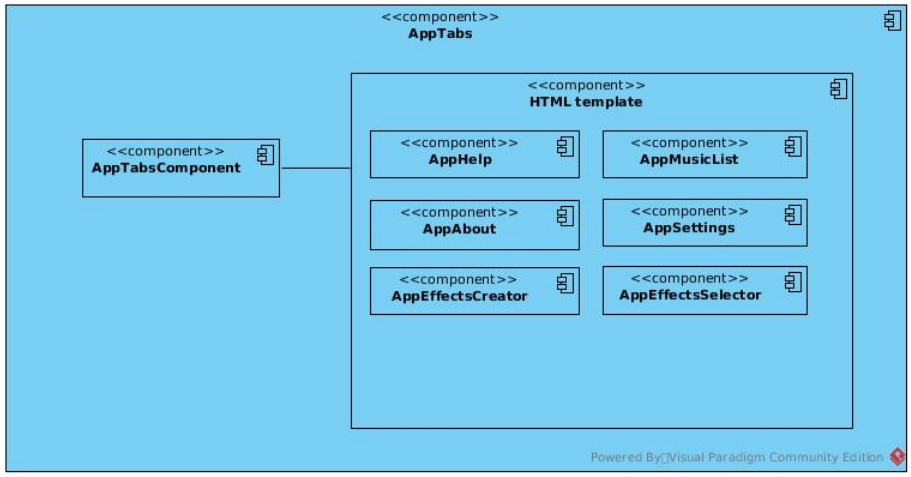


Figura 6.14: Componente AppTabs

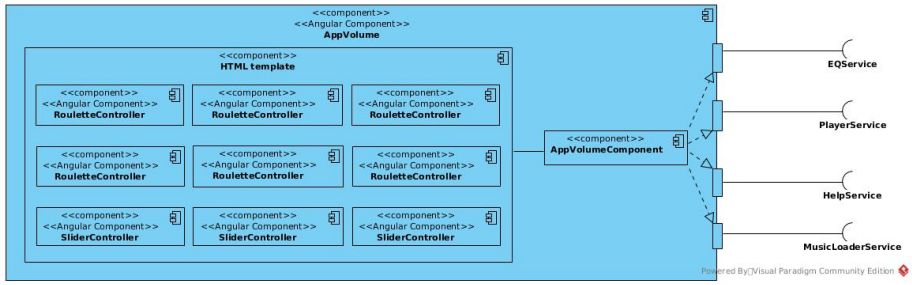


Figura 6.15: Componente AppVolume

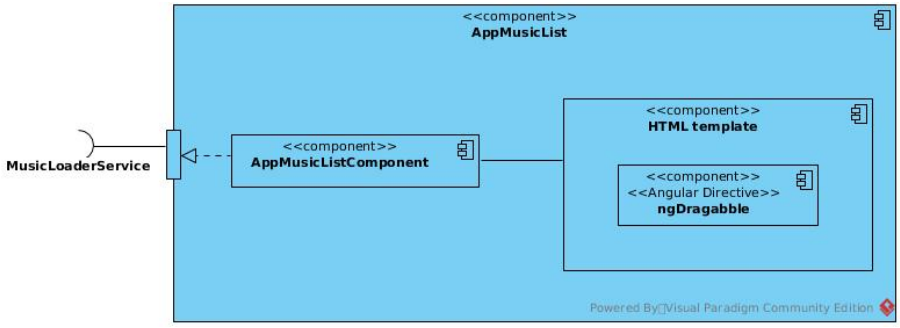


Figura 6.16: Componente AppMusicList

- EffectsService: Se encarga de gestionar los efectos existentes y permitir crear nuevos efectos o eliminarlos. También se encarga de guardar los efectos en el almacenamiento interno del navegador para conservarlos entre sesiones y aporta 6 efectos base si es la primera vez que se inicia la aplicación. Representado en la Figura 6.19.

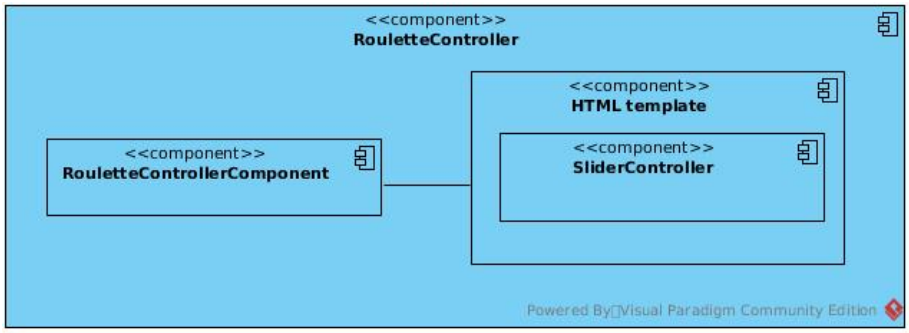


Figura 6.17: Componente RouletteController

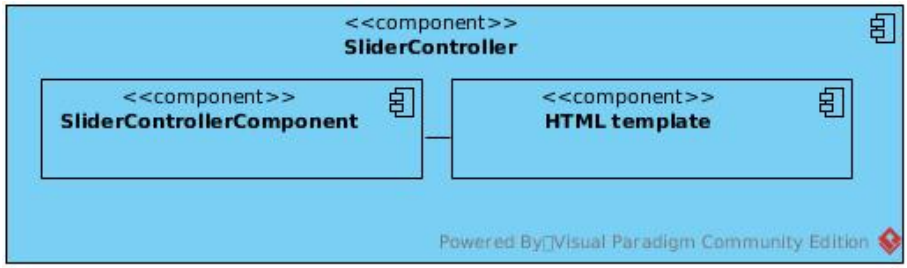


Figura 6.18: Componente SliderController

- EQService: Se encarga de crear los filtros necesarios para tener el efecto de ecualizador. Representado en la Figura 6.19.

- HelpService: Se encarga de notificar a los componentes asociados qué elementos de la interfaz de usuario deben destacar cuando se interactúa con el componente de ayuda. Representado en la Figura 6.20.

- TranslationService: Se encarga de gestionar el idioma en uso. Es un recubrimiento alrededor de la biblioteca ngx-translate. Representado en la figura 6.21.

- SizeService: Se encarga de controlar el tamaño de la interfaz. Pensado para si se crea una versión de escritorio, pues actualmente se ajusta al tamaño disponible. Representado en la figura 6.22.

En los diagramas se representan los servicios PlayerService, MusicLoaderService, EffectsService y EQService juntos, pues juntos se encargan de las funcionalidades principales de la aplicación.

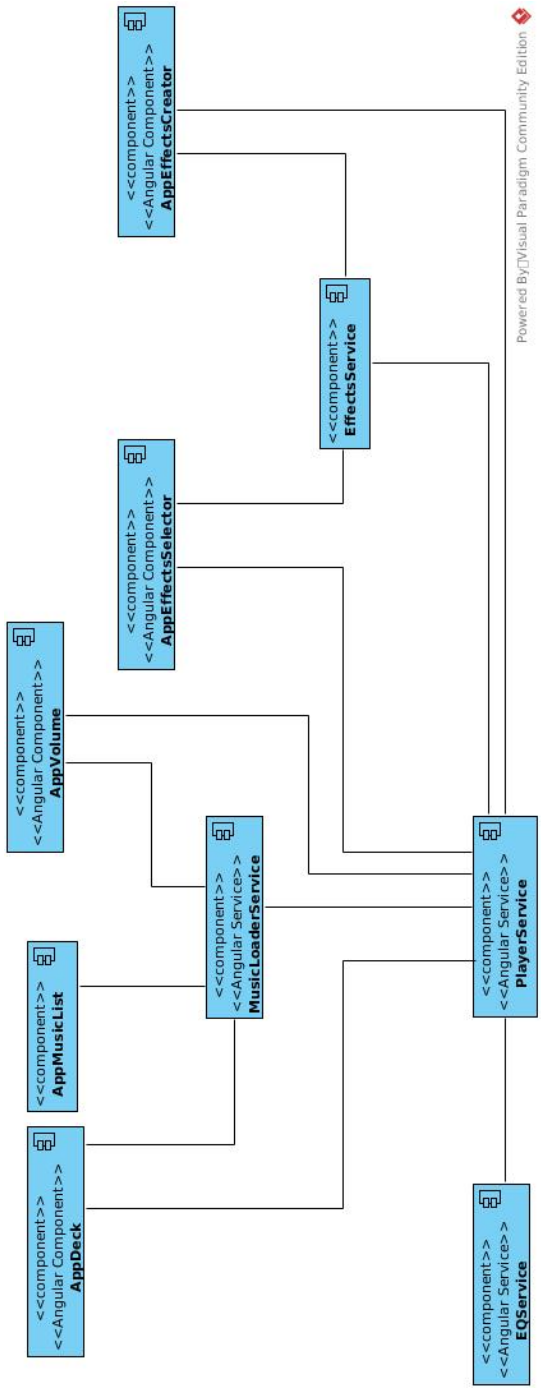


Figura 6.19: Servicios de funcionalidad principal

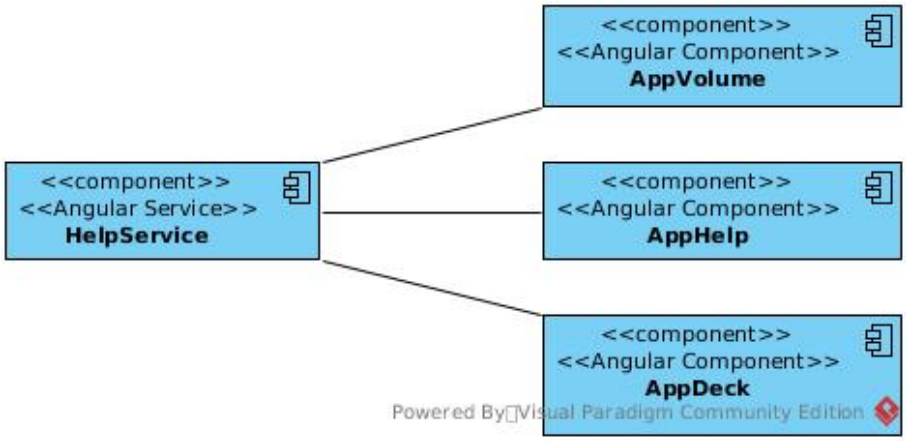


Figura 6.20: HelpService

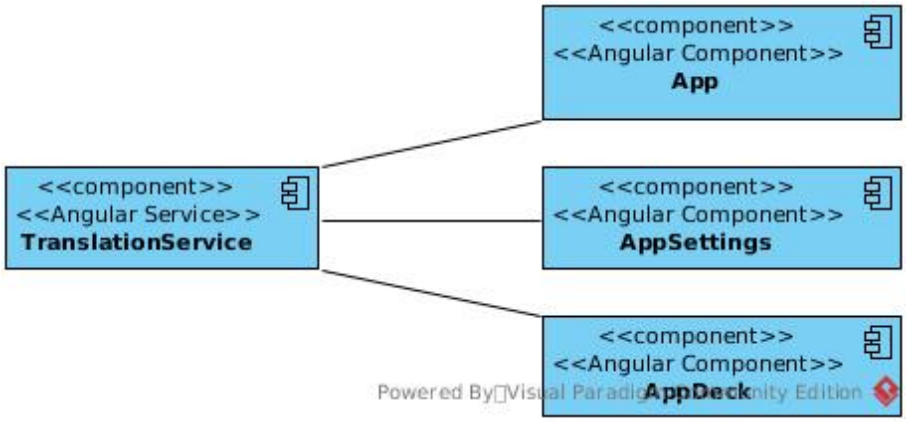


Figura 6.21: TranslationService

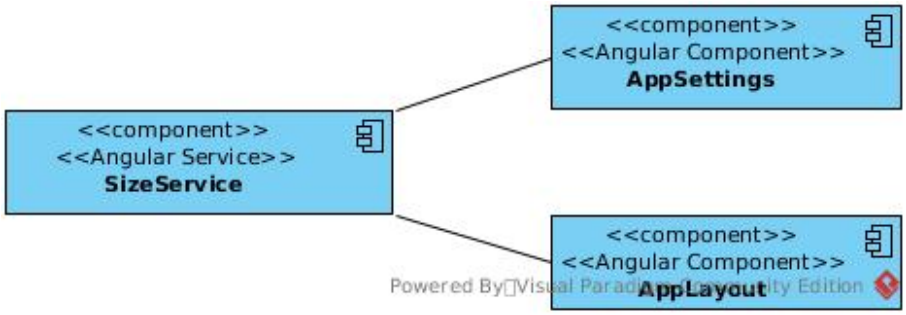


Figura 6.22: SizeService

## 6.3. Diagrama de clases

Habiendo visto la estructura de la aplicación en tiempo de ejecución mediante componentes y la comunicación entre los mismos, queda detallar las clases que realizan en tiempo de compilación las interfaces y funcionalidad de los componentes representados. No se representa el módulo principal AppModule, porque es una clase cuya única función en este proyecto es agrupar todos los componentes, servicios y módulos y es una clase vacía.

Los clases presentes son:

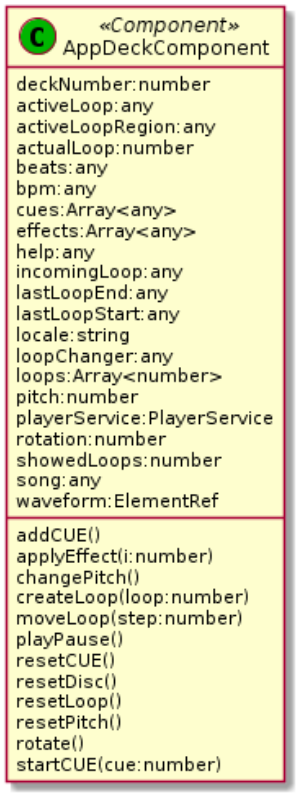


Figura 6.23: AppDeckComponent class

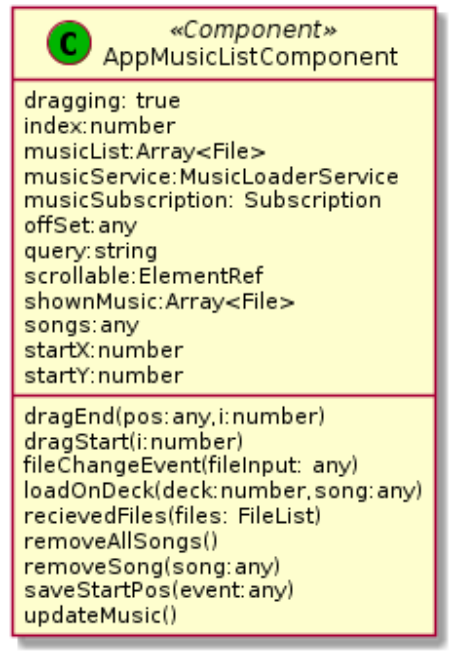


Figura 6.24: AppMusicListComponent class

- AppComponent: la clase del componente principal. Se encarga de inicializar el servicio de traducciones y evitar que el evento drop se propague más arriba en la jerarquía del DOM. Esto es necesario para evitar salir de la página al intentar cargar música arrastrando. Detallada en la Figura 6.34.

- AppLayoutComponent: Se encarga de la gestión del tamaño de ventana. Detallada en la Figura 6.30.

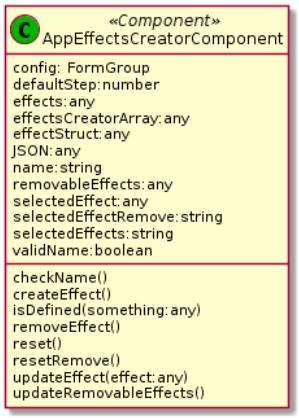


Figura 6.25: AppEffectsCreatorComponent class

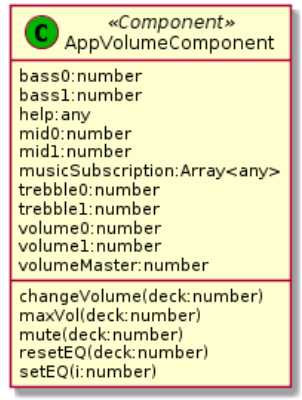


Figura 6.26: AppVolumeComponent class


Figura 6.27: AppTabsComponent class

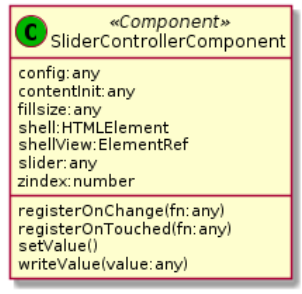


Figura 6.28: SliderControllerComponent class

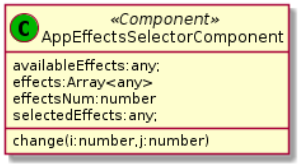


Figura 6.29: AppEffectsSelectorComponent class

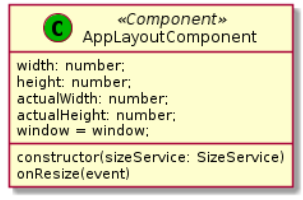


Figura 6.30: AppLayoutComponent class

- AppDeckComponent: Se encarga de conectar los componentes visuales de la platina con los servicios asociados. Interactúa de forma activa con el PlayerService y recibe datos del TranslationService, del MusicLoaderService y del HelpService. Detallada en la Figura 6.23.

- AppVolumeComponent: Se encarga de conectar los componentes visuales de volumen

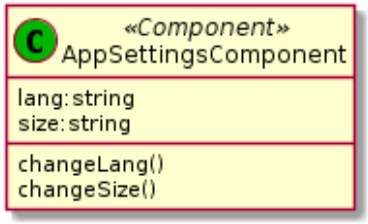


Figura 6.31: AppSettingsComponent class

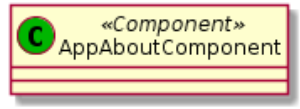


Figura 6.32: AppAboutComponent class


Figura 6.33: AppHelpComponent class


Figura 6.34: AppComponent class

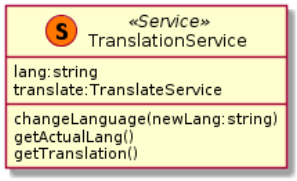


Figura 6.35: TranslationService class


Figura 6.36: PlayerService class

con los servicios asociados. Interactúa de forma activa con el PlayerService y recibe datos del MusicLoaderService y del HelpService. Detallada en la Figura 6.26.

- AppTabsComponent: Se encarga de gestionar el menú de pestañas. También gestiona los eventos necesarios para cargar música desde el exterior arrastrando. Por tanto necesita comunicarse de forma activa con el MusicLoaderService. Detallada en la Figura 6.27.

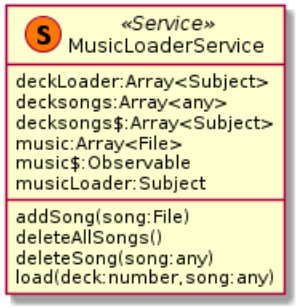


Figura 6.37: MusicLoaderService class

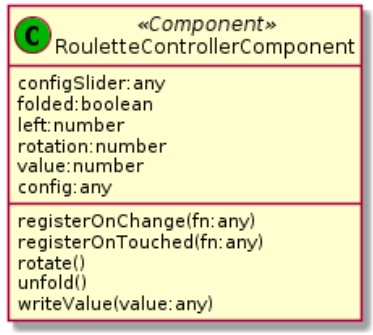


Figura 6.38: RouletteControllerComponent class

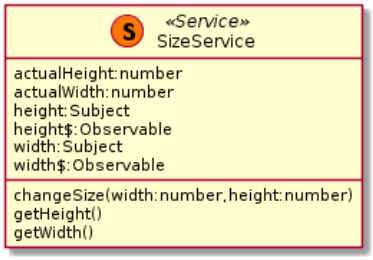


Figura 6.39: SizeService class

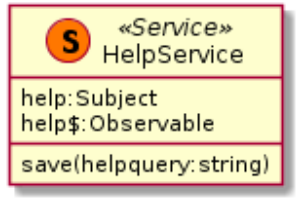


Figura 6.40: HelpService class

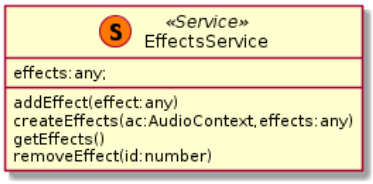


Figura 6.41: EffectsService class

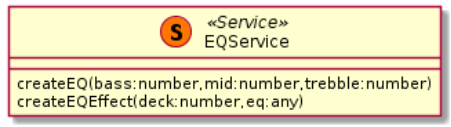


Figura 6.42: EQService class

- AppMusicListComponent: Se encarga de gestionar los eventos de Javascript necesarios para cargar música en una platina arrastrando desde la música disponible, así como de conectar los componentes visuales de la lista de música y los servicios. También se comunica de forma activa con el MusicLoaderService. Detallada en la Figura 6.24.

- AppEffectsSelectorComponent: Se encarga de conectar los componentes visuales de selección de efectos con los servicios asociados. Se comunica de forma activa con el PlayerService y recibe datos del EffectsService. Detallada en la Figura 6.29.

- AppEffectsCreatorComponent: Se encarga de conectar los componentes visuales del formulario de creación de efectos con los servicios asociados. Se comunica activamente con el EffectsService y recibe datos del PlayerService. Detallada en la Figura 6.25.

- AppSettingsComponent: Se encarga de conectar los componentes visuales de ajustes con los servicios asociados. Se comunica de forma activa con el TranslationService y con el SizeService. Detallada en la Figura 6.31.
- AppAboutComponent: No hace nada, pues es un componente puramente visual. Detallada en la Figura 6.32.
- AppHelpComponent: Se encarga de conectar los componentes visuales de ayuda con los servicios asociados. Interactúa de forma activa con el HelpService. Detallada en la Figura 6.33.
- SlideControllerComponent: Este componente es un caso especial. Su template es un recubrimiento para el componente input de tipo range (básico de HTML), que permite darle estilo. Es necesario porque cada navegador implementa el componente de una forma distinta y esto imposibilita darle estilos con facilidad. La clase implementa la lógica necesaria para adaptar el recubrimiento pero no necesita comunicarse con ningún servicio. Detallada en la Figura 6.28.
- RouletteControllerComponent: De forma similar al componente SlideControllerComponent añade un recubrimiento a mayores para darle un estilo diferente y añadir un desplegable. La clase sólo implementa esta lógica y tampoco necesita de ningún servicio. Detallada en la Figura 6.38.

Las clases de los servicios representan los componentes de tipo servicio Angular explicados anteriormente.

La directiva presente es:

- AngularDraggableDirective: Es una directiva copiada del repositorio angular2-draggable. Se necesitó realizar algunas modificaciones para adaptar el comportamiento.

## 6.4. Organización del código

La organización de código final ha quedado así:

```
.
├── package.json
├── package-lock.json
├── public
│   +── ...
├── README.md
├── src
│   ├── app
│   │   ├── app.component.ts
│   │   ├── app.module.ts
│   │   ├── app-routing.module.ts
```

```
|    |    |-- ...
|    |    |-- components
|    |    |    |-- app-about
|    |    |    |    |-- app-about.component.css
|    |    |    |    |-- app-about.component.html
|    |    |    |    +-- app-about.component.ts
|    |    |    +-- ...
|    |    |-- data
|    |    |    +-- ...
|    |    |-- directives
|    |    |    +-- ...
|    |    |-- form-components
|    |    |    |-- roulette-controller
|    |    |    +-- ...
|    |    |-- scripts
|    |    |    +-- beats.script.ts
|    |    +-- services
|    |         |-- effects.service.ts
|    |         +-- ...
|    |-- assets
|    |    |-- cssreset.css
|    |    |-- i18n
|    |    |    |-- en.json
|    |    |    +-- es.json
|    |    |-- images
|    |    |    |-- ...
|    |    +-- testing
|    |         |-- testSong1.mp3
|    |         +-- testSong2.m4a
|    |-- browserslist
|    |-- environments
|    |    |-- environment.prod.ts
|    |    +-- environment.ts
|    |-- index.html
|    |-- styles.css
|    +-- ...
+-- ...
```

En la raíz podemos ver algunos fichero de configuración y un README así como los directorios `public` y `src`.

El directorio `public` contiene la documentación generada por CompoDoc y tiene este nombre para facilitar la creación de la documentación con las páginas de GitLab. En caso de haber realizado una compilación también existiría el directorio `dist`, en el que estaría el artefacto listo para el despliegue.

En el directorio `src` vemos tres directorios más:

- `environments`, que contiene los entornos para la compilación de la aplicación.
- `app`, que contiene el código fuente del proyecto.
- `assets`, que contiene imágenes, hojas de estilos, traducciones y otros ficheros necesarios.

También hay otros ficheros de configuración, así como el HTML y el CSS principal.

Dentro de `app` tenemos los archivos necesarios del componente AppComponent, un fichero de enrutado, que no tiene uso en este proyecto y el modulo principal. También nos encontramos con varios directorios:

- `components`: contiene los componentes principalmente visuales.
- `data`: contiene ficheros de datos que se usan en uno o varios componentes (para evitar sobrecargar el código de estos).
- `directives`: contiene la directiva AngularDraggableDirective.
- `form-components`: contiene los dos componentes 'especiales' (SlideControllerComponent y RouletteControllerComponent) que no son componentes de la interfaz si no que sirven de apoyo a estos.
- `scripts`: contiene un script usado para calcular los beats per minute y el array de beats de una canción. Es necesario tenerlos fuera de los componentes pues los web-workers de HTML 5 requieren código totalmente externo a nuestros componentes.
- `services`: contiene los servicios de nuestra aplicación.

Los directorios `components` y `form-components` tienen un subdirectorio por cada componente. Dentro de cada uno de estos subdirectorios tenemos la clase Typescript, el template HTML y si es necesario el archivo de estilos CSS y el archivo de test unitario de ese componente.

## 6.5. Despliegue

En cuanto al despliegue, hay que tener en cuenta que finalizada la implementación no se tenía ninguna funcionalidad dependiente de BackEnd. Por lo tanto para desplegar basta con tener un servidor web con los archivos compilados de Angular, como se representa en la Figura 6.43.

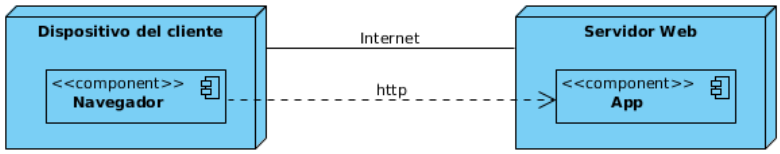


Figura 6.43: Diagrama de despliegue

# Capítulo 7

# Implementación

## 7.1. CI/CD

Antes de empezar el desarrollo se preparó un pipeline de integración continua y despliegue continuo, para automatizar algunos procesos necesarios para la ingeniería de software. Esto se realizo con el formato usado por GitLab. El fichero final de integración continua es el siguiente:

```
image: node:latest

cache:
  paths:
    - node_modules/

test_unit:
  before_script:
    - apt-get update -yqqq
    - apt-get install -y xvfb
    - apt-get install iceweasel -yqq
    - apt-get install fonts-liberation libappindicator3-1
      ↪ libasound2 libatk-bridge2.0-0 libatspi2.0-0 libgtk
      ↪ -3-0 libnspr4 libnss3 libxss1 libxtst6 lsb-release
      ↪ xdg-utils -y
    - wget https://dl.google.com/linux/direct/google-chrome-
      ↪ stable_current_amd64.deb
    - dpkg -i google-chrome-stable_current_amd64.deb; apt-get -
      ↪ fy install
    - Xvfb :99 -ac &
    - export DISPLAY=:99
    - npm install --silent
```

```
    - ./node_modules/.bin/webdriver-manager update --versions.
      ↪ gecko=v0.17.0
  script:
    - npm run test:ci
# test_e2e:
#   before_script:
#     ...
#   script:
#     - npm run e2e

deploy_stage:
  stage: deploy
  environment: Stage
  only:
    - develop
  before_script:
    - npm install --silent
  script:
    - rm ./package-lock.json
    - ./node_modules/@angular/cli/bin/ng build --progress false
      ↪  --prod --base-href http://virtualdj-stage.surge.sh
    - ./node_modules/.bin/surge -p dist/TFG --domain virtualdj-
      ↪ stage.surge.sh

deploy_prod:
  stage: deploy
  environment: Production
  only:
    - master
  before_script:
    - npm install --silent
  script:
    - rm ./package-lock.json
    - ./node_modules/@angular/cli/bin/ng build --progress false
      ↪  --prod --base-href http://virtualdj.surge.sh
    - ./node_modules/.bin/surge -p dist/TFG --domain virtualdj.
      ↪ surge.sh

pages:
  stage: deploy
  script:
    - echo 'Nothing to do...'
  artifacts:
    paths:
      - public
  only:
    - master
```

### 7.1.1. Preparación

El primer parámetro que vemos es image:

```
image: node:latest
...
```

Esto le indica al worker de GitLab qué imagen docker debe coger para desplegar el entorno de CI y CD. En este caso cogemos node:latest. Esto coge la última versión del contenedor oficial node [60]. Este suele ser un linux bastante limpio con lo necesario para correr node desde el primer momento.

El siguiente parámetro es cache:

```
...
cache:
  paths:
    - node_modules/
...
```

Esto le indica al worker que tras la ejecución debe guardar la carpeta en caché y al principio de la ejecución debe cargar esa copia (si esta disponible).

Después de eso tenemos los trabajos de CI y CD.

### 7.1.2. Trabajos de testing

Hay dos trabajos de testing pero uno de ellos esta desactivado por ahora. El trabajo disponible es test_unit:

```
...
test_unit:
  before_script:
    - ...
  script:
    - ...
```

Se ha decidido separar cada trabajo en dos partes por claridad. La preparación del script y el script en sí. El parámetro before_script indica que debe hacerse antes de ejecutar el script. En este apartado se suelen instalar los paquetes necesarios, tanto del sistema operativo como de npm.

En nuestro caso lo primero que se hace es actualizar la lista de paquetes y dependencias y después actualizar los paquetes:

```
...
    - apt-get update -yqqq
    - apt-get install -y xvfb
...
```

Tras eso se instala iceweasel, que es un fork de Firefox que permite ejecutar los tests.

```
...
    - apt-get install iceweasel -yqq
...
```

También se instala Chrome (y todos los paquetes necesarios), para tener dos navegadores distintos a la hora de realizar tests:

```
...
    - apt-get install fonts-liberation libappindicator3-1
      ↪ libasound2 libatk-bridge2.0-0 libatspi2.0-0 libgtk
      ↪ -3-0 libnspr4 libnss3 libxss1 libxtst6 lsb-release
      ↪ xdg-utils -y
    - wget https://dl.google.com/linux/direct/google-chrome-
      ↪ stable_current_amd64.deb
    - dpkg -i google-chrome-stable_current_amd64.deb; apt-get -
      ↪ fy install
...
```

Una vez tenemos los dos navegadores instalados debemos crear una instancia de X virtual framebuffer (un servidor X11 que ejecuta todas las operaciones gráficas en memoria, sin mostrar nada por pantalla) y decirle a nuestro sistema que es nuestro display principal. Esto es necesario para poder ejecutar un navegador en un worker sin display.

```
...
    - Xvfb :99 -ac &
    - export DISPLAY=:99
...
```

Hecho esto solo queda instalar los paquetes npm y actualizar el paquete webdriver-manager para usar una versión compatible con el Chrome y Firefox que hemos instalado.

```
...
    - npm install --silent
    - ./node_modules/.bin/webdriver-manager update --versions.
      ↪ gecko=v0.17.0
...
```

Ahora ya estamos listos para ejecutar los tests unitarios. El comando test:ci ejecuta los tests sin el parámetro watch y sin el parámetro progress, para evitar sobrecargar los logs.

```
  ...
  script:
    - npm run test:ci
```

El trabajo de los tests e2e sería similar en el apartado before_script y solo se cambiaría el comando final.

```
  ...
  script:
    - npm run e2e
```

En estos trabajos no hemos visto el parámetro stage. Este indica en qué etapa debe ejecutarse el comando, pero por defecto stage vale test por tanto en los trabajos de test no se necesita.

### 7.1.3. Trabajos de despliegue

Tenemos dos trabajos de despliegue de la aplicación y un trabajo de despliegue de la documentación. Los dos trabajos de despliegue de la aplicación son muy similares:

```
deploy_stage:
  stage: deploy
  environment: Stage
  only:
    - develop
  before_script:
    - npm install --silent
  script:
    - rm ./package-lock.json
    - ./node_modules/@angular/cli/bin/ng build --progress false
      ↪  --prod --base-href http://virtualdj-stage.surge.sh
    - ./node_modules/.bin/surge -p dist/TFG --domain virtualdj-
      ↪ stage.surge.sh

deploy_prod:
  stage: deploy
  environment: Production
  only:
    - master
  before_script:
    - npm install --silent
  script:
    - rm ./package-lock.json
    - ./node_modules/@angular/cli/bin/ng build --progress false
      ↪  --prod --base-href http://virtualdj.surge.sh
```

```
    - ./node_modules/.bin/surge -p dist/TFG --domain virtualdj.
      ↪ surge.sh
```

Lo primero que vemos es el parámetro stage. Como queremos que solo se despliegue si la etapa de tests ha sido un éxito aquí debemos elegir la etapa deploy, que va después.

```
...
  stage: deploy
...
```

Después vemos el parámetro environment. Este sirve para crear un entorno en GitLab (Figura 7.1) que más tarde se puede consultar (Figura 7.2):

```
...
  environment: Stage
...
  environment: Production
...
```

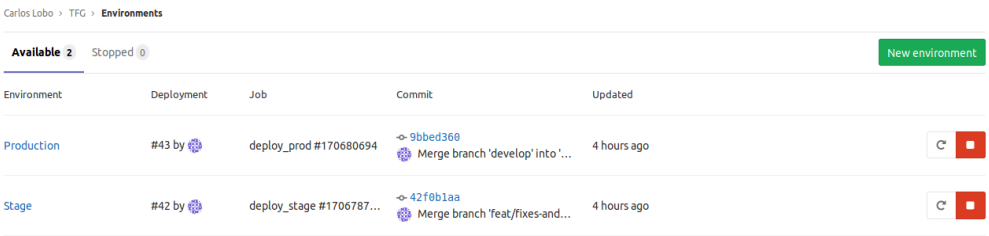


Figura 7.1: Entornos de gitlab

Tras el parámetro environment tenemos el parámetro only. Este nos especifica las ramas o tags con las que ejecutar este trabajo. En nuestro caso deploy_stage solo se ejecuta para develop y deploy_prod para master.

```
...
deploy_stage:
  ...
  only:
    - develop
deploy_prod:
  ...
  only:
    - master
...
```

Una vez configurados estos trabajos, la ejecución es muy simple. Antes del script se instalan los paquetes npm necesarios y tras esto se construye la aplicación y mediante el binario ofrecido por surge se despliega cada una en el dominio elegido.

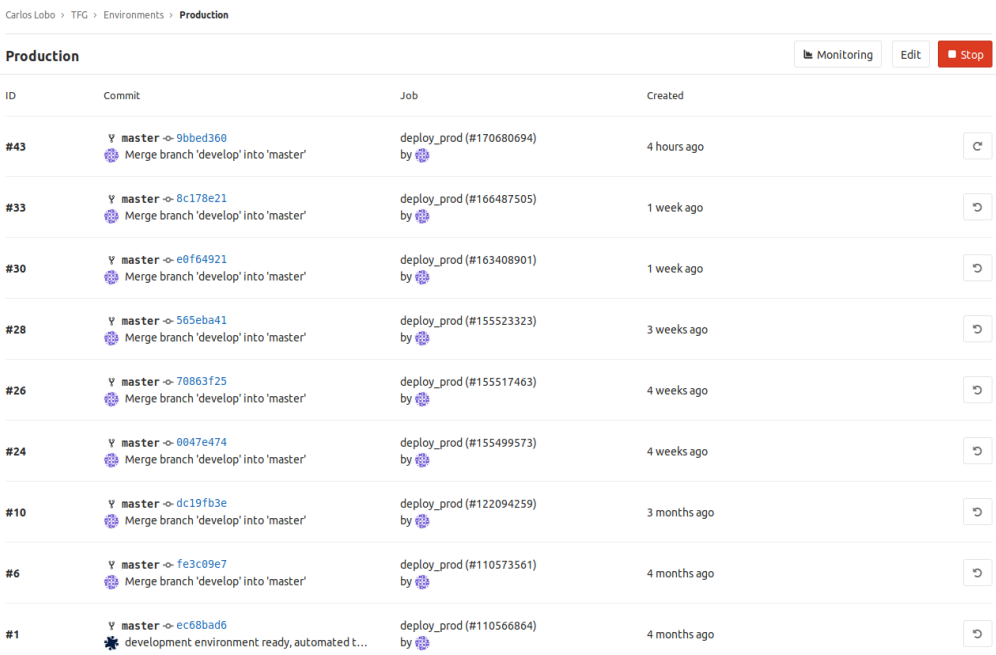


Figura 7.2: Entorno producción

```
...
  before_script:
    - npm install --silent
  script:
    - rm ./package-lock.json
    - ./node_modules/@angular/cli/bin/ng build --progress false
      ↪  --prod --base-href http://virtualdj-stage.surge.sh
    - ./node_modules/.bin/surge -p dist/TFG --domain virtualdj-
      ↪ stage.surge.sh
...
```

El tercer trabajo de la etapa de despliegue es un trabajo propio de GitLab que se encarga de desplegar la documentación siempre que se hace un push a master. No hace nada pues solo crea el artefacto que después GitLab despliega en tu repositorio.

```
pages:
  stage: deploy
  script:
    - echo 'Nothing to do...'
  artifacts:
    paths:
      - public
```

```
  only:
    - master
```

## 7.2. Bocetos

El primer paso en el desarrollo de una WebApp es la realización de bocetos de la interfaz gráfica de usuario (Figura 7.3). Para una aproximación inicial se tienen 2 decks (Figura 7.4) en los que reproducir música, un controlador de volumen (Figura 7.5), una búsqueda de Youtube y una lista de música En los decks estaría el control del pitch, la onda de sonido, los

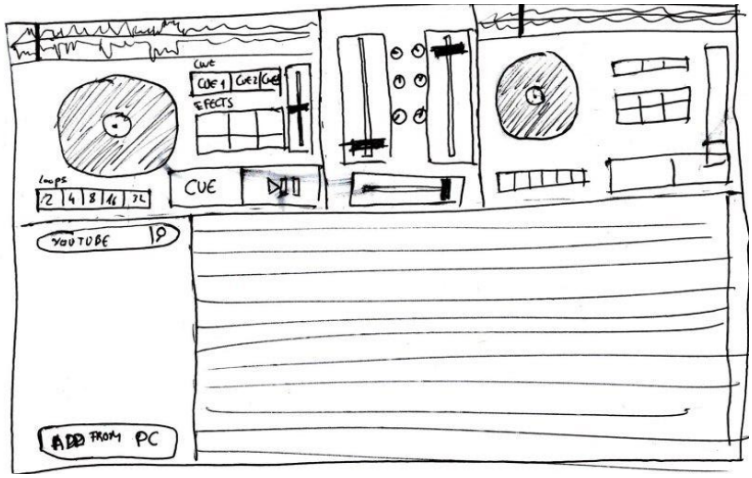


Figura 7.3: Boceto de interfaz gráfica de usuario

botones de reproducción/pausa y de crear CUEs, los botones de loops, los botones de efectos y los botones de retomar CUEs.

## 7.3. Desarrollo de la interfaz de usuario

Durante la creación de la interfaz han surgido algunos problemas. Por ejemplo, es difícil dar estilos a componentes básicos de HTML, lo que resulta en la creación de un componente Angular específico para poder, mediante el uso de JavaScript, dar el estilo deseado. Esto ha resultado en la creación de dos componentes, slider controller (Figura 7.6) y roulette controller ( Figura 7.7 y Figura 7.8).

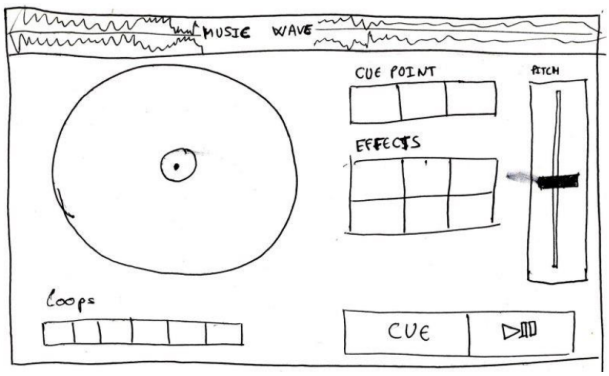


Figura 7.4: Detalle de boceto de interfaz gráfica, deck

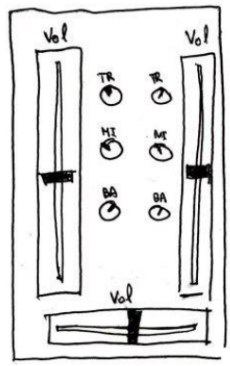


Figura 7.5: Detalle de boceto de interfaz gráfica, controlador de volumen

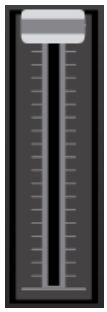

Figura 7.6: Slider controller

Figura 7.7: Roulette controller a) al máximo

Figura 7.8: Roulette controller b) en el medio

### 7.3.1. Implementación inicial

Durante la implementación inicial se ha modificado una de las ideas de los bocetos originales. En el apartado de la lista de música se han añadido pestañas y 2 secciones nuevas. Una será para la configuración de diversos parámetros (lenguaje, aspecto, efectos elegidos para cada botón de efectos) y una pestaña de información, en la que se mostrará información sobre la aplicación, así como información sobre atribución de contenidos.

La implementación inicial de la interfaz, a falta de la representación de las ondas (que se hará cuando se reproduzca música):

- Deck o platina, Figura 7.9.
- Volumen, sin texto por encima de las ruletas, Figura 7.10.

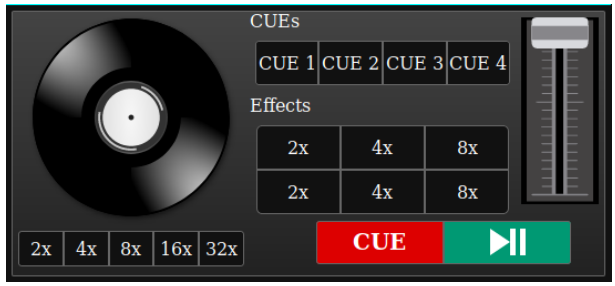


Figura 7.9: Implementación inicial del deck

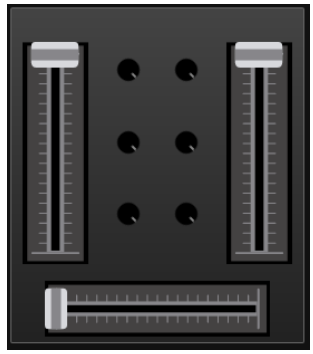

Figura 7.10: Implementación inicial del volumen

- Búsqueda, Figura 7.11.


Figura 7.11: Implementación inicial de la búsqueda

- Lista de música, Figura 7.12.
- Configuración, Figura 7.13.

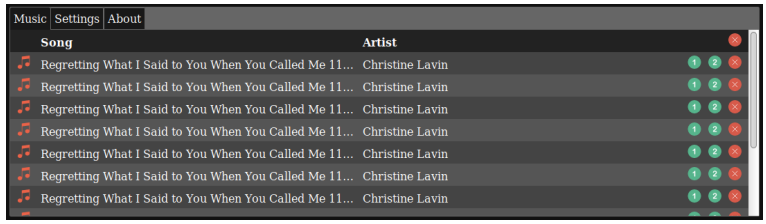


Figura 7.12: Implementación inicial de la lista de música

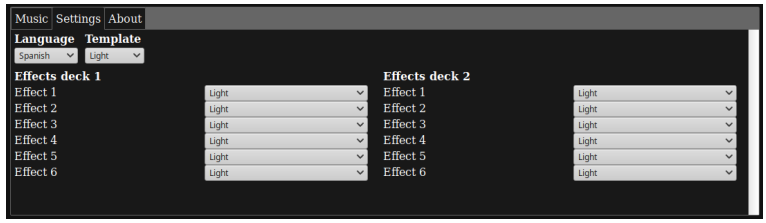


Figura 7.13: Implementación inicial de la configuración

- About, Figura 7.14.

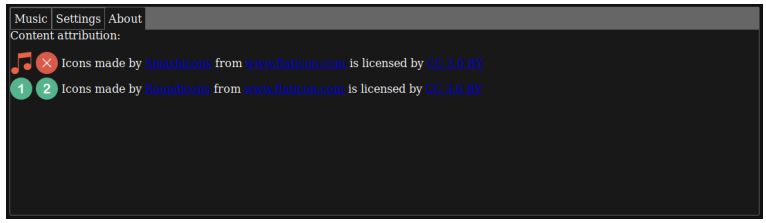


Figura 7.14: Implementación inicial de about

## 7.4. Carga y reproducción de música

Las primeras tareas realizadas tras tener la interfaz de usuario han sido la carga y la reproducción de música:

- Carga de música: La carga de música inicial se realiza desde el ordenador local. Para esto se ha creado un servicio de carga de música que se ha conectado a los componentes de interfaz. La carga se puede realizar pulsando un botón desde la búsqueda de música o bien arrastrando música a la lista de música. Una vez cargada la música se ve aparecer en la lista de música y se podrá cargar en uno de los dos decks.
- Reproducción: Para la reproducción se ha creado una instancia de Wavesurfer para cada deck, permitiendo reproducir la música. Durante este proceso han surgido proble-

mas, pues la compatibilidad entre Angular y Wavesurfer no está del todo pulida, pues Wavesurfer intenta modificar el HTML manualmente, pero Angular crea los elementos de forma dinámica, lo que trae problemas a la hora de dibujar la onda de sonido.

Esto resulta en la necesidad de crear la instancia de Wavesurfer dentro de una llamada a la función `requestAnimationFrame` [44] después de iniciar el componente (en el hook `ngAfterViewInit`).

Una vez resuelto este problema y creada la instancia, ésta se guarda en el servicio PlayerService, que es el que se encarga de permitir a los distintos componentes comunicarse con el reproductor de Wavesurfer. Esto facilitará futuras tareas y además simplifica bastante el código.

### 7.4.1. Vistas renovadas

Se han realizado algunas mejoras en la vista de la lista de música:

- Cuando no hay ninguna canción se ven unas breves instrucciones de uso. Se puede ver en la Figura 7.15.

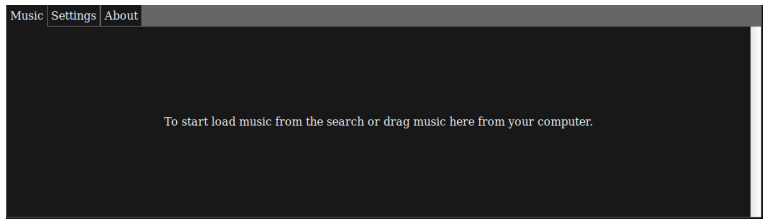


Figura 7.15: Lista de música vacía

- Se ha añadido una caja de búsqueda para filtrar las canciones. Se puede ver en la Figura 7.16.

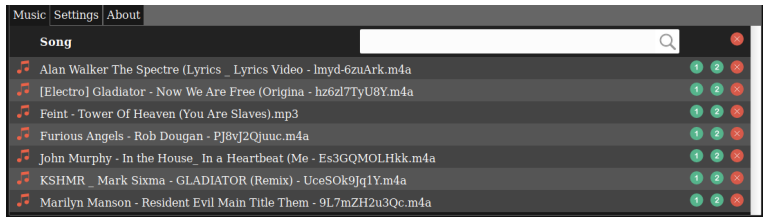


Figura 7.16: Lista de música con búsqueda

- Se ha creado una vista para cuando se están arrastrando archivos por encima de la página. Se puede ver en la Figura 7.17.

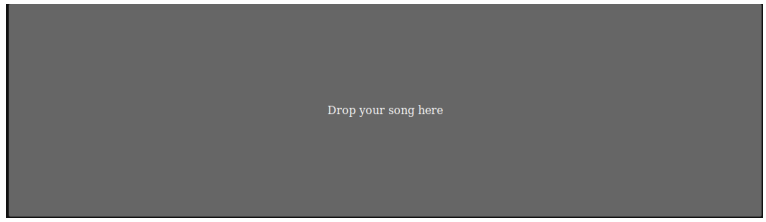


Figura 7.17: Lista de música mientras arrastras

También se ha mejorado el aspecto visual de los decks, para incluir la onda de la canción que se está reproduciendo, así como el título.

## 7.5. Efectos y volumen avanzado

Durante la planificación del quinto sprint se ha elegido trabajar en estas dos funcionalidades a la par porque el volumen avanzado es en realidad 10 efectos aplicados juntos (se ha separado la onda de sonido en varios intervalos de frecuencia desde graves hasta agudos y por cada uno de ellos se ha aplicado un filtro bicuadrado).

- Volumen avanzado: Se empezó creando un servicio para la gestión de efectos, en él se crea la estructura que indica las frecuencias y el tipo y potencia del filtro:

```
[
    {
      f: 32,
      type: 'lowshelf',
      value: 0
    },
    {
      f: 64,
      type: 'peaking',
      value: -10
    },
    ...
]
```

  Esta estructura se modifica desde el componente de volumen. Una vez se tiene esta estructura en el servicio de reproducción se usa para crear un array de efectos de WebAudio (se necesita la instancia del reproductor, por eso se guarda en el servicio de reproducción). Una vez se tiene el array de efectos se aplica a la reproducción.

- Efectos: Para crear los efectos se ha hecho uso de la biblioteca tuna. Esta biblioteca permite crear efectos de WebAudio de forma simple. Dado que se ha pensado en poder implementar la creación de efectos personalizables (modificando parámetros de los efectos que provee tuna) se ha creado una estructura de datos:

```
[
    {
      name: 'Basic moog filter',
      type: 'MoogFilter',
      id: 1,
      active: false,
      config: {
        cutoff: 0.065, // 0 to 1
        resonance: 3.5, // 0 to 4
        bufferSize: 4096 // 256 to 16384
      }
    },
    {
```

```
            name: 'Basic ping pong',
            type: 'PingPongDelay',
            id: 2,
            active: false,
            config: {
              cutoff: 0.065, // 0 to 1
              resonance: 3.5, // 0 to 4
              bufferSize: 4096 // 256 to 16384
            }
          },
          ...
    ]
```

Donde name es el nombre que se muestra en la selección de efectos, type es el tipo de efecto (en tuna), el id es un numero único para facilitar la selección y búsqueda de efectos, active indica si está activo y config es la configuración que necesita tuna para crear el efecto.

Una vez se tiene el servicio de efectos para trabajar con esta estructura, es fácil crear efectos nuevos (tarea a realizar en los siguientes sprints).

## 7.6. Creación de efectos

### 7.6.1. Estructura de datos

Durante el sexto sprint se realizó la creación de efectos personalizables. Para ello lo primero que se hizo fue crear una estructura de datos que describiera los efectos:

```
[
    {
      name: 'Chorus',
      effect: 'Chorus',
      description: 'A basic chorus effect.',
      configs: [
        { name: 'rate', min: 0.01, max: 8, default: 1.5 },
        { name: 'feedback', min: 0, max: 1, default: 0.2 },
        { name: 'bypass', values: [0, 1], default: 0 },
        ...
      ]
    },
    ...
]
```

En ella vemos un nombre, que se mostrará en el creador de efectos, así como una descripción del efecto. El campo effect es el nombre por el que conoce Tuna los efectos y el campo

de configs es un array con las distintas configuraciones para cada efecto. Estas configuraciones pueden ser un número en un rango o un selector de posibles valores. En caso de ser un número dentro de un rango posee mínimo, máximo, número por defecto y opcionalmente step (unidad mínima aceptable en el número) que tiene 0.01 como valor por defecto. En caso de ser un selector tendríamos un array con los posibles valores y el valor elegido por defecto.

El objetivo de crear esta estructura de datos es delimitar cómo debe ser la estructura que describe el efecto (para que sea igual a la estructura descrita en la sección anterior).

### 7.6.2. Mejoras a la interfaz

Antes de empezar a crear un formulario de creación/eliminación de efectos se decidió añadir una mejora a la interfaz: la posibilidad de ampliar los menús. Esto no consigue una mejora instantánea, pero se valoró que para la creación de efectos resultaría útil o incluso necesario. Las mejoras se pueden ver en la Figura 7.18 y la Figura 7.19.

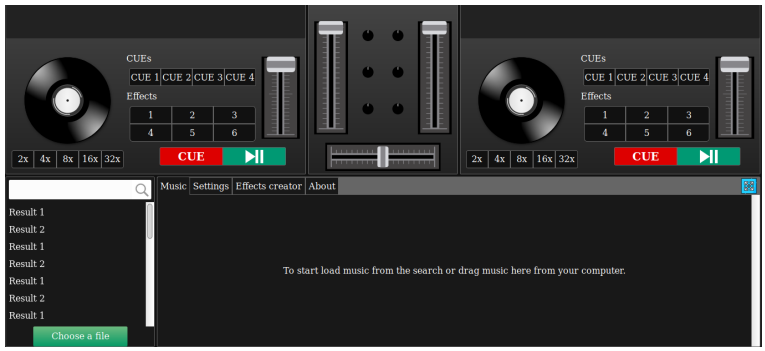


Figura 7.18: Menús con tamaño normal

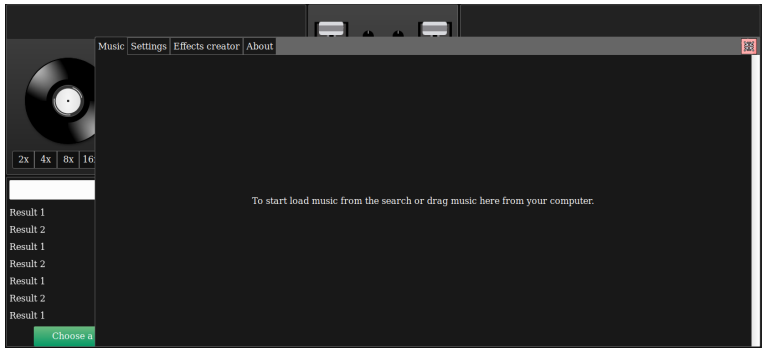


Figura 7.19: Menús maximizados

### 7.6.3. Formulario de creación

Tras realizar esta modificación se procedió al desarrollo del formulario de creación de efectos. Éste deberá pedir primero el tipo de efecto que se quiere crear. Una vez seleccionado, deberá pedir un nombre y valores para la configuración del efecto. También se añadió un formulario para eliminar efectos. La interfaz de estos formularios se puede ver en la Figura 7.20, la Figura 7.21 y la Figura 7.22.


Figura 7.20: Crear efectos

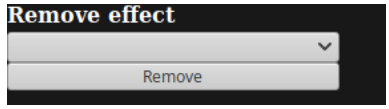


Figura 7.21: Eliminar efectos 1

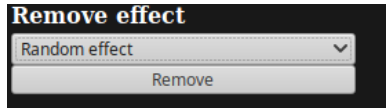


Figura 7.22: Eliminar efectos 2

## 7.7. Descarga de música desde plataformas

Durante el sprint 7 se ha investigado la descarga de música desde plataformas como Youtube, Soundcloud y otras alternativas.

### 7.7.1. Limitaciones legales

Durante la investigación se han leído términos y condiciones de Youtube [47]. En ellos se puede ver que no está permitida la descarga de contenido a no ser que se posea el permiso directo de Youtube o de los propietarios legales del contenido descargado, o que el contenido

en cuestión esté exento de copyright. Por tanto debería añadirse en los términos de uso de nuestra aplicación esta limitación.

Esto añade un gran problema a la descarga de música, pues el contenido sin copyright (más fácilmente accesible) es bastante limitado.

También se revisaron los términos y condiciones de Soundcloud [48] encontrando directamente la prohibición de uso del contenido en webs externas:"No debe utilizar scraping o técnicas similares para agregar, reconvertir, volver a publicar o hacer cualquier otro uso de ningún contenido."

Esto hace que sea inviable directamente usar Soundcloud dado que actualmente no ofrecen claves para consultar a la API directamente y la única forma de usarlo sería mediante scraping, y eso está prohibido.

### 7.7.2. Limitaciones técnicas

Habiendo descartado Soundcloud, se procedió a intentar conectar la aplicación con Youtube.

Lo primero que se intentó fue una conexión directa descargando desde Youtube un vídeo en formato mp4 y transformando a mp3. Esto fue frustrado pues Youtube no permite las peticiones desde otros sitios web mediante CORS [49]. Youtube bloquea la descarga de contenido desde cualquier fuente que no sea Youtube.

Esto supone la necesidad de añadir un servidor entre Youtube y nuestra aplicación. Para comprobar la viabilidad de la creación de un servidor se procedió a crear un servidor de prueba en Node.js, usando el mismo lenguaje que se ha usado para nuestra aplicación de Frontend, Javascript.

El problema que se encontró en ese momento fue la necesidad de descargar el vídeo sin audio, la transformación y más tarde la transferencia al cliente. Esto añadía una latencia bastante alta (varios minutos) desde que se empezaba la descarga en el servidor hasta que llegaba al cliente, y esto teniendo un solo cliente. En el caso de añadir varios clientes más en paralelo el tiempo necesario aumentaría. Esto, para la aplicación que se está diseñando, es inaceptable. También podría suponer un problema legal en caso de que algún usuario decidiera descargar contenido con copyright, pues al descargarlo previamente en nuestro servidor para después transformarlo a mp3 estaríamos incumpliendo sus términos y condiciones y descargando contenido ilegal.

### 7.7.3. Evaluación final

Tras la investigación y prueba de concepto se ha decidido descartar la funcionalidad de descarga de música desde un servidor, pues puede causar problemas legales (siempre en Soundcloud y en Youtube en caso de descargar contenido protegido por copyright), y tiene

problemas técnicos a la hora de implementarse, causando una mala experiencia de usuario a la hora de utilizarse (tardando varios minutos desde que se pide una canción hasta que consigues poder usarla en un caso básico).

Por tanto, en estos momentos solo se puede usar la carga de música desde local, que por otra parte funciona bastante rápido (tardando pocos segundos en cargar la música a nuestra interfaz tras seleccionarla), por tanto la pérdida de esta funcionalidad no es tan problemática.

### 7.7.4. Cambios a la interfaz

En vista de la eliminación de la búsqueda de música se eliminó la parte correspondiente de la interfaz, resultando la nueva interfaz vista en la Figura 7.23.

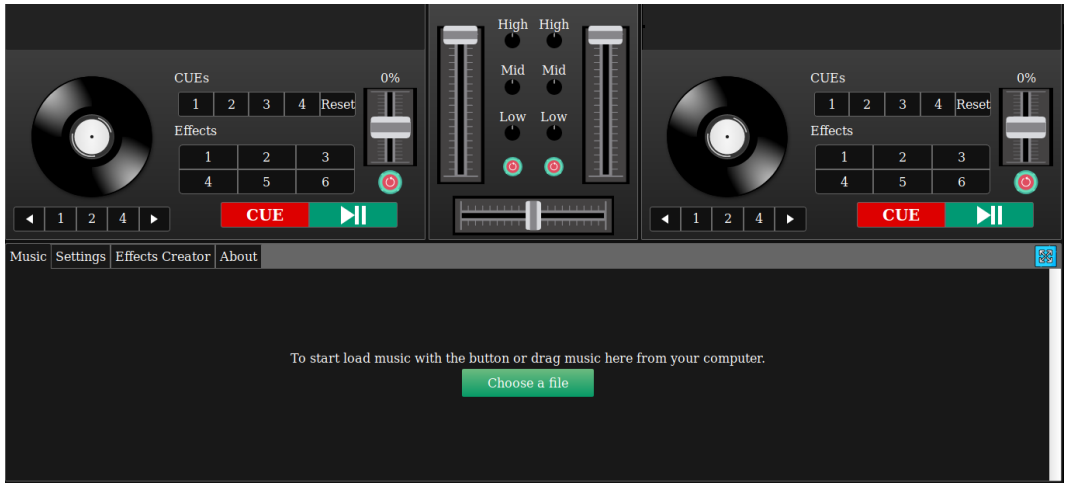


Figura 7.23: Interfaz sin búsqueda

## 7.8. Implementación del pitch

Para terminar el sprint 7 se implementó el pitch. Dado que los componentes de interfaz ya estaban preparados solo se añadieron un par de funciones que se ejecutaban al usar estos componentes.

## 7.9. Implementación de CUE, loops y retoques finales

Durante el sprint 8 se implementó el CUE. El objetivo de los CUEs es crear un punto para retomar la canción. La implementación final permite guardar un punto mediante el botón de

CUE, hasta un total de 4 puntos guardados y luego retomar la canción desde cualquiera de estos puntos mediante 4 botones, así como resetear los CUE.

También se han realizado algunos cambios a la interfaz, como añadir botones de reset, indicador del pitch actual, control de resolución y indicación de la incompatibilidad con un layout vertical.

Tras eso, se implementaron los loops, que permiten reproducir pequeños fragmentos de la canción repetidas veces.

Finalmente se ha añadido al menú una nueva pestaña que explica las diversas funcionalidades de la aplicación, con la posibilidad de mostrar cuál es cada una de las funcionalidades en la interfaz de usuario. También se ha enriquecido con aspectos de localización para mejorar la usabilidad del software con muchos más usuarios de diferente procedencia.

## 7.10. Final del desarrollo

Al finalizar el sprint 8 se había terminado el desarrollo inicial de la aplicación, teniendo un artefacto que cumplía todos los requisitos pedidos en las User Stories. Por tanto durante el siguiente sprint se pasaría al testing.

## 7.11. Licencia

Tras publicar la última versión de código en el repositorio se decidió publicar el software como software libre. Para ello se eligió la licencia GNU GPLv3 [63]. Esto se ha decidido porque no se quiere restringir al usuario que usa el software; se quiere poder ofrecer el software para que cualquiera pueda usarlo, modificarlo y compartirlo, así como compartir sus propias modificaciones.

# Capítulo 8

# Testing

Durante el sprint 9 se ha planificado realizar tests programáticos así como pruebas con usuarios.

## 8.1. Tests programáticos

Durante la preparación del entorno de desarrollo en el sprint 1 se eligió Jest como herramienta para el testing programático. Esto pareció una buena decisión pues Jest era más rápido que Karma, la otra alternativa. Al empezar a realizar los tests se descubrió que Jest no era compatible con las APIs de Canvas (usada para renderizar la onda de sonido de música) y WebAudio (usada para reproducir música) de HTML 5. Por lo tanto se volvió a preparar el entorno con Karma.

Una vez preparado se empezaron a realizar tests. Durante la realización de los primeros tests se solucionaron unos cuantos problemas de calidad de código. Angular trabaja con el DOM [56] (la API que permite desde Javascript interactuar con el HTML) de una forma determinada y espera que el desarrollador use las herramientas que Angular pone a su disposición para ello. Al no usarlas se están poniendo en practica antipatrones que conducen a errores. Estos errores no son notables al ejecutar la aplicación, pues desde que carga hasta que se empieza a usar pueden pasar algunos segundos, que permiten a Angular recuperarse de estos errores sin que el usuario se dé cuenta. En cambio a la hora de testear se intenta aprovechar el tiempo lo mas posible, empezando el testing milisegundos después de tener cargada la página. Esto resulta en errores que provocan fallos en los tests, pero que poco tienen que ver con el test en sí.

### 8.1.1. Configuraciones importantes

Dado que durante el proceso de preparación del entorno de desarrollo se prepararon trabajos de CI, se modificó la configuración de Karma para testear en dos navegadores importantes (Firefox y Chrome). El fichero de configuración de Karma, karma.conf.js, quedó así:

```
module.exports = function(config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-firefox-launcher'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output
        ↪ visible in browser
    },
    coverageIstanbulReporter: {
      dir: require('path').join(__dirname, '../coverage'),
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['FirefoxHeadless', 'ChromeHeadlessNoSandbox'],

    customLaunchers: {
      FirefoxHeadless: {
        base: 'Firefox',
        flags: ['-headless']
      },
      ChromeHeadlessNoSandbox: {
        base: 'ChromeHeadless',
        flags: ['--no-sandbox']
      }
    },
    singleRun: false
  });
```

```
};
```

Las configuraciones que se han añadido son:

- require('karma-firefox-launcher') y require('karma-chrome-launcher'): Esto carga configuraciones necesarias para poder usar los dos navegadores elegidos.
- customLaunchers: Esto permite crear instancias de navegadores con una configuración personalizada. En este caso se usó para poder abrir los navegadores en modo headless y sandbox.
- browsers: Esto permite elegir qué navegadores utilizar, a la hora de realizar la batería de tests.

### 8.1.2. Resultados de los test programáticos

El código asociado a tests unitarios se encuentra en la carpeta propia de cada componente. Los tests e2e, de haberlos, se encontrarían en una carpeta llamada `e2e` a la altura del código fuente.

En total se realizaron 20 tests unitarios y 0 tests e2e.

Cabe destacar un test asociado a la reproducción de canciones. Gracias a este test se descubrió el uso de antipatrones. La implementación original de la reproducción estaba modificando el DOM directamente. Esto causaba errores de los cuales Angular podía recuperarse, resultando invisibles al usuario, pero el uso continuado de las funciones acababa resultando en fallos graves de memoria. Tras investigar, se sustituyeron las modificaciones directas del DOM por funciones especificas de Angular, resolviendo el problema.

## 8.2. Pruebas con usuarios

Para realizar las pruebas con usuario primero hay que preparar la metodología y un formulario de preguntas que realizar. La metodología usada sería:

- Realizar preguntas sobre el perfil de usuario (conocimiento sobre este tipo de software). Las preguntas realizadas son:
  - ¿Conocías este tipo de software?
  - ¿Cuántas veces has usado este tipo de software?
- Dejar entre 5 y 10 minutos al usuario con el software para probarlo. Esto permite evaluar cómo de intuitivo es de usar el software.

- Pedir al usuario que realice distintas acciones con el software evaluando la facilidad con la que los realiza. Se evalúa del 1 al 5 siendo 1 la no realización de la acción y 5 la realización inmediata. Las acciones que se pide al usuario:
  - Cargar una canción en la lista de música.
  - Cargar otra canción (cuando cambia de lugar el botón).
  - Utilizar la búsqueda de canciones.
  - Cargar una canción en una platina.
  - Reproducir la canción.
  - Cambiar el volumen de una platina.
  - Cambiar el volumen de balance entre platinas.
  - Modificar el volumen de los ecualizadores.
  - Resetear los ecualizadores.
  - Utilizar los CUEs.
  - Modificar el pitch.
  - Resetear el pitch.
  - Usar los bucles.
  - Usar un efecto.
  - Crear un nuevo efecto.
  - Modificar los efectos disponibles.
  - Modificar el idioma.
- Una vez ha probado el software realizar varias preguntas y se apunta su respuesta:
  - ¿Cómo consideras la disposición de objetos?
  - ¿Has visto la pestaña de ayuda? ¿Te ha resultado útil?
  - Sugerencias de mejora
  - Opinión del usuario
  - Otro (esta parte se reservaba para apuntar errores o bugs encontrados.

### 8.2.1. Resultados del testing con usuarios

Los resultados más inmediatos del testing fueron la resolución de bugs y la implementación de mejoras.

Se han descubierto y corregido múltiples bugs, como son:

- El plato se para después de cargar la canción.
- Si el usuario crea un efecto, lo carga en los efectos disponibles, lo usa y mientras está en uso lo borra, el programa deja de responder correctamente.

- Si el usuario intenta arrastrar imágenes la lista de música deja de funcionar correctamente.
- Si el usuario cambia la resolución a una resolución en la que aparece el mensaje de advertencia, al volver a una resolución válida crearía un nuevo componente visual. Pero el anterior seguiría existiendo, resultando en un fallo de memoria.

También se recibieron algunas sugerencias interesantes, las cuales se implementaron:

- Separar la selección de efectos de la pestaña de ajustes, para hacerlo más claro
- El botón de reset no estaba lo suficientemente claro
- Añadir un título y un favicon (icono presente al lado del nombre en las pestañas de los navegadores)
- Añadir texto para indicar cuáles son los loops
- Un botón de ampliación más intuitivo
- Indicación en la pestaña de ayuda que se puede pulsar
- Eliminación de la restricción de número de caracteres en los nombres de efectos
- Botones en la pestaña de ayuda más claros
- Carga de canciones arrastrando desde la lista
- Botones para silenciar y maximizar el volumen

Por otra parte tras realizar las pruebas se procedió a estudiar los resultados.

Se realizaron pruebas un total de 18 usuarios. En resumen:

- El 83,3 % (15) de los usuarios conocía este tipo de software
- En total contaríamos con 6 usuarios que hemos calificado como expertos (han usado este tipo de software al menos 10 veces) y con 7 usuarios totalmente novatos (nunca han usado este tipo de software).
- Todos los usuarios han completado al instante las tareas de cargar música, búsqueda, carga en la platina, reproducción, cambio de volumen de platinas y reset del pitch.
- Las tareas más complicadas, que al menos un usuario no fue capaz de completar son, modificar los ecualizadores y usar los loops.
- En las demás tareas no hubo mucha complicación en general, con un máximo de 2 o 3 usuarios no completándolas de forma instantánea.
- En general la disposición se considera correcta (esto es especialmente importante en los usuarios expertos, pues resulta similar al software que han utilizado).

- La pestaña de ayuda a veces pasaba desapercibida o resultaba innecesaria, pero para los usuarios novatos fue útil.
- La opinión en general es buena. Se coincide en que es fácil de usar, intuitivo y como software de iniciación está bien.

En general las pruebas salieron bien pues se pudo ver que la mayoría de funcionalidades son intuitivas. También se descubrieron múltiples bugs y se añadieron algunas funcionalidades interesantes.

# Capítulo 9

# Conclusión y líneas de trabajo futuras

## 9.1. Conclusiones

Tras la finalización del tiempo marcado para el proyecto se puede decir que el desarrollo ha sido satisfactorio. Aunque no se han podido completar todos los objetivos planeados originalmente, no se debe a fallos en la planificación, si no a la imposibilidad de implementación. Volviendo atrás, si miramos los objetivos iniciales:

- Múltiples decks (2 a 4): Se han implementado 2 decks.
- Nivel medio de personalización (volumen general y específico de graves, agudos y medios, loops, pitch, efectos): Este objetivo se ha conseguido con creces, pues se ha añadido la posibilidad de crear efectos personalizables.
- Webapp: Es una webapp e incluso está desplegada a día 16 de mayo de 2019 aquí.
- Carga de música local y desde plataformas musicales (en un principio Youtube): Por desgracia debido a los términos y condiciones de Youtube o Soundcloud, solo se ha podido cargar música desde el ordenador local.
- Compatibilidad táctil al menos para resolución alta (al menos 1024x768): Se ha conseguido la compatibilidad en dispositivos táctiles (probado en un ordenador con pantalla táctil).

En cuanto a los objetivos personales, se han logrado todos:

- Mejorar mi entendimiento de SCRUM: Completar un proyecto del que se ha realizado un seguimiento semanal me ha ayudado a entender cómo se trabaja de esta forma.

- Reforzar mis conocimientos de desarrollo *Frontend*: he aprendido conceptos sobre WebAudio, testing unitario web y he mejorado mi conocimiento sobre Angular y Typescript.
- Familiarizarme con todos los pasos a seguir durante el desarrollo de un software: Aunque no me he sumergido de lleno en todos los pasos a seguir, GitLab me ha permitido obtener un conocimiento básico sobre éstos durante la realización del proyecto. También ha despertado mi interés en estos procesos para proyectos futuros.

## 9.2. Mejoras futuras

Las mejoras futuras más claras serían aquellas citadas en la introducción :

- Versión para escritorio: Con Electron esto no sería muy complicado.
- Efectos personalizables: Parte del trabajo está hecho, pero estaría bien añadir la modificación de los efectos existentes, pues ahora mismo solo se permite la creación y destrucción. También estaría bien poder importar y exportar efectos.
- Gestión de usuarios: Se podría usar esta funcionalidad para guardar efectos y ajustes en la cuenta del usuario para poder acceder a ellos desde otros ordenadores fácilmente.
- Listas de reproducción: Para poder funcionar como un reproductor de música.
- Compatibilidad con hardware de DJ: Esta mejora sería bastante costosa en tiempo pues no se tienen los conocimientos necesarios.

Otras funcionalidades posibles que se han ido pensando durante el desarrollo son:

- Conversión en una Progressive WebApp. Esto permitiría mejorar la experiencia de usuario permitiendo cargar la aplicación offline, además de añadir otras características deseables.
- Aumento de los idiomas disponibles, con el objetivo de llegar a más público.
- Despliegue en servicio propio para evitar tener un nombre de URL extraño o difícil de recordar.
- Adición de publicidad no intrusiva.

# Apéndice A

# Manual de instalación del entorno para desarrolladores

## A.1. Requisitos previos

Para preparar el entorno de desarrollo se requiere:

- Node JS versión 8+
- npm versión 6.4+
- git (en caso de estar descargando desde un repositorio)

## A.2. Instalación

Para preparar el entorno:

- Descargamos el código fuente mediante un git clone o copiando desde el CD.
- En la carpeta donde se tiene el código se instalan las dependencias npm:

```
$ npm install
```

Con esto tendríamos el entorno preparado para empezar a desarrollar.

## A.3. Comandos útiles

Una vez tenemos todo listo podemos usar distintos comandos:

- `npm run start`: Es un alias para `ng serve`. Este comando permite ejecutar un servidor de desarrollo de nuestra aplicación. Levanta en el puerto 4200 de nuestra maquina local (localhost:4200) una copia de la aplicación.
- `npm run test`: Es un alias para `ng test --no-watch`. Este comando permite ejecutar los tests unitarios una vez.
- `npm run playground`: Es un alias para `angular-playground`. Este comando permite ejecutar un servidor de desarrollo con muestras de algunos de los componentes desarrollados.
- `npm run build` o `npm run build:prod`: Son alias para `ng build` y `ng build --prod` respectivamente. Permiten construir la aplicación para desarrollo y para producción.

# Apéndice B

# Manual de despliegue

Para poder desplegar se necesita un artefacto. Para generar este artefacto se requiere tener preparado el entorno de desarrollo y ejecutar:

```
$ npm run build:prod
```

Una vez hecho esto se habrá generado la carpeta dist. Dentro de ella encontramos la carpeta TFG. Para desplegar nuestra WebApp se debe publicar el contenido de esta carpeta en un servidor web.

# Apéndice C

# Contenido del CD-ROM

Los contenidos del CD-ROM:

- La carpeta `tfg`: Contiene el código fuente de la aplicación desarrollada (como explica 6.4), así como la documentación del proyecto.
- La carpeta `memoria`: contiene la presente memoria en formato PDF, así como varias imágenes, que por su tamaño pueden resultar difíciles de leer.

# Bibliografía

[1] Web de la Escuela de Ingeniería Informática de Valladolid, *Guía docente de TRABAJO DE FIN DE GRADO – MENCIÓN I.S.*, Consultado el 12/10/2018, Disponible en `https://www.inf.uva.es/wp-content/uploads/2016/06/G46976.pdf`

[2] Wikipedia, *European Credit Transfer and Acummulation System*, Consultado el 12/10/2018, Disponible en `https://en.wikipedia.org/wiki/European_Credit_Transfer_and_Accumulation_System`

[3] Virtual DJ, *Pagina oficial de Virtual DJ*, Consultado el 12/10/2018, Disponible en `https://es.virtualdj.com/`

[4] you.dj, *Aplicación web you.dj*, Consultado el 12/10/2018, Disponible en `https://you.dj/es`

[5] Youtube-dj, *Aplicación web YoutubeDJ*, Consultado el 12/10/2018, Disponible en `https://youtube-dj.com/`

[6] Joel Francia - Scrum.org, *¿Qué es Scrum?*, Consultado el 13/10/2018, Disponible en `https://www.scrum.org/resources/blog/que-es-scrum`

[7] proyectosagiles.org, *Requisitos para poder utilizar Scrum*, Consultado el 13/10/2018, Disponible en `https://proyectosagiles.org/requisitos-de-scrum/`

[8] beagilemyfriend.com, *¿Qué es un Sprint en Scrum?*, Consultado el 13/10/2018, Disponible en `https://www.beagilemyfriend.com/que-es-un-sprint/`

[9] Marc Bara - obs-edu.com, *Las 5 etapas en los "Sprints" de un desarrollo Scrum*, Consultado el 13/10/2018, Disponible en `https://www.obs-edu.com/es/blog-investigacion/project-management/las-5-etapas-en-los-sprints-de-un-desarrollo-scrum`

[10] softeng.es/, *Proceso y Roles de Scrum*, Consultado el 13/10/2018, Disponible en `https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html`

[11] Scrum Manager BoK, *Artefactos*, Consultado el 13/10/2018, Disponible en `https://www.scrummanager.net/bok/index.php?title=Artefactos`

[12] Universidad de Alicante, *Modelo vista controlador (MVC)*, Consultado el 13/10/2018, Disponible en `https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html`

[13] Microsoft developer network, *Desarrollo de Software basado en Componentes*, Consultado el 13/10/2018, Disponible en `https://msdn.microsoft.com/es-es/library/bb972268.aspx`

[14] Julio Pari, *Alta cohesión y bajo Acoplamiento – Diseño de Software*, Consultado el 13/10/2018, Disponible en `http://blog.juliopari.com/alta-cohesion-y-bajo-acoplamiento-diseno-de-software/`

[15] GitLab, *What is GitLab?*, Consultado el 13/10/2018, Disponible en `https://about.gitlab.com/what-is-gitlab/`

[16] GitLab, *Plan*, Consultado el 13/10/2018, Disponible en `https://about.gitlab.com/stages-devops-lifecycle/plan/`

[17] GitLab, *Create*, Consultado el 13/10/2018, Disponible en `https://about.gitlab.com/stages-devops-lifecycle/create/`

[18] GitLab, *Verify*, Consultado el 13/10/2018, Disponible en `https://about.gitlab.com/stages-devops-lifecycle/verify/`

[19] Vincent Driessen, *A successful Git branching model*, Consultado el 13/10/2018, Disponible en `https://nvie.com/posts/a-successful-git-branching-model/`

[20] Wikipedia, *Daily build*, Consultado el 13/10/2018, Disponible en `https://en.wikipedia.org/wiki/Daily_build`

[21] Wikipedia, *HTML5*, Consultado el 13/10/2018, Disponible en `https://es.wikipedia.org/wiki/HTML5`

[22] Javier Flores Herrera - Código Facilito, *¿Qué es HTML?*, Consultado el 14/10/2018, Disponible en `https://codigofacilito.com/articulos/que-es-html`

[23] Wikipedia, *Hoja de estilos en cascada*, Consultado el 14/10/2018, Disponible en `https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada`

[24] MDN Web Docs, *¿Qué es JavaScript?*, Consultado el 14/10/2018, Disponible en `https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript`

[25] Pagina oficial de TypeScript, *TypeSCript*, Consultado el 14/10/2018, Disponible en `https://www.typescriptlang.org/`

[26] Angular.io, *JavaScript*, Consultado el 14/10/2018, Disponible en `https://angular.io/`

[27] Boletín oficial del estado, *XVII Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública*, Consultado el 15/10/2018, Disponible en `https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf`

[28] Elvira Ramajo - Ramajo Asesores, *¿Cuanto le cuesto a mi empresa?*, Consultado el 15/10/2018, Disponible en `https://www.ramajoasesores.com/2014/12/cuanto-le-cuesto-a-mi-empresa/`

[29] Agencia tributaria, *Tabla de coeficientes de amortización lineal.*, Consultado el 15/10/2018, Disponible en `https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml`

[30] Angular Documentation, *Architecture overview*, Consultado el 16/10/2018, Disponible en `https://angular.io/guide/architecture`

[31] Angular Documentation, *Introduction to components*, Consultado el 16/10/2018, Disponible en `https://angular.io/guide/architecture-components`

[32] Sobre GitLab, *GitLab pricing*, Consultado el 16/10/2018, Disponible en `https://about.gitlab.com/pricing/`

[33] Pagina oficial de Jest, *Jest*, Consultado el 19/10/2018, Disponible en `https://jestjs.io/`

[34] Repositorio oficial de Prettier, *Prettier, Opinionated Code Formatter*, Consultado el 19/10/2018, Disponible en `https://github.com/prettier/prettier`

[35] Repositorio oficial de husky, *husky*, Consultado el 19/10/2018, Disponible en `https://github.com/typicode/husky`

[36] Pagina oficial de surge, *surge*, Consultado el 19/10/2018, Disponible en `https://surge.sh/`

[37] Pagina oficial de compodoc, *Compodoc*, Consultado el 20/10/2018, Disponible en `https://compodoc.app/guides/getting-started.html`

[38] Pagina oficial de GitLab, *Static vs Dynamic Websites*, Consultado el 20/10/2018, Disponible en `https://about.gitlab.com/2016/06/03/ssg-overview-gitlab-pages-part-1-dynamic-x-static/`

[39] Project Management for Instructional Designers, *Risk Management Process*, Consultado el 20/10/2018, Disponible en `https://pm4id.org/chapter/11-2-risk-management-process/`

[40] Angular documentation, *The Ahead-of-Time (AOT) compiler*, Consultado el 20/10/2018, Disponible en `https://angular.io/guide/aot-compiler`

[41] MDN web docs, *Web Audio API*, Consultado el 20/11/2018, Disponible en `https://developer.mozilla.org/es/docs/Web_Audio_API`

[42] Wavesurfer Official Page, *Wavesurfer*, Consultado el 21/11/2018, Disponible en `https://wavesurfer-js.org/`

[43] howler.js Official Page, *Howler JS*, Consultado el 21/11/2018, Disponible en `https://howlerjs.com/`

[44] WAVESURFER GITHUB REPOSITORY, *It doesn't support Angular 2*, Consultado el 21/11/2018, Disponible en `https://github.com/katspaugh/wavesurfer.js/issues/1200`

[45] THEODEUS, *tuna*, Consultado el 08/12/2018, Disponible en `https://github.com/Theodeus/tuna`

[46] FHACEB OOK, *EL FILTRO BICUADRADO*, Consultado el 26/12/2018, Disponible en `https://es.scribd.com/document/74185703/EL-FILTRO-BICUADRADO`

[47] YOUTUBE, *Términos y Condiciones del Servicio*, Consultado el 13/01/2019, Disponible en `https://www.youtube.com/static?gl=GB&template=terms`

[48] SOUNCLOUD, *Términos y Condiciones de Uso de SoundCloud*, Consultado el 13/01/2019, Disponible en `https://soundcloud.com/terms-of-use#Aceptaci%C3%B3n-de-los-T%C3%A9rminos-y-Condiciones-de-Uso`

[49] MDN WEB DOCS, *Control de acceso HTTP (CORS)*, Consultado el 13/01/2019, Disponible en `https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS`

[50] NPM ARCHIVE, *Music Tempo*, Consultado el 20/01/2019, Disponible en `https://www.npmjs.com/package/music-tempo`

[51] BACKTRACK.FM, *Beats Per Minute*, Consultado el 21/01/2019, Disponible en `https://backtracks.fm/resources/podcast-dictionary/beats+per+minute`

[52] SIMON DIXON, *Automatic Extraction of Tempo and Beat from Expressive Performances*, Consultado el 21/01/2019, Disponible en `http://www.eecs.qmul.ac.uk/~simond/pub/2001/jnmr.pdf`

[53] NPM ARCHIVE, *ngx-web-worker*, Consultado el 26/01/2019, Disponible en `https://www.npmjs.com/package/ngx-web-worker`

[54] MDN WEB DOCS, *Using Web Workers*, Consultado el 26/01/2019, Disponible en `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers`

[55] PAGINA OFICIAL DE KARMA, *Karma*, Consultado el 17/02/2019, Disponible en `https://karma-runner.github.io/latest/index.html`

[56] MDN WEB DOCS, *DOM*, Consultado el 18/02/2019, Disponible en `https://developer.mozilla.org/es/docs/Glossary/DOM`

[57] PÁGINA OFICIAL DE PLANTUML, *Guía de Referencia del Lenguaje PlantUML*, Consultado el 02/03/2019, Disponible en `http://plantuml.com/es/guide`

[58] REPOSITORIO OFICIAL DE NGX-TRANSLATE, *@ngx-translate/core*, Consultado el 02/03/2019, Disponible en `https://github.com/ngx-translate/core`

[59] DOCUMENTACIÓN OFICIAL DE GITLAB, *GitLab Runner*, Consultado el 02/03/2019, Disponible en `https://docs.gitlab.com/runner/`

[60] Docker Hub, *node*, Consultado el 03/03/2019, Disponible en `https://hub.docker.com/_/node/`

[61] proyectosagiles.org, *Facilitador (Scrum Master)*, Consultado el 17/03/2019, Disponible en `https://proyectosagiles.org/facilitador-scrum-master/`

[62] CONECTART, *Metodologías ágiles*, Consultado el 17/03/2019, Disponible en `https://blog.conectart.com/metodologias-agiles/`

[63] Página oficial de GNU, *A Quick Guide to GPLv3*, Consultado el 02/04/2019, Disponible en `https://www.gnu.org/licenses/quick-guide-gplv3.html`

[64] Página oficial de Visual Paradigm, *Visual Paradigm*, Consultado el 02/04/2019, Disponible en `https://www.visual-paradigm.com/`

# Glosario

**acoplamiento** En diseño de software el acoplamiento es la medida en que los cambios de un componente tiende a necesitar cambios de otro componente. 27, 56

**beats per minute** Beats per minute es un termino usado para medir el tempo de una canción. "Tempo" es un termino musical para la velocidad o ritmo de la canción. Beats per minute es la unidad usada para medir el tempo. Un "beat" es la unidad estándar de medición de longitud de una pieza de musica.) [51]. 2, 35, 71

**cadencia** Manera regular de ocurrir algo en períodos de tiempo que se repiten. 10

**cohesión** En diseño de software la cohesión es la medida en la que un componente se dedica a realizar solo la tarea para la cual fue creado, delegando las tareas complementarias a otros componentes. 27, 56

**CUE** Un CUE en un entorno de dj es un punto de ruptura en una canción, que te permite retomar la canción en ese punto. XIV, 15, 91, 92

**data-binding** En Angular el data-binding es cómo se conoce a la comunicación unidireccional (o bidireccional) entre template y clase, que permite reaccionar a eventos presentes en el template o modificar la información mostrada por el template. 27

**deck** Un deck o platina en un entorno de DJ es un reproductor de discos, originalmente vinilos. Hoy en día suele ser un reproductor o la representación de uno de estos reproductores, sea de forma física o de forma virtual. 15

**filtro bicuadrado** Filtro compuesto por un filtro paso bajo, otro paso alto, y un paso banda (filtra las frecuencias que estén por debajo de un umbral, deja pasar las de dentro de un rango y filtra las que están por encima de otro umbral) [46]. 86

**lazy-loading** Lazy-loading en Angular es un proceso de carga de módulos en el cual los módulos se descargan y posteriormente cargan en memoria los módulos sólo cuando se requieren, no durante la carga inicial de la pagina. 27

**loop** Un loop en un entorno de dj es la reproducción de una parte de la canción en bucle. XIV, 15, 91, 92, 97

**marco** Conjunto de elementos que rodean una realidad no material y que sirven para acotarla, encuadrarla, comprenderla, etc... 9, 18

**nightly** En desarrollo de software una versión nightly [20] es una versión que se construye "cada noche" (night en ingles) o en intervalos muy cortos, usada para testeo por usuarios. 19

**pitch** El pitch, hablando de música, es un atributo de los tonos musicales. Representa la percepción de la frecuencia, una modificación del pitch resultará en un cambio de la velocidad de la música que esta sonando. XIV, 15, 91