eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

CONTRIBUCIONES A UN PROYECTO OPEN SOURCE DE ÁMBITO INTERNACIONAL: AMPACHE

Alumno/Alumna: González, Llaguno, Urtzi

Director/Directora: Pereira, Varela, Juanan

Curso: 2020-2021

Fecha: Bilbao, 30, Junio, 2021

Resumen

Castellano

Este trabajo consiste en la contribución al desarrollo de Ampache, un *software* libre para la gestión de contenido multimedia. Teniendo como objetivo aportar valor añadido a un *software* usado por miles de usuarios, se han desarrollado mejoras propuestas por usuarios de la comunidad — la visualización de la duración total de una lista de reproducción y la opción de definir una imagen de fondo personalizada —, y también se ha contribuido a la remodelación de la interfaz de usuario.

Euskara

Lan honetan Ampache multimedia-edukia kudeatzeko *software* librearen garapenean ekarpenak egin dira. Milaka erabiltzailek erabilitako *softwareari* balio erantsia ematea helburu izanik, komunitateko erabiltzaileek proposatutako hobekuntzak egin dira — erreprodukzio-zerrenda baten iraupen osoa bistaratzea eta horma irudi pertsonalizatua definitzeko aukera —, baita erabiltzailearen interfazea birmoldatzen lagundu da ere.

English

This work consists of the contribution to the development of Ampache, a free software project for multimedia content management. Aiming to add value to a software used by thousands of people, new features proposed by community members have been developed - displaying the total duration of playlists and the ability to specify a custom background image - as well as helping in the redesign of the user interface.

Prefacio

La motivación de este proyecto surge del interés de aplicar los conocimientos y competencias adquiridas en el grado a un proyecto del mundo real, fuera del marco académico. Contribuyendo a un proyecto existente con una comunidad de usuarios, se consigue aportar un valor añadido difícilmente alcanzable por otros trabajos que parten desde cero. Además, existen retos transversales tales como la comunicación con otros contribuidores experimentados y la comprensión del código ya existente, que hacen que este trabajo se asemeje mejor al escenario en el que los nuevos graduados de ingeniería informática nos encontraremos.

El proyecto en cuestión es Ampache, un administrador de archivos y servidor de *streaming* multimedia libre, que funciona sobre un servidor web. Lanzado en 2001, sigue teniendo un desarrollo activo y una comunidad involucrada en la constante mejora del servicio. Está compuesto de más de 500.000 líneas de código programadas en PHP bajo la licencia AGPLv3.

Índice general

Resumen I

Prefacio III

Índice de figuras VII

Índice de tablas IX

1. Introducción 1
- 1.1. Contexto . . . 2
- 1.2. Motivaciones y beneficios . . . 3
- 1.3. Ampache . . . 4
- 1.4. Objetivos . . . 9

2. Metodología 11
- 2.1. Planificación . . . 12
- 2.2. Cronograma . . . 27
- 2.3. Herramientas utilizadas . . . 29
- 2.4. Evaluación económica . . . 31
- 2.5. Análisis de riesgos . . . 36

3. Ejecución 41
- 3.1. Captura de requisitos . . . 42
- 3.2. Interfaz . . . 48
- 3.3. Análisis y diseño . . . 52
- 3.4. Desarrollo . . . 70

4. Resultados 81
- 4.1. Pruebas de calidad . . . 82
- 4.2. Integración de los aportes . . . 91

5. Conclusión **95**
5.1. Diferencia entre estimación y tiempo real 96
5.2. Futuras líneas de trabajo 98
5.3. Licencia 98
5.4. Reflexión personal 99

Acrónimos **101**

Glosario **103**

Bibliografía **107**

Índice de figuras

1.3. Arquitectura de Ampache. 6
1.4. Estructura interna de Ampache. 7

2.1. Visión general del diagrama EDT. 12
2.2. Paquetes de trabajo de la agrupación «Planificación». 13
2.3. Paquetes de trabajo de la agrupación «Formación». 15
2.4. Paquetes de trabajo de la agrupación «Captura de requisitos». 17
2.5. Paquetes de trabajo de la agrupación «Análisis y diseño». . . . 19
2.6. Paquetes de trabajo de la agrupación «Implementación». . . . 21
2.7. Paquetes de trabajo de la agrupación «Pruebas». 23
2.8. Paquetes de trabajo de la agrupación «Documentación». . . . 25
2.9. Diagrama Gantt de la planificación del proyecto. 28

3.1. Jerarquía de actores. 42
3.2. Casos de uso generales. 43
3.3. Página principal de la herramienta *issue*. 44
3.4. Etiquetas de la *issue* #2527. 45
3.5. Etiquetas de la *issue* #2582. 46
3.6. Página de inicio. 48
3.7. Página de todos los artistas. 49
3.8. Página de un artista. 50
3.9. Página de un álbum. 51
3.10. Página de una canción. 51
3.11. Diagrama ER de la *issue* 2527. 53
3.12. Diagrama de clases de la *issue* 2527. 54
3.13. Interfaz de la página `show_playlists`. 55
3.14. Interfaz de la página `show_playlist`. 56
3.15. Diagrama de secuencia de la *issue* 2527. 57
3.16. Diagrama ER de la *issue* 2582. 59
3.17. Diagrama de clases de la *issue* 2582. 60
3.18. Interfaz de la página `login.php`. 61

3.19. Interfaz de la página `show_preferences.inc.php`. 62
3.20. Diagrama de secuencia del proceso de actualización de la *issue* 2582. 63
3.21. Interfaz de actualización. 64
3.22. Diagrama de secuencia del proceso de configuración de la *issue* 2582. 65
3.23. Estado actual de la página de álbumes. 66
3.24. Diagrama de secuencia de la página de álbumes. 67
3.25. Estado actual de la funcionalidad favoritos. 68
3.26. Diagrama de secuencia de la funcionalidad favoritos. 69
3.27. Cambios en la página `show_playlists.inc.php` para mostrar la duración total. 71
3.28. Cambios en la página `show_playlist.inc.php` para mostrar la duración total. 72
3.29. Cambios en la página `show_preferences.inc.php` para mostrar la nueva preferencia. 76
3.30. Página de inicio de sesión tras definir un fondo. 76
3.31. Página de álbumes tras el desarrollo. 78

4.1. *Log* (reducido) de las pruebas automatizadas sobre el *pull request* 2527. 84
4.2. Comprobación de la representación visual de la *issue* 2582 en una resolución de 1080p. 86
4.3. Comprobación de la representación visual de la *issue* 2582 en una resolución móvil de 720p. 86
4.4. Lista de canciones de un álbum. 89
4.5. Canción marcada como favorita. 89
4.6. Canción desmarcada como favorita. 89
4.7. Query antes de marcar la canción como favorita. 90
4.8. Query después de marcar la canción como favorita. 90
4.10. Comentario inicial del *pull request* de la *issue* 2527. 92
4.11. *Commit* en el repositorio de Ampache del *pull request* 2527. . 92
4.12. Fallo del *commit* `40c0bb2` en el test automático de Travis CI. 93
4.13. Éxito en el test automático de Travis CI tras realizar los cambios de estilo al código. 93

5.1. Tiempo planificado y tiempo real. 96
5.2. Diagrama Gantt del desglose de horas real del proyecto. 97

Índice de tablas

2.1. Paquete de trabajo «Diagrama EDT». 13
2.2. Paquete de trabajo «Cronograma». 14
2.3. Paquete de trabajo «Reuniones». 14
2.4. Paquete de trabajo «Emails». 14
2.5. Paquete de trabajo «Ampache». 15
2.6. Paquete de trabajo «PHP». 16
2.7. Paquete de trabajo «Git». 16
2.8. Paquete de trabajo «Travis CI». 16
2.9. Paquete de trabajo «LaTeX». 17
2.10. Paquete de trabajo «Comunicación». 18
2.11. Paquete de trabajo «Casos de uso». 18
2.12. Paquete de trabajo «Análisis y diseño Issue 2527». 19
2.13. Paquete de trabajo «Análisis y diseño Issue 2582». 20
2.14. Paquete de trabajo «Análisis y diseño React». 20
2.15. Paquete de trabajo «Implementación Issue 2527». 21
2.16. Paquete de trabajo «Implementación Issue 2582». 22
2.17. Paquete de trabajo «Implementación React». 22
2.18. Paquete de trabajo «Pruebas Issue 2527». 23
2.19. Paquete de trabajo «Pruebas Issue 2582». 24
2.20. Paquete de trabajo «Pruebas React». 24
2.21. Paquete de trabajo «Memoria». 25
2.22. Paquete de trabajo «Defensa». 25
2.23. Resumen de tiempos y precedencias del EDT. 26
2.24. Resumen de los gastos asociados al proyecto. 35

3.1. Parámetros aceptados para la acción `Albums` de la API. 78
3.2. Parámetros aceptados para la acción `Flag` de la API. 80

4.1. Comparación en los tiempos de carga del *pull request* 2527. . . 85

1. Introducción

Durante las décadas de los años 1960 y 1970 gran parte del desarrollo de *software* se producía dentro de un contexto de investigación, en laboratorios académicos y corporativos. Para los científicos e ingenieros desarrolladores era habitual compartir libremente el código fuente, creando así un entorno de colaboración mutua.

En 1980, el Instituto de Tecnología de Massachusetts comenzó a vender la propiedad intelectual del *software* creado bajo licencias privativas. En consecuencia, se restringió el acceso al código fuente y se prohibió su modificación por la comunidad académica (von Hippel and von Krogh [2009]).

Esta acción creó el descontento de los investigadores involucrados. Entre ellos se encontraba Richard Stallman, programador para el MIT Artificial Intelligence Laboratory. Stallman, que veía las nuevas limitaciones como un impedimento para el desarrollo del conocimiento, fundó en 1985 la Free Software Foundation.

La innovadora idea de la FSF era la creación de un mecanismo legal que protegiese el acceso libre al *software*. Para la implementación de esta idea se creó la GNU General Public License[1], también conocida como GPL. Una licencia que respeta la libertad de aprendizaje, modificación, ejecución y distribución del código fuente.

Bajo la licencia GPL se han publicado una gran variedad de programas informáticos. Entre los más destacables se encuentra el *kernel* Linux, una alternativa libre que domina el mercado de los ordenadores de alto rendimiento (Meuer [2008]).

[1] The GNU General Public License v3.0 (`www.gnu.org/licenses/gpl-3.0.html`)

1.1. Contexto

A lo largo del grado en Ingeniería Informática de Gestión y Sistemas de Información es frecuente la realización de proyectos académicos con el fin de adquirir un conocimiento técnico-práctico que sirva como complemento al contenido teórico impartido en el aula. La dinámica habitual de estos proyectos consiste en comenzar a trabajar desde cero — sin tener en cuenta trabajo previo de terceros — siguiendo un guión y de manera individual, en parejas o grupos de tamaño reducido formados por los alumnos.

Esta metodología es adecuada dentro de un contexto académico, sin embargo, no se asemeja a un entorno real de desarrollo de *software*. Un escenario habitual, tanto en el mundo laboral como de investigación, consta de: la colaboración con entidades externas (universidades extranjeras, empresas colaboradoras, etc.); la comunicación entre distintos departamentos compuestos de decenas de personas; y la expansión, en base al trabajo previo, tanto del *software* existente como del cuerpo de conocimiento presente. Es por ello, que en este trabajo se busca la integración en un entorno de desarrollo real, no académico, buscando formarse y experimentar los beneficios y problemas asociados. Valorando el éxito de los trabajos de fin de grado presentados el año pasado (Albizuri [2019] y Jaio [2019]), surge la idea para este proyecto como propuesta del alumno Urtzi González al profesor Juanan Pereira, miembro del departamento de LSI en la Escuela de Ingeniería de Bilbao y tutor de ambos trabajos mencionados (Pereira [2021]).

1.2. Motivaciones y beneficios

«*Eman ta zabal zazu*»[2] es el lema de la Universidad del País Vasco. El propio Eduardo Chillida — quien definió tanto el lema como el emblema de la universidad — lo describe como (Chillida [2003]): «Me pidieron que hiciera un símbolo, y lo hice, un árbol y su fruto. Esa es la idea, *Eman ta zabal zazu.*».

Una de las ideas que engloba el lema de la universidad es el derecho al conocimiento universal como herramienta de interés social público, es aquí donde surge un nexo con el movimiento del *software* libre.

El concepto fundamental del FOSS consiste en la libertad de acceso y distribución de la información. Se trata de un modelo de transparencia que elimina barreras geográficas, monetarias y políticas de obstrucción en la difusión del conocimiento.

La motivación principal para la elección de este trabajo consiste en contribuir a un proyecto de *software* libre que esté en línea con las ideas previamente descritas.

1.2.1. Selección de proyecto FOSS

Entre las propuestas presentadas al tutor del proyecto se encontraban: la plataforma de música Ampache, la aplicación de notas Turtl[3] y la extensión de navegador JavaScript Reddit Enhancement Suite[4].

Dado que Ampache es una aplicación que utilizo a diario y reúne mi interés por la informática con mi afición por la música, resulta una opción interesante a la que poder realizar contribuciones.

Los beneficios son claros, contribuyendo a un *software* libre se genera valor para la comunidad de usuarios que lo utilizan, que en este caso es abierta e ilimitada.

[2]Traducido al castellano como «Dalo y difúndelo».

[3]Turtl: the secure, collaborative notebook (`www.turtlapp.com`)

[4]Reddit Enhancement Suite: community-driven unofficial browser extension for Reddit (`www.redditenhancementsuite.com`)

1.3. Ampache

Ampache es una aplicación web destinada a la transmisión de audio y vídeo que realiza la función de administración de archivos y metadatos.

Figura 1.1: Logotipo de Ampache.[5]

Entre las características destacables de Ampache están incluidas (Ampache [2020]):

- **Almacenamiento de música**: catalogación y administración de colecciones de música a través de una interfaz web.
- ***Streaming* audiovisual**: transmisión de contenido a reproductores multimedia y reproducción directa mediante la página web con el reproductor HTML5.
- ***Software* libre**: respeta la libertad del usuario y el código es libremente accesible bajo la licencia AGPLv3[6].
- **Compatibilidad**: posibilidad de conexión a aplicaciones de terceros mediante protocolos de transmisión de información.

[5]Publicado por el autor SUTJael bajo la licencia pública GPLv2.

[6]La licencia AGPLv3 es una versión modificada de la licencia GPLv3, que contiene una única clausula adicional: al ejecutar un programa modificado en un servidor, el servidor debe permitir la descarga del código fuente correspondiente a la versión modificada que se ejecuta (Foundation [2015]).

1.3.1. Situación del proyecto

Publicado en 2001 por Scott Kveton, miembro de la Oregon State University, el proyecto Ampache ha tenido múltiples mantenedores y más de 100 contribuidores a lo largo de sus dos décadas de existencia (Figura 1.2). Desde 2019 el proyecto es mantenido por el usuario australiano «lachlan-00».

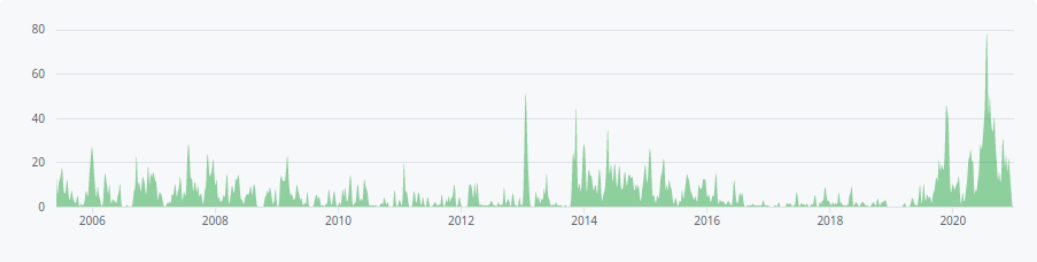


Figura 1.2: Gráfico de actividad en el desarrollo de Ampache.[7]

Teniendo en cuenta que la mayoría de proyectos *open source* tienen una esperanza de vida inferior a los dos años (Liao et al. [2017]), la longevidad de Ampache se convierte en una característica muy destacable del proyecto[8].

A fecha de redacción de esta memoria, la última versión publicada es la 4.4.3 (5 de junio del 2021). La versión 5.0.0 — que incluirá cambios notables en la API y una modernización de la interfaz de usuario — se encuentra en desarrollo actualmente.

En paralelo al desarrollo de nuevas versiones, se está realizando un nuevo *frontend* basado en el *framework* React — al que también se contribuirá con este trabajo — con el objetivo de obtener una interfaz de usuario que tenga un aspecto más actualizado.

[7] Las aportaciones anteriores a 2006 no se realizaron en el repositorio actual de GitHub, por lo que no hay registro de actividad para esa franja de años.

[8] El *efecto Lindy*, descrito por el matemático Benoit Mandelbrot y ampliado por Nassim Taleb, indica que una tecnología — o cualquier cosa no perecedera — aumenta su esperanza de vida con cada día adicional de existencia. Un *software* que ha existido durante 20 años es probable que siga existiendo otros 20 años (Taleb [2012]).

1.3.2. Arquitectura

Ampache sigue un un modelo de arquitectura cliente-servidor (Figura 1.3). Los usuarios (cliente) interaccionan con la aplicación (servidor) mediante dos posibles vías: la interfaz web HTML5 o los conectores API para clientes de terceros.

Esta lógica se ejecuta sobre un servidor HTTP (Apache, NGINX, lighttpd, etc.) el cual se encarga de la comunicación con el resto de elementos y la base de datos.

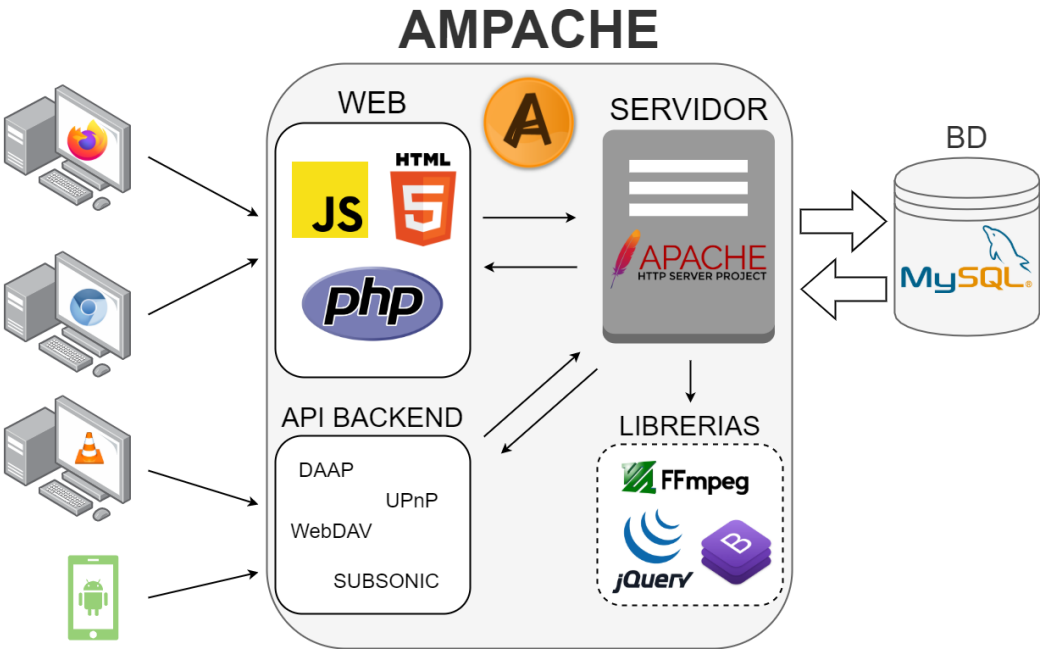


Figura 1.3: Arquitectura de Ampache.

La interfaz web, desde la que acceden los navegadores, utiliza PHP y JavaScript para la parte lógica y HTML5 junto a CSS3 para la estructuración de contenido. Dispone de un reproductor de audio y vídeo y permite la gestión de la configuración de la aplicación.

La estructura interna de Ampache (Figura 1.4) está dividida en una jerarquía de directorios. La lógica de negocio se encuentra en la carpeta «`lib`», es aquí donde están definidas las clases y métodos principales.

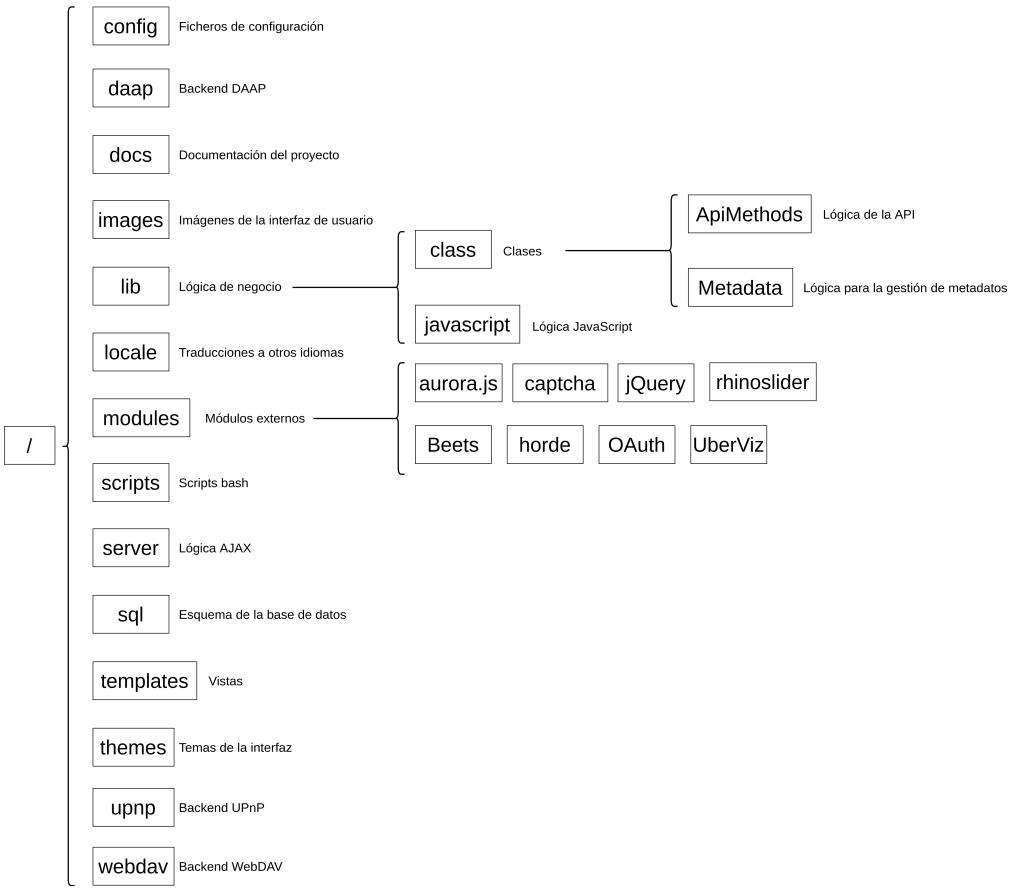


Figura 1.4: Estructura interna de Ampache.

El código base no está desarrollado sobre ningún *framework*, aunque sí que se utilizan algunas funcionalidades especificas de algunos *frameworks* para tareas concretas.

Se hace uso de múltiples librerías externas que facilitan el desarrollo de funcionalidades. Por ejemplo, se hace uso de la librería FFmpeg[9] para la conversión de *codecs*; jQuery[10] para la manipulación de elementos HTML; y Bootstrap[11] para la interfaz gráfica. Otras librerías externas utilizadas son:

[9]FFmpeg: a complete, cross-platform solution to record, convert and stream audio and video (`www.ffmpeg.org`)

[10]jQuery: the Write Less, Do More, JavaScript Library (`www.jquery.com`)

[11]Bootstrap: the most popular HTML, CSS, and JS library in the world (`www.getbootstrap.com`)

- **Aurora.js**[12]: *framework* para la decodificación y reproducción de audio.
- **Beets**[13]: módulo para el manejo de archivos multimedia, cataloga colecciones y gestiona metadatos.
- **Captcha**: módulo de seguridad para proteger los servicios de ataques automatizados.
- **Horde**[14]: *framework* de uso general que ofrece funcionalidades para la gestión de preferencias, compresión, detección de navegador, etc.
- **OAuth**[15]: estándar de seguridad para la delegación de acceso.
- **Rhinoslider**[16]: librería para la creación de efectos especiales.
- **UberViz**[17]: generador de efectos visuales en base a audio.

Por otra parte, los conectores permiten la comunicación con otras aplicaciones. Aplicaciones como Subsonic[18] o DSub[19] son compatibles con Ampache y habilitan una experiencia nativa para dispositivos Android.

Para ello se establece un canal basado en un protocolo de comunicación (UPnP, WebDAV, DAAP, etc.) que transmite la información entre ambas aplicaciones. De esta manera, se abre la posibilidad de conectarse a Ampache desde un dispositivo móvil con una aplicación nativa a su sistema operativo.

[12]Aurora.js: JavaScript audio decoding framework (`www.github.com/audiocogs/aurora.js`)

[13]Beets: media library management system for obsessive music geeks (`https://beets.io/`)

[14]Horde: general-purpose web application framework in PHP (`www.horde.org`)

[15]OAuth: industry-standard protocol for authorization (`www.oauth.net`)

[16]Rhinoslider: the most flexible jQuery slider/slideshow (`www.rhinoslider.com`)

[17]UberViz: custom real-time audio-reactive music visualizers (`www.uberviz.io`)

[18]Subsonic: stream music and video from your home computer to your phone (`www.subsonic.org/pages/apps.jsp`)

[19]DSub: music streaming app for Subsonic servers (`www.github.com/daneren2005/Subsonic`)

1.4. Objetivos

A lo largo de la formación en el grado de Ingeniería Informática de Gestión y Sistemas de Información se han adquirido multitud de competencias. En este trabajo se busca poner en práctica las competencias y conocimientos adquiridos en el grado para ayudar a corregir errores y añadir nuevas funcionalidades a un *software* profesional.

1.4.1. Aplicar las competencias obtenidas en el grado

Se busca demostrar la capacidad de las tecnologías vistas en el grado para su implementación en un proyecto real. Siendo capaz de definir requisitos a nivel de *software* que se ajusten a las especificaciones y estándares de la industria.

Esto implica un conocimiento de la arquitectura y las distintas fases de desarrollo de un proyecto *software*, entre las que se encuentran: la captura de requisitos establecidos por el cliente, el análisis y diseño siguiendo los requisitos establecidos, la implementación lógica mediante lenguajes de programación, y la realización de pruebas de calidad que verifiquen los resultados obtenidos.

Todo ello debe quedar adecuadamente documentado para permitir la comprensión por terceros y facilitar un desarrollo colaborativo, esta documentación debe tener un lenguaje científico-técnico que se adecúe al contexto informático.

1.4.2. Contribuir a un *software* libre

Existe una gran diferencia entre comenzar un nuevo proyecto desde *tabula rasa*, a tener que comprender el trabajo que decenas de personas han realizado durante décadas, como es el caso de Ampache.

Es una propiedad frecuente de los proyectos de gran envergadura encontrar secciones que contengan tecnología desfasada y con poca o ninguna documentación. Lo recomendable en esas situaciones es realizar una refac-

torización del código con el objetivo de reducir la deuda técnica (Arif and Rana [2020]).

También es necesario un conocimiento del marco legal en el que se desarrolla un proyecto *open source*. Para ello es necesario un estudio de la permisividad de distribución, modificación y uso comercial que ofrecen los distintos tipos de licencias existentes (AGPL, Apache, Creative Commons, MIT, etc.).

1.4.3. Capacidad de relación interpersonal en una lengua no materna

Realizar contribuciones a un *software open source* supone un reto adicional, ya que es necesaria una comunicación — generalmente en inglés — con personas de distintas nacionalidades, culturas y géneros. Dada la problemática de esta tarea, es frecuente encontrar códigos de conducta en los proyectos *open source* indicando explícitamente unas normas que permitan un trato respetuoso con el resto de personas.

1.4.4. Aprendizaje autónomo de nuevas tecnologías

Por muy aptos que resulten los conocimientos adquiridos en el grado, estos no son suficientes para trabajar en un campo tan dinámico como es la informática.

El ámbito informático es uno donde la innovación en la creación de nuevas metodologías y herramientas es constante. Es por ello que es necesaria una formación continua que se adapte a la dirección en la que evoluciona el campo.

Tomando como base los conocimientos obtenidos en el grado, mediante el desarrollo de este trabajo se busca expandir y profundizar en las capacidades técnicas tanto de las tecnologías vistas en el grado como de tecnologías en las que será necesaria una nueva formación autónoma individual.

2. Metodología

2.1. Planificación

Con el objetivo de facilitar la planificación de este trabajo, los distintos módulos que lo componen han sido descompuestos mediante un diagrama EDT.

El diagrama EDT resultante está compuesto de las siguientes agrupaciones (Figura 2.1): planificación, formación, captura de requisitos, análisis y diseño, desarrollo, pruebas y documentación.

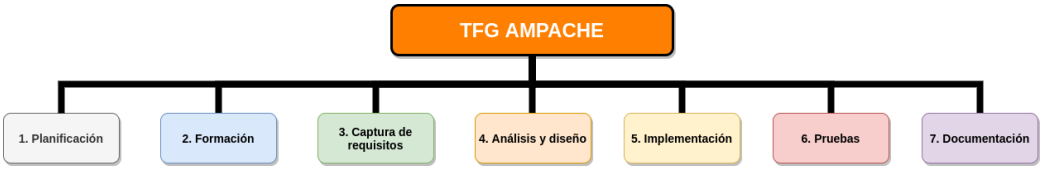


Figura 2.1: Visión general del diagrama EDT.

1. **Planificación y seguimiento**: esta agrupación incluye las actividades de coordinación para la realización del proyecto.

2. **Formación**: en esta agrupación están incluidas las herramientas y tecnologías que requieren de un aprendizaje autónomo adicional.

3. **Captura de requisitos**: agrupación de la información requerida para poder afrontar los requerimientos de los *stakeholders*.

4. **Análisis y diseño**: esta agrupación hace referencia a la fase de modelado y definición de la estructura del *software* a ser desarrollado.

5. **Implementación**: agrupación que incluye la fase de programación del código siguiendo el diseño previamente definido.

6. **Pruebas**: en esta agrupación están contenidas las distintas pruebas de calidad y usabilidad relativas al código desarrollado.

7. **Documentación**: agrupación que contiene los elementos relacionados a la realización de la memoria del proyecto.

2.1.1. Planificación

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.2 y Tablas 2.1, 2.2, 2.3, 2.4) que ofrecen un soporte a la gestión de la ejecución del proyecto.

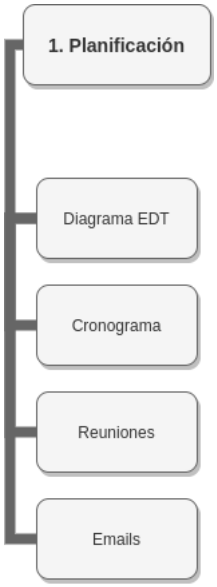


Figura 2.2: Paquetes de trabajo de la agrupación «Planificación».

Diagrama EDT
Agrupación: Planificación.
Duración estimada: 5 horas.
Descripción: la realización de este diagrama EDT, donde se descomponen en paquetes de trabajo las tareas a realizar a lo largo de la elaboración del proyecto.
Salidas/Entregables: diagrama EDT.
Recursos necesarios: *software* «diagrams.net».

Tabla 2.1: Paquete de trabajo «Diagrama EDT».

Cronograma
Agrupación: Planificación.
Duración estimada: 5 horas.
Descripción: realización del diagrama Gantt donde se especifica el orden de las tareas a realizar y las dependencias necesarias para su ejecución.
Salidas/Entregables: diagrama Gantt.
Recursos necesarios: *software* «GanttProject».

Tabla 2.2: Paquete de trabajo «Cronograma».

Reuniones
Agrupación: Planificación.
Duración estimada: 15 horas.
Descripción: reuniones presenciales o virtuales con el tutor del proyecto para la coordinación y seguimiento del proyecto. Estas reuniones sirven para tomar decisiones sobre las *issues* a realizar, recibir recomendaciones sobre la elaboración del proyecto y resolver dudas puntuales.
Salidas/Entregables: apuntes sobre los temas tratados a lo largo de la reunión.
Recursos necesarios: -

Tabla 2.3: Paquete de trabajo «Reuniones».

Emails
Agrupación: Planificación.
Duración estimada: 10 horas.
Descripción: correos electrónicos periódicos intercambiados con el tutor con el objetivo de realizar un seguimiento continuo del estado del proyecto.
Salidas/Entregables: mensajes intercambiados.
Recursos necesarios: -

Tabla 2.4: Paquete de trabajo «Emails».

2.1.2. Formación

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.3 y Tablas 2.5, 2.6, 2.7, 2.8, 2.9) relativos al aprendizaje continuo realizado sobre las herramientas y tecnologías especificas de este proyecto.

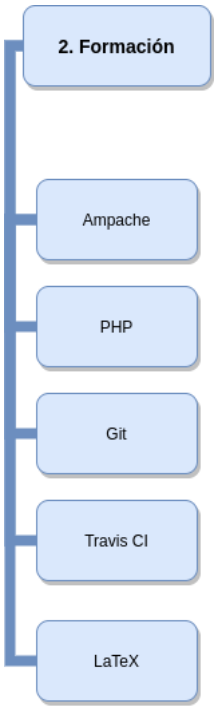


Figura 2.3: Paquetes de trabajo de la agrupación «Formación».

Ampache
Agrupación: Formación.
Duración estimada: 20 horas.
Descripción: familiarización con la arquitectura, estilo de código y funcionamiento de Ampache.
Salidas/Entregables: -
Recursos necesarios: acceso al repositorio de Ampache.

Tabla 2.5: Paquete de trabajo «Ampache».

PHP
Agrupación: Formación.
Duración estimada: 30 horas.
Descripción: aprendizaje continuo del lenguaje de programación PHP para poder programar el código de las funcionalidades.
Salidas/Entregables: -
Recursos necesarios: documentación de PHP.

Tabla 2.6: Paquete de trabajo «PHP».

Git
Agrupación: Formación.
Duración estimada: 15 horas.
Descripción: aprendizaje sobre el flujo de trabajo en Git para posibilitar la creación de nuevas ramas de desarrollo y su posterior integración con el repositorio principal.
Salidas/Entregables: -
Recursos necesarios: documentación de Git.

Tabla 2.7: Paquete de trabajo «Git».

Travis CI
Agrupación: Formación.
Duración estimada: 5 horas.
Descripción: obtener conocimiento acerca del funcionamiento de la herramienta de integración continua Travis CI, que es utilizada por Ampache para comprobar la validez de los aportes realizados.
Salidas/Entregables: -
Recursos necesarios: documentación de Travis CI.

Tabla 2.8: Paquete de trabajo «Travis CI».

LaTeX
Agrupación: Formación.
Duración estimada: 10 horas.
Descripción: formación en la herramienta de preparación de documentos LaTeX para la elaboración de la memoria.
Salidas/Entregables: -
Recursos necesarios: documentación de LaTeX.

Tabla 2.9: Paquete de trabajo «LaTeX».

2.1.3. Captura de requisitos

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.4 y Tablas 2.10, 2.11) relativos a la recopilación de los requisitos de las funcionalidades que serán desarrolladas.

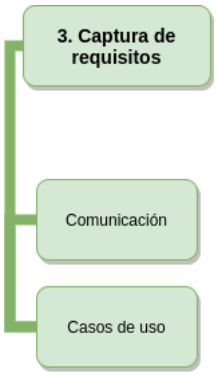


Figura 2.4: Paquetes de trabajo de la agrupación «Captura de requisitos».

Comunicación
Agrupación: Captura de requisitos.
Duración estimada: 8 horas.
Descripción: interacción mediante GitHub con los creadores originales de las *issues* para poder definir adecuadamente los casos de uso a abordar.
Salidas/Entregables: mensaje intercambiados.
Recursos necesarios: cuenta de usuario en la plataforma GitHub.

Tabla 2.10: Paquete de trabajo «Comunicación».

Casos de uso
Agrupación: Captura de requisitos.
Duración estimada: 5 horas.
Descripción: definición textual y esquemática del funcionamiento que deberá tener cada funcionalidad a ser implementada.
Salidas/Entregables: casos de uso definidos.
Recursos necesarios: información recopilada sobre los requisitos.

Tabla 2.11: Paquete de trabajo «Casos de uso».

2.1.4. Análisis y diseño

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.5 y Tablas 2.12, 2.13, 2.14) relativos a la transformación de los requerimientos obtenidos en especificaciones mediante diagramas que sigan estándares UML.

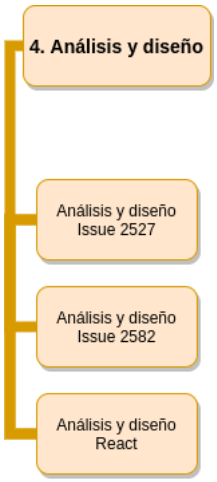


Figura 2.5: Paquetes de trabajo de la agrupación «Análisis y diseño».

Análisis y diseño Issue 2527
Agrupación: Análisis y diseño.
Duración estimada: 20 horas.
Descripción: una vez dispuesta la captura de requisitos, elaboración y desarrollo del diagrama ER, los diagramas de clase, diagramas asociativos y diagramas de secuencia de modo que quede especificado el funcionamiento para la posterior fase de desarrollo.
Salidas/Entregables: diagramas ER, diagramas de clase, diagramas asociativos y diagramas de secuencia relativos a la *issue* 2527.
Recursos necesarios: caso de uso 2527, *software* «diagrams.net» y *software* «MySQL Workbench».

Tabla 2.12: Paquete de trabajo «Análisis y diseño Issue 2527».

Análisis y diseño Issue 2582
Agrupación: Análisis y diseño.
Duración estimada: 15 horas.
Descripción: una vez dispuesta la captura de requisitos, elaboración y desarrollo de diagramas ER, diagramas de clase y diagramas de secuencia.
Salidas/Entregables: diagramas generados.
Recursos necesarios: caso de uso 2582 y *software* «diagrams.net».

Tabla 2.13: Paquete de trabajo «Análisis y diseño Issue 2582».

Análisis y diseño React
Agrupación: Análisis y diseño.
Duración estimada: 24 horas.
Descripción: definición de las clases, métodos y flujos de comunicación relacionados a las funcionalidades del nuevo *frontend*.
Salidas/Entregables: diagramas generados.
Recursos necesarios: casos de uso relacionados al desarrollo React y *software* «diagrams.net».

Tabla 2.14: Paquete de trabajo «Análisis y diseño React».

2.1.5. Implementación

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.6 y Tablas 2.15, 2.16, 2.17) relativos a la programación lógica del código en base a los diseños concebidos previamente.

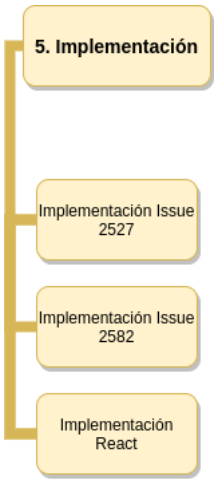


Figura 2.6: Paquetes de trabajo de la agrupación «Implementación».

Implementación Issue 2527
Agrupación: Implementación
Duración estimada: 12 horas
Descripción: proceso de programación en el lenguaje PHP siguiendo el diseño definido en la fase de análisis.
Salidas/Entregables: código relativo a la *issue* 2527.
Recursos necesarios: diagramas ER, diagramas de clase, diagramas asociativos y diagramas de secuencia relativos a la *issue* 2527.

Tabla 2.15: Paquete de trabajo «Implementación Issue 2527».

Implementación Issue 2582
Agrupación: Implementación
Duración estimada: 8 horas
Descripción: definición de la lógica de la *issue* 2582 mediante código.
Salidas/Entregables: código relativo a la *issue* 2582.
Recursos necesarios: diagramas ER, diagramas de clase, diagramas asociativos y diagramas de secuencia relativos a la *issue* 2582.

Tabla 2.16: Paquete de trabajo «Implementación Issue 2582».

Implementación React
Agrupación: Implementación
Duración estimada: 15 horas.
Descripción: desarrollo de código en el *framework* React siguiendo las especificaciones de diseño definidas.
Salidas/Entregables: código relativo al desarrollo React.
Recursos necesarios: diagramas definidos en el análisis y diseño de React.

Tabla 2.17: Paquete de trabajo «Implementación React».

2.1.6. Pruebas

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.7 y Tablas 2.18, 2.19, 2.20) relativos a la comprobación de la calidad y usabilidad del código generado.

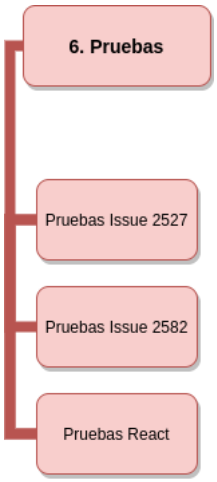


Figura 2.7: Paquetes de trabajo de la agrupación «Pruebas».

Pruebas Issue 2527
Agrupación: Pruebas.
Duración estimada: 10 horas.
Descripción: verificación de la calidad del código, de modo que se identifiquen y solucionen los errores encontrados antes de integrar la implementación con el producto final.
Salidas/Entregables: documentación de las pruebas realizadas sobre la implementación de la *issue* 2527.
Recursos necesarios: código relativo a la *issue* 2527.

Tabla 2.18: Paquete de trabajo «Pruebas Issue 2527».

Pruebas Issue 2582
Agrupación: Pruebas.
Duración estimada: 5 horas.
Descripción: comprobación de que la implementación realizada alcanza un nivel de calidad y usabilidad aceptable.
Salidas/Entregables: documentación de las pruebas realizadas sobre la implementación de la *issue* 2582.
Recursos necesarios: código relativo a la *issue* 2582.

Tabla 2.19: Paquete de trabajo «Pruebas Issue 2582».

Pruebas React
Agrupación: Pruebas.
Duración estimada: 15 horas.
Descripción: testeo de la implementación realizada con el objetivo de comprobar errores en el diseño o desarrollo.
Salidas/Entregables: documentación de las pruebas realizadas sobre la implementación del desarrollo React.
Recursos necesarios: código relativo al desarrollo React.

Tabla 2.20: Paquete de trabajo «Pruebas React».

2.1.7. Documentación

En esta agrupación están incluidos los paquetes de trabajo (Figura 2.8 y Tablas 2.21, 2.22) relativos a la investigación y redacción de los documentos relacionados a la ejecución del proyecto.

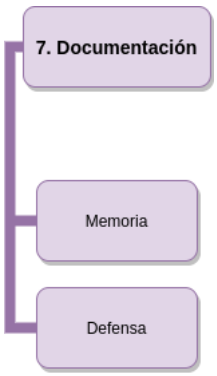


Figura 2.8: Paquetes de trabajo de la agrupación «Documentación».

Memoria
Agrupación: Documentación.
Duración estimada: 160 horas.
Descripción: investigación, redacción y maquetación del proceso completo de la elaboración del proyecto.
Salidas/Entregables: documento de memoria del TFG.
Recursos necesarios: todas las figuras, textos, tablas, diagramas y código generados a lo largo del proyecto.

Tabla 2.21: Paquete de trabajo «Memoria».

Defensa
Agrupación: Documentación.
Duración estimada: 8 horas.
Descripción: creación de las diapositivas para la presentación de la defensa del trabajo.
Salidas/Entregables: conjunto de diapositivas.
Recursos necesarios: figuras, textos, tablas y diagramas más relevantes del proyecto.

Tabla 2.22: Paquete de trabajo «Defensa».

2.1.8. Resumen de la planificación

Realizando una suma de la duración individual de cada paquete de trabajo, se obtiene una duración total de 420 horas para la ejecución del proyecto completo (Tabla 2.23).

ID	Nombre de la tarea	Precendecia	Duración estimada (horas)
1.1	Diagrama EDT	-	5
1.2	Cronograma	1.1	5
1.3	Reuniones	-	15
1.4	Emails	-	10
2.1	Ampache	-	20
2.2	PHP	-	30
2.3	Git	-	15
2.4	Travis CI	-	5
2.5	LaTeX	-	10
3.1	Comunicación	-	8
3.2	Casos de uso	3.1	5
4.1	Análisis y diseño Issue 2527	3.2	20
4.2	Análisis y diseño Issue 2582	3.2	15
4.3	Análisis y diseño React	3.2	24
5.1	Implementación Issue 2527	4.1	12
5.2	Implementación Issue 2582	4.2	8
5.3	Implementación React	4.3	15
6.1	Pruebas Issue 2527	5.1	10
6.2	Pruebas Issue 2582	5.2	5
6.3	Pruebas React	5.3	15
7.1	Memoria	De 1.1 a 6.3	160
7.2	Defensa	De 1.1 a 6.4	8
Duración total			420

Tabla 2.23: Resumen de tiempos y precedencias del EDT.

2.2. Cronograma

El proyecto tiene como fecha inicio el día 1 de septiembre de 2020 y fecha de cierre el 23 de julio de 2021[1].

Existen ciertas tareas — como la redacción de la memoria, las reuniones y el intercambio de *emails* — que se realizan a lo largo de todo el proyecto y otras — como la formación en las herramientas — que solo se realizan a lo largo del desarrollo de las *issues*.

Se han definido tres hitos, cada uno está asociado a la completitud del desarrollo e integración de una *issue*.

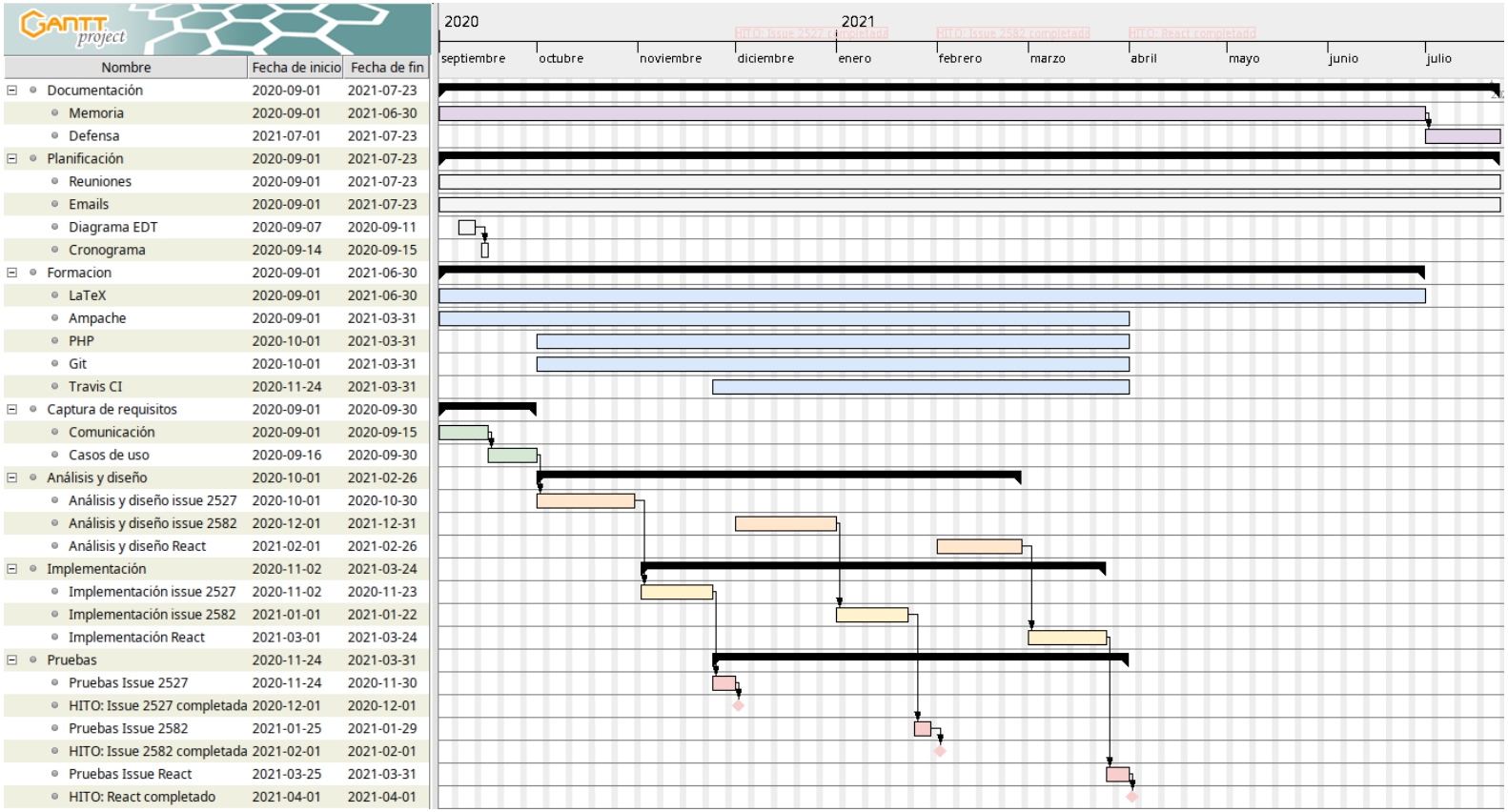
- **Hito 1 - *Issue* 2527 completada**: habiendo comenzado en octubre, se espera tener finalizada la primera *issue* para el día 1 de diciembre de 2020.

- **Hito 2 - *Issue* 2582 completada**: el desarrollo de la segunda *issue* comienza inmediatamente después de terminar con la primera. Se establece como fecha objetivo tener su integración realizada para el primer día de febrero de 2021.

- **Hito 3 - Funcionalidades React completadas**: dado que el último hito se realiza tras la completitud de los dos anteriores, existe la posibilidad de que su plazo de ejecución varíe en mayor medida. No obstante, se ha establecido como meta finalizar su desarrollo para el 1 de abril de 2021.

Mediante un diagrama Gantt (Figura 2.9) es posible representar visualmente las fechas mencionadas en esta sección y los paquetes de trabajo descritos en el EDT de la sección anterior.

[1]La fecha de cierre puede variar entre el 19 y 23 de julio dependiendo de la fecha de defensa asignada por la universidad.



Nombre	Fecha de inicio	Fecha de fin
Documentación	2020-09-01	2021-07-23
Memoria	2020-09-01	2021-06-30
Defensa	2021-07-01	2021-07-23
Planificación	2020-09-01	2021-07-23
Reuniones	2020-09-01	2021-07-23
Emails	2020-09-01	2021-07-23
Diagrama EDT	2020-09-07	2020-09-11
Cronograma	2020-09-14	2020-09-15
Formacion	2020-09-01	2021-06-30
LaTeX	2020-09-01	2021-06-30
Ampache	2020-09-01	2021-03-31
PHP	2020-10-01	2021-03-31
Git	2020-10-01	2021-03-31
Travis CI	2020-11-24	2021-03-31
Captura de requisitos	2020-09-01	2020-09-30
Comunicación	2020-09-01	2020-09-15
Casos de uso	2020-09-16	2020-09-30
Análisis y diseño	2020-10-01	2021-02-26
Análisis y diseño issue 2527	2020-10-01	2020-10-30
Análisis y diseño issue 2582	2020-12-01	2021-12-31
Análisis y diseño React	2021-02-01	2021-02-26
Implementación	2020-11-02	2021-03-24
Implementación issue 2527	2020-11-02	2020-11-23
Implementación issue 2582	2021-01-01	2021-01-22
Implementación React	2021-03-01	2021-03-24
Pruebas	2020-11-24	2021-03-31
Pruebas Issue 2527	2020-11-24	2020-11-30
HITO: Issue 2527 completada	2020-12-01	2020-12-01
Pruebas Issue 2582	2021-01-25	2021-01-29
HITO: Issue 2582 completada	2021-02-01	2021-02-01
Pruebas Issue React	2021-03-25	2021-03-31
HITO: React completado	2021-04-01	2021-04-01

Figura 2.9: Diagrama Gantt de la planificación del proyecto.

2.3. Herramientas utilizadas

Para facilitar las labores relacionadas a este trabajo, se ha hecho uso de herramientas y servicios que facilitan la elaboración del mismo. Entre ellas, las más destacables y directamente relacionadas con el trabajo han sido:

- **Apache**: servidor web. En la configuración utilizada, la aplicación de Ampache se ejecuta sobre un servidor montado en un servicio Apache.
- **diagrams.net**: *software* de creación de diagramas. Los diagramas de clases, diagramas asociativos y diagramas de secuencia han sido realizados con esta herramienta.
- **GanttProject**: aplicación de planificación de proyectow. Su uso ha permitido la realización del diagrama Gantt utilizado para planificar este proyecto.
- **Git**: sistema de control de versiones. Se ha utilizado junto a la plataforma GitHub[2] para gestionar las distintas versiones de código generadas.
- **IntelliJ IDEA**: entorno de desarrollo integrado. Gracias a las funcionalidades que ofrece para el análisis de código, su uso ha facilitado la comprensión y desarrollo.
- **LaTeX**: sistema de preparación de documentos. La documentación asociada al trabajo ha sido redactada en el servicio Overleaf[3] el cual utiliza el sistema LaTeX.
- **MySQL Workbench**: *software* para el diseño visual de bases de datos. Su uso ha permitido una mejor comprensión de la base de datos de Ampache. Los diagramas ER han sido generados mediante esta herramienta.
- **Telegram**: servicio de mensajería instantánea[4]. Es el canal de comunicación general de los miembros de la comunidad.

[2] GitHub: where the world builds software (`www.github.com`)
[3] Overleaf: online LaTeX Editor (`www.overleaf.com`)
[4] Telegram: a new era of messaging (`www.telegram.org`)

- **Travis CI**: servicio de integración continua[5]. Ha sido utilizado para la realización de pruebas automatizadas que verifican la validez del código.
- **Vim** y **Gedit**: editores de texto. El código desarrollado ha sido escrito utilizando estos dos editores de texto que facilitan la presentación y edición utilizando colores apropiados para la sintaxis de cada lenguaje.
- **Visual Paradigm**: herramienta de modelado UML. El diseño de los diagramas de clases ha sido realizado con esta herramienta.

También se han utilizado otras herramientas más comunes, indirectamente relacionadas con el trabajo tales como: Blackboard Collaborate y email para la comunicación con el tutor; Firefox como navegador donde se ejecuta la aplicación web de Ampache; y Linux como sistema operativo servidor de Apache.

[5]Travis CI: test and deploy your code with confidence (`www.travis-ci.org`)

2.4. Evaluación económica

Todo proyecto tiene asociado un coste de realización. En esta sección se realiza un análisis económico de los gastos relacionados a la ejecución del proyecto. Este análisis sirve como referencia para una posible comercialización del trabajo realizado.

2.4.1. Mano de obra

En un entorno laboral, la realización de este trabajo la llevaría acabo una persona con un puesto de trabajo de programador y analista de aplicaciones informáticas. En las tablas salariales publicadas en la disposición 14977 del Boletín Oficial del Estado (Tablas salariales y plus convenio para los años 2018, 2019 y 2020 BOE [2019]), se establece un salario base de 1.244,93 € mensuales para el desempeño de este puesto laboral. Dado que este salario está dispuesto en 14 mensualidades, el salario neto ascendería a 1.452,41 € mensuales.

El salario base indicado en el BOE está establecido en base a una jornada laboral completa de 40 horas semanales. El sueldo por hora se puede obtener mediante la siguiente fórmula (Fórmula 2.1):

$$Sueldo_{hora} = \frac{Sueldo_{mes}}{Horas\ de\ trabajo_{día} \times Días\ laborables_{mes}} \tag{2.1}$$

Un mes promedio contiene 30 días de los cuales 8 son festivos por fin de semana, realizando la resta se obtienen 22 días laborables y una jornada laboral habitual consta de 8 horas de trabajo. Una vez establecidas estas variables, es posible resolver la ecuación y obtener el salario por hora (Fórmula 2.2):

$$Sueldo_{hora} = \frac{1452{,}41\,€}{8\ horas \times 22\ días} = 8{,}25\,€ \tag{2.2}$$

Conociendo el número total de horas estimadas a la realización de este proyecto (Sección «2.1 Planificación») y el salario por hora de un puesto de trabajo semejante a las características del proyecto, es posible realizar el computo económico del coste del proyecto (Fórmula 2.3):

$$\begin{aligned} Coste\ mano\ de\ obra &= Sueldo\ hora \times Horas\ estimadas \\ &= 8{,}25\,€ \times 420\ horas = 3465\,€ \end{aligned} \tag{2.3}$$

Por lo se obtiene un coste de 3.465 € para la mano de obra asociada al proyecto.

2.4.2. Gastos de *software*

En linea con el objetivo de desarrollar y utilizar *software* libre, la mayoría de herramientas (Sección «2.3 Herramientas utilizadas») asociadas al desarrollo de este proyecto poseen licencias de *software* libre, es por ello que gran parte de los programas utilizados son gratuitos.

El único programa utilizado que requiere de una licencia de pago es Visual Paradigm. En este caso, la propia universidad ha comprado las licencias para los alumnos. Sin embargo, en un proyecto comercial se necesitaría adquirir una licencia Modeler[6] con un coste mensual asociado de 6 €. Como este proyecto tiene una duración de 9 meses el precio total ascendería a 54 €.

En cuanto a los servicios (GitHub, Overleaf, Travis CI, etc.) utilizados, todos ellos disponen de opciones gratuitas y opciones de pago. Como no se necesitan las funcionalidades avanzadas que ofrecen las opciones de pago, el coste de estos servicios también resulta gratuito.

En resumen, el gasto de *software* asciende a 54 €.

[6]También existen licencias más avanzadas (Standard, Professional y Enterprise) y la opción de adquirir licencias permanentes en lugar de mensuales. Sin embargo, el cálculo del coste se ha realizado con la opción adecuada más económica posible.

2.4.3. Gastos materiales

El proyecto se ha realizado sobre dos equipos informáticos: un ordenador personal utilizado para el desarrollo y un servidor utilizado para servir Ampache.

A continuación se realiza el cálculo de los gastos de amortización asociados a estos dispositivos.

Ordenador personal

Este proyecto se ha realizado sobre un ordenador personalizado cuyo valor asciende a 1500 € y tiene una vida útil de 8 años.

La pérdida de valor mensual de este activo se puede obtener mediante la siguiente fórmula (Fórmula 2.4):

$$Amortización\ mensual = \frac{Coste}{Vida\ útil} = \frac{1500\,€}{96\ meses} = 15{,}63\,€ \tag{2.4}$$

Considerando que la duración del proyecto es de 9 meses, la depreciación total asciende a (Fórmula 2.5):

$$\begin{aligned} Amortización\ total &= Amortización\ mensual \times Duración \\ &= 15{,}63\,€ \times 9\ meses = 140{,}67\,€ \end{aligned} \tag{2.5}$$

Servidor

Se ha utilizado una máquina separada para albergar y servir Ampache con alta disponibilidad. Esta máquina tiene un coste de 500 € y una vida estimada de 10 años.

La amortización se calcula del mismo modo que la amortización mensual del ordenador personal calculada previamente (Fórmula 2.6):

$$Amortización\ mensual = \frac{Coste}{Vida\ útil} = \frac{500\,€}{120\ meses} = 4{,}17\,€ \tag{2.6}$$

Multiplicando la amortización mensual por los 9 meses que dura el proyecto se obtiene una amortización total de (Fórmula 2.7):

$$\begin{aligned} Amortización\ total &= Amortización\ mensual \times Duración \\ &= 4{,}17\,€ \times 9\ meses = 37{,}5\,€ \end{aligned} \tag{2.7}$$

2.4.4. Gastos indirectos

La última categoría de gastos son aquellos que no están directamente relacionados con el proyecto. Dentro de esta categoría entran utilidades como la luz y la conexión internet.

Luz

Para calcular el gasto de luz es necesario considerar el gasto que realiza cada dispositivo mencionado en la categoría de gastos materiales.

El ordenador personal posee una fuente de alimentación con 800W de potencia y se utiliza durante las 420 horas (Subsección «2.1.8 Resumen de la planificación») destinadas al proyecto.

La fuente de alimentación del servidor tiene una potencia de 300W y está activa las 24 horas del día de los 9 meses del proyecto. Realizando la multiplicación[7] se obtienen 6480 horas de actividad.

[7] 24 h/día × 30 días/mes × 9 meses

El precio del kilovatio hora establecido por la compañía eléctrica es de 0.155 €. Por tanto, el coste total de electricidad es de (Fórmula 2.8):

$$\begin{aligned} Coste\ eléctrico &= \sum Horas\ activas \times Consumo\ enérgetico \times Precio \\ &= 420\ horas \times 0,800\ kWh \times 0{,}155\,€/kWh \\ &+ 6480\ horas \times 0,300\ kWh \times 0{,}155\,€/kWh \\ &= 353{,}4\,€ \end{aligned} \tag{2.8}$$

Internet

Se tiene contratada una tarifa plana de 600Mbps de fibra óptica simétrica por el precio de 40 € mensuales. Considerando que la duración del proyecto es de 9 meses, el coste total de la conexión es de 360 €.

2.4.5. Resumen de los gastos

A continuación se muestra un resumen (Tabla 2.24) de todos los gastos descritos en esta sección.

Concepto	Coste
Mano de obra	3465 €
Software - Licencia Visual Paradigm	54 €
Materiales - Ordenador personal	140,67 €
Materiales - Servidor	37,5 €
Indirectos - Luz	353,4 €
Indirectos - Internet	360 €
Total	**4410,57 €**

Tabla 2.24: Resumen de los gastos asociados al proyecto.

En resumen, la estimación en base al análisis económico realizado da como resultado un coste total de 4410,57 €.

2.5. Análisis de riesgos

Tal y como establece la familia de estándares para la gestión del riesgo ISO 31000, se busca una integración de normativas en los procesos (Figura 2.10), con el objetivo de identificar y gestionar los riesgos existentes en la ejecución de un proyecto.

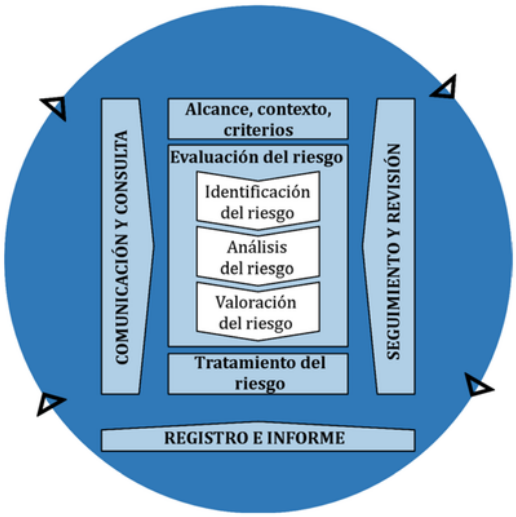


Figura 2.10: Proceso de gestión de riesgos.[8]

Para ello, se identifican los principales riesgos existentes y se define para cada uno:

- **Prevención**: pautas para reducir la posibilidad de que el riesgo se materialice.
- **Plan de contingencia**: en caso de producirse, pasos a tomar para minimizar el impacto causado.
- **Probabilidad**: estimación de la probabilidad de que suceda el riesgo.
- **Impacto**: consecuencias de su materialización.
- **Riesgo**: multiplicación de la probabilidad por el impacto para obtener una cifra de evaluación del riesgo.

[8]Publicado por la International Organization for Standardization (ISO [2018]).

Planificación incorrecta

Una definición demasiado ambiciosa del alcance, una formación lenta en las nuevas herramientas o una estimación general imprecisa del tiempo necesario para la ejecución, son motivos que generan un riesgo en la realización del proyecto.

Prevención

- Realizar un cronograma de los tiempos marcados.
- Permitir holgura entre tareas para paliar posibles inconvenientes.
- Definir con el tutor del trabajo un alcance adecuado.

Plan de contingencia

- Redefinir el alcance del trabajo.
- Aumentar el número de horas dedicadas a su realización.

Probabilidad

- Planificación ligeramente incorrecta - probabilidad estimada 50 %.
- Planificación excesivamente incorrecta - probabilidad estimada 2 %.

Impacto

- Planificación ligeramente incorrecta: 5 días $\times$ 3 horas/día = 15 horas. Impacto medio.
- Planificación excesivamente incorrecta: 30 días $\times$ 3 horas/día = 90 horas. Impacto alto.

Riesgo

- Planificación ligeramente incorrecta: 50 % $\times$ 15 horas = 7,5. Riesgo medio.
- Planificación excesivamente incorrecta: 2 % $\times$ 90 horas = 1,8. Riesgo bajo.

Cancelación o abandono del proyecto principal

Al estar trabajando sobre un proyecto real y dado que no existe ningún contrato formal, existe la posibilidad de que el desarrollador principal abandone el proyecto sin previo aviso, bien sea por motivo de fuerza mayor (fallecimiento, enfermedad, etc.) o por voluntad propia.

Prevención

- Comunicarse con el desarrollador principal para conocer su compromiso con el mantenimiento del proyecto.

Plan de contingencia

- Buscar otro proyecto al que poder contribuir.
- Animar a otros desarrolladores con experiencia a tomar la posición de desarrollador principal.

Probabilidad

- Baja temporal del desarrollador principal 5 %.
- Ausencia permanente del desarrollador principal 1 %.

Impacto

- Baja temporal del desarrollador principal: 10 días $\times$ 3 horas/día = 30 horas. Impacto medio.
- Ausencia permanente del desarrollador principal: 90 días $\times$ 3 horas/día = 270 horas. Impacto muy alto.

Riesgo

- Baja temporal del desarrollador principal: 5 % $\times$ 30 horas = 1,5. Riesgo bajo.
- Ausencia permanente del desarrollador principal: 1 % $\times$ 270 horas = 2,7. Riesgo bajo.

Incapacidad temporal

Baja por enfermedad o accidente que imposibilita temporalmente la capacidad para trabajar.

Prevención

- **Derivadas del trabajo**
  - Evitar movimientos crónicos repetitivos.
  - Establecer un espacio de trabajo ergonómico.
  - Establecer pausas destinadas al descanso.
- **No derivadas del trabajo**
  - Seguir las pautas de salud general recomendadas (descanso, dieta, distanciamiento social, higiene, etc.) por los expertos en salud.

Plan de contingencia

- Solicitar una valoración de la lesión en un centro médico homologado.
- Seguir las pautas recibidas en la valoración solicitada.

Probabilidad

- Lesión leve (una semana de baja) - probabilidad estimada 20 %.
- Lesión grave (un mes de baja) - probabilidad estimada 2 %.

Impacto

- Lesión leve: 7 días × 3 horas/día = 21 horas. Impacto medio.
- Lesión grave: 30 días × 3 horas/día = 90 horas. Impacto alto.

Riesgo

- Lesión leve: 20 % × 21 horas = 4,2. Riesgo bajo.
- Lesión grave: 2 % × 90 horas = 1,8. Riesgo bajo.

Pérdida del trabajo realizado

Existen multitud de motivos (virus o *bugs* en el *software*; fallo, pérdida o robo del *hardware*; error humano) por los que se puede dar un deterioro o pérdida completa del trabajo realizado, tanto a nivel de documentación como de código.

Prevención

- Realizar copias de seguridad periódicas.
- Mantener redundancia de los ficheros en distintos lugares.
- Utilizar versiones estables y actualizadas de *software*.
- Utilizar adecuadamente el equipo informático.

Plan de contingencia

- Volver a realizar el trabajo perdido.
- En caso de ser necesario, aplazar las fechas estimadas de ejecución.

Probabilidad

- Pérdida leve (una semana de trabajo) - probabilidad estimada 10 %.
- Pérdida grave (tres meses de trabajo) - probabilidad estimada 1 %.

Impacto

- Pérdida leve: 7 días $\times$ 3 horas/día = 21 horas. Impacto medio.
- Pérdida grave: 90 días $\times$ 3 horas/día = 180 horas. Impacto muy alto.

Riesgo

- Pérdida leve: 10 % $\times$ 21 horas = 2,1. Riesgo bajo.
- Pérdida grave: 1 % $\times$ 180 horas = 1,8. Riesgo bajo.

3. Ejecución

3.1. Captura de requisitos

En esta sección se describen los actores que interactúan con el sistema y los casos de uso asociados. También se presentan las *issues* escogidas junto a una descripción de sus requisitos y objetivos a alcanzar.

3.1.1. Jerarquía de actores

Existen tres tipos de actores que utilizan el sistema (Figura 3.1):

- **Usuario Anónimo**: este actor hace referencia a los participantes que acceden a la aplicación de Ampache sin haberse identificado con una cuenta registrada en el sistema.
- **Usuario**: una vez un «Usuario Anónimo» se identifica con unas credenciales válidas en la página de *login* este pasa a ser un «Usuario».
- **Administrador**: existe una tercera categoría destinada a las cuentas de servicio para la administración y mantenimiento, para acceder al sistema como «Administrador» es necesario utilizar las credenciales de la cuenta de servicio creada en la instalación de la aplicación.

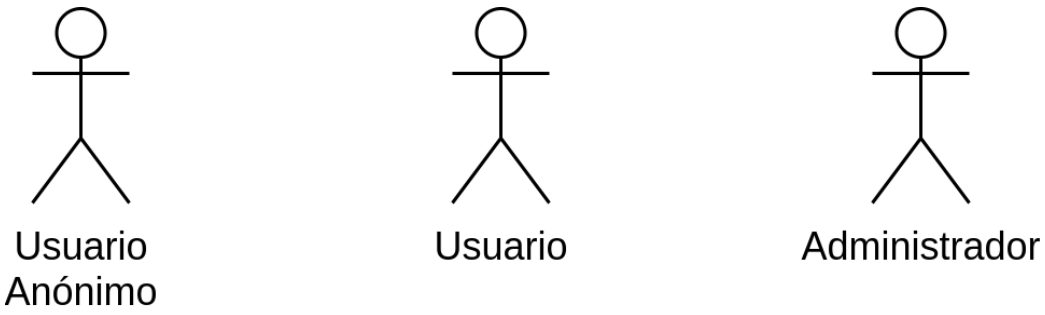


Figura 3.1: Jerarquía de actores.

3.1.2. Casos de uso

Los actores definidos pueden realizar las siguientes acciones (Figura 3.2) en la aplicación:

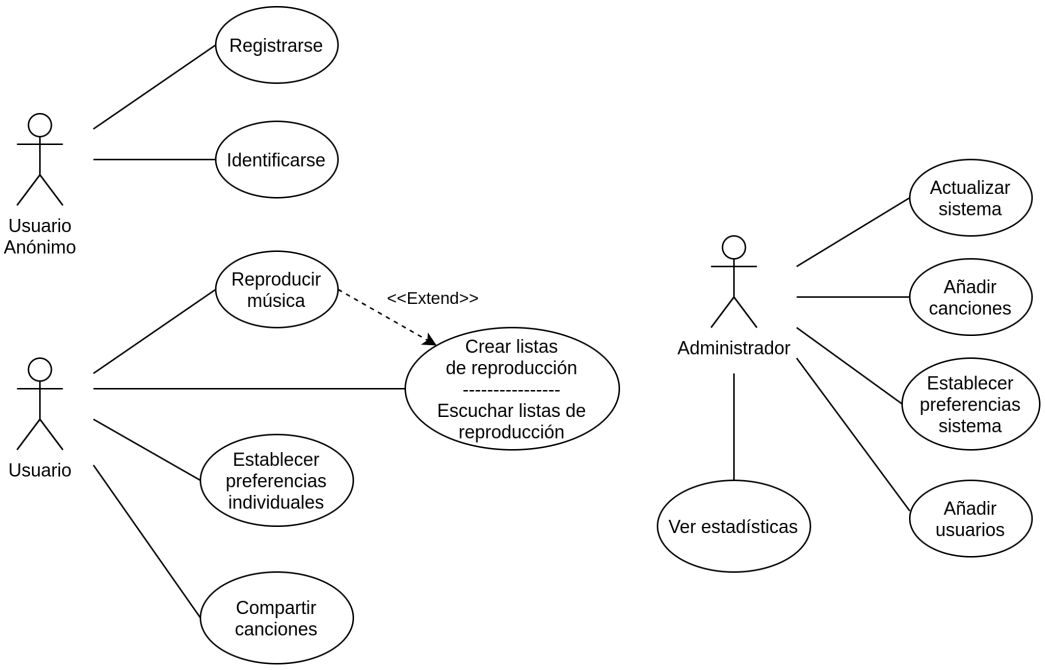


Figura 3.2: Casos de uso generales.

3.1.3. Selección de *issues*

Siendo un proyecto público, es la propia comunidad de usuarios quien propone las nuevas características e informa de los fallos o *bugs* existentes de manera colaborativa. Esta gestión de requisitos y sus respectivas soluciones se gestiona mediante la herramienta de *issues* de GitHub[1] (Figura 3.3).

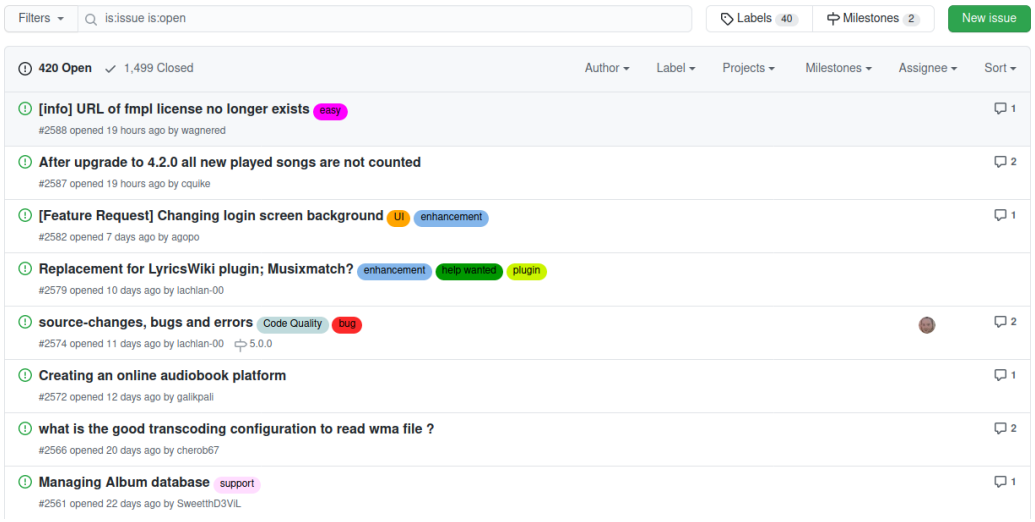


Figura 3.3: Página principal de la herramienta *issue*.

El sistema de rastreo de *issues* permite centralizar en un único lugar todas las incidencias y aporta granularidad puesto que cada incidencia es tratada de manera individual. Una *issue* contiene los siguientes elementos:

- **Título y descripción** indicando cual es el tema a tratar.
- Una **persona apoderada** que será la responsable de su resolución.
- Una **sección de comentarios** que facilite la comunicación entre desarrolladores y usuarios.
- Opcionalmente, es posible añadir **etiquetas** (p.e. «Interfaz», «Urgente», «Mejora») que faciliten la categorización.

[1] GitHub Guides: mastering issues (`https://guides.github.com/features/issues`) (Guides [2020])

#2527 [Feature Request] Total time

Esta incidencia se abrió en septiembre de 2020 por el usuario «4phun» de Ohio, Estados Unidos (4phun [2020]). Posee las etiquetas: interfaz, base de datos y mejora (Figura 3.4). La descripción inicial menciona lo siguiente:

> «Hi, I think it would be a great addition to the web UI to show the total time for playlists, albums, and wherever else it may apply.»

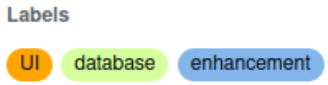


Figura 3.4: Etiquetas de la *issue* #2527.

Se trata de una incidencia para solicitar una nueva característica en la interfaz de usuario. En concreto, se solicita que las colecciones de canciones como los álbumes y listas de reproducción muestren su duración total.

#2582 [Feature Request] Changing login screen background

En este caso, el usuario «agopo» de Alemania (agopo [2020]) indica que le gustaría poder definir un *background* para la página de inicio de sesión sin necesidad de alterar las hojas de estilo de la aplicación:

> «I'd like to be able to change the login screen background without having to tamper with the main theme.»

También especifica el flujo de usabilidad deseado, indicando que desearía que al acceder como administrador del sistema a los ajustes de preferencias, existiese una opción para determinar una URL que dirija a una imagen y que esta imagen se aplique como fondo.

Considerando los requisitos, el desarrollador principal incluye esta *issue* dentro del proyecto «Ampache Improvement and Modernization» y le asigna las etiquetas: UI y enhancement (Figura 3.5).

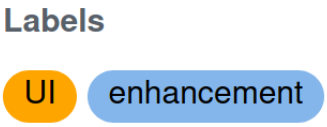


Figura 3.5: Etiquetas de la *issue* #2582.

Cliente React

La interfaz actual de Ampache se diseñó en 2008 mediante HTML, CSS y JavaScript y se apoya sobre el *framework* Bootstrap[2] para facilitar las tareas de diseño.

Desde entonces, la tecnología de diseño de interfaces ha evolucionado hasta el punto de considerarse desfasada la interfaz actual de la aplicación. Es por ello que surge la necesidad de rediseñar un nuevo *frontend* para la aplicación siguiendo patrones de diseño modernos.

El usuario australiano Mitch Ray comenzó el proyecto para la creación de una nueva interfaz utilizando el *framework* React[3]. Esta nueva interfaz se encuentra todavía en una fase muy inicial de desarrollo y carece de funcionalidades básicas (marcadores, cambios de ajustes, puntuaciones, etc.).

Una característica muy destacable de esta nueva interfaz es que hace uso de la API de Ampache. Esto facilita considerablemente el desarrollo, la legibilidad y el mantenimiento, ya que en lugar de tener que desarrollar métodos para interactuar con el *backend*, es suficiente con realizar llamadas a las funciones definidas en la API.

[2]Bootstrap: the most popular HTML, CSS, and JS library in the world (`www.getbootstrap.com`)

[3]React: a JavaScript library for building user interfaces (`www.reactjs.org`)

El objetivo para la tercera fase de desarrollo de este TFG consiste en contribuir a la creación de este nuevo cliente. Al ser una característica en fase de prueba y todavía no estar publicada oficialmente, no existen *issues* relacionadas a ella. Sin embargo, sí que existen multitud de tareas a realizar, tal y como se indica en el documento README de la rama del proyecto.

En concreto, se van a desarrollar dos funcionalidades pendientes de implementación:

- **Página de álbumes**: actualmente esta página está sin implementar. El objetivo consistirá en mostrar, siguiendo el estilo establecido para las demás páginas, todos los álbumes incluidos en un catálogo.
- **Favoritos**: esta funcionalidad permitirá al usuario marcar una canción de su catalogo como destacada. Para ello, se mostrará junto al título de la canción un icono con forma de corazón, si el usuario pulsa sobre él, se añadirá dicha canción a favoritos.

3.2. Interfaz

A continuación, se muestra mediante capturas las páginas más relevantes de la interfaz de Ampache.

3.2.1. Página de inicio

Cuando un usuario inicia sesión correctamente, este es automáticamente redirigido a la página de inicio (Figura 3.6):

- En la parte superior de la página de inicio, se permite la opción de navegar a la lista de canciones, álbumes o artistas.
- La ventana «Now Playing» muestra la canción (o canciones) que estén siendo actualmente reproducidas por los usuarios del sistema.
- «Albums of the Moment» son los álbumes más escuchados recientemente.
- Por último, «Recently Played» muestra las diez últimas canciones reproducidas, juntos a su información (álbum, artista, año).

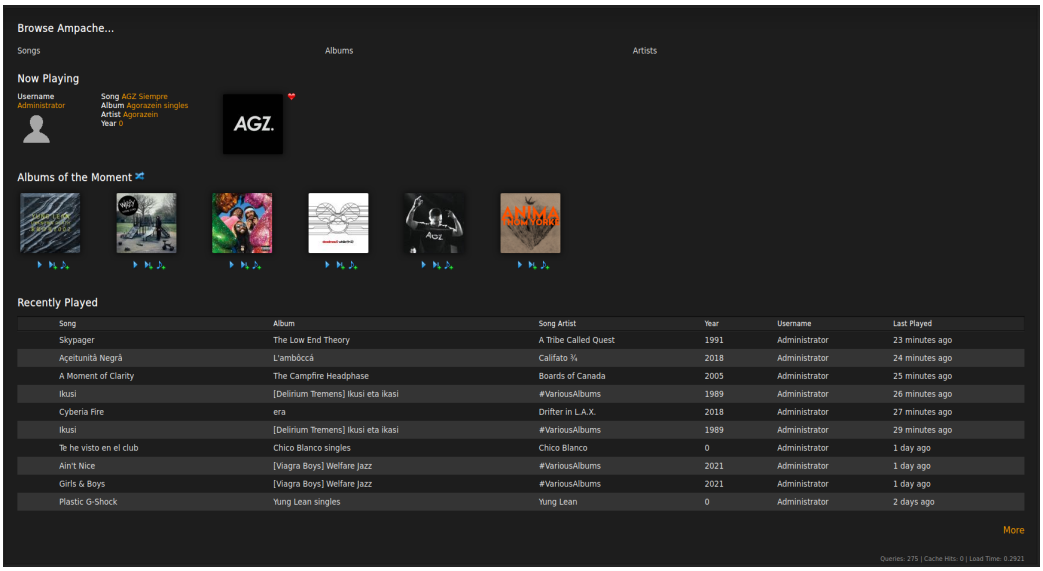


Figura 3.6: Página de inicio.

3.2.2. Página de artistas

La página de artistas (Figura 3.7) consiste de una lista ordenada alfabéticamente (con la opción de elegir orden ascendente o descendente) que contiene todos los artistas incluidos en un catálogo.

Para mejorar el rendimiento y facilitar la navegación, la lista está paginada cada n elementos (por defecto 50).

Cada fila contiene: una imagen, el nombre del artista, la cantidad de canciones y álbumes, el tiempo de reproducción total de todas las canciones contenidas, los géneros musicales[4] y un emoticono con forma de corazón para indicar si es un artista favorito.

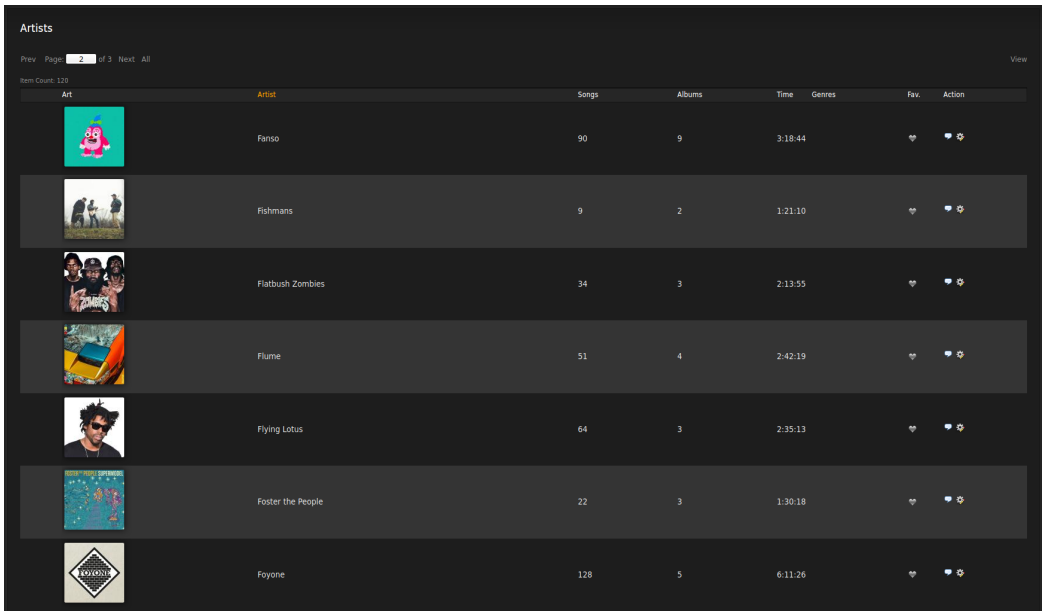


Figura 3.7: Página de todos los artistas.

Las páginas que muestran los álbumes y canciones son muy similares, por lo que se obvia mostrarlas.

[4]En mi catálogo no hay definidos géneros musicales, por lo que en la captura este campo se muestra vacio.

3.2.3. Página de artista

Una vez seleccionado un artista, se muestra una página (Figura 3.8) que contiene una descripción del artista y una lista con los álbumes publicados.

Esta página también permite acciones como: reproducir todas las canciones, editar los metadatos del artista y hacer comentarios.

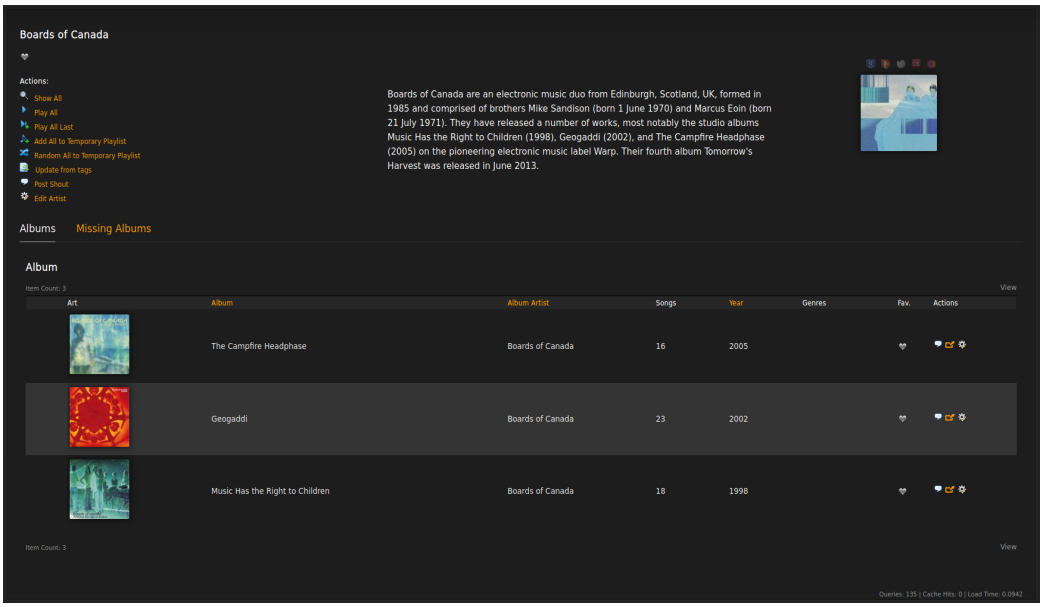


Figura 3.8: Página de un artista.

3.2.4. Página de álbum

De forma análoga, al seleccionar un álbum de un artista, se muestra la página relativa a ese álbum (Figura 3.9). Esta página contiene la carátula del álbum y una lista de posibles acciones.

En la parte inferior, se listan las canciones que contiene el álbum ordenadas por el número de pista establecido por el artista. También se muestran otros metadatos como el título, el nombre del artista y álbum, el año de publicación, el género musical, la duración de cada pista y un emoticono con forma de corazón para indicar las canciones favoritas.

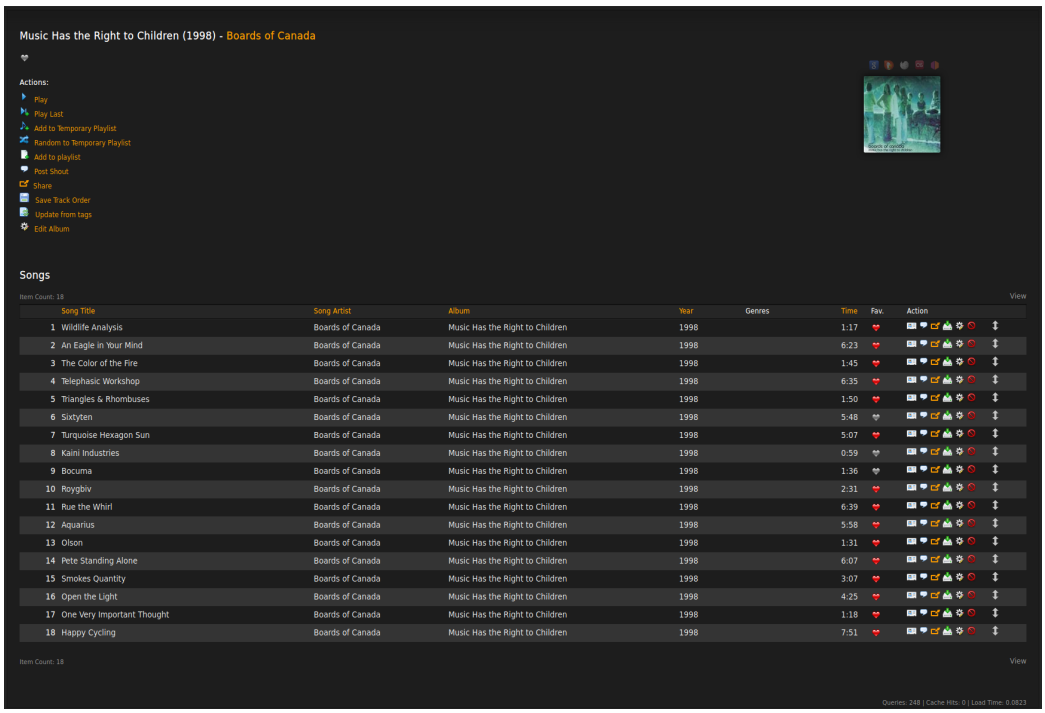


Figura 3.9: Página de un álbum.

3.2.5. Página de canción

Si se selecciona una canción, se navega a una página (Figura 3.10) que muestra los metadatos asociados a ella.

Los botones de acciones permiten reproducir la canción, añadirla a una lista de reproducción, compartirla, descargarla y editar sus datos.

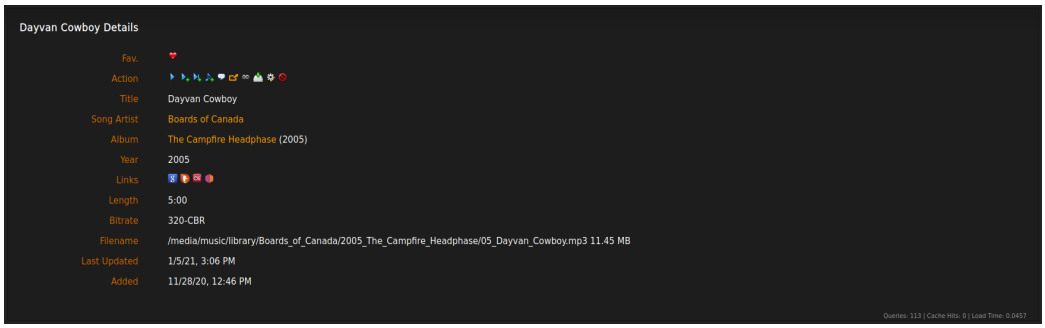


Figura 3.10: Página de una canción.

3.3. Análisis y diseño

Esta sección trata los aspectos de análisis y diseño correspondientes a cada funcionalidad implementada.

Al tratarse de un proyecto de gran magnitud que posee cientos de clases y tablas resulta imposible representar mediante un único diagrama todas las clases del sistema, por lo que se ha decidido mostrar con cada funcionalidad las clases y tablas pertinentes a la misma.

A continuación, se muestran los diagramas relacionales, diagramas de clase y diagramas de secuencia de cada *issue*.

3.3.1. #2527 [Feature Request] Total time

Una visión de alto nivel de esta *issue* consiste en: obtener la duración total de una lista de reproducción mediante la suma de la duración sus canciones y mostrar dicha información mediante la interfaz de usuario.

Para ello, se ha analizado la arquitectura del programa. Identificando los lugares donde se almacenan de datos, y la comunicación entre clases que permita acceder a ellos y mostrarlos.

Diagrama ER (#2527)

En primer lugar, ha sido necesario identificar las tablas de la bases de datos que contienen la información sobre la duración de las canciones (Figura 3.11). La tabla `song` contiene la duración de una canción y está relacionada con la tabla `playlist` mediante la relación `playlist_data`. Además toda canción pertenece a un artista y está contenida en un catalogo.

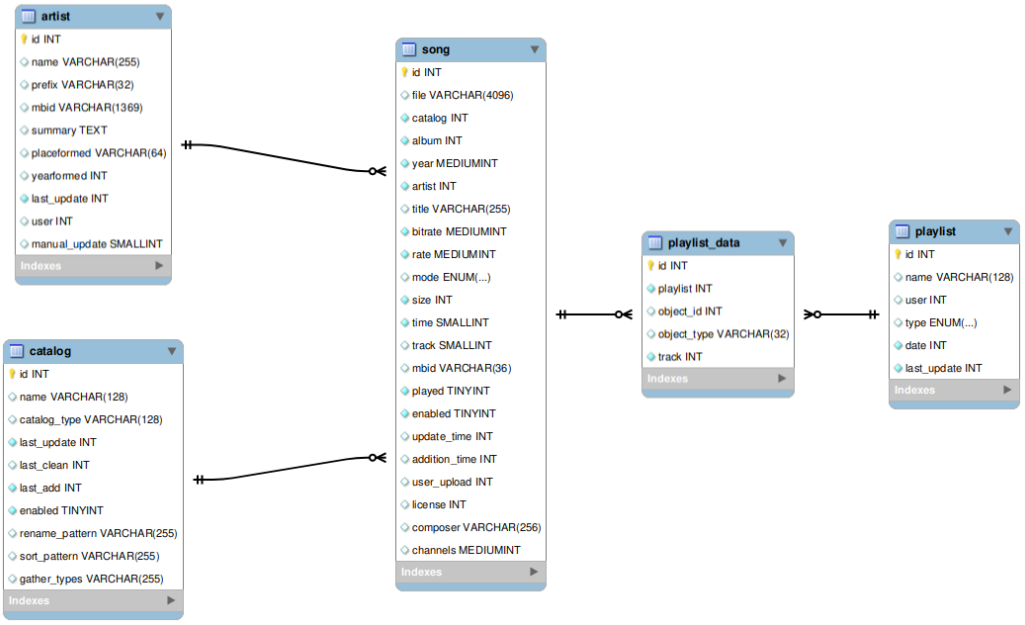


Figura 3.11: Diagrama ER de la *issue* 2527.

Diagrama de clases (#2527)

A continuación, se ha analizado como obtener esta información mediante las clases adecuadas. En el diagrama de clases realizado (Figura 3.12), se muestra la jerarquía de las clases que implementan las listas de reproducción.

La clase `Playlist` hereda de la clase abstracta `playlist_object`, la cual a su vez, hereda de la clase abstracta `database_object`.

También se observa que la clase abstracta `playlist_object` implementa la interfaz `library_item`, la cual hereda los atributos de `playable_item`.

Las clases `Browse` y `Search` son utilizadas para realizar las consultas sobre la base de datos y obtener la duración total.

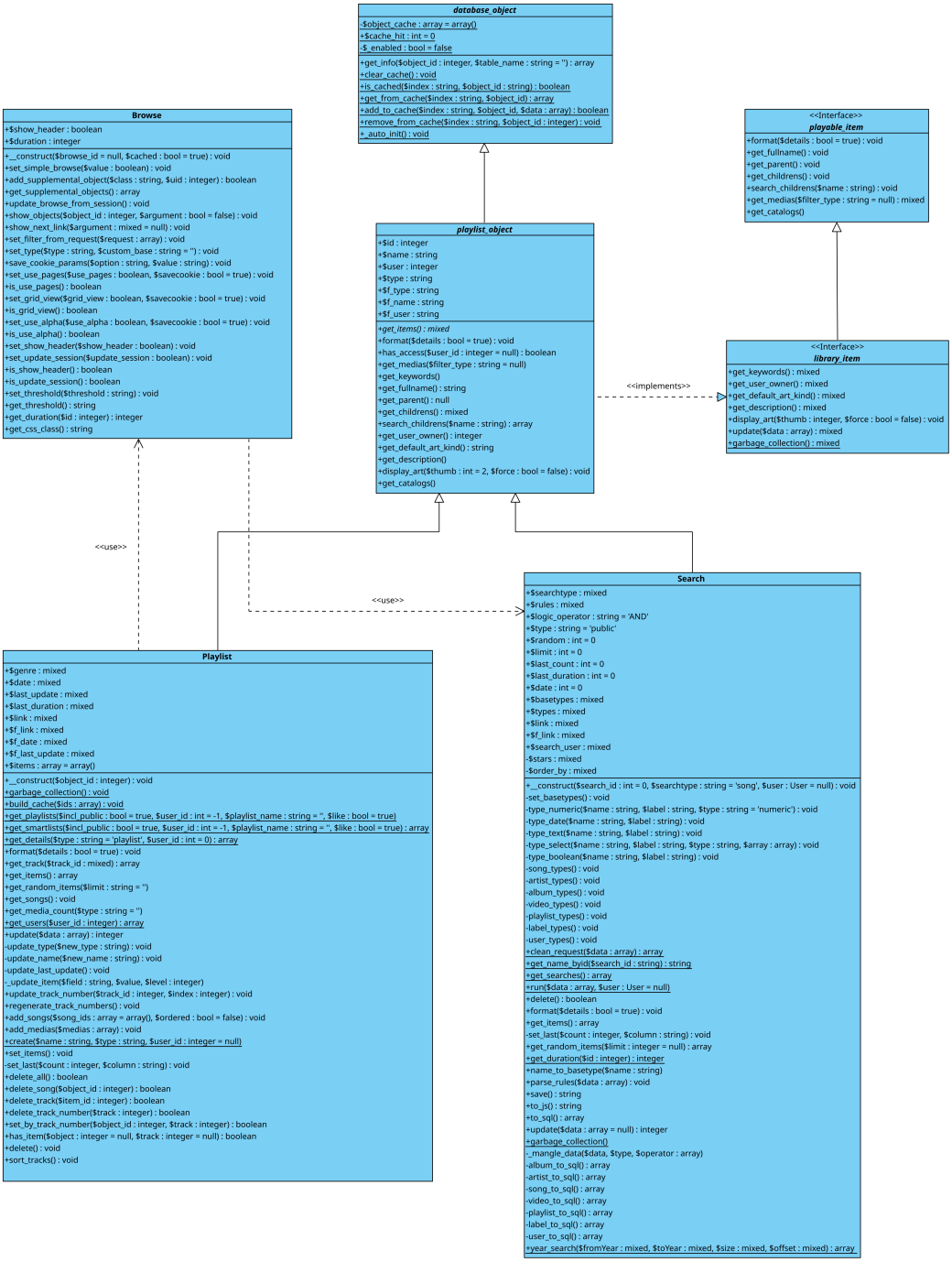


Figura 3.12: Diagrama de clases de la *issue* 2527.

Diagramas asociativos (#2527)

Tras obtener la información, es necesario mostrarla mediante la interfaz de usuario. Para facilitar esta labor, se han elaborado diagramas que asocian los elementos de la interfaz de usuario con las clases del sistema.

En este caso, dos páginas de la interfaz muestran la información relativa a las listas de reproducción: (1) el *script* `show_playlists` (Figura 3.13) se encarga de mostrar una lista de las listas de reproducción existentes y (2) el *script* `show_playlist` (Figura 3.14) muestra los elementos o canciones contenidas en una lista de reproducción concreta.

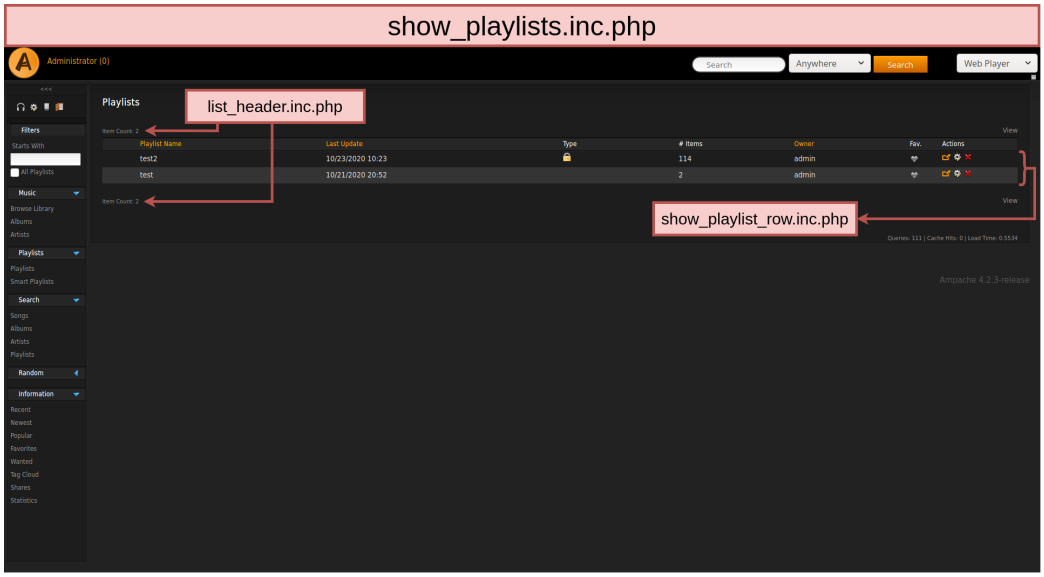


Figura 3.13: Interfaz de la página `show_playlists`.

Figura 3.14: Interfaz de la página `show_playlist`.

Diagrama de secuencia (#2527)

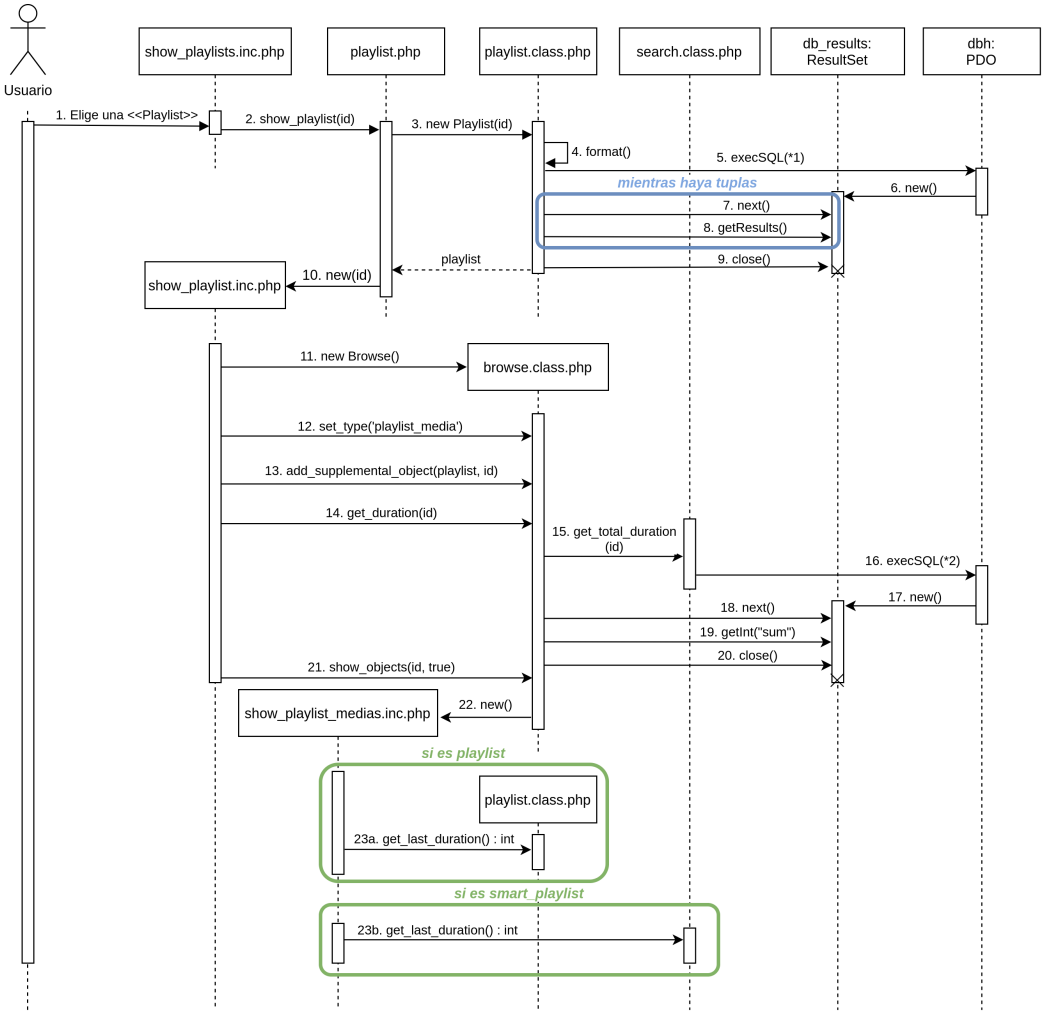
Es posible conseguir una visión más en detalle mediante el empleo de un diagrama de secuencia donde se describa la secuencia de acciones (Figura 3.15).

El *script* `show_playlists` muestra al usuario las listas de reproducción existentes (Figura 3.13) y el usuario selecciona una de las listas. Tras hacerlo, se instancia a la clase `Playlist`, la cual solicita a la base de datos la información de la lista de reproducción.

Tras la instanciación del objeto `Playlist` seleccionado, la información es mostrada por el *script* `show_playlist.inc.php` (Figura 3.14). En esta página se muestra tanto la información relativa a una lista de reproducción especifica, como las canciones que contiene.

En `show_playlist.inc.php` se delega la obtención de información a la clase `Browse`, la cual conociendo el atributo `id` de la `Playlist`, realiza una consulta al SGBD y obtiene la duración total (en segundos).

Finalmente, la clase `Browse` llama al *script* `show_playlist_medias.inc.php` el cual muestra al usuario la duración total de la lista de reproducción.



*1 : "SELECT `id`, `object_id`, `object_type`, `track` FROM `playlist_data` WHERE `playlist`= %id% ORDER BY `track`"

*2 : "SELECT SUM(`time`) FROM `song` WHERE `id` = %id%"

Figura 3.15: Diagrama de secuencia de la *issue* 2527.

3.3.2. #2582 [Feature Request] Changing login screen background

Tras realizar un análisis de esta *issue*, se ha identificado que para su implementación es necesario añadir una nueva opción de configuración a los ajustes de Ampache.

Examinando las opciones de configuración ya existentes para otros ajustes, se ha realizado un diseño que se asemeje al funcionamiento de las mismas.

Diagrama ER (#2582)

Primero se ha identificado cuáles son las tablas de la bases de datos que almacenan los ajustes de las preferencias de usuario.

Esta información es almacenada en las tablas: `user`, `user_preference` y `preference`. La tabla `user`, que almacena los usuarios del sistema, está relacionada con múltiples valores de la tabla `user_preference` la cual sirve de nexo con la tabla `preference` que es donde se almacenan los atributos de cada ajuste (Figura 3.16).

Por lo que es necesario realizar dos inserciones: la primera en la tabla `preference` con los atributos necesarios para poder almacenar el fondo de pantalla que defina el usuario y la segunda en la tabla `user_preference` para enlazar el ajuste con el usuario que lo haya definido.

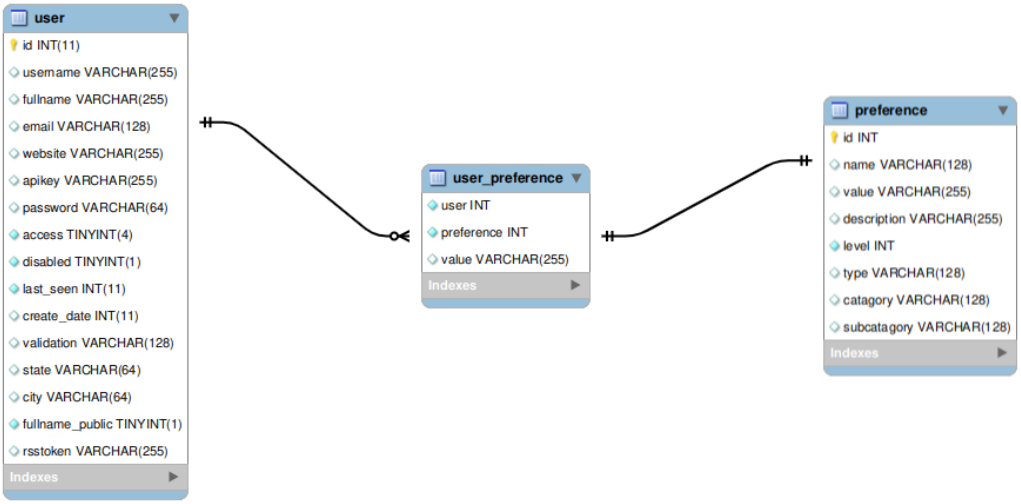


Figura 3.16: Diagrama ER de la *issue* 2582.

Diagrama de clases (#2582)

Esta *issue* involucra la modificación de tres clases: `Update`, `Preference` y `UI` (Figura 3.17).

`Update` es la clase encargada de las actualizaciones del esquema de la bases de datos, por lo que es en ella donde se introducen las modificaciones mencionadas en la subsección anterior.

La clase `Preference`, que hereda de la clase abstracta `database_object`, es donde se gestionan las preferencias de Ampache. Es en esta clase donde se habilitará la lógica que permita al usuario modificar la nueva preferencia introducida desde el panel de control de Ampache.

Al ser una funcionalidad que muestra un elemento visual (el fondo de pantalla), también es necesaria una modificación de la clase `UI` de modo que se muestre gráficamente la imagen definida por el usuario.

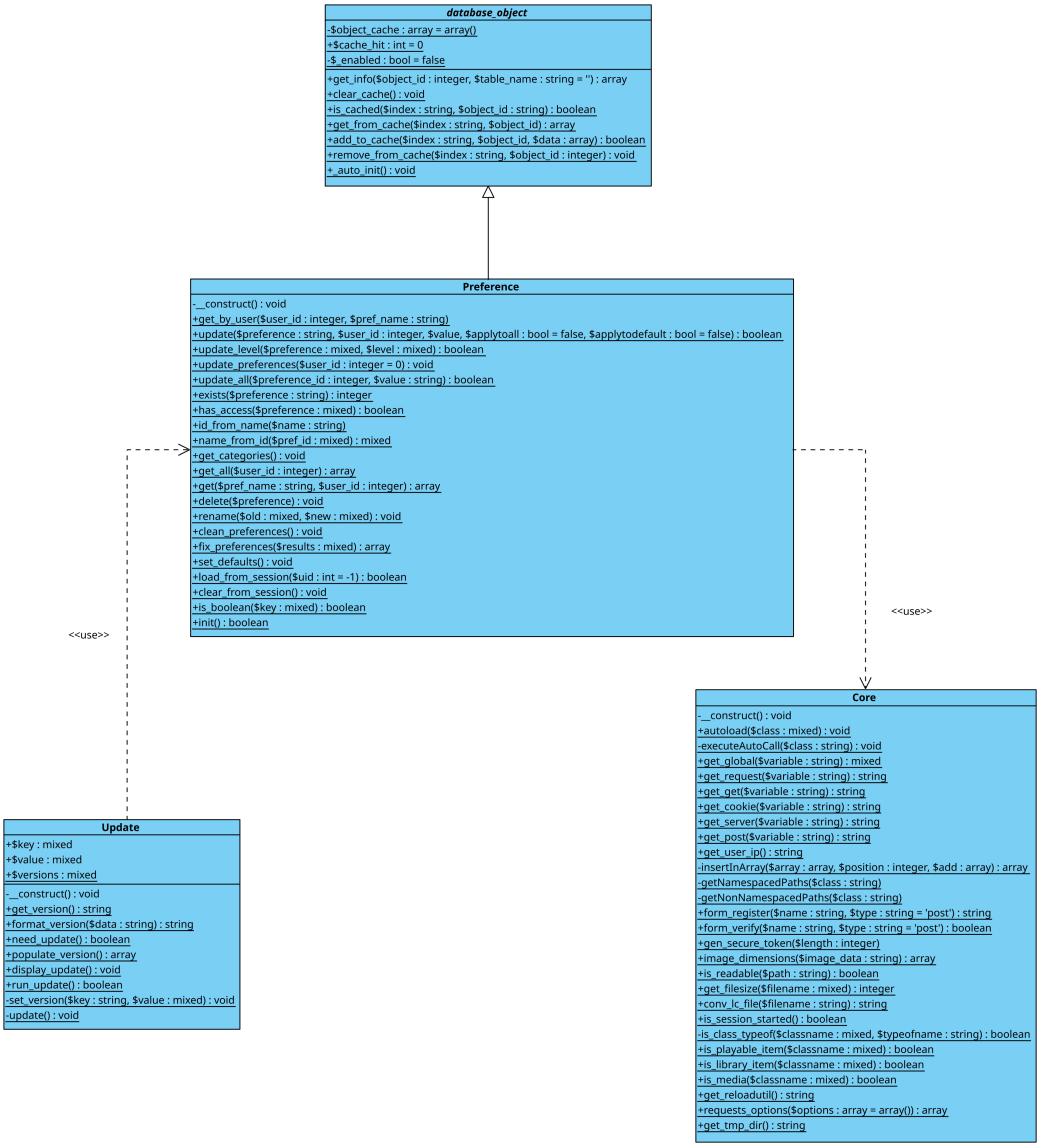


Figura 3.17: Diagrama de clases de la *issue* 2582.

Diagrama asociativos (#2582)

Con el objetivo de obtener una representación visual que facilite la comprensión de los elementos a modificar, se han generado unos diagramas (Figuras 3.18 y 3.19) que asocian los elementos de la interfaz a las clases correspondientes.

Por una parte, se ha identificado que la lógica de la página de inicio de sesión se encuentra en el archivo `login.php`. Sin embargo, para mostrar un fondo de pantalla, no es necesaria la modificación de la lógica de inicio de sesión, sino la introducción de un nuevo estilo en la clase `UI.class.php`.

Por otra parte, es necesaria la introducción de una nueva preferencia y que esta sea accesible desde la página de ajustes `preferences.php`. Para ello se insertará una nueva variable `custom_login_backgound` que permita la modificación de esta nueva preferencia desde la página de preferencias.

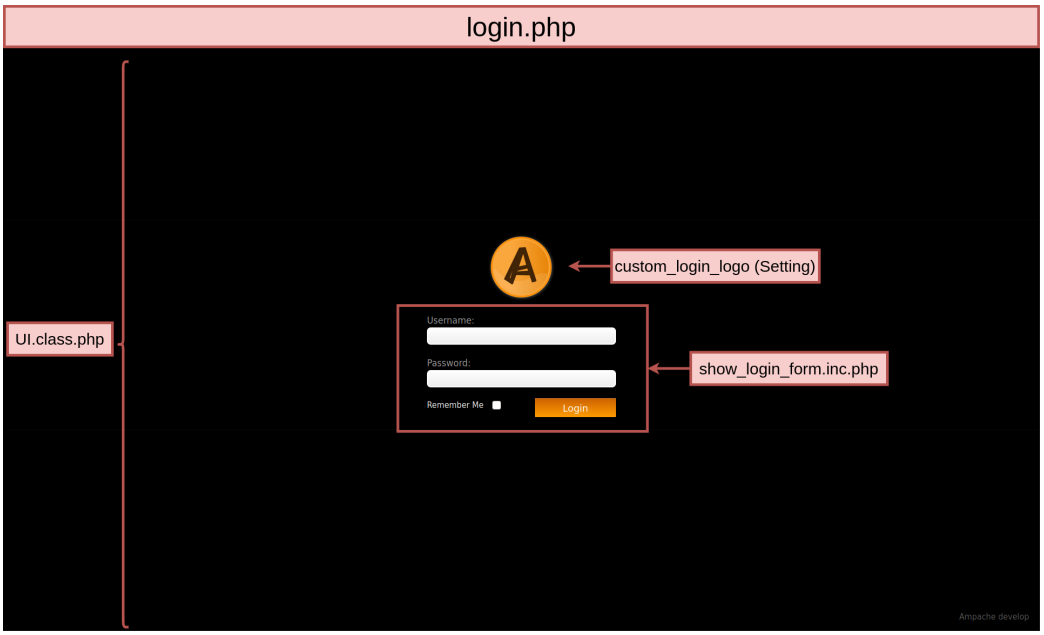


Figura 3.18: Interfaz de la página `login.php`.

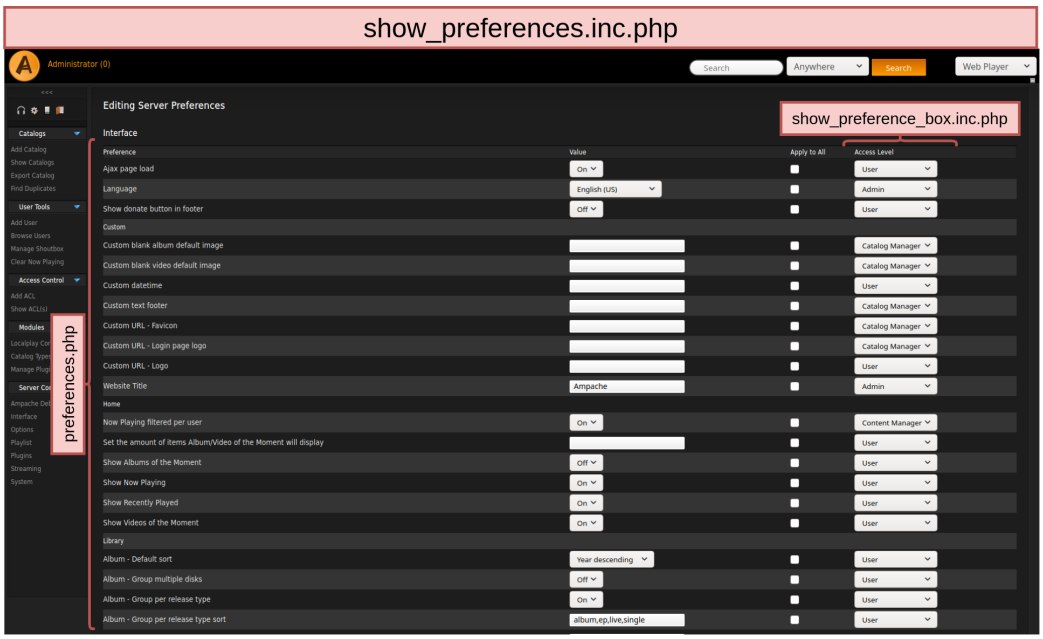


Figura 3.19: Interfaz de la página `show_preferences.inc.php`.

Diagramas de secuencia (#2582)

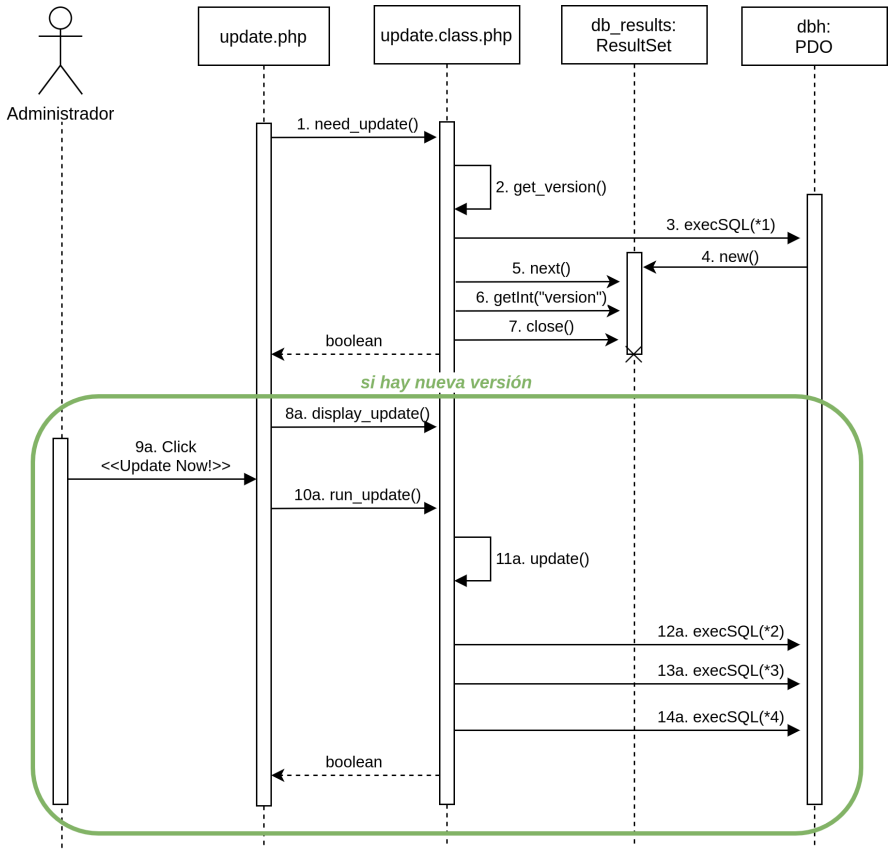
Se presentan dos diagramas de secuencia para esta funcionalidad. El primero representa el proceso de actualización de versión que inserta en la base de datos la nueva preferencia. El segundo representa los pasos que sigue un usuario para definir la nueva preferencia y comprobar que ha sido establecida correctamente.

Proceso de actualización

El flujo del proceso de actualización (Figura 3.20) comienza con una comprobación en la que el *script* `update.php` solicita a la clase `Update` la comprobación de si es necesaria una actualización.

La clase `Update` realiza una *query* a la base de datos — que es donde se almacena la versión actual — para obtener la versión. Si la versión actual instalada es inferior a la última versión disponible, se le muestra al usuario una pantalla de actualización (Figura 3.21) en la que el usuario debe seleccionar el botón «Update Now!» para comenzar la instalación de la actualización.

Es entonces cuando se insertará en las tablas `preference` y `user_preference` de la base de datos la nueva preferencia definida para establecer como ajuste el fondo de pantalla. Finalmente se actualiza la versión, evitando así repetir nuevamente este proceso.



*1 : "SELECT * FROM `update_info` WHERE `key`='db_version'"

*2 : "INSERT INTO `preference` (`name`, `value`, `description`, `level`, `type`, `catagory`, `subcatagory`)
VALUES ('custom_login_background', '', 'Custom URL - Login page background', 75, 'string', 'interface', 'custom')"

*3 : "INSERT INTO `user_preference` VALUES (-1, ?, ' ')"

*4 : "UPDATE update_info SET value='$value' WHERE `key`='db_version'"

Figura 3.20: Diagrama de secuencia del proceso de actualización de la *issue* 2582.

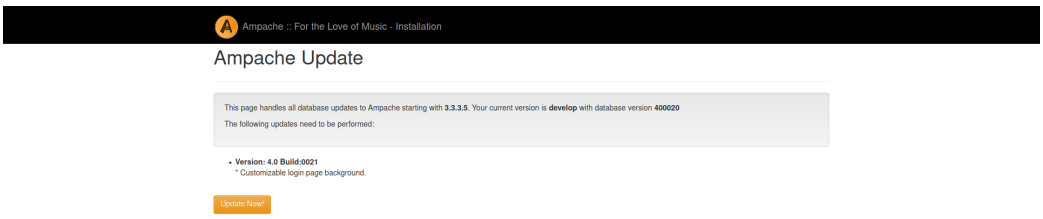


Figura 3.21: Interfaz de actualización.

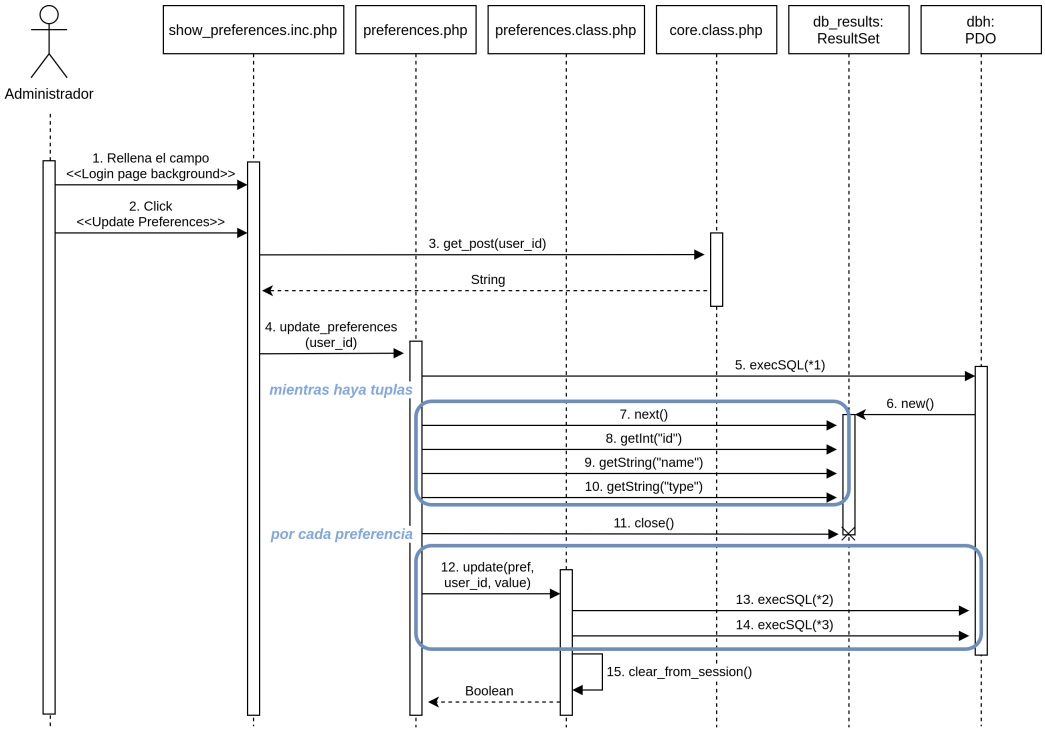
Proceso de configuración

Para configurar la nueva preferencia introducida (Figura 3.22), el usuario accede a la página de ajustes `show_preferences` 3.19.

Es en esta página donde rellena el campo «Login page background» indicando la ruta a la imagen y pulsa el botón «Update Preferences» para guardar los cambios realizados.

El *script* `show_preferences.inc.php` solicita el identificador del usuario que ha realizado el cambio y lanza una petición al módulo `preferences.php` para que actualice en la base de datos la nueva preferencia.

Primero se ejecuta una consulta para obtener el id, nombre y tipo de preferencia a modificar. Después, se actualizan las tablas `preference` y `user_preference` con los valores introducidos por el usuario en la interfaz.



*1 : "SELECT `id`, `name`, `type` FROM `preference`"

*2 : "UPDATE `preference` SET `value`='$value' WHERE `id`='$pref_id'"

*3 : "UPDATE `user_preference` SET `value`='$value' WHERE `preference`='$pref_id'$user_check"

Figura 3.22: Diagrama de secuencia del proceso de configuración de la *issue* 2582.

3.3.3. Cliente React - Página álbumes

Actualmente, al acceder a esta página se obtiene un mensaje de error (Figura 3.23) dado que todavía no está implementada.

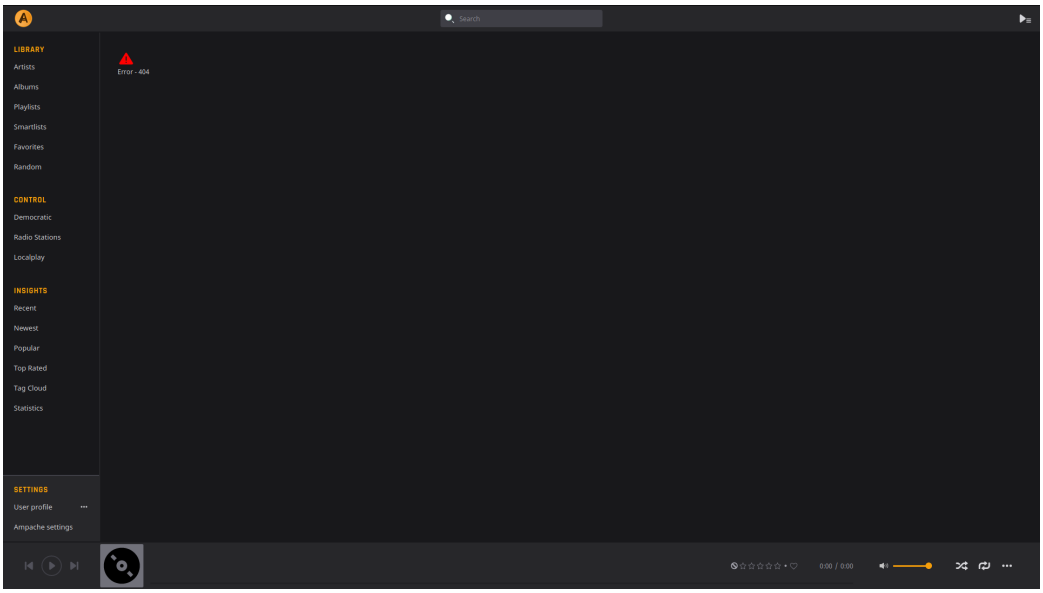


Figura 3.23: Estado actual de la página de álbumes.

Por tanto, el objetivo es obtener y mostrar una cuadrícula con la información de todos los álbumes almacenados.

Diagramas de secuencia (Página álbumes)

En el diagrama de secuencia (Figura 3.24) se muestra cómo cuando un usuario accede a la página de álbumes, esta solicita la información necesaria para mostrarlos.

Para ello, realiza una llamada a la función `getAlbums()` de la clase `Albums` de la parte lógica y suministra como parámetro la variable de entorno `authKey`. Esta es la única variable necesaria para realizar llamada a la API y obtener la información solicitada en un formato JSON.

Los posibles errores en la comunicación — como no llegar a obtener una respuesta de la API u obtener una respuesta JSON que se encuentre vacía — son tratados y se le informa al usuario de su occurrencia.

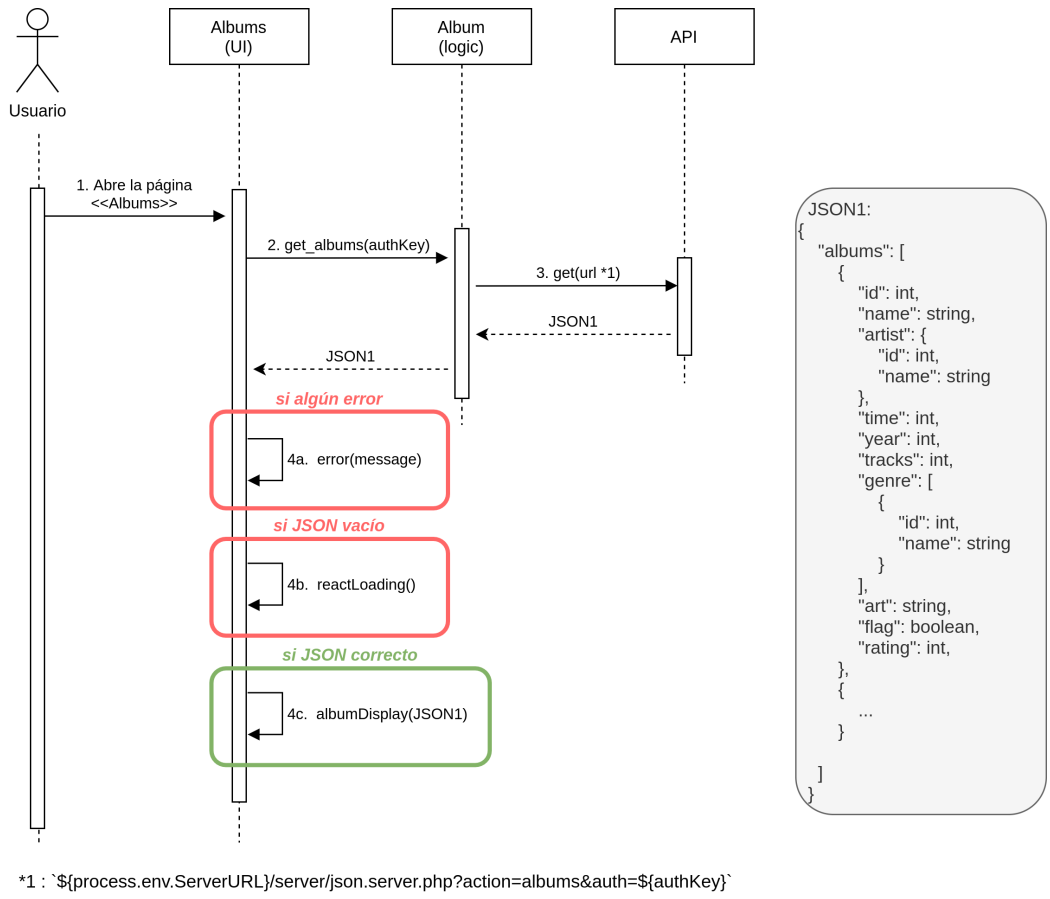


Figura 3.24: Diagrama de secuencia de la página de álbumes.

3.3.4. Cliente React - Favoritos

Al acceder a una lista de canciones (pistas de un álbum, canciones de una lista de reproducción, etc.), un usuario dispone de un botón con forma de corazón para marcar una canción como favorita (Figura 3.25).

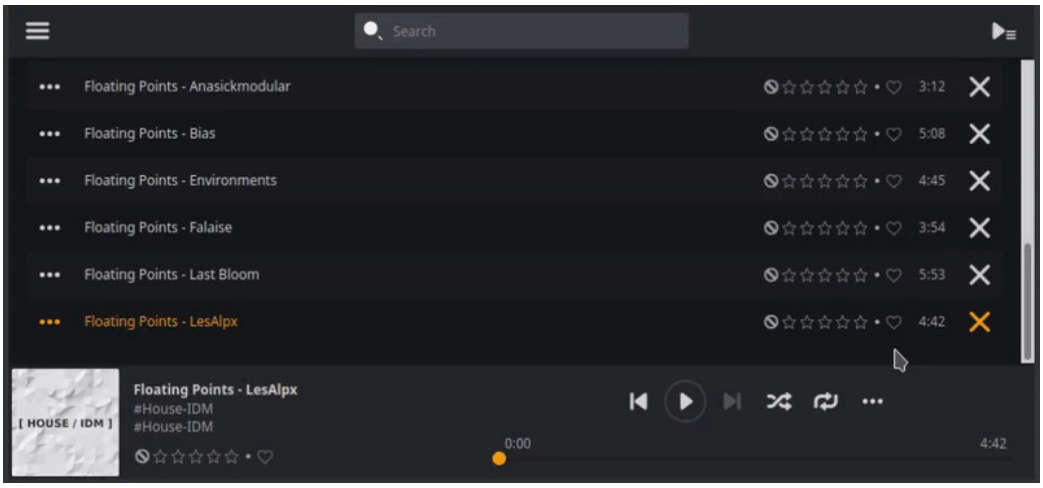


Figura 3.25: Estado actual de la funcionalidad favoritos.

En el presente, la funcionalidad de este botón no está implementada, por lo que, si se pulsa, no se produce ninguna acción.

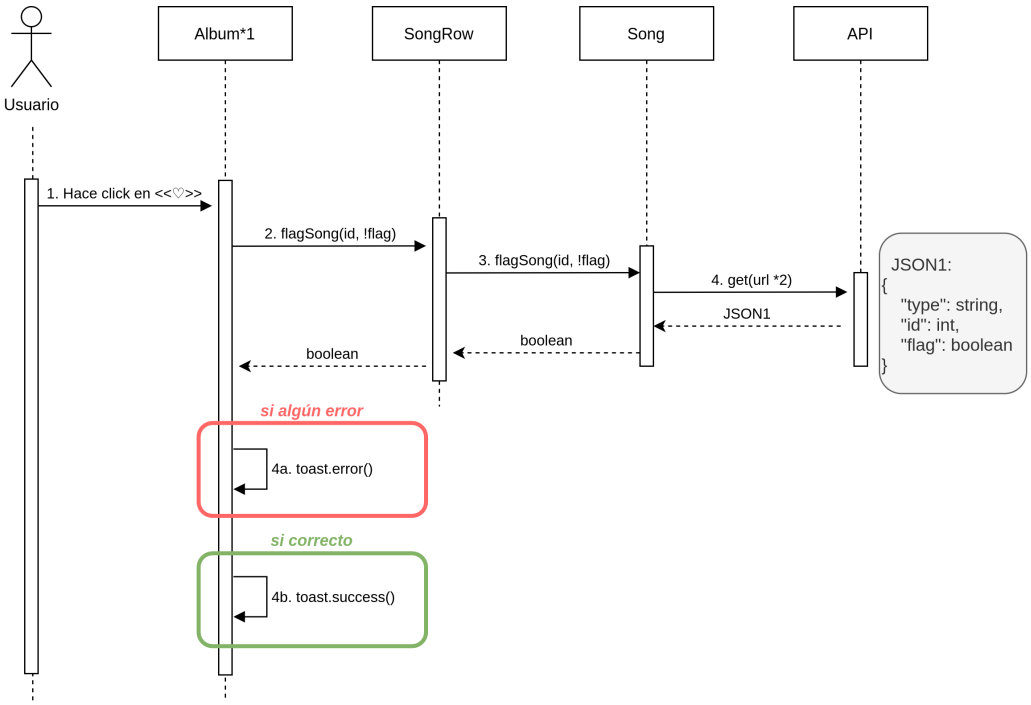
El objetivo consiste en conectar el *frontend* React con el *backend* de modo que quede registrada esta acción.

Diagramas de secuencia (Favoritos)

Cuando un usuario accede a una lista de canciones (álbumes, *playlists*, etc.), cada canción se muestra en una fila mediante el elemento `SongRow`.

Para añadir o eliminar una canción de favoritos, un usuario puede pulsar el botón con forma de corazón. Esta acción lanza una llamada a la API para invertir el valor existente de la etiqueta favoritos y devuelve en un JSON el valor actualizado (Figura 3.26).

Al usuario se le informa del resultado de esta acción, tanto si el proceso es exitoso como si sucede algún error, mediante un *pop-up* con estilo *toast*.



*1 : También sirven otro tipo de listas de canciones que contengan el elemento SongRow, como una playlist o una búsqueda filtrada.

*2 : `${process.env.ServerURL}/server/json.server.phpaction=flag&type=song&id=${songID}&flag=${Number(favorite)}`

Figura 3.26: Diagrama de secuencia de la funcionalidad favoritos.

3.4. Desarrollo

Esta sección detalla el proceso individual seguido en la implementación de cada *issue*. Se trata de un proceso iterativo que parte de una solución inicial, incluye cambios en base a la retroalimentación de la comunidad y concluye con un resultado final.

Se incluye también una explicación de los problemas encontrados y las soluciones propuestas en este proceso de desarrollo.

3.4.1. #2527 [Feature Request] Total time

Primera aproximación

Para evitar el desarrollo de código redundante, se ha realizado una búsqueda de funciones ya definidas que faciliten la implementación de esta funcionalidad.

En este caso, se ha encontrado que existe una función `get_total_duration()` (Código 3.1) la cual realiza una suma de tiempos a nivel de BD.

Partiendo de ella, se ha incluido una nueva columna en la página `show_playlists.inc.php`. Esta columna, muestra los segundos calculados por `get_total_duration()` en un formato `HORAS:MINUTOS:SEGUNDOS` para una mejor comprensión (Figura 3.27).

```php
/**
* get_total_duration
* Get the total duration of all songs.
* @return string|null
*/

public function get_total_duration()
{
    $songs  = $this->get_songs();
    $idlist = '(' . implode(',', $songs) . ')';
    if ($idlist == '()') {
        return null;
    }
    $sql        = "SELECT SUM(`time`) FROM `song` WHERE `id`
    IN $idlist";
    $db_results = Dba::read($sql);

    $results = Dba::fetch_row($db_results);

    return $results['0'];
}
```

Código 3.1: Función `get_total_duration()`.

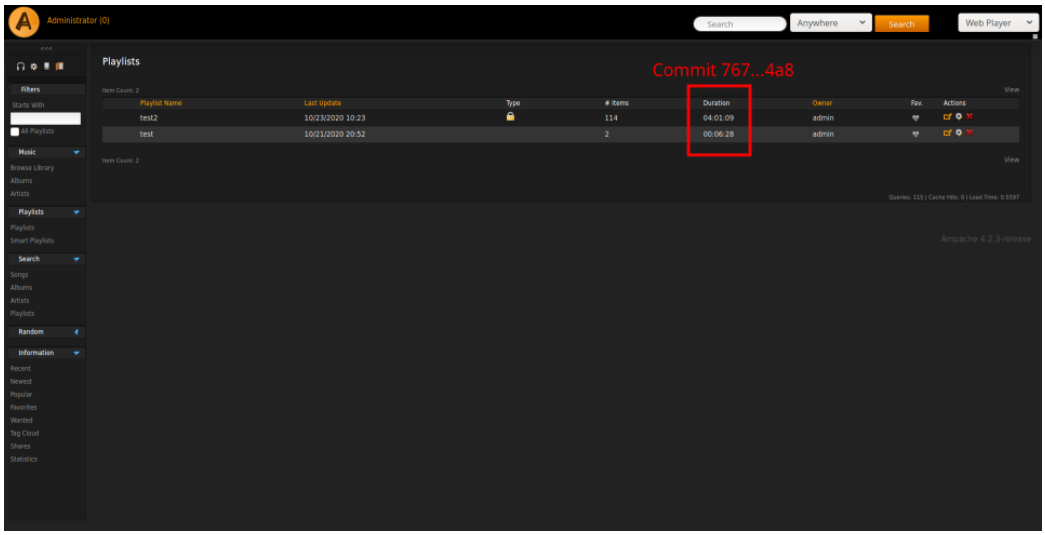


Figura 3.27: Cambios en la página `show_playlists.inc.php` para mostrar la duración total.

Cambios en base a retroalimentación

Tras realizar esta primera aproximación, el usuario que inicialmente abrió la *issue* propone incluir[5] esta misma información también en la página `show_playlist.inc.php`.

Siguiendo su propuesta, se realiza un nuevo *commit* en el que se incluye el tiempo total junto al elemento `Item Count` de la interfaz (Figura 3.28).

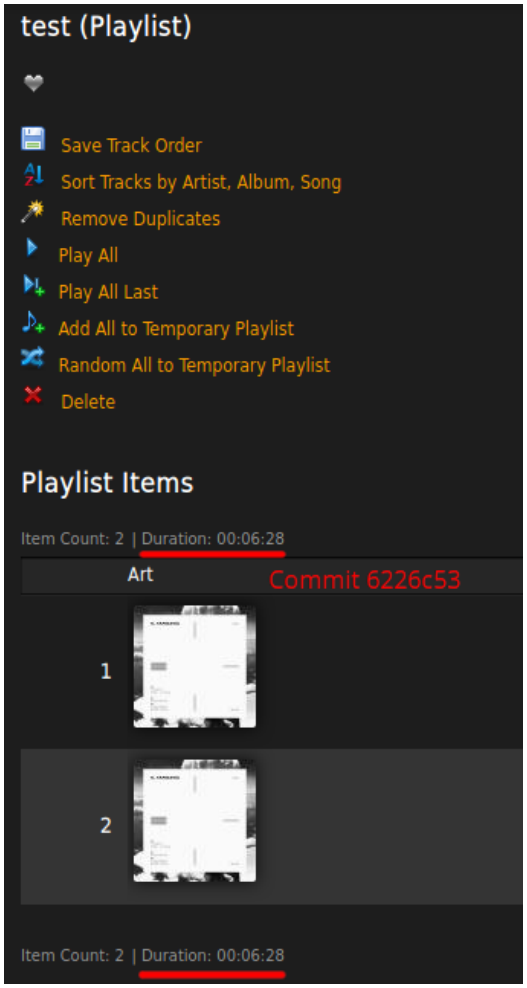


Figura 3.28: Cambios en la página `show_playlist.inc.php` para mostrar la duración total.

[5] @4phun (creador de la *issue*)«Hi, this is cool! Can we get it on the playlist screen too? Where the songs are displayed.»

Problemas y soluciones con la implementación

Con los nuevos cambios se descubre un problema: la página `show_playlist.inc.php` solamente funciona con las listas de reproducción de tipo `Playlist`, mientras que la página no carga para las de tipo `Smart Playlist`.

Las `Smart Playlist` son listas de reproducción generadas en base a filtros (p. ej. en base a rangos en las fechas de publicación). Debido a este motivo, son objetos de tipo `Search` en lugar de tipo `Playlist`. Esto hace que la llamada a la función `get_total_duration()` genere un error[6] dado que no encuentra el objeto `Playlist` esperado.

Este problema se ha solucionado realizando una comprobación del tipo de objeto recibido. Comprobando mediante un operador ternario cual es el tipo de objeto recibido para poder llamar a la función `get_total_duration()` con los parámetros adecuados (Código 3.2).

```
1 $playlist_duration = $playlist === NULL ? $search->
    get_total_duration($object_ids) : $playlist->
    get_total_duration();?>
```

Código 3.2: Comprobación de clase mediante operador ternario.

Tras su resolución, se ha procedido a la realización de pruebas descritas en la sección «4.1 Pruebas de calidad».

[6] `PHP Fatal error: Uncaught Error: Call to a member function on null`

3.4.2. #2582 [Feature Request] Changing login screen background

Primera aproximación

En primer lugar se han analizado las preferencias existentes en busca de una con un funcionamiento similar a tener como referencia. Tras esta búsqueda se ha encontrado la preferencia «Custom URL - Login page logo», la cual permite modificar el logotipo de Ampache por uno definido por el usuario.

Siguiendo la misma lógica se ha añadido en la función `show_custom_style()` (Código 3.3) un estilo para mostrar gráficamente el fondo de pantalla si este está definido.

```
1 public static function show_custom_style()
2 {
3     if (AmpConfig::get('custom_login_background')) {
4         echo "<style> body { background-image: url('" .
    AmpConfig::get('custom_login_background') . "') !important
    ; }</style>";
5     }
6
7 }
```

Código 3.3: Adición realizada a la función `show_custom_style()`.

La inserción de la nueva preferencia en la base de datos se ha definido mediante la función `update_400020()` (Código 3.4). Esta función muestra el código de una migración de la base de datos donde se añade la entrada `custom_login_background` en la tabla `preference` de la base de datos.

```
1  /**
2   * update_400020
3   *
4   * Customizable login background image
5   */
6
7  public static function update_400020()
8  {
9      $retval = true;
10
11     $sql = "INSERT INTO `preference` (`name`, `value`, `
       description`, `level`, `type`, `catagory`, `subcatagory`)
       " .
12         "VALUES ('custom_login_background', '', 'Custom URL -
        Login page background', 75, 'string', 'interface', '
       custom')";
13     $retval &= Dba::write($sql);
14     $row_id = Dba::insert_id();
15     $sql    = "INSERT INTO `user_preference` VALUES (-1,?,
       '')";
16     $retval &= Dba::write($sql, array($row_id));
17
18     return $retval;
19 }
```

Código 3.4: Función `update_400020()`.

El número `400020` sigue la nomenclatura[7] utilizada para mantener el control de las migraciones de la base de datos. Las migraciones se realizan de manera secuencial y el número `400020` de esta inserción representa la versión 4.0 actualización 20.

También se ha añadido el *string* «Custom URL - Login page background» a los archivos de traducción `messages.pot` y `translatable_database_strings.txt` para permitir su traducción a los más de 20 idiomas en los que está disponible Ampache.

[7] «This class mainly handles schema updates for the database. Versions are a monotonically increasing integer: First column(s) are the major version, followed by a single column for the minor version and four columns for the build number. 3.6 build 1 is 360000; 10.9 build 17 is 1090017.»

Tras realizar los cambios mencionados, se ha conseguido añadir exitosamente la nueva preferencia «Custom URL - Login page background» en la página de ajustes (Figura 3.29) y que se muestre la imagen en la página de inicio de sesión (Figura 3.30).

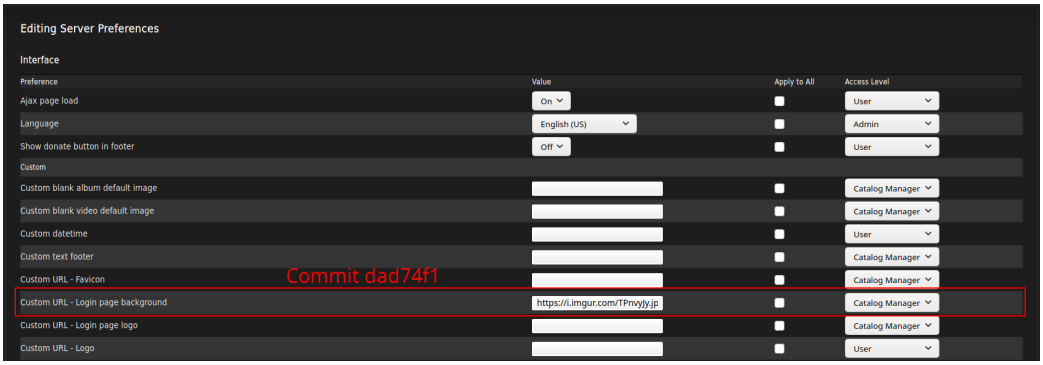


Figura 3.29: Cambios en la página `show_preferences.inc.php` para mostrar la nueva preferencia.

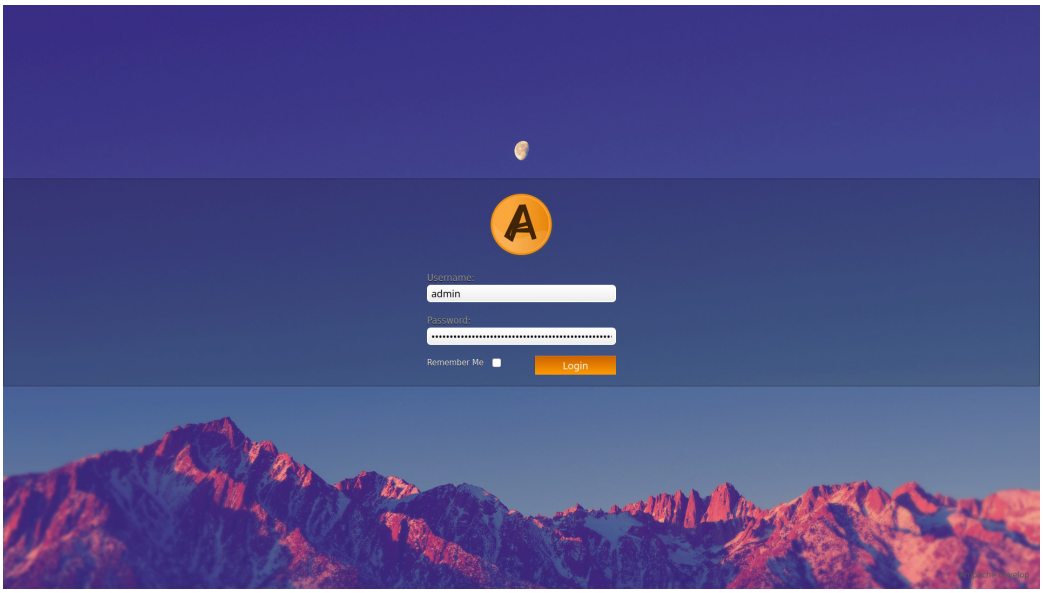


Figura 3.30: Página de inicio de sesión tras definir un fondo.

3.4.3. Cliente React - Página álbumes

En las *issues* anteriores se han añadido funcionalidades a una página ya existente. En cambio, en esta ocasión se está desarrollando el módulo principal de una página.

Primero de todo, se ha añadido una regla de enrutamiento en `router.tsx` de tal modo que cuando un usuario acceda a la ruta `/albums` deje de obtener un error 404 y, en su lugar, se muestre la página de álbumes.

La información relativa a los álbumes se obtiene mediante una llamada a la API. Para realizar esta llamada, se ha definido la función `getAlbums` en la capa lógica del cliente (Código 3.5).

```
1  const getAlbums = (authKey: AuthKey, includeSongs = false) =>
       {
2      let includeString = '';
3      if (includeSongs) {
4          includeString += '&include[]=songs';
5      }
6      const getUrl = `${process.env.ServerURL}/server/json.
       server.php?action=albums&auth=${authKey}${includeString}&
       version=400001`;
7      return axios.get(getUrl).then((response) => {
8          const JSONData = response.data;
9          if (!JSONData) {
10             throw new Error('Server Error');
11         }
12         if (JSONData.error) {
13             throw new AmpacheError(JSONData.error);
14         }
15         return JSONData.album as Album[];
16     });
17 };
```

Código 3.5: Función `getAlbums`.

En la linea 6 está definida la llamada a la API. En concreto, se está haciendo uso de la acción `albums` (Tabla 3.1) y suministrando la variable `authKey` que contiene la sesión del usuario.

La respuesta esperada es un texto JSON que contiene la información (id, nombre, artista, etc.) de todos los álbumes. Esta información es introducida en el *array* `Album[]` y devuelta a la capa de vista del cliente.

albums

This returns albums based on the provided search filters

Input	Type	Description	Optional
'filter'	string	Value is Alpha Match for returned results, may be more than one letter/number	YES
'exact'	boolean	if true filter is exact rather then fuzzy	NO
'add'	set_filter	ISO 8601 Date Format (2020-09-16)	YES
		Find objects with an 'add' date newer than the specified date	
'update'	set_filter	ISO 8601 Date Format (2020-09-16)	YES
		Find objects with an 'update' time newer than the specified date	
'offset'	integer	Return results starting from this index position	YES
'limit'	integer	Maximum number of results to return	YES
'include'	string	'albums', 'songs' will include nested in the album JSON	YES

Tabla 3.1: Parámetros aceptados para la acción `Albums` de la API.

Una vez el *array* que contiene la información de los álbumes es recibido en la vista, se realiza un mapeo con la función genérica `map()` y se muestran en forma de cuadrícula (Figura 3.31).

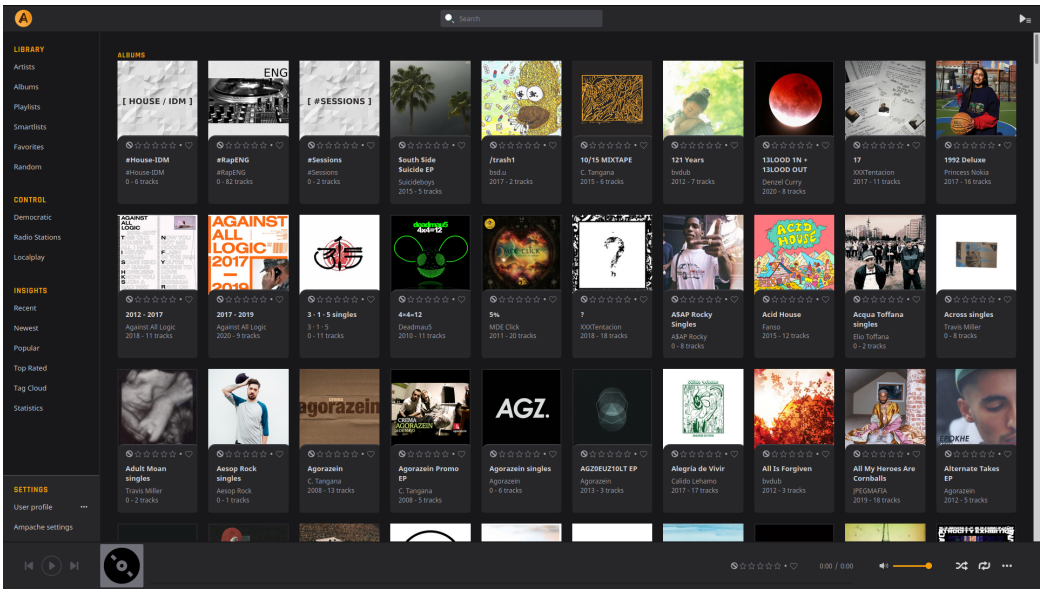


Figura 3.31: Página de álbumes tras el desarrollo.

3.4.4. Cliente React - Favoritos

Esta funcionalidad cuenta con una interfaz ya implementada *a priori*, en la fase de desarrollo se ha definido la lógica necesaria para recoger, comunicar y almacenar las acciones del usuario.

Para ello, ha sido necesario que cada fila `SongRow` de una lista de canciones conozca los atributos `song`, `fav` y tenga accesible la función `flagSong`. El botón con forma de corazón contiene un *listener* de eventos, de modo que cuando el usuario lo pulsa se acciona la función `flagSong`.

Al llamar a la función `flagSong` (Código 3.6) se niega el valor actual del atributo `fav`[8]. De este modo, si la canción no está en favoritos y se hace `click` en el botón con forma de corazón, la canción pasará a estar marcada como favorita y viceversa.

```
export const flagSong = (songID: number, favorite: boolean,
    authKey: AuthKey) => {
     return axios
         .get(
             `${process.env.ServerURL
             }/server/json.server.php?action=flag&type=song&id
    =${songID}&flag=${Number(
                 favorite)}&auth=${authKey}&version=400001`
         )
         .then((response) => {
             const JSONData = response.data;
             if (!JSONData) {
                 throw new Error('Server Error');
             }
             if (JSONData.error) {
                 throw new AmpacheError(JSONData.error);
             }
             return true;
         });
};
```

Código 3.6: Función `flagSong`.

[8] `props.flagSong(props.song.id, !props.song.flag);`

La función `flagSong` realiza una llamada a la acción `flag` (Tabla 3.2) de la API indicando que se trata de un elemento tipo `song` usando el `id` y `flag` recibidos en los parámetros de la función.

flag

This flags a library item as a favorite

- Setting flag to true (1) will set the flag
- Setting flag to false (0) will remove the flag

Input	Type	Description	Optional
'type'	string	'song', 'album', 'artist', 'playlist',	NO
		'podcast', 'podcast_episode', 'video'	
		'tvshow', 'tvshow_season'	
'id'	integer	$object_id	NO
'flag'	boolean	0, 1	NO

Tabla 3.2: Parámetros aceptados para la acción `Flag` de la API.

4. Resultados

4.1. Pruebas de calidad

Es posible que los resultados obtenidos tras la ejecución del código desarrollado diverjan de los resultados esperados o incluso que el código contenga *bugs*.

Además, dada la complejidad de ciertos proyectos *software*, introducir cambios o nuevas funcionalidades puede generar conflictos con el código existente. Debido a ello, surge la necesidad de realizar pruebas (unitarias, de regresión, de rendimiento, etc.) para asegurar que el código alcanza un nivel de calidad óptimo.

4.1.1. Pruebas automatizadas

Ampache hace uso del sistema de integración continua y testeo Travis CI[1]. Travis CI se configura mediante un archivo `.yml`[2] (Código 4.1) en el directorio raíz del repositorio.

Este archivo especifica el lenguaje de programación utilizado y el entorno de construcción y prueba deseado, incluidas las dependencias que deben instalarse antes de que se pueda compilar y probar el *software*.

[1]Travis CI: test and deploy your code with confidence (`www.travis-ci.org`)

[2]YAML es un formato de serialización estándar de datos para todos los lenguajes de programación. (YAML [2020])

```
language: php
php:
 - 7.1
 - 7.3
 - 7.4

os: linux
dist: xenial

before_install:
 - export PATH="$PATH:$HOME/.composer/vendor/bin"
 - composer global require friendsofphp/php-cs-fixer:~2.15.0
 -
before_script:
 - chmod +x scripts/tests/syntax.sh
 - chmod +x scripts/tests/codestyle.sh

script:
 - scripts/tests/syntax.sh
 - scripts/tests/codestyle.sh
```

Código 4.1: Configuración `.yaml` de Ampache en la herramienta Travis CI.

En este caso, se generan tres entornos de ejecución con distintas versiones de PHP (7.1, 7.3 y 7.4) y cada entorno se ejecuta sobre un sistema GNU/-Linux Ubuntu (versión 16.04 LTS Xenial Xerus).

Tras hacer un *pull request*, Travis CI se encarga de compilar y ejecutar automáticamente los nuevos cambios de código. Además, comprueba que tanto la sintaxis (`syntax.sh`) como el estilo de código (`codestyle.sh`) siguen las directrices establecidas por el proyecto[3].

Todo ello genera un *log* (Figura 4.1). Si el código de salida es «0» el resultado es exitoso, en caso contrario se han producido uno o varios errores.

[3] «Ampache Coding Standards» y «Ampache Code Philosophy».

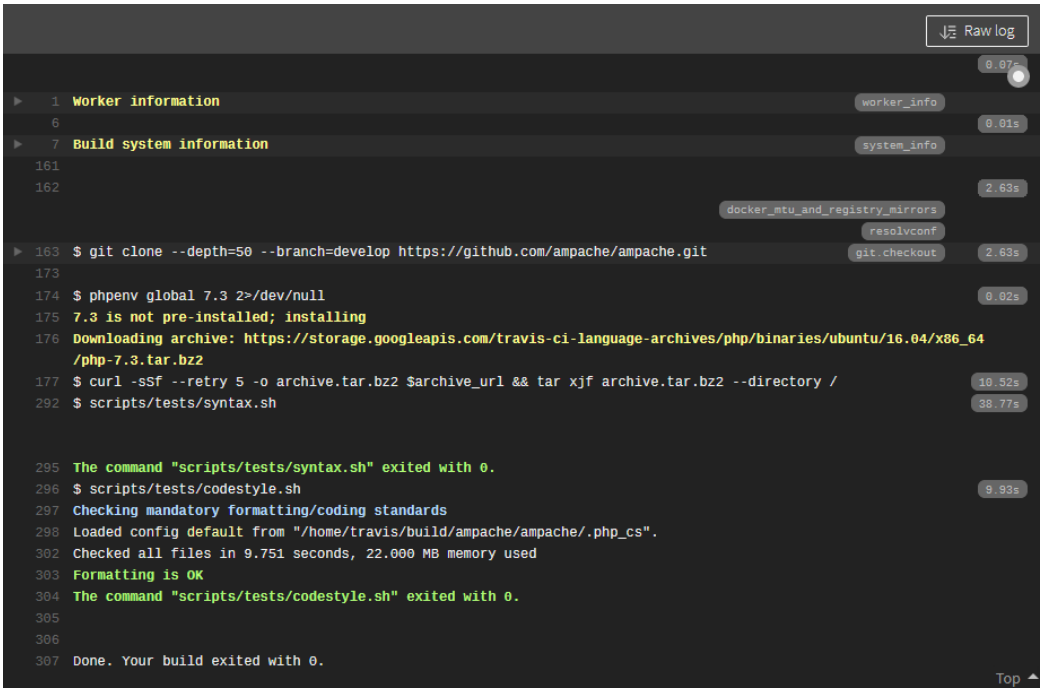


Figura 4.1: *Log* (reducido) de las pruebas automatizadas sobre el *pull request* 2527.

Es requisito necesario — pero no suficiente — superar todas las pruebas automatizadas para que un *pull request* pueda ser aceptado. Todos los *pull requests* propuestos en este trabajo han superado las pruebas de integración.

4.1.2. Pruebas manuales

#2527 [Feature Request] Total time

En primer lugar, se han realizado pruebas con distintos números de elementos (1, 5 y 100) para comprobar que tanto la suma total en segundos, como su representación en el formato `HORAS:MINUTOS:SEGUNDOS` sea correcta.

Realizando estas pruebas, se ha encontrado que en la solución inicial el número de horas hacía un *overflow* al llegar al valor 24 debido a representar los valores mediante la función `gmdate()`, la cual está diseñada para representar las horas de un día.

Para solucionar este problema, se ha dividido la suma total de segundos entre 3600 obteniendo así el número de horas y anexando a ello los minutos y segundos obtenidos mediante `gmdate()`.

Tras comprobar que el tiempo se muestra correctamente, se ha verificado si el tiempo de carga de la página `playlists.php` se mantiene en un margen aceptable. Para ello, se ha comparado el tiempo de carga previo con el tiempo de carga tras introducir los nuevos cambios (Tabla 4.1).

Elementos	Tiempo Previo	Tiempo Posterior
2	0.3539s	0.3582s
115	3.9702s	4.0244s
315	5.8598s	5.9271s

Tabla 4.1: Comparación en los tiempos de carga del *pull request* 2527.

Se encuentra una diferencia de alrededor de un 1 % en los tiempos de carga. Al no tratarse de una diferencia notable, se aceptan los resultados obtenidos.

#2582 [Feature Request] Changing login screen background

Usabilidad

En primer lugar, se ha realizado el proceso de actualización de versión desde el punto de vista de usuario para comprobar que la inserción del nuevo atributo en la tabla de la base de datos sea correcta.

Al tratarse de una funcionalidad con un elemento visual, se ha realizado una comprobación de su presentación en distintas resoluciones. Para ello se han tomado dos dispositivos como referencia: un monitor con una resolución 1080p[4] (Figura 4.2) y un dispositivo móvil con una resolución nativa[5] (Figura 4.3).

[4] 1920 *píxeles* de ancho y 1080 *píxeles* de alto.
[5] 400 *píxeles* de ancho y 720 *píxeles* de alto.

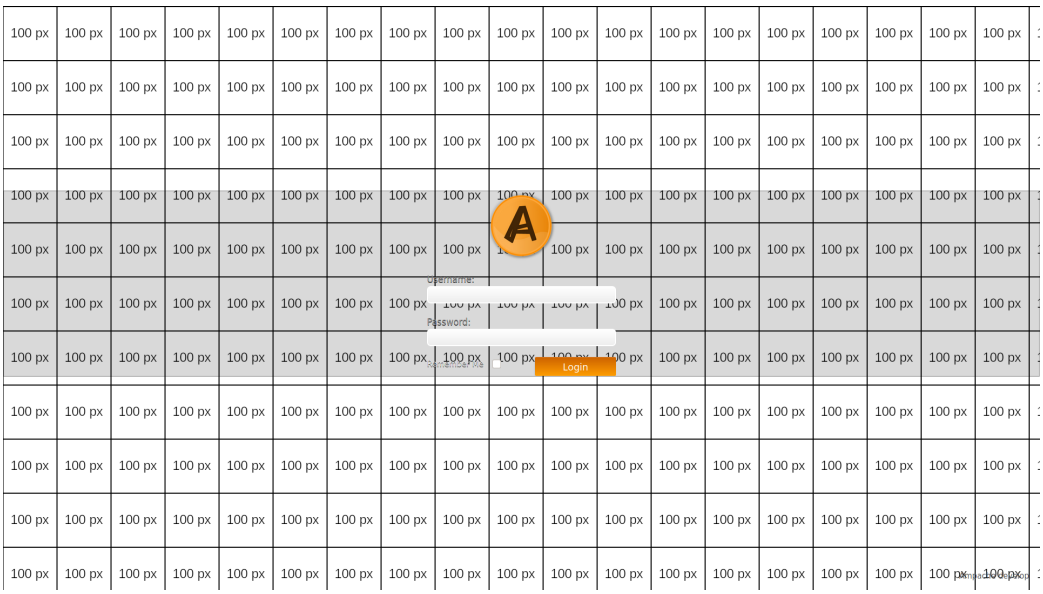


Figura 4.2: Comprobación de la representación visual de la *issue* 2582 en una resolución de 1080p.

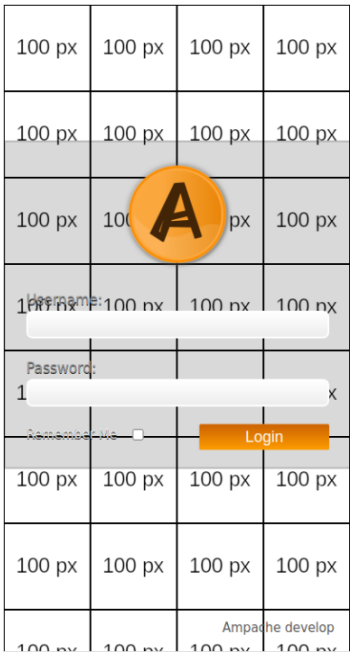


Figura 4.3: Comprobación de la representación visual de la *issue* 2582 en una resolución móvil de 720p.

Se puede apreciar que la imagen elegida — en este caso un rectángulo de 100 *píxeles* — se replica *ad infinitum* hasta cubrir toda la resolución disponible. Este es el comportamiento esperado de la función de PHP `background-image` utilizada para mostrar el fondo de pantalla.

Seguridad

Por otra parte, se ha comprobado el aspecto de seguridad de esta nueva característica. Al tratarse de un campo con entrada de texto libre — que además introduce un valor en la base de datos — existe la posibilidad de ataques tipo inyección SQL o Cross Site Scripting (XSS).

Un ejemplo de ataque tipo inyección SQL, podría consistir en introducir en el campo de entrada la sentencia:

```
imagen.png'); DROP TABLE preferences;--
```

con el objetivo de eliminar la tabla `preferences` de la base de datos.

Sin embargo, tras intentar realizar este ataque, no se ha conseguido penetrar la seguridad del sistema. Esto es debido a que la función `update()` de Ampache — que actualiza las preferencias — filtra adecuadamente la entrada esperada[6], salvaguardando así la seguridad del sistema de inyecciones maliciosas.

En cuanto a los ataques XSS, estos consisten en lograr la ejecución de un *script* malicioso en la parte del cliente que accede al sistema. Se ha probado a introducir el siguiente *script* inofensivo para comprobar si este vector de ataque es posible:

```
<script>alert("hacked")</script>
```

Nuevamente se ha verificado que este tipo de ataque no es posible en el sistema. En este caso es la función `get_post()` de la clase genérica `core.class.php` la que realiza la sanitización de la entrada eliminando los caracteres especiales (`<`, `>`, `/`, etc.) que pueda contener.

[6]Para ello utiliza la función genérica de PHP `filter_var()` con el filtro `FILTER_SANITIZE_STRING` que codifica o elimina caracteres especiales (`www.php.net/manual/en/function.filter-var.php`)

Cliente React - Página álbumes

Con el objetivo de comprobar tiempos de carga y posibles errores, se ha analizado el funcionamiento de la página con distintas cantidades de álbumes.

En caso de no tener ningún álbum en el catalogo, la página de álbumes se muestra vacía correctamente. Tanto con 1 como con 50 álbumes el tiempo de carga es adecuado y la página se muestra como es esperado (Figura 3.31).

Sin embargo, a partir de 100 álbumes el tiempo de carga se convierte excesivo, el cliente tarda demasiado en obtener y mostrar la información. Para solventar esta problemática, el desarrollador principal ha implementado un paginamiento a modo de *scroll infinito* de modo que la información se obtenga progresivamente en agrupaciones de 50 álbumes.

En caso de suceder un error obteniendo las información de los álbumes, se le informa al usuario mediante una notificación con el mensaje «Something went wrong getting the albums».

Cliente React - Favoritos

Se han comprobado dos aspectos fundamentales de esta integración: (1) que la interfaz sea actualizada correctamente tras pulsar el botón, es decir, que el icono con forma de corazón se rellena y vacíe respectivamente y (2) que la información quede almacenada en la tabla correspondiente de la base de datos.

Para facilitar esta tarea e informar al usuario de que su acción ha sido registrada, se han añadido dos notificaciones emergentes con los mensajes «Song added to favorites» y «Song removed from favorites».

Partiendo de una lista de canciones (Figura 4.4), cuando el usuario pulsa el botón de favoritos, el icono se rellena (Figura 4.5) y vacía (Figura 4.6) correctamente.

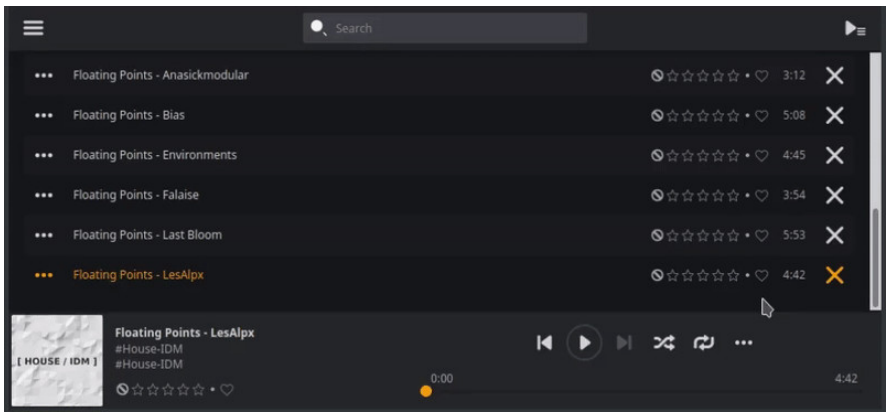


Figura 4.4: Lista de canciones de un álbum.

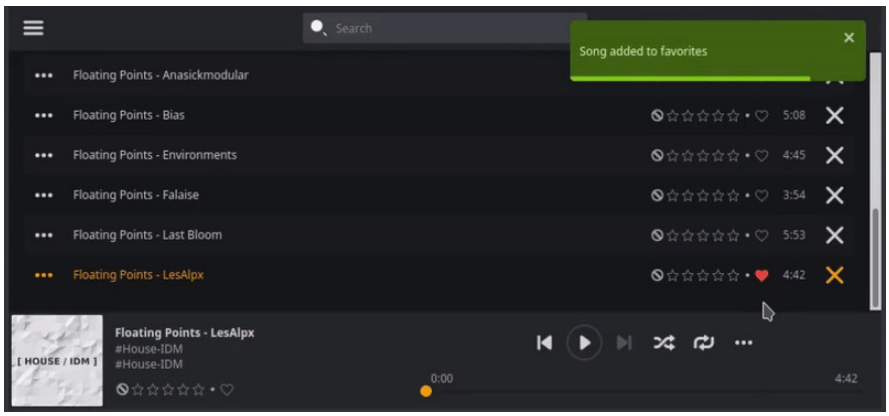


Figura 4.5: Canción marcada como favorita.

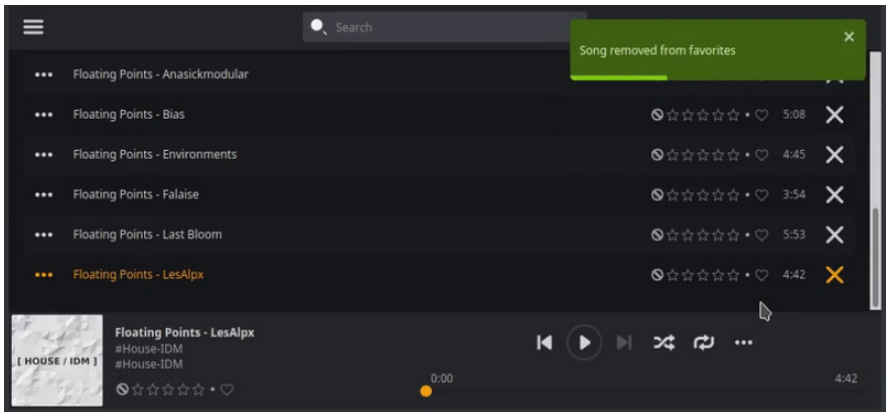


Figura 4.6: Canción desmarcada como favorita.

En cuanto a la base de datos, *a priori* la canción no está marcada como favorita, por lo que al realizar una *query* se obtiene un conjunto vacío (Figura 4.7).

```
mysql> SELECT id FROM `song` WHERE title = 'Floating Points - LesAlpx';
+------+
| id   |
+------+
| 9445 |
+------+
1 row in set (0.02 sec)

mysql> SELECT * FROM `user_flag` WHERE object_id = 9445;
Empty set (0.00 sec)
```

Figura 4.7: Query antes de marcar la canción como favorita.

Tras marcarla como favorita, al volver a realizar la *query* se obtiene la entrada correspondiente (Figura 4.8).

```
mysql> SELECT id FROM `song` WHERE title = 'Floating Points - LesAlpx';
+------+
| id   |
+------+
| 9445 |
+------+
1 row in set (0.03 sec)

mysql> SELECT * FROM `user_flag` WHERE object_id = 9445;
+-------+------+-----------+-------------+------------+
| id    | user | object_id | object_type | date       |
+-------+------+-----------+-------------+------------+
| 18846 |    1 |      9445 | song        | 1619182465 |
+-------+------+-----------+-------------+------------+
1 row in set (0.00 sec)
```

Figura 4.8: Query después de marcar la canción como favorita.

4.2. Integración de los aportes

La integración de los aportes realizados al repositorio del proyecto es un proceso elemental en una contribución a un proyecto *open source* (Figura 4.9).

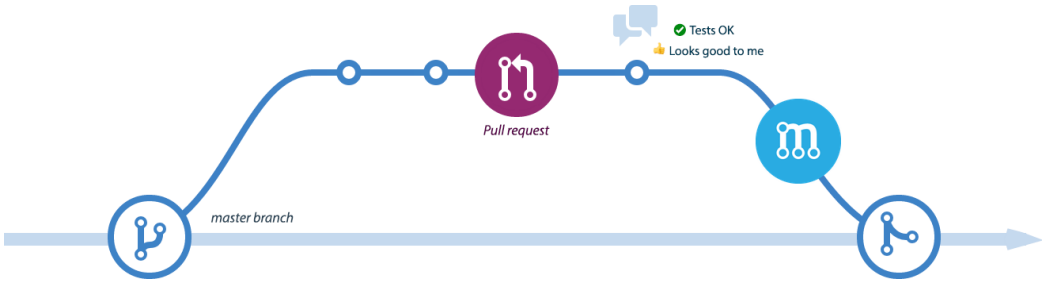


Figura 4.9: Proceso de integración de aportes.[7]

El proceso está compuesto de los siguientes elementos:

1. **Creación un *fork***: el primer paso consiste en clonar el repositorio del proyecto a un nuevo repositorio del que se será propietario y se tendrá permisos de edición.
2. **Creación una rama**: en este paso se crea una nueva rama de desarrollo y será en esta nueva rama donde se incluyan los cambios pertinentes al código existente.
3. **Desarrollo código**: tras realizar adiciones y cambios al código existente, estas modificaciones se introducen en *commits*, los cuales se sincronizan con la rama del repositorio que contiene el código.
4. **Solicitud de *pull request***: una vez terminado el desarrollo, se solicita la integración con el repositorio original mediante un *pull request*.
5. **Aprobación o denegación de la solicitud**: los cambios propuestos son revisados por la persona que mantiene del proyecto y será esta persona quien decida si los cambios son aptos o no.

[7]Publicado por el autor Julien Danjou. (`https://julien.danjou.info/content/images/size/w2000/2018/06/github-branching.png`)

4.2.1. #2527 [Feature Request] Total time

El primer *pull request* fue solicitado en noviembre de 2020. Está compuesto de 7 *commits* que afectan a 10 archivos del repositorio. En la solicitud se indicaron los cambios y pruebas realizadas (Figura 4.10).

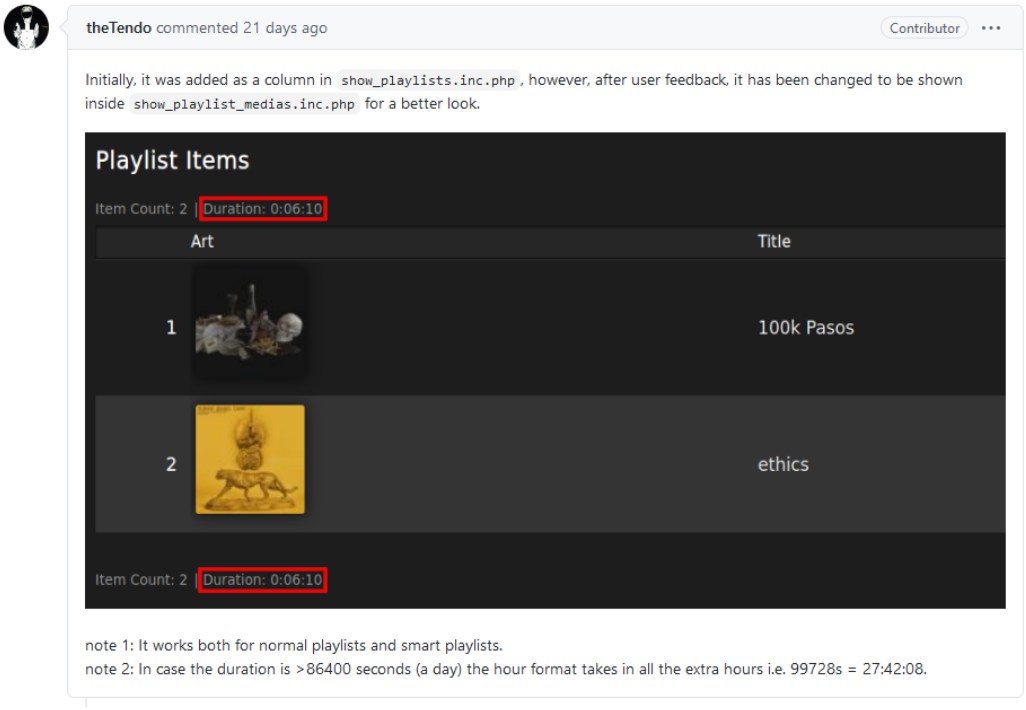


Figura 4.10: Comentario inicial del *pull request* de la *issue* 2527.

El desarrollador principal del proyecto realizó una pregunta[8] respecto a la usabilidad, la cual fue respondida. Tras realizar unos cambios menores al *pull request* propuesto, se aceptó su integración con la rama *development* del proyecto Ampache (Figura 4.11).

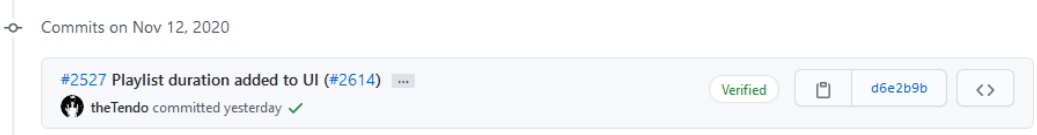


Figura 4.11: *Commit* en el repositorio de Ampache del *pull request* 2527.

[8] @lachlan: «For large lists are these hour minutes seconds still?»

4.2.2. #2582 [Feature Request] Changing login screen background

El *pull request* realizado está compuesto de 6 *commits* que modifican un total de 5 archivos.

Surgió un problema con las pruebas automatizadas realizadas por Travis CI (Figura 4.12) dado que el código propuesto inicialmente no cumplía la normativa de estilo.

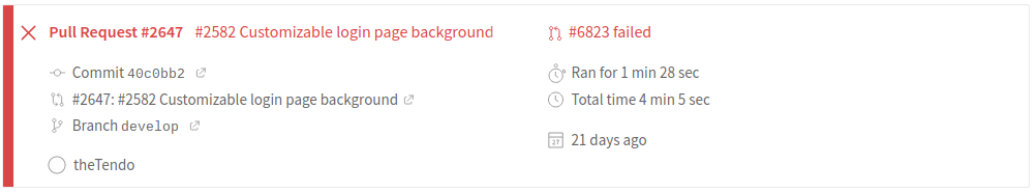


Figura 4.12: Fallo del *commit* `40c0bb2` en el test automático de Travis CI.

Después de realizar los cambios necesarios para adecuar el código al estilo establecido — se estaban utilizando 8 espacios para la indentación de una porción de código que debía contener 4 espacios — los cambios se aceptaron (Figura 4.13) e integraron en la rama *development* del repositorio.

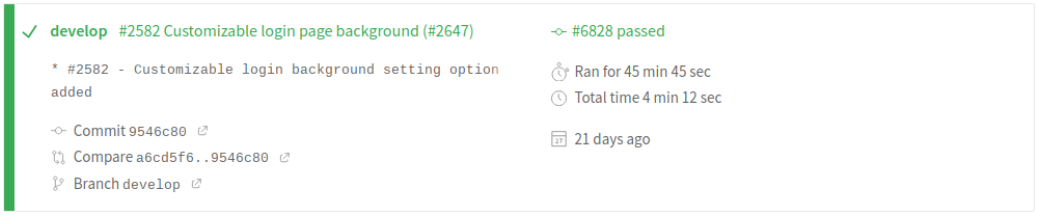


Figura 4.13: Éxito en el test automático de Travis CI tras realizar los cambios de estilo al código.

Tras integrar los cambios, el usuario que inicialmente abrió la *issue* indicó estar satisfecho con los cambios realizados[9].

[9] @agopo (usuario creador de la *issue*): «Hey, I honestly didn't expect this to be implemented so quickly (if at all). Thanks a lot for your work, I'm looking forward to it!»

4.2.3. Cliente React - Página álbumes

El desarrollo del cliente React es supervisado por un desarrollador distinto al mantenedor principal. Al no estar modificando código existente, la integración de está funcionalidad fue bastante sencilla.

Se abrió un *pull request* que añadía dos nuevos ficheros `index.tsx` y `index.styl` donde está definida la lógica y el estilo de la página de álbumes. Junto a ello, se introdujo la definición de la ya mencionada función `getAlbums` (Código 3.5) y la regla de enrutamiento en `router.tsx`.

El mismo día que fue creado el *pull request*, Ashot Nazaryan (responsable del cliente React), indicó que el desarrollo era adecuado[10] y fusionó los cambios con la rama oficial del cliente React.

4.2.4. Cliente React - Favoritos

La integración de esta funcionalidad ha sido la que mayor retroalimentación ha recibido por parte de los desarrolladores principales, con un total de 16 comentarios.

El desarrollador Ashot Nazaryan planteó 5 cuestiones respecto a distintos fragmentos del *pull request*. Preguntando el motivo de uso de una función *callback*[11]; proponiendo mejoras en el estilo del código[12] e indicando errores menores de la implementación[13,14].

Una vez resueltos todos los comentarios recibidos mediante un *commit*, se procedió a la integración de los mismos el 17 de febrero, cerrando así este último *pull request*.

[10] @AshotN (desarrollador principal): «Looks about right. I'll merge this and check it out later. Thank you!»

[11] @AshotN: «Is there any specific reason this is a useCallback?»

[12] @AshotN: «The else could be avoided by using a return»

[13] @AshotN: «You declared the interface but didn't apply it. This should be `const MusicControl: React.FC<MusicControlProps>= (props) =>{`»

[14] @AshotN: «I just realized, this shouldn't be optional.»

5. Conclusión

5.1. Diferencia entre estimación y tiempo real

A continuación, se muestra una comparativa entre la estimación de tiempo inicial realizada y la dedicación real desempeñada (Figura 5.1).

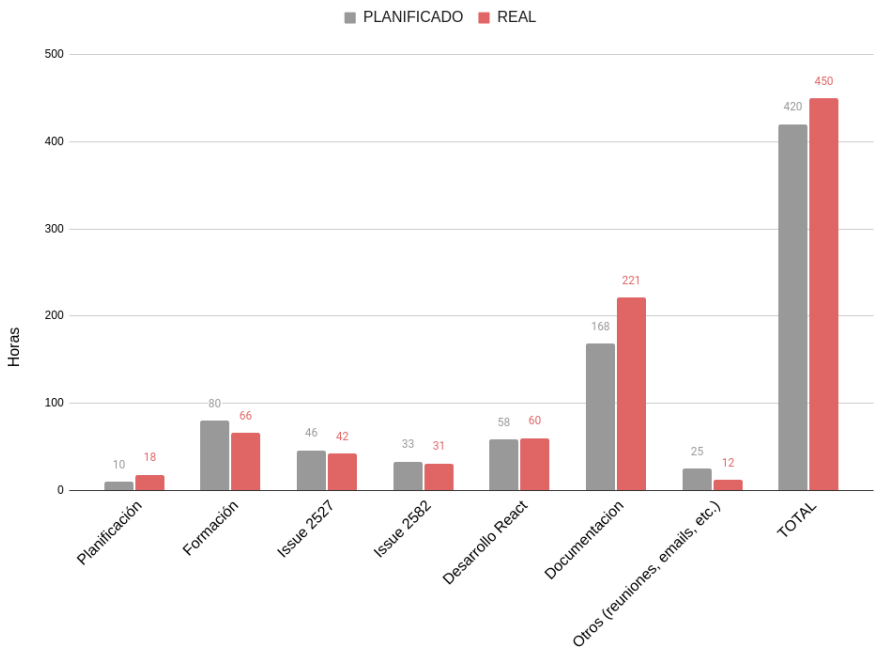


Figura 5.1: Tiempo planificado y tiempo real.

En términos generales, la planificación inicial muestra bastante semejanza al tiempo real destinado. Las 420 horas iniciales planificadas han resultado ser 450 horas en la realidad, lo que supone una desviación del 7 %.

La mayor diferencia absoluta se ha dado en el conjunto de tareas relacionadas a la documentación, sobrepasando 53 horas el tiempo estimado. Uno de los factores al que se debe este exceso, es el no haber considerado correctamente los tiempos de revisión y corrección relacionados a la memoria.

Se ha recreado el diagrama Gantt considerando las horas reales invertidas (Figura 5.2). La planificación estimada (Sección «2.2 Cronograma») se asemeja considerablemente a los plazos reales.

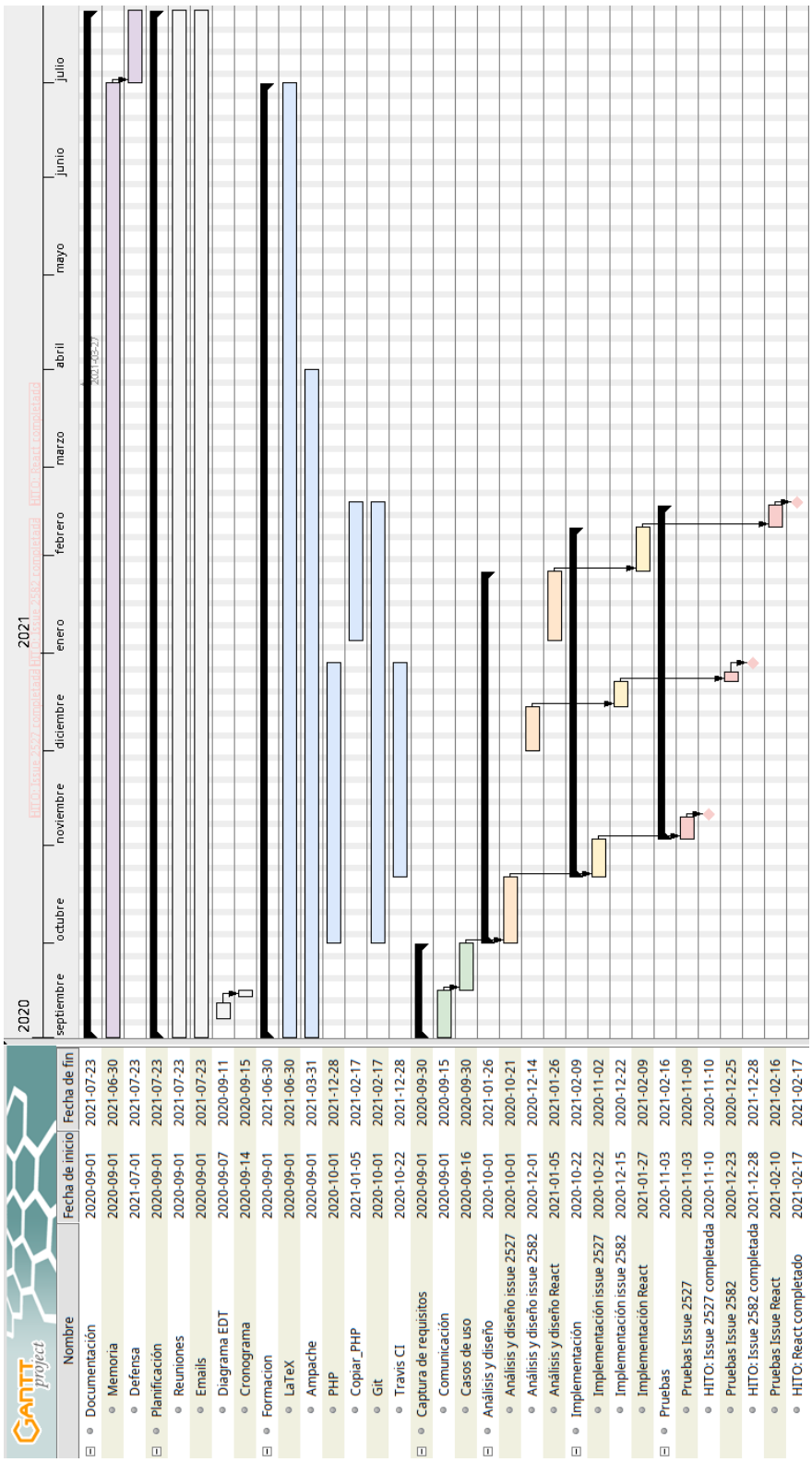


Figura 5.2: Diagrama Gantt del desglose de horas real del proyecto.

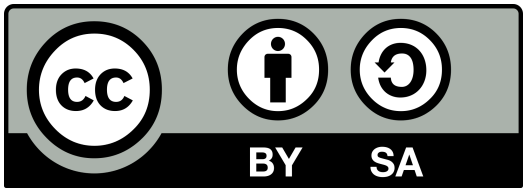
5.2. Futuras líneas de trabajo

Ampache es un proyecto activo con distintos frentes de desarrollo abiertos. Utilizando este trabajo como base, se puede expandir el desarrollo en la misma dirección resolviendo algunas de las más de 400 *issues* que están abiertas actualmente o continuando con la creación del cliente React.

También existe la posibilidad de expandir las funcionalidades de la API. Existen métodos pendientes de implementación, *bugs* por resolver y documentación que realizar.

Alternativamente, al tratarse de un proyecto de gran tamaño que tiene varías décadas, seguramente una refactorización a la base del código sería una buena opción de contribución al proyecto.

5.3. Licencia



El contenido de esta memoria tiene licencia Creative Commons Attribution 4.0 International[1].

Está permitido: (1) compartir, copiar y redistribuir el material en cualquier medio o formato; (2) adaptar, remezclar, transformar y construir a partir del material para cualquier propósito, incluso comercialmente.

Bajo los términos: (1) atribución — dar crédito de manera adecuada, ofrecer un enlace a la licencia, e indicar si se han realizado cambios; (2) compartir igual — cualquier remezcla, transformación o creación a partir del material se debe distribuir bajo la la misma licencia del original.

[1] Creative Commons: Attribution-ShareAlike 4.0 International (`www.creativecommons.org/licenses/by-sa/4.0/`)

5.4. Reflexión personal

Concluyo la memoria con una valoración subjetiva del trabajo realizado.

Seguramente este sea el trabajo que con mayor entusiasmo he realizado en toda mi trayectoria académica. Tener la oportunidad de contribuir a un proyecto de *software* libre dentro del contexto académico, y la libertad de elegir el proyecto deseado, ha hecho que más que tomarme el TFG como una obligación, lo haya tenido como un *hobby*.

La primera vez que revisé la basé de código de Ampache para comenzar el desarrollo me pareció algo abrumador, pero tenía claro que no iba a dar un paso atrás y cambiar de proyecto. Con el paso del tiempo me fui familiarizando con la estructura interna hasta tener mi primera aportación lista para presentar. Aún recuerdo la emoción del día en el que mi primer *pull request* fue aceptado.

En el proceso he aprendido sobre una considerable cantidad de herramientas: desde los flujos de integración de código de Git/GitHub, hasta los sistemas de integración continua como Travis CI junto a otras competencias transversales como la preparación de documentos en LaTeX o el hecho de aprender a navegar por estructuras de códigos ajenas.

He tenido la suerte de encontrar una comunidad activa de personas muy simpáticas y colaborativas. Además, también hemos tenido la oportunidad de hablar de temas relacionados con la música más allá del desarrollo de Ampache.

En definitiva, ha sido una experiencia muy positiva. Recomiendo sin duda alguna a futuros alumnos que encuentren un proyecto de su agrado y contribuyan en él.

Acrónimos

API

Application Programming Interface. 5, 46, 66–68, 77, 80, 98

BOE

Boletín Oficial del Estado. 31

CSS

Cascading Style Sheets. 46

EDT

Estructura de descomposición del trabajo. 12, 13, 27

ER

Entidad Relación. 19, 21, 22, 29

FOSS

Free and open-source software. 3

FSF

Free Software Foundation. 1

HTML

Hypertext Markup Language. 46

ISO

International Organization for Standardization. 36

JSON

JavaScript Object Notation. 66–68, 78

LSI

Lenguajes y Sistemas Informáticos. 2

MIT

Massachusetts Institute of Technology. 1

SGBD

Sistema de gestión de bases de datos. 56

SQL

Structured Query Language. 104

TFG

Trabajo de Fin de Grado. 25, 47

UML

Unified Modeling Language. 19, 30

URL

Uniform Resource Locator. 45, 104

XSS

Cross Site Scripting. 87

Glosario

***software* libre**

«Software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software»(Foundation [2019]). 3, 32

backend

Capa lógica de acceso a los datos en un *software*, generalmente se refiere al servidor de la aplicación. 46, 68

background

Imagen utilizada a modo de fondo de pantalla en una interfaz gráfica. 45

bug

Agujero, brecha o falta de seguridad de un programa de computación. 40, 44, 82, 98

codec

Programa que codifica contenido y permite comprimir información. 7

commit

Dentro del contexto de un sistema de control de versiones, se trata de la operación que envía los cambios realizados al repositorio. 72, 91–94

compilar

Proceso que convierte el código fuente en binario para su posterior ejecución por una computadora. 82, 83

código fuente

Versión del *software* tal y como fue escrita originalmente por un humano en texto plano. Es la base desde la que se compilan los binarios ejecutables que utilizan los ordenadores (Project [2006]). 1, 103, 105

DAAP

Acrónimo de «Digital Audio Access Protocol». Es un protocolo propietario de Apple para compartir archivos multimedia a través de una red local. 8

framework

Entorno de trabajo que provee funciones genéricas modificables con el objetivo de facilitar el desarrollo del *software*. 5, 7, 8, 22, 46

frontend

Capa de presentación de una aplicación *software*. 5, 20, 46, 68

hardware

Componentes físicos de un sistema informático, como la unidad central de procesamiento (CPU) y los dispositvos periféricos. 40

inyección SQL

Vulnerabilidad que consiste en introducir sentencias SQL en campos de texto o URLs para actuar maliciosamente sobre una base de datos. 87

issue

Así se denomina en GitHub a una resolución de un error o una propuesta de mejora. 14, 18, 19, 21–24, 27, 42, 44, 45, 47, 52, 58, 59, 70, 72, 77, 93, 98

kernel

Programa que constituye el núcleo central de un sistema operativo. Tiene control total sobre todo lo que ocurre en el sistema (Project [2005]). 1

log

Archivo que contiene un registro de eventos. 83

login

Método de autenticación en un sistema informático mediante credenciales asociadas a una cuenta. 42

metadatos

Información que describe el contenido, calidad, condiciones, historia, disponibilidad y otras características de los datos (secretaría de Gobierno Digital Peruana [2020]). 4, 8, 50, 51

open source

Se llama así a los programas informáticos en los que el código fuente está disponible para su acceso público. 5, 10, 91

overflow

Sucede cuando de una operación aritmética se obtiene un valor fuera del rango de valores representables. 84

P2P

Acrónimo de «Peer-to-peer». Es una arquitectura de red que distribuye el procesamiento entre los nodos que la forman. 106

playlist

Lista de canciones y piezas musicales. Traducción: lista de reproducción. 68

pop-up

Ventana emergente que se superpone al resto de elementos de la interfaz. 69

pull request

Proceso de integración que permite informar a otros usuarios sobre los cambios realizados en una rama de un repositorio. 83, 84, 91–94

píxel

Unidad básica de representación en una imagen digital. 85, 87

query

Consulta realizada sobre una base de datos. 62, 90

README

Documento que presenta una información general sobre un proyecto, sirve a modo de carta de presentación. 47

repositorio

Lugar donde se almacenan archivos los archivos de código. 15, 16, 82, 91–93, 103, 105

script

Lista de comandos para la automatización de tareas que se ejecutan en tiempo de ejecución. 55, 56, 62, 64, 87

software

«Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora»(RAE [2020]). 1, 2, 9, 10, 12–14, 19, 20, 29, 32, 40, 82, 103, 104

stakeholder

Individuo, grupo u organización que se ve afectado por el resultado de un proyecto. 12

string

Secuencia de caracteres. 75

UPnP

Acrónimo de «Universal Plug and Play». Es un conjunto de protocolos propietarios P2P diseñados para permitir la conectividad entre dispositivos de distintos proveedores. 8

WebDAV

Acrónimo de «Web Distributed Authoring and Versioning». Es un conjunto de extensiones del protocolo HTTP que permite a los usuarios editar y administrar archivos de forma colaborativa en servidores web (Whitehead [2010]). 8

Bibliografía

4phun. #2527 [feature request] total time. `www.github.com/ampache/ampache/issues/2527`, 2020.

agopo. #2582 [feature request] changing login screen background. `www.github.com/ampache/ampache/issues/2582`, 2020.

Oihane Albizuri. Contribuciones a un proyecto open source de ámbito internacional: Ganttproject. *Universidad del País Vasco*, 2019.

Ampache. Music streaming service. `www.ampache.org`, 2020.

A. Arif and Z. A. Rana. Refactoring of code to remove technical debt and reduce maintenance effort. *2020 14th International Conference on Open Source Systems and Technologies (ICOSST)*, 2020.

BOE. *Resolución de 7 de octubre de 2019, de la Dirección General de Trabajo, por la que se registra y publica el XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos.* Ministerio de Trabajo, Migraciones y Seguridad Social, 2019.

Eduardo Chillida. *Elogio del horizonte. Conversaciones con Eduardo Chillida.* Destino, 2003.

Free Software Foundation. Why the affero gpl. `www.gnu.org/licenses/why-affero-gpl.en.html`, 2015.

Free Software Foundation. ¿qué es el software libre? `www.gnu.org/philosophy/free-sw.es.html`, 2019.

GitHub Guides. Mastering issues. `https://guides.github.com/features/issues`, 2020.

ISO. *Gestión del riesgo — Directrices.* International Organization for Standardization, 2018.

Anaitz Jaio. Contribuciones a un proyecto open source de ámbito internacional: Ganttproject. *Universidad del País Vasco*, 2019.

Zhifang Liao, Benhong Zhao, Shengzong Liu, Haozhi Jin, Dayu He, Liu Yang, Jinsong Wu, and Yan Zhang. A prediction model of the project life-span in open source software ecosystem, 2017.

Hans Werner Meuer. Looking back over 15 years of supercomputing experience. *The TOP500 Project*, 2008.

Juanan Pereira. Leveraging final degree projects for open source software contributions. *Electronics*, 10(10), 2021. ISSN 2079-9292. URL `www.mdpi.com/2079-9292/10/10/1181`.

The Linux Information Project. Kernel definition. `www.linfo.org/kernel.html`, 2005.

The Linux Information Project. Source code definition. `www.linfo.org/source_code.html`, 2006.

RAE. *Diccionario de la lengua española*. RAE, 2020.

La secretaría de Gobierno Digital Peruana. ¿qué son los metadatos? `www.geoidep.gob.pe/conoce-las-ides/metadatos/que-son-los-metadatos`, 2020.

Nassim Taleb. *Antifragile: Things That Gain From Disorder*. Random House, 2012.

Eric von Hippel and Georg von Krogh. Open source software and the "private-collective" innovation model: Issues for organization science. *Massachusetts Institute of Technology*, 2009.

Jim Whitehead. Webdav resources. `www.webdav.org`, 2010.

YAML. The official yaml web site. `www.yaml.org`, 2020.