

Plan de gestión, análisis, diseño y memoria del proyecto
Proyecto software

Álvaro García Díaz, Óscar Baselga Lahoz, Alberto Calvo Rubió,
Saúl Flores Benavente, Luis García Garcés, Alejandro Facorro Loscos,
Germán Garcés Latre

Grupo 10. Karen Spärck Jones

Universidad de Zaragoza

15 de abril de 2020



Índice

| | |
|---|-----------|
| 1. Introducción | 3 |
| 2. Organización del proyecto | 4 |
| 3. Plan de gestión del proyecto | 5 |
| 3.1. Procesos | 5 |
| 3.1.1. Procesos de inicio del proyecto | 5 |
| 3.1.2. Procesos de ejecución y control del proyecto | 5 |
| 3.1.3. Procesos técnicos | 7 |
| 3.2. Planes | 9 |
| 3.2.1. Plan de gestión de configuraciones | 9 |
| 3.2.2. Plan de construcción y despliegue del software | 10 |
| 3.2.3. Plan de aseguramiento de la calidad | 11 |
| 3.2.4. Calendario del proyecto y división del trabajo | 14 |
| 4. Análisis y diseño del sistema | 17 |
| 4.1. Análisis de requisitos | 17 |
| 4.1.1. Sistema: Requisitos no funcionales | 17 |
| 4.1.2. Backend y Frontend: Requisitos funcionales | 17 |
| 4.1.3. Backend y Frontend: Requisitos no funcionales | 18 |
| 4.2. Diseño del sistema | 19 |
| 5. Memoria del proyecto | 36 |
| 5.1. Inicio del proyecto | 37 |
| 5.2. Ejecución y control del proyecto | 37 |
| 5.3. Cierre del proyecto | 38 |
| 6. Conclusiones | 40 |
| 7. Anexo I. Glosario | 40 |
| 8. Anexo II. Actas de todas las reuniones realizadas | 40 |
| 9. Anexo III..Otros anexos que se consideren necesarios | 40 |

1. Introducción

En este proyecto se va a desarrollar una aplicación que ofrece un servicio enfocado a la reproducción de música y podcasts en streaming.

Se trata de una aplicación que permite a los usuarios disfrutar de las obras de artistas, lo que hace que sea una forma rápida y sencilla de que los músicos que están empezando en el mundillo puedan abrirse un hueco en él. Los usuarios consumidores disfrutan sin límites del contenido que es publicado, consiguiendo así horas y horas de entretenimiento.

La aplicación permite a los usuarios crear sus listas de reproducción (privadas o públicas) y compartirlas con sus amigos, reproducir canciones o listas de otros usuarios, seguir a sus artistas favoritos, y muchas posibilidades más. También cuenta con un ámbito social, dado que cada usuario puede compartir contenido con amigos desde la propia aplicación o a través de otras redes sociales, como puede ser Whatsapp.

El sistema es accesible desde la aplicación Android o desde un navegador Web, como Google Chrome o Mozilla Firefox. Así, el usuario tiene la opción de conectarse desde distintos dispositivos sin importar sus sistema operativo. Existen detalles que consiguen mejorar la usabilidad del sistema, como por ejemplo mantener el estado de la última canción escuchada al terminar la sesión, pudiendo así continuar la próxima vez desde donde se dejó.

El proyecto va a contar con una serie de hitos. El primero consiste en la finalización de una primera versión del ‘Plan de gestión, análisis, diseño y memoria’ del proyecto, programado para el 14 de marzo. El siguiente hito trata de una segunda versión del plan anterior y de la primera propuesta de código, en la cual se tendrá implementado un reproductor de música capaz de comunicarse con la base de datos. Está programada para el 15 de abril. El último hito corresponde a la entrega de la memoria (actual documento) y de la aplicación final. Se encuentra programada para el 8 de junio.

Por último, se va a realizar un resumen del resto de los apartados de este documento. Primero se presentan a los miembros del equipo (punto 2) que van a desarrollar el proyecto; después, se va a explicar el llamado plan de gestión del proyecto (punto 3), dividido en dos partes: los procesos (punto 3.1), que se encargan de describir cómo se van a llevar a cabo las distintas tareas que hay que ir realizando a lo largo del proyecto, y los planes (punto 3.2), que establecen un objetivo y qué es lo que se necesita para conseguir alcanzarlo; el apartado que sigue consiste en el análisis y diseño del sistema (punto 4), en él se van a plantear los requisitos que el sistema deberá cumplir en su versión final y cómo han sido diseñados para su incorporación; seguidamente aparece la memoria del proyecto (punto 5), en la cual se realiza una descripción de cómo ha ido evolucionando el proyecto, incluyendo cambios respecto a la versión inicial, imprevistos surgidos, etc.; se termina con un apartado de conclusiones (punto 6) que engloba tanto personales como grupales.

2. Organización del proyecto

El equipo está formado por 7 alumnos de Ingeniería Informática de la Universidad de Zaragoza:

Backend:

- Saúl Flores Benavente: Servidor y API
- Alberto Calvo Rubió: Base de Datos, despliegue (Docker y Heroku) y apoyo al servidor

Frontend Android:

- Luis García Garcés: aplicación Android
- Alejandro Facorro Loscos: aplicación Android
- Óscar Baselga Lahoz: aplicación Android

Frontend Web:

- Álvaro García Díaz: aplicación Web
- Germán Garcés Latre: aplicación Web

Para gestionar el proyecto, se han establecido diferentes roles:

- Director de Proyecto: Alberto Calvo. Se encarga de liderar y coordinar a todos los miembros, velar por el progreso del proyecto, responsable de la documentación entregable (revisión y asignación de apartados), investigar y plantear estrategias y metodologías de trabajo para el proyecto. Tiene un visión global del proyecto y sirve de enlace con el cliente.
- Líderes de las diferentes partes del sistema: coordinan un pequeño equipo (reparto de tareas y control de horas), mantienen el repositorio específico, revisan su código y son encargados de crear, revisar y cerrar los issues de su equipo. Permite que en algunas reuniones sólo acudan los distintos líderes.
 - Líder Backend: Saúl Flores
 - Líder Frontend Android: Luis García
 - Líder Frontend Web: Álvaro García

Para el desarrollo, algunos conocimientos que van a ayudar a realizar este proyecto que los integrantes poseen son:

- Conocimientos sobre programación backend y frontend en el desarrollo de un sistema de información con Java
- Experiencia en el desarrollo de aplicaciones Android
- Conocimientos de bases de datos (Oracle, MySQL y PostgreSQL)

3. Plan de gestión del proyecto

3.1. Procesos

3.1.1. Procesos de inicio del proyecto

El proyecto se iniciará con la división en dos grupos, **Front-End y Back-End**, también se decidirá a qué serie de dispositivos irá nuestra aplicación y con qué **lenguaje de programación y frameworks** se trabajará, una vez decidido, se creará la organización de cada grupo y del proyecto mediante la votación de los diferentes **organizadores** y la asignación de cada miembro del proyecto a cada grupo. Por último, se crearán las distintas vías de **comunicación** entre los miembros del proyecto.

Respecto a la creación de **repositorios en Github**, cada organizador del proyecto creará el propio repositorio siguiendo las directrices del proyecto y dando permiso a cada uno de los integrantes de dicho grupo en concreto.

En cuanto a la **formación** de cada uno de los grupos, cada grupo sigue una lista de tutoriales para cada uno de los frameworks y lenguajes de programación elegidos.

- Flask [Episodios 1-5 y 8] [16]
- SQLAlchemy y API [17]
- Angular [18]
- TypeScript y HTML [19]
- Android Studio [20]
- Flutter [21]

Para la **formación de usuarios nuevos en la plataforma**, se tendría que comunicar con el Director del Proyecto para unirlo a los diferentes chat de comunicación del proyecto e indicar en cuál de los diferentes grupos se comenzaría a trabajar, en dicho grupo se le indicaría que tutoriales seguir, qué herramientas instalar y la asignación tanto de tareas como de permisos en diferentes plataformas.

Por último, se decanta por utilizar **Heroku** como servidor cloud propio de la aplicación, para ello se utiliza la versión gratuita, la cual nos brinda: 10.000 filas para la base de datos, un proceso para ejecutar, 512MB de RAM y 550h de límite de tiempo de los dynos(contenedores). Para la obtención de dicho servidor, solo se tuvo que realizar el registro y elegir la opción gratuita por defecto; la persona que tiene acceso a dicho servidor mediante la contraseña y nombre de usuario, es el propio Director del Proyecto y el líder de backend, también se tuvo que instalar el add-on Heroku Postgres en la aplicación creada en Heroku.

3.1.2. Procesos de ejecución y control del proyecto

Las comunicaciones internas se llevarán a cabo mediante distintos medios. El encargado de administrar y dar de alta al personal es el Director de Proyecto.

- **Grupo de Whatsapp:** conversaciones acerca de dudas, avisos y temas de menor importancia, evitando que las decisiones queden reflejadas aquí ya que se suelen perder con el tiempo.
- **Canal de Discord:** tras los sucesos ocurridos (cuarentena por COVID-19) se ha necesitado un medio por el que realizar reuniones mediante llamadas de voz. Una vez empezadas las reuniones en Discord, el Director de Proyecto guía la reunión para evitar que se alarguen y que se cumpla el objetivo. Todos los participantes deben guardar silencio e intervenir de forma ordenada para llevar a cabo la reunión de forma correcta. También se puede utilizar como medio de conversación informal y resolución de dudas cuando no se sea el momento de reunión.
- **Github issues:** cada líder de repositorio(web, android, backend y documentación) crea un issue por cada tarea en la que se pueda dividir el trabajo, intentando modularizar de forma que se independice lo máximo posible una tarea de otra. Se asignan a cada uno de los miembros de cada equipo según la valoración del líder basándose en los conocimientos del asignado, su dedicación de horas y se tendrá en cuenta dentro de lo posible el horario de cada uno. Se priorizará el equilibrio de horas dedicadas de cada uno. Quedará constancia del progreso mediante comentarios cada vez que se realice un progreso o se termine la tarea, en cuyo caso el líder correspondiente revisará la tarea/issue y determinará si cerrarlo o no.
- **Google Calendar:** herramienta utilizada para mostrar el calendario del proyecto como entregas y reuniones. Utilizando funciones de recordatorio mediante notificaciones y correos.

Además, se realizarán **actas de reuniones formales** (ya sean de grupo o con el profesor/cliente) en los que quedan reflejados: participantes, ausencias, duración, orden del día, acciones tomadas, objetivos para la siguiente reunión y fecha de la siguiente reunión. Se utiliza la plantilla estándar de Google Docs “Notas de la reunión - Escritor moderno”. Normalmente, en cada reunión hay dos **encargados de redactar**, Germán Garcés y Luis García.

Mediante estos elementos se espera un correcto funcionamiento de grupo de forma que todos estén notificados de todas las actividades y se mantenga una serie de **registros formales** en los que poder identificar problemas y sacar conclusiones, evitando los problemas que puedan surgir y corrigiéndolos lo antes posible.

Respecto a la **resolución de disputas** se solucionará mediante la conversación entre las personas que tienen la disputa. Si fuese necesario, se recurrirá al líder del correspondiente grupo (backend, android, web) y en todo caso, al Director de Proyecto. Si se debe tomar una decisión que involucra a todos los miembros del grupo se realizará una votación entre todos ellos. El ganador será el que obtenga mayoría de votos. En caso de empate, recaerá la decisión sobre el Director de Proyecto.

Las **medidas de progreso** a tener en cuenta son los **issues** cerrados respecto a los totales, los **requisitos cumplidos** frente al total y las **líneas de código**. Se utilizará una hoja de cálculo en la que anotar las métricas y se calculen datos y gráficas de forma automática (Disponible en Google Drive en la carpeta de Control de Proyecto). Estos progresos serán medidos semanalmente en la reunión semanal(normalmente sábado o domingo) por el director de proyecto y se revisarán para obtener una visión global del progreso. Para valorar cuantitativamente la dificultad de los issues y requisitos se les asigna un valor 1(Fácil), 2(Medio) y 3(Difícil) consiguiendo unos cálculos más próximos a la realidad. Los issues los valoran los líderes teniendo en cuenta tiempo aproximado, desconocimiento de la solución y experiencia propia. Se les añadirá una etiqueta con un determinado color correspondiente en Github Issues(*low difficult/fff000, medium difficult/ff7000 y hard*

difficult/ff0000). A los requisitos se les asigna la valoración en conjunto por todos los miembros. Para contar las líneas de código de todo el proyecto se han de descargar en una carpeta los tres repositorios y eliminar el entorno virtual de Flask(no queremos contar sus librerías). Se utiliza la herramienta cloc en la carpeta y se recoge el total de líneas de código mostrado por pantalla. Además, se guardará una copia de todo el desglose de análisis del código en un fichero y se guardará en Google Drive en la carpeta de Control de proyecto por si se necesita su revisión posteriormente.

En el caso de que se detecten problemas de rendimiento, avance insuficiente o desviaciones respecto al plan inicial, se hará un reajuste de tareas al personal para solucionar el problema de progreso de forma más ajustada, intentando aprovechar el máximo de horas posibles que cada miembro disponga. Incluso se puede mover personal de un grupo a otro para intentar ayudar en tareas que no sean específicas de la tecnología propia (ya que probablemente se desconozca), intentando ayudar en documentación u otras tareas. Se prioriza la entrega de todos los requisitos pactados, pero en caso de no poder llegar a la entrega final con todo desarrollado con buena calidad, se recortará en requisitos prevaleciendo la calidad de los demás.

3.1.3. Procesos técnicos

El despliegue del sistema se realizará en Heroku mediante contenedores. Se deberá ejecutar el docker-compose correspondiente que mediante Dockerfile creará las imágenes y resolverá las dependencias necesarias indicadas en los ficheros de cada servidor.

■ **Proceso de Configuración y Construcción:** en cada uno de los pasos siguientes se trabaja siempre desde la rama master de cada repositorio como se indica en el apartado 3.2.1. En la parte de **frontend web**, al trabajar con Angular, se requiere de utilizar el comando ”npm install.^al principio del proyecto para instalar las **dependencias** que se emplean indicadas en el fichero package.json. Este comando, crea una nueva carpeta, denominada ”node_modules.^{en} la que se descargarán todas las dependencias. En nuestro caso, ha hecho falta instalar **dos dependencias** extra para hacer uso de ficheros javascript (ecualizador y control de volumen) mediante ”npm install jquery^z ”npm install bootstrap”. A continuación, para lanzar la página web se usa ”ng serve TuneIT”, siendo TuneIT el nombre del proyecto. Así, podemos acceder a localhost:4200 y observar la interfaz web. Para la parte de **frontend móvil**, es necesario instalarse la versión 3.5.3 de Android Studio y 44.0.1 de Flutter, siendo este un plugin de Android Studio. También es necesario haberse instalado esta serie de bibliotecas:

- http: ^0.12.0 + 2
- rx dart: ^0.23.1
- flutter_exoplayer: ^0.5.5
- bloc_pattern: 2.1.0
- path_provider: ^1.1.0
- path_provider_macos: ^0.0.1 f
- firebase_messaging: ^6.0.13
- provider: ^3.0.0 + 1

Dichas **bibliotecas** están definidas en el fichero “pubspec.yaml” dentro del apartado “**dependencies**” de este mismo. Una vez definido el fichero, aparecerá un desplegable pidiendo si quiere que dichas dependencias se **actualicen**, una vez actualizadas, se habrá **instalado** la nueva biblioteca.

En pubspec.yaml no solo se indican las dependencias, esta aplicación utiliza imágenes por defecto:



Esta imagen es LogoApp.png, se encuentran en las carpetas de **assets** dentro de nuestro proyecto. Para indicarle a Android Studio que ha de poder utilizar esa imágenes en la aplicación ha de añadirse: assets/ - assets/ En el fichero pubspec.yaml.

Para backend es necesario instalar la versión de Python 3.7.5 y la herramienta pip 18.1. El proyecto de Flask dispone del fichero requirements.txt en el que se indican todas las dependencias. Estas se instalan en un entorno virtual de python venv (por actualizar posteriormente el fichero sin introducir otras personales) que no se subirá al repositorio. Estas se instalan mediante “pip -r requirements.txt”. En caso de instalarse una nueva se deberá actualizar el fichero mediante “pip freeze >requirements.txt” que escribirá todas las librerías y su versión del entorno venv. Además son necesarias las librerías “libcurl4-openssl-dev libssl-dev libpq-dev” instalables en debian y derivados con “apt install” como en producción (para desarrollo de forma local depende del OS del usuario, pudiendo cambiar los nombres de las librerías).

- **Proceso de Despliegue:** para desplegar la interfaz web, hay que ejecutar ”ng build –prod”, que generará una carpeta denominada ”dist” subir dicha carpeta. En este proyecto, hay un fichero JSON denominado ”package.json” que especifica todas las versiones de las distintas dependencias empleadas y del código, por lo tanto, ambos integrantes tienen las mismas versiones al tener todo el proyecto en un repositorio de GitHub común.

Para realizar el despliegue en cloud de las aplicaciones de Angular y Flask es necesario primero que se puedan desplegar localmente mediante el fichero **docker-compose** del repositorio de backend.

1. Construir la imagen (*docker-compose build*) y despliegue(*docker-compose up*). caso de no surgir errores de construcción se realizarán las pruebas automáticas descritas en los procesos de pruebas.

2. Tras pasar las pruebas satisfactoriamente se procede a desplegar los contenedores en Heroku mediante **Heroku CLI** (instalación ¹). Para cada contenedor:
 - a) Login de Heroku (*heroku login*). Credenciales personales no reflejadas aquí (Contactar con Director de Proyecto).
 - b) Logueo y actualización de contenedor (*heroku container:login*, *heroku container:push web*).
 - c) Liberar nueva versión(*heroku container:release web*).

Para utilizar la **aplicación** en el móvil se ha de instalar un **archivo .apk**. Este archivo se genera desde la terminal de Android Studio. Se ejecuta “*flutter build apk*” en el directorio de nuestro proyecto. Tras ejecutar esto se habrá creado un ejecutable .apk que se copiará en los dispositivos compatibles y se instalará.

■ **Proceso de Pruebas:** las pruebas se realizarán de forma específica para cada **servidor** y la **base de datos** mediante scripts/ficheros que ejecutarán las **pruebas automáticamente**. En algunos casos se realizarán pruebas de forma manual para casos puntuales.

A la hora de **comprobarlo localmente**, se podrá probarlo en cada uno de los **diferentes entornos de trabajo** respectivamente a los grupos de aplicación Web y Móvil. Para probar la aplicación de forma conjunta se utilizará tanto **los navegadores web y dispositivos móviles compatibles** según los requisitos no funcionales y dichos dispositivos se comunican con el servidor web que estará desplegado en **Heroku**.

Las pruebas en el **servidor** consisten en **tests unitarios** que se ejecutan con la base de datos vacía, ya que los datos necesarios son introducidos durante los tests y luego se borran. Para ejecutarlos se usan los comandos de python específicos para tests unitarios, en este caso se usará: *python -m unittest discover -s test*

Este comando buscará ficheros que importen la librería de tests unitarios de python, encontrando nuestro fichero de pruebas *test_server.py* y ejecutando su contenido. Tras la ejecución, se muestra en pantalla el número de tests ejecutados y el número de tests exitosos y fallidos. El objetivo es que todos los tests sean correctos, lo que corresponde a que todas las peticiones realizadas al servidor han devuelto el resultado esperado.

3.2. Planes

3.2.1. Plan de gestión de configuraciones

Para el código que se desarrolla en el proyecto se han creado **tres repositorios en GitHub**. Dos de estos para la parte de Frontend de la aplicación y otra para el Backend y lógica del servidor. Los administradores de dichos repositorios son los encargados de coordinar cada apartado. En el caso del Frontend para móvil el administrador será Luis García, para frontend web Álvaro García y para administrar el repositorio de Backend será Saúl Flores. Para la documentación también se ha creado un repositorio de GitHub y utilizaremos las incidencias para llevar un control del proceso de documentación. El administrador de este repositorio es el coordinador general del proyecto, Alberto Calvo. Los administradores de dichos repositorios serán los encargados de gestionar las incidencias

¹<https://devcenter.heroku.com/articles/heroku-cli>

y llevar el control del desarrollo. Esto se hará semanalmente , una vez a la semana los coordinadores mirarán sus respectivos repositorios y comprobarán el avance o la actualización de las incidencias.

Los ficheros correspondientes a Frontend, tanto web como móvil, y Backend, código del servidor y la base de datos, serán nombrados según el **estándar SnakeCase**. La documentación que acompaña al proyecto también seguirá este estándar. Además, para la escritura de código se seguirá el estándar recomendado por **Google** en el caso de **Android** [12] y Backend, y el recomendado por la **W3Schools** para **Frontend** [13].

Se va a utilizar la gestión de incidencias de GitHub para organizar y llevar un seguimiento del trabajo. **Semanalmente** los coordinadores repartirán el trabajo, indicando qué ha de hacer cada miembro esa semana, todos los **domingos**. Cuando se haya completado una incidencia, la persona que la haya terminado adjuntará un comentario a la incidencia, indicando que la ha terminado, se pondrá en contacto con el coordinador y este se encargará de valorar si se ha terminado y la cerrará.

En lo relacionado al código, las incidencias serán añadidas al repositorio por los administradores y se corresponderá al trabajo relacionado con el proyecto que se desarrollará esa semana, esto se realizará todos los domingos. Las incidencias estarán en su estado inicial antes de comenzar el trabajo, este estado inicial es “To Do”. Cuando un miembro del grupo comience dicha incidencia, este mismo actualizará su estado a “In progress” . Una vez la haya terminado dicho trabajo, el coordinador pondrá dicha incidencia como finalizada, pasando esta incidencia al estado final “Done”, en el caso de cumplir los requisitos establecidos.

Los requisitos que ha de cumplir una incidencia para darse por finalizada se acordarán previamente entre el coordinador y los miembros de grupo. Estos serán requisitos de funcionalidad y estética, el código entregado ha de compilar y no interrumpir el desarrollo que el resto de compañeros están llevando a cabo. **Cada issue tendrá sus propios requisitos**, estos se anotarán en la descripción del issue una vez haya sido creada por parte del coordinador.

En lo relacionado a la **documentación** el control de issues será similar, aunque en este caso sí que será necesaria una **revisión del coordinador**. Lo cual implica que las incidencias tengan un estado anterior al estado “Done”, este estado será “Revisión pendiente”. Para que una de estas incidencias pase al estado de “Done” ha de ser aprobada por el coordinador del proyecto. El director de proyecto es el encargado de actualizar los documentos del repositorio una vez estén terminados, ya que su modificación se lleva a cabo en Google Drive en un doc de Google.

Vamos a utilizar un **flujo de trabajo centralizado** para los cuatro repositorios de GitHub que empleamos en este proyecto. Trabajaremos sobre una sola rama, la **rama master** y todos los cambios se confirman en dicha rama. Se ha llevado a cabo esta decisión ya que los repositorios de GitHub cuentan con un máximo de 3 integrantes y se parten las tareas de forma que el número de conflictos se minimice.

3.2.2. Plan de construcción y despliegue del software

El software se construye y se prueba mediante los procesos indicados en el 3.1.3. En la etapa temprana del proyecto se planea hacerla de **forma unitaria** de cada repositorio sin interactuar con las demás partes. Una vez se han alcanzado las etapas intermedia y final del proyecto se espera poder construir y desplegar localmente de forma conjunta mediante varios contenedores. De esta forma, cada desarrollador puede desplegar mediante **docker-compose** su versión que esté modificando y **simultáneamente** la última versión estable de las otras partes incluyendo su propia base de datos.

Con este método se asemeja más al entorno de producción en el que se despliega con contenedores, **Heroku**.

En Heroku se dispone de una cuenta gratuita creada por el Director de proyecto con un límite en cada contenedor de 500 MB de RAM y 1 worker. Se disponen de 450h de funcionamiento, contando que cada vez que se despliega, la aplicación sigue activa durante 30 min hasta que se suspende. Además, se dispone del addon Heroku Postgres en la versión “hobby” que permite almacenar 10000 filas y realiza mantenimientos eventuales no programables en los que se desactiva temporalmente (5 min aproximadamente).

Para almacenar las canciones se dispone de una cuenta de estudiante de **AWS S3** durante 12 meses con 5GB de almacenamiento, 20000 solicitudes GET y 2000 solicitudes POST. En ella se almacenarán las canciones mediante **buckets**.

Los despliegues e integración de todo el software están **programados semanalmente**. Se deberá haber probado localmente antes con la configuración de los contenedores establecidos. En cada reunión semanal se realizará una presentación de las nuevas funcionalidades desplegadas. Se valorará si la aplicación en ese momento es funcional respecto a los demás repositorios (para poder desarrollar de forma estable) y en caso positivo será la que se mantendrá desplegada también en Heroku para realizar pruebas. De esta forma, se plantea una especie de integración continua un poco más relajada sin realizar entregas semanales.

3.2.3. Plan de aseguramiento de la calidad

Para atraer usuarios y tener un **diseño común** entre las dos interfaces (web y móvil) se han consultado varias guías de estilo y/o pautas para desarrolladores de diferentes aplicaciones relacionadas con la música, tales como Spotify, Shazam o Deezer. Con el fin de conseguir usuarios, se han seguido las guías de estilo para facilitar a que el cambio no sea tan brusco respecto a una aplicación de música habitual y sea más fácil identificar dónde se encuentra cada función disponible.

Una de las guías de estilo más empleada es la de **Spotify** [1] debido a que se trata de la mayor aplicación de reproducción de música y es la que posee una mayor base de usuarios. En este caso, se han seguido tanto la guía de estilo y algunas **pautas para desarrolladores** [2]. Al seguir sus recomendaciones, se han llegado a las siguientes conclusiones junto a las pantallas afectadas:

- Si se muestra la imagen de un álbum, la imagen ha de ser cuadrada (pantallas de álbum, artista, listas de reproducción y todas aquellas que muestren como mínimo una canción).
- En la barra superior, en la parte izquierda se mostrará el logo de la aplicación seguido de su nombre (presente en todas las pantallas).
- El logo ha de encontrarse en un fondo blanco (*FFFFFF*), negro (*000000*) o que no esté en solo dos tonos diferentes (presente en todas las pantallas, destacando la de iniciar sesión y registro).
- El logo sólo puede encontrarse de tres maneras: el logo con el nombre de la aplicación a la derecha (barra superior), con el nombre de la aplicación abajo (inicio sesión y registro) o solo el logo (imagen de la aplicación móvil o ícono de la pestaña en interfaz web).
- Los colores predominantes serán el blanco (*FFFFFF*), negro (*000000*), los colores del logo (azul (214B8B), morado (833A8E)) y una escala de grises (presente en todas las pantallas).



Otra guia de estilo consultada ha sido la de **Shazam** [3], con la que se han obtenido nuevas conclusiones:

- La **tipografía** será ligeramente mayor para el nombre de la canción y menor para el nombre del artista cuando se muestren en forma vertical, es decir, el nombre de la canción y debajo el nombre del artista (barra inferior y pantallas de canción y álbum). También, será mayor cuando se muestre el título de una lista de reproducción en comparación con las canciones de esa lista.

- La tipografía predominante será Moderne Sans [10] y podrá presentarse en negrita (pantalla de canción en el nombre de la canción).

Para consultar varios de los **bocetos** de las pantallas, en el punto 4.2 se muestran antes de seguir las decisiones tomadas en este punto, por lo que las pantallas se verán modificadas en el resultado final, aunque la estética será parecida.

En cuanto a la **usabilidad**, todas las páginas principales de la aplicación se pueden acceder desde el menú de la izquierda o desde la barra superior, reduciendo la búsqueda del usuario al querer acceder a una página en concreto al encontrarse todo en dos sitios. Cuando no se ha iniciado sesión, en la pantalla información en el caso de la interfaz web, los botones que redirigen a las páginas de inicio de sesión y registro se encuentran en el centro de la pantalla para que sea fácil identificarlas y cueste menor tiempo encontrarlas. En el caso de la interfaz móvil, se accede directamente a la pantalla de iniciar sesión, a través de la cual se puede acceder a la página registro.

Ambas interfaces (web y móvil) presentan una **estética común**, permitiendo que el cambio de interfaz no sea tan brusco. Esta estética común se compone de una barra superior con páginas y el logo con el nombre de la aplicación, a la izquierda un menú desplegable con las páginas principales y una barra inferior con la reproducción actual y permitiendo gestionar el audio, como parar la canción o pasar a la siguiente. En la barra superior se puede encontrar la página “Ayuda”, la cual contiene una serie de preguntas frecuentes que pueden ayudar al usuario a resolver la duda o problema surgido. Esta estética no se encuentra presente en las pantallas que se muestran cuando no se ha iniciado sesión, presentando solamente la barra superior.

En cada parte del proyecto, se harán una serie de **tests automáticos**, que pueden tratarse de acciones realizadas por un usuario o probando datos que podrían ser inválidos (introducir en la pantalla de registro valores que sobrepasen el límite o valores ya repetidos en la base de datos) y de pruebas manuales, que consistirán en la prueba de cada funcionalidad de forma más detallada por parte de los encargados de esa parte y explicándoles al resto del grupo qué es lo que han hecho, cómo funciona y si van a implementar alguna mejora. Si al grupo le ha convencido lo propuesto y no ha surgido alguna duda o mejora, se considera como finalizada. En la parte de backend tienen una serie de pruebas para comprobar que las peticiones producen los resultados esperados. Todas las pruebas siguen el mismo formato, se añade un elemento a la base de datos, se ejecuta la petición a probar, se comprueba el resultado y se elimina ese elemento para restaurar la base de datos. En cuanto a frontend, se comprueba que cada funcionalidad realiza lo esperado, como es la reproducción en bucle o aleatoria. Para comprobar las diferentes funcionalidades relacionadas con el reproductor, se precisa, como mínimo, de una canción. Como en backend, las pruebas siguen un formato, se prueba la funcionalidad, se comprueba y se prueba con un diferente número de canciones.

Cuando un miembro no líder da por **terminada una funcionalidad**, el líder de la parte correspondiente, se asegura de que funciona correctamente e informa de que dicha función ha sido completada o que falta alguna parte para finalizar. Para considerarla como finalizada, se emplean las **issues** de GitHub en el que se añade una descripción para enfatizar las acciones que hay que realizar para completarla y cerrarla. Aunque se haya aceptado, quedará pendiente la demostración al resto del grupo explicada en el apartado anterior.

Un código limpio y ordenado es más fácil de comprender y de encontrar fallos, por lo tanto para cada parte se empleará un **analizador estático** y **formateadores de código**, asegurándose de tener un código con la menor cantidad de errores o avisos. En la parte de Frontend web se va a emplear Prettier [7] como formateador de código y codelyzer [8] como analizador estático, en Frontend

Android se ha escogido dartanalyzer [9] como analizador estático y en cuanto al formateador de código se usa el comando “dartfmt” y en el backend el propio IDE PyCharm reorganiza el código y como analizador estático se emplea el plugin Pylint [15].

Para llevar la cuenta de la cantidad de líneas de cada tipo de lenguaje, teniendo en cuenta líneas en blanco y comentarios, se emplea la herramienta “cloc” [11] con la carpeta que contiene los ficheros como parámetro.

Otra parte importante del proyecto es la **documentación**, por lo que hay que tener en cuenta el diseño de esta memoria. Para ello, cuando cada miembro ha finalizado sus partes asignadas, el líder del proyecto revisa cada apartado, saca defectos y se los comunica a las personas implicadas. Cuando no encuentre ningún defecto, se pasa toda la memoria a LaTeX para conseguir un diseño más formal.

3.2.4. Calendario del proyecto y división del trabajo

Para la organización del trabajo del equipo, se ha realizado un **diagrama de Gantt**. Este diagrama enumera las **issues** publicadas en GitHub para cada una de las partes del proyecto: frontend web, frontend móvil y backend.

Se ha establecido que para la primera entrega, deben estar realizados todos los requisitos relacionados con un reproductor de música o podcasts y sus equivalentes, por ejemplo, el crear listas de reproducción, el poder buscar podcasts... Mientras que para la segunda entrega se ha decidido que se completarán los requisitos restantes, es decir, aquellos requisitos relacionados con los usuarios y las interacciones entre ellas.

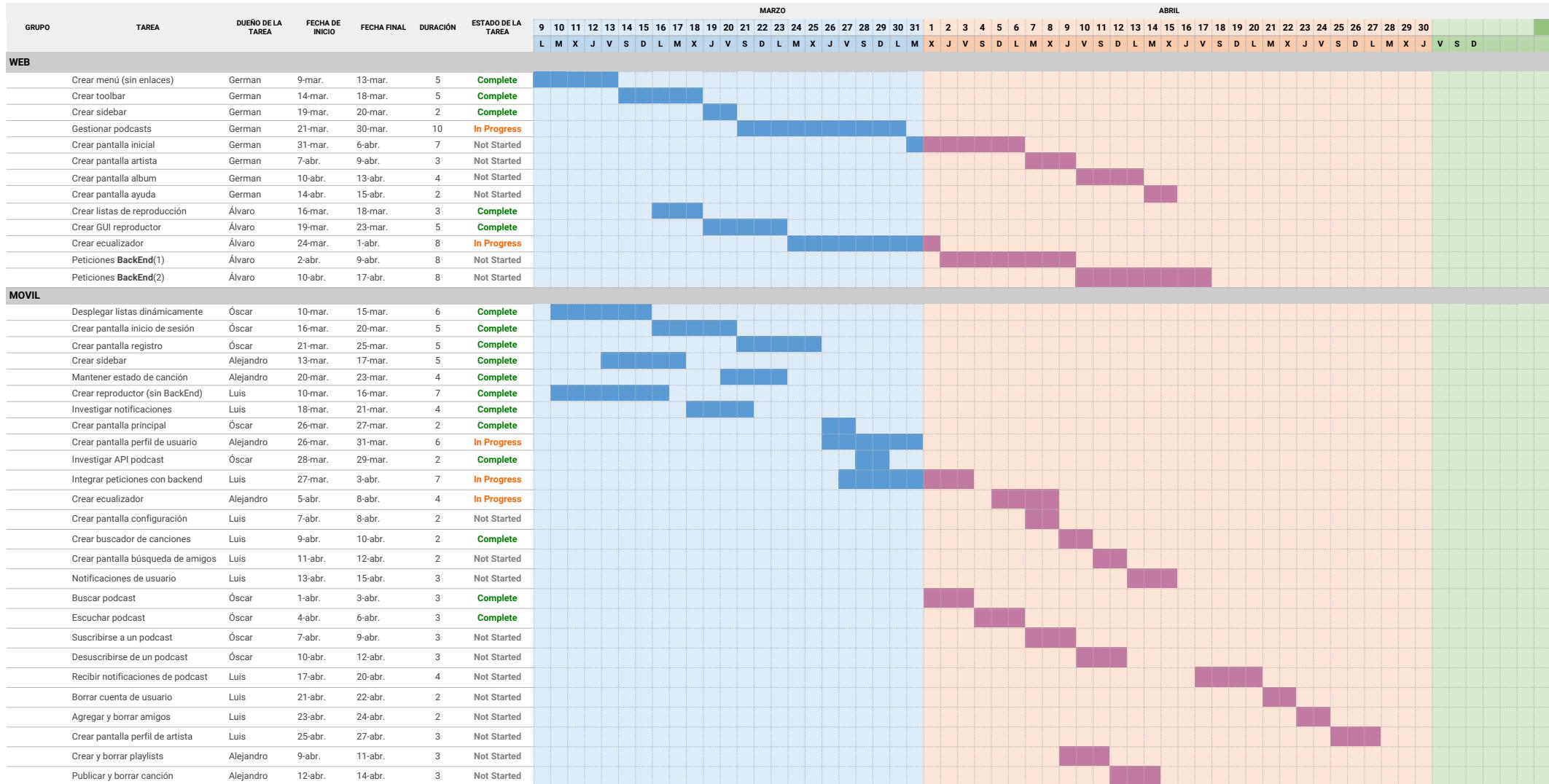
En resumen, para la **primera entrega** se tiene previsto tener completados los siguientes requisitos: Del RNF1 al RNF4 y del RNF8 al RNF 10. Del RF1 al RF5, RF13 y del RF32 AL RF34.

Y para la **segunda entrega** se realizarán los requisitos restantes: Del RF6 al RF12 y del RF14 AL RF31.

DIAGRAMA DE GANTT

| | |
|----------------------------|---------------|
| TITULO DEL PROYECTO | Tunelt |
| PROJECT MANAGER | Alberto Calvo |

| | |
|------------------------|--------------------|
| NOMBRE COMPAÑÍA | Karen Sparck Jones |
| FECHA | 10/2/20 |



4. Análisis y diseño del sistema

4.1. Análisis de requisitos

4.1.1. Sistema: Requisitos no funcionales

1. El sistema tendrá una versión para Android y otra para Web.
2. El sistema soportará la versión de Android 6.0 y de la Web los navegadores Chrome (71.0.3578.80) y Firefox (75.0).
3. El sistema se conectará a un servidor para obtener las canciones/podcasts.
4. El sistema necesita conexión a Internet.

4.1.2. Backend y Frontend: Requisitos funcionales

1. Se permite reproducir canciones y podcasts.
2. Se permite la reproducción en bucle, aleatoria, empezar, pausar y avanzar a la anterior o siguiente canción/podcast.
3. Se permite el uso de un ecualizador para cambiar las ganancias en diferentes frecuencias.
4. Una canción/podcast se compone de un audio, un título, un artista, un álbum y una categoría.
5. Una lista de reproducción está compuesta por una serie de canciones en un orden específico, creadas por un usuario o por el sistema, que contiene un título y una descripción.
6. Los usuarios pueden crear o eliminar listas de reproducción.
7. Los usuarios pueden añadir o eliminar una canción de una lista de reproducción propia previamente creada.
8. Los usuarios pueden ordenar las canciones presentes en una lista de reproducción propia.
9. Los usuarios poseen una lista de canciones/podcasts favoritas.
10. Los usuarios pueden añadir o eliminar canciones/podcasts de favoritos.
11. Los usuarios pueden ordenar las canciones/podcasts presentes en favoritos.
12. Los usuarios pueden buscar canciones/podcasts.
13. Los podcasts se agrupan en series de distintos episodios.
14. Un usuario está conformado por un nombre de usuario (*nick*), un correo electrónico único (es decir, no puede haber dos usuarios con el mismo correo), una contraseña para acceder, una fecha de nacimiento y su país.
15. Los usuarios se pueden registrar en el sistema.

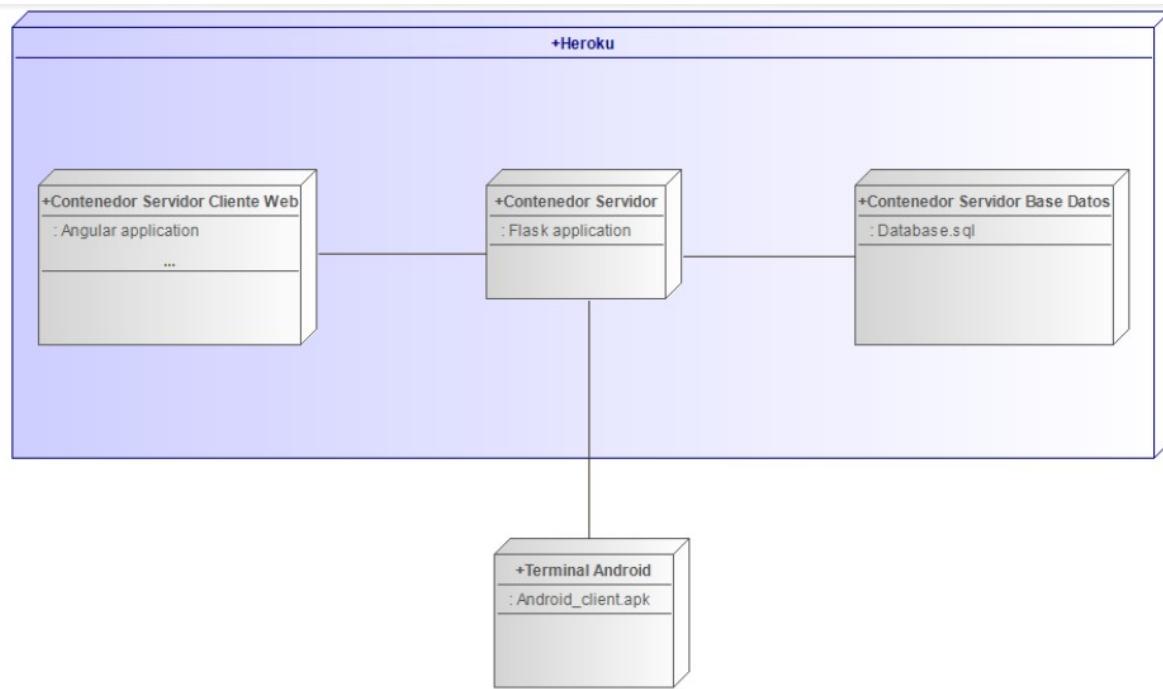
16. Los usuarios registrados pueden iniciar sesión.
17. Los usuarios pueden buscar a otros usuarios.
18. Los usuarios pueden agregar a otros usuarios como amigos.
19. El sistema permite consultar los usuarios amigos.
20. El sistema permite eliminar a otros usuarios amigos.
21. El sistema permite consultar las peticiones de amistad recibidas.
22. Los usuarios pueden compartir canciones/podcasts o listas de reproducción con otro usuario amigo.
23. Los usuarios pueden agregar a sus listas de reproducción otras listas creadas por otros usuarios.
24. Los usuarios pueden modificar su nombre de usuario, su correo electrónico y su contraseña.
25. Los usuarios pueden eliminar su cuenta.
26. Los usuarios se pueden suscribir a los artistas.
27. Se guarda el tiempo en el que se ha pausado o cerrado la aplicación de la última canción/podcast.
28. Se guarda la última lista reproducida.
29. La reproducción de canciones/podcast se sincronizará entre dispositivos de un mismo usuario.
30. Se recupera el tiempo de la última canción/podcast escuchado cuando se abre el sistema.
31. Se recupera la última lista de reproducción cuando se abre el sistema.
32. El sistema permite buscar información externa asociada a una canción/podcast mediante Google.
33. La aplicación permite listar canciones/podcasts.
34. La aplicación mostrará el estado de la canción/podcast.

4.1.3. Backend y Frontend: Requisitos no funcionales

5. El *nick* del usuario debe tener entre 3 y 50 caracteres.
6. Las contraseñas del usuario se encuentran cifradas.
7. Los usuarios inician sesión mediante el correo electrónico y la contraseña.
8. Se muestran las últimas canciones añadidas al sistema.
9. El reproductor en la parte inferior solo aparece cuando se está reproduciendo una canción o un podcast.
10. La búsqueda de canciones/podcasts se realiza mediante palabras clave con un mínimo de una palabra y una longitud mayor o igual a 3 caracteres.

4.2. Diseño del sistema

Diagrama de despliegue:



El protocolo empleado para la comunicación entre Backend y Frontend será HTTP, usando los métodos estándar de HTTP.

Las tecnologías elegidas son variadas. Para el Backend se han elegido **Flask**, **SQLAlchemy** y **PostgreSQL**. Flask ha sido elegido ya que se trata de un framework en python que permite el desarrollo de proyectos de forma sencilla, rápida y en muy pocas líneas de código. **SQLAlchemy** ha sido elegido ya que permite interactuar con la base de datos de forma mucho más simple. Evita utilizar lenguaje SQL y ayuda a pasar las relaciones y clases de la base de datos a Python, de esta forma se simplifica la interacción con **Flask**. Para el desarrollo del Frontend web se va a utilizar el framework **Angular**, desarrollado en TypeScript, de código abierto y mantenido por Google. Los lenguajes con los que trabajaremos en Angular son principalmente TypeScript junto a **HTML** y **SCSS**.

Finalmente para desarrollar el Frontend móvil utilizaremos **Flutter**. Es un SDK código fuente abierto de desarrollo de aplicaciones móviles creado por Google que emplea el lenguaje Dart. Haremos uso de una API web externa con la que se conectarán nuestro sistema. Utilizaremos la API que ofrece **Listen Notes** [6], esta API nos permite realizar hasta 5000 consultas de forma totalmente gratuita, de esta forma tendremos acceso a los últimos podcasts que se hayan publicado.

El resultado final será una **aplicación web** y una **aplicación móvil (Android)**, para web se ha decidido usar tecnologías web para facilitar todo el proceso de implementación de los requisitos

funcionales. Para la web se ha decidido usar **Angular**, un framework para aplicaciones web desarrollado en **TypeScript** mientras que para móvil se ha usado **Flutter**, un SDK para la generación de código nativo.

En cuanto a la base de datos, se va a utilizar una **base de datos Postgresql**, de tipo relacional, ya que no se va a trabajar con grandes cantidades de datos, ni se tiene que escalar horizontalmente ni se necesitan todas las otras ventajas que ofrece una base de datos NoSQL frente a las SQL.

A la hora de almacenar la información, la mayoría de los datos se almacenarán en la base de datos, mientras que las canciones, que son los elementos más pesados, se almacenarán en un servicio cloud de almacenamiento de **Amazon S3**, guardando en la base de datos únicamente la ruta necesaria para poder alcanzarlas, que es además lo que se necesita para poder realizar el streaming.

La **API** no va a ser de tipo **RESTful**, es suficientemente sencilla como para que no sea necesario, aunque sí contará con algunas de sus características, como ser cliente-servidor y no guardar estado, sino que las peticiones contengan la información necesaria para llevarlas a cabo.

La seguridad de la información de los usuarios que se va a almacenar se asegurará cifrando las contraseñas de los mismos, de forma que nunca se tratan en la base de datos las contraseñas planas sino que se trabajará con los hashes.

El despliegue de la aplicación se realizará en **heroku** [14], donde se alojarán 2 contenedores, uno donde se instala el servidor flask y otro para el servidor que se usará de host para el cliente web en angular. La base de datos Postgresql se despliega mediante el addon Heroku Postgres.

Se accede a la aplicación mediante los navegadores web instalados en los equipos a través de la url que Heroku proporcionará al servidor web. Para la aplicación Android, se simulará el cliente en los equipos de los desarrolladores mediante las herramientas que ofrece Android studio y una vez haya madurado lo suficiente, se instale en algún dispositivo de forma que no provoque errores graves.

Diagrama de clases del Cliente:

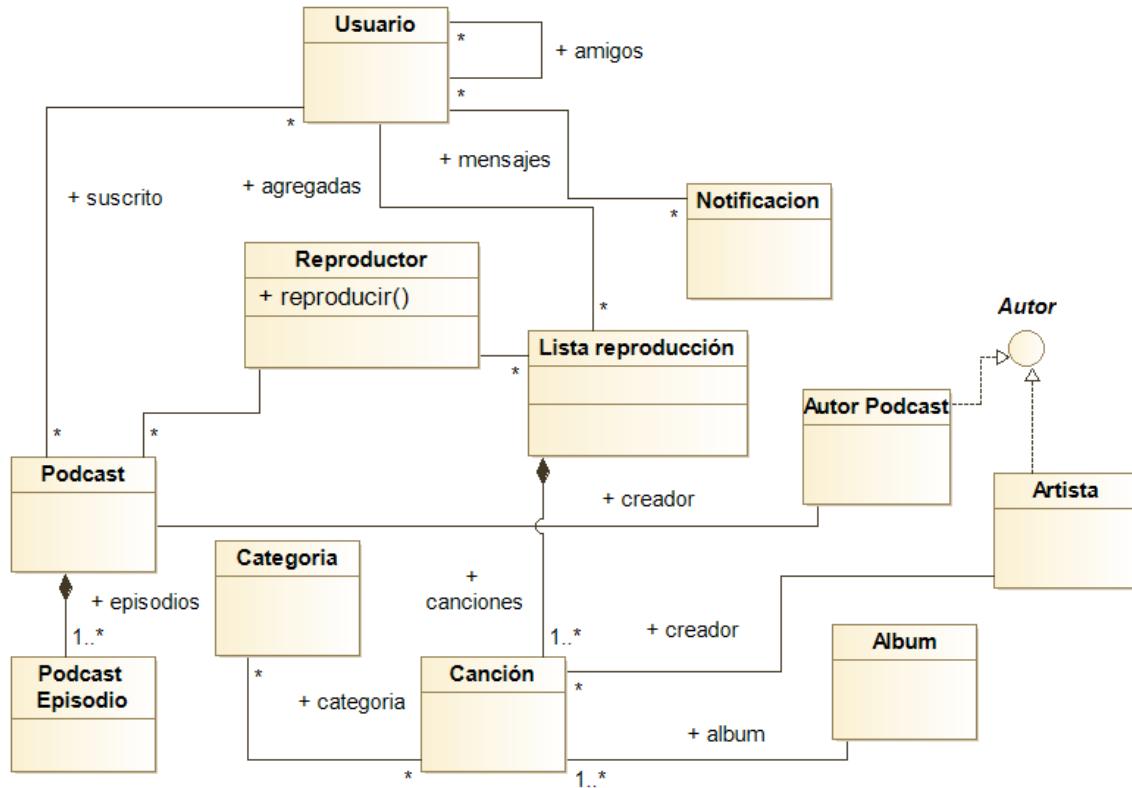


Diagrama de clases del Servidor:

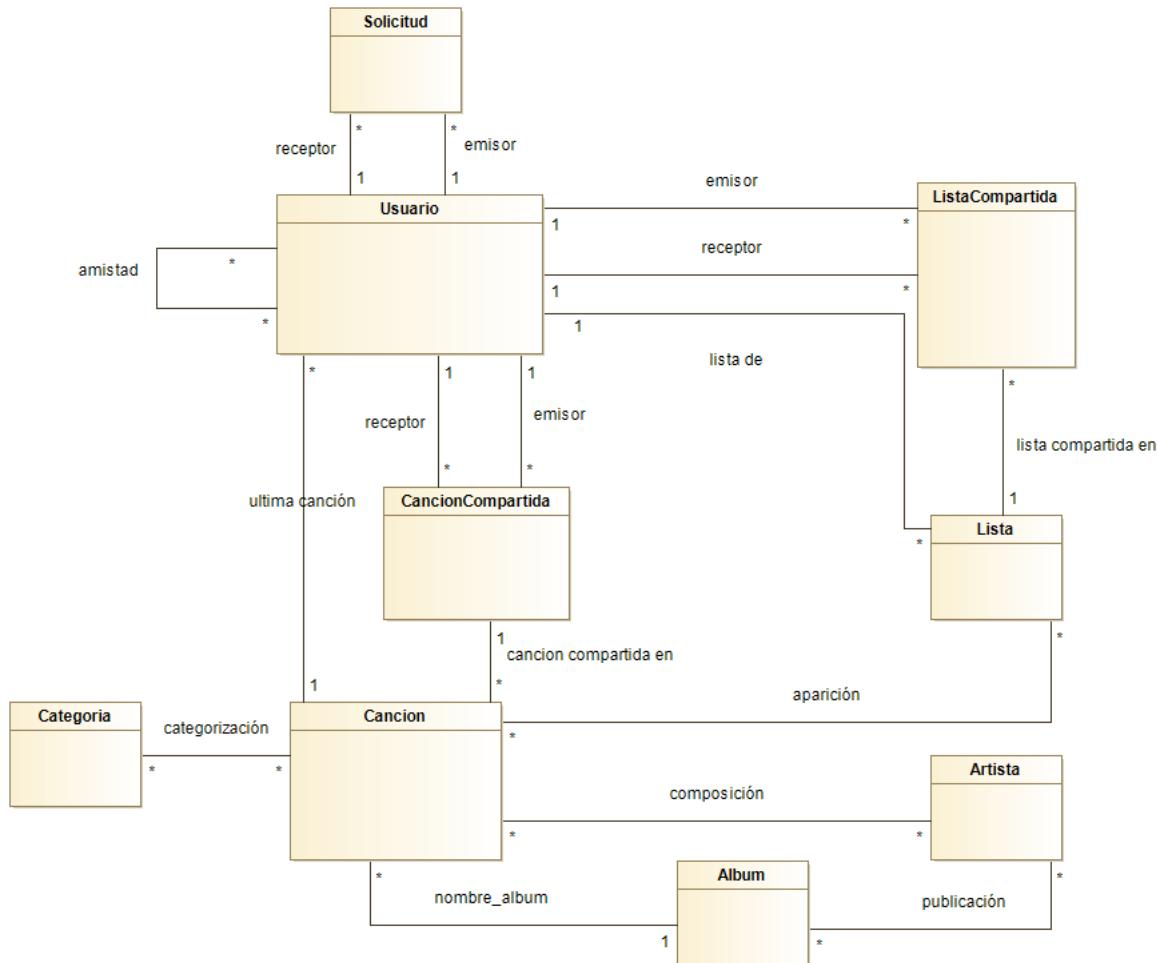


Diagrama de clases de objetos de la base de datos en el servidor. Las clases objeto almacenan los datos que se extraen de la base de datos, y son los que usa la aplicación para acceder a la información. De esa forma, todos los objetos están relacionados con la app y la database.

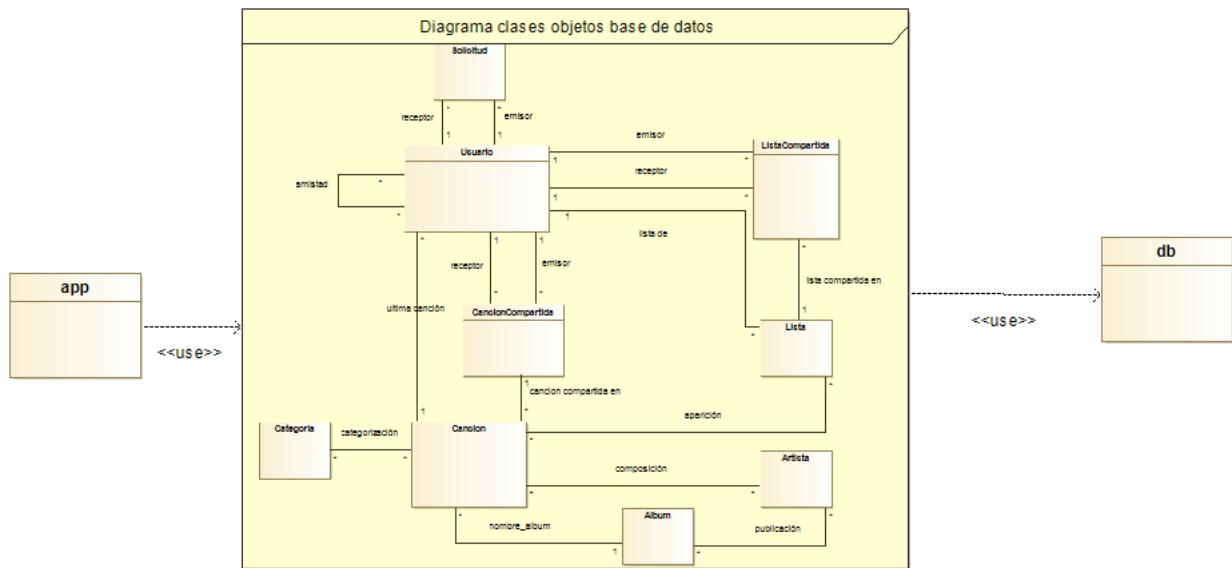
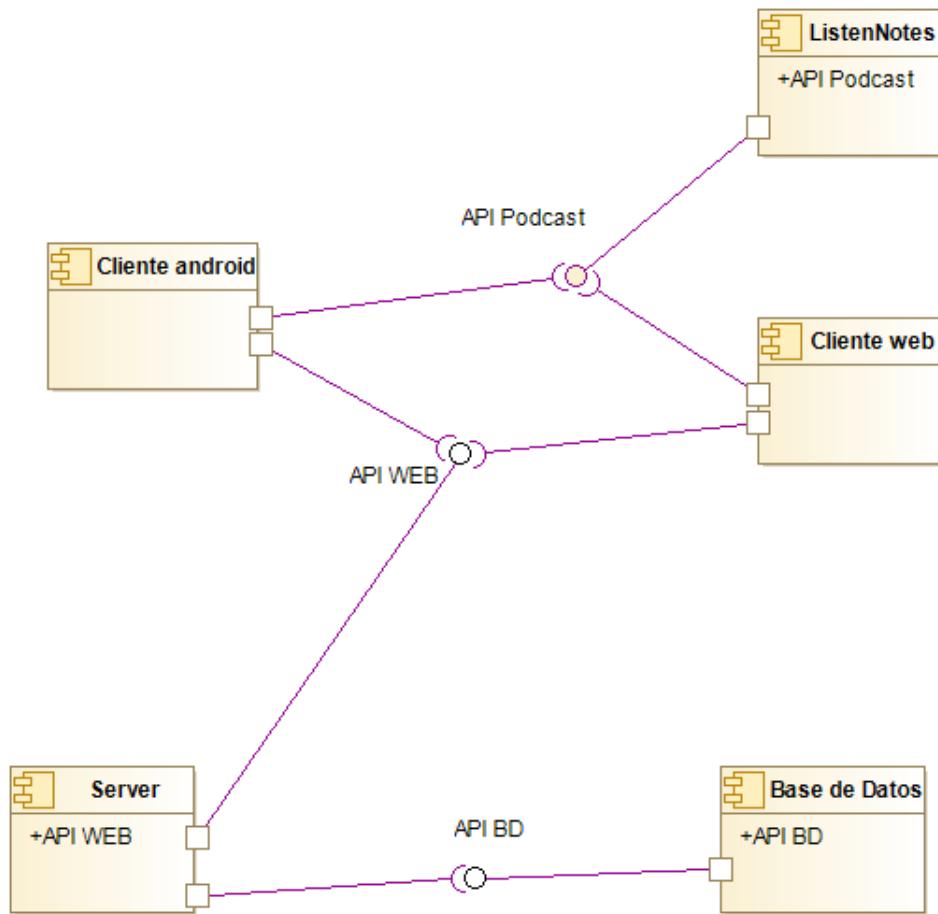
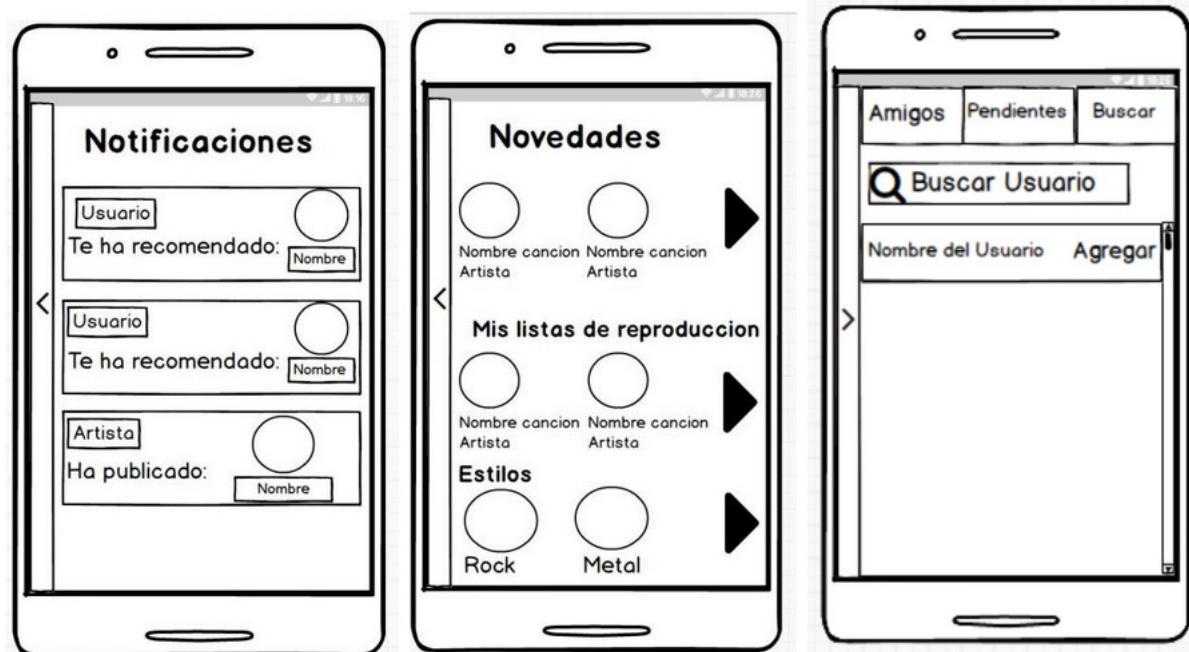
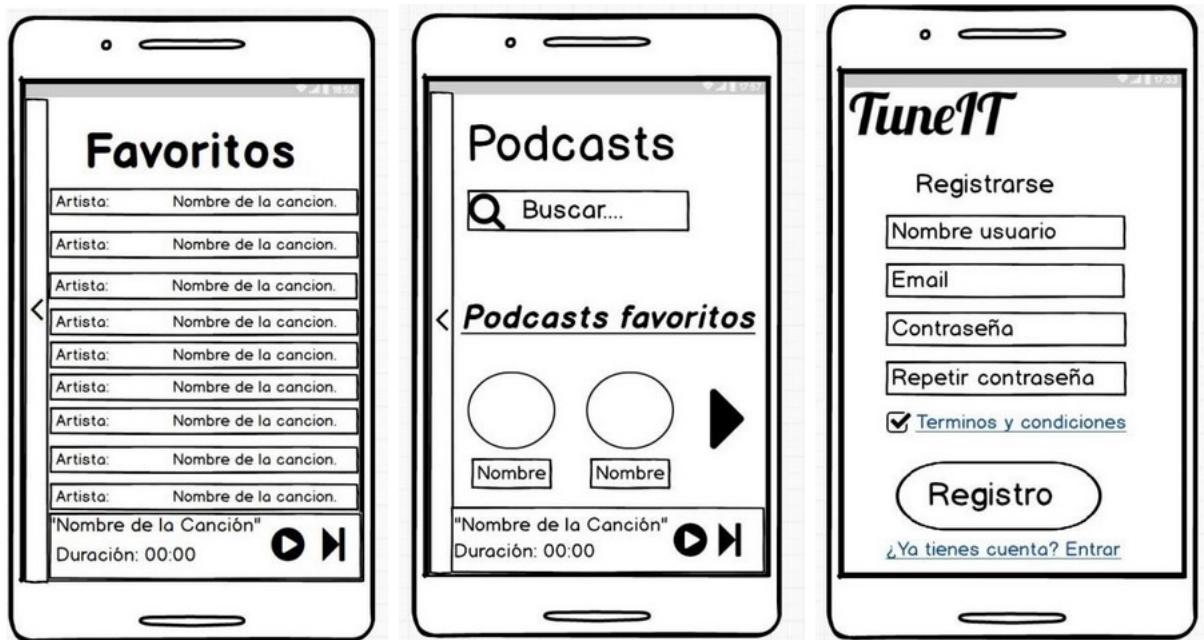
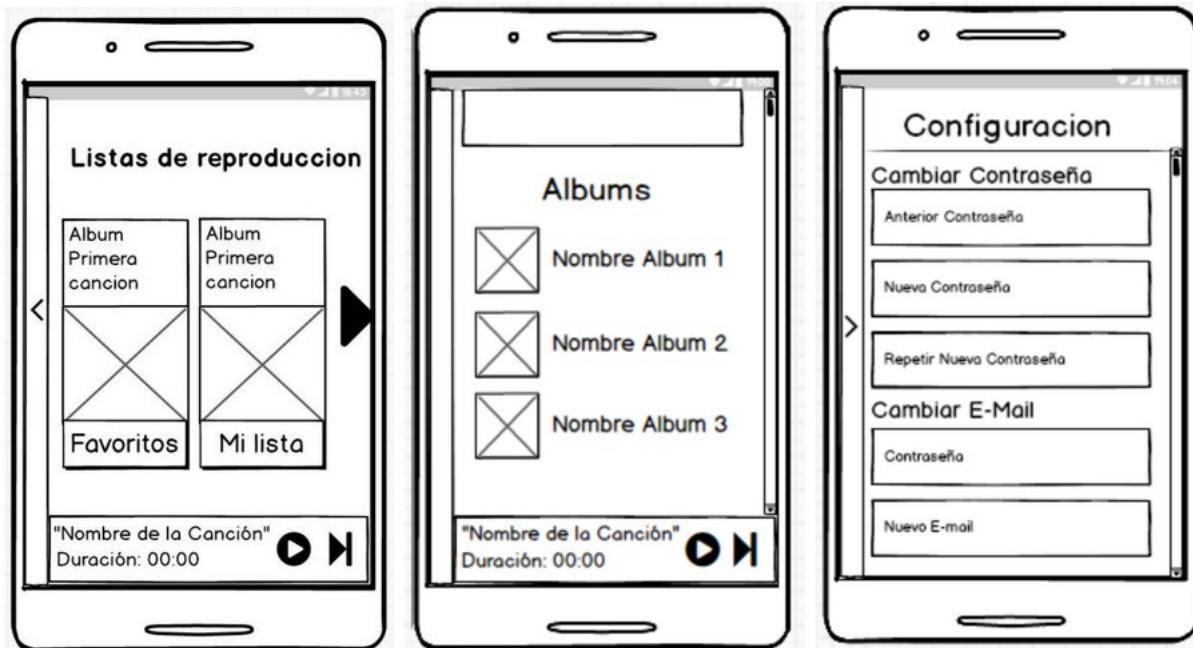


Diagrama de componentes y conectores:



Interfaces de Android:







Interfaces de Web:

TuneIT

NOTIFICACIONES AYUDA CONFIGURACIÓN

Buscar

Música Podcasts Listas de reproducción Amigos Perfil Ecualizador

@2020 CodeMusic

Canción Canción Canción Canción Canción Canción

Imagen álbum

Nombre álbum
Artista

0:59 3:25

Canción Artista

TuneIT

NOTIFICACIONES AYUDA CONFIGURACIÓN

Buscar

Música Podcasts Listas de reproducción Amigos Perfil Ecualizador

@2020 CodeMusic

Canción Artista

Descripción

Suscribirse

Álbums

Álbum 1 Álbum 2

Nombre artista

0:59 3:25

Canción Artista



NOTIFICACIONES

AYUDA

CONFIGURACIÓN

 Buscar

Música

Podcasts

Listas de reproducción

Amigos

Perfil

Ecualizador

@2020 CodeMusic

 Canción
Artista

Configuración

Cambiar contraseña

 Anterior contraseña Nueva contraseña Repetir nueva contraseña

Cambiar correo electrónico

 Contraseña Nuevo correo electrónico

Cambiar nombre de usuario

 Nuevo nombre

Cancelar

Aceptar

0:59 —————— 3:25 |◀ ▶| ❤️ ⌂ 🔊



NOTIFICACIONES

AYUDA

CONFIGURACIÓN

 Buscar

Música

Podcasts

Listas de reproducción

Amigos

Perfil

Ecualizador

@2020 CodeMusic

 Canción
ArtistaCanción
Artista

0:59 —————— 3:25



0:59 —————— 3:25 |◀ ▶| ❤️ ⌂ 🔊

Música para todos.

[Iniciar sesión](#)[Registrarse](#)

The mobile application interface for TuneIT is shown. On the left is a vertical sidebar with the following menu items:

- Música
- Podcasts
- Listas de reproducción
- Amigos
- Perfil
- Ecualizador

At the bottom of the sidebar is the copyright notice: ©2020 CodeMusic.

The main screen area has a header "Podcast" and a search bar labeled "Buscar". Below the search bar, it says "Podcasts de tus artistas favoritos". There are three circular placeholder cards, each with the word "Foto" and "Nombre". At the bottom of the main screen is a control bar with the following icons from left to right: a square icon labeled "Canción Artista", a progress bar from 0:59 to 3:25, a back arrow, a play/pause button, a forward arrow, a heart icon, a list icon, and a volume icon.



NOTIFICACIONES

AYUDA

CONFIGURACIÓN

Buscar

- Música
- Podcasts
- Listas de reproducción
- Amigos
- Perfil
- Ecualizador

@2020 CodeMusic

Canción
Artista

0:59 ━━━━ 3:25 |◀ □ ▶| ❤️ ⌂ 🔊



NOTIFICACIONES

AYUDA

CONFIGURACIÓN

Buscar

Nombre usuario

Última canción escuchada: Canción de Artista

- Música
- Podcasts
- Listas de reproducción
- Amigos
- Perfil
- Ecualizador

@2020 CodeMusic

Canción
Artista

Listas de reproducción



Álbum de la
primera
canción

Nombre



Álbum de la
primera
canción

Nombre



Álbum de la
primera
canción

Nombre



NOTIFICACIONES

AYUDA

CONFIGURACIÓN

 Buscar

Música

Podcasts

Listas de reproducción

Amigos

Perfil

Ecualizador

@2020 CodeMusic

 Canción
Artista

Notificaciones

Usuario te ha recomendado:Canción
ArtistaArtista ha publicado:Canción
Artista

0:59 ————— ●————— 3:25 |◀ □ ▶| ❤️ ⌂ 🔊

Notificaciones Informacion Ayuda

 Buscar

Listas de reproducción



Favoritos



MiLista



MiLista



MiLista



MiLista



MiLista



MiLista



MiLista

Musica

Podcasts

Listas de reproducción

Amigos

Perfil

Ecualizador

 Título
Artista

0:59



2:21

◀ □ ▶ ⌂ 🔊 ⟲ ⟳

TuneIT

Notificaciones Informacion Ayuda

Buscar

Musica Podcasts Listas de reproduccion Amigos Perfil Ecualizador

Mi lista

Añadir canciones +

Artista - Cancion Artista - Cancion

0:59 2:21 ▶◀ ▶ ▶▶ ❤ ⌂ 🔍

TuneIT

NOTIFICACIONES AYUDA CONFIGURACIÓN

Buscar

Música Podcasts Listas de reproducción Amigos Perfil Ecualizador

©2020 CodeMusic

Novedades

Mis listas de reproducción

Géneros

0:59 3:25 ▶◀ ▶ ▶▶ ❤ ⌂ 🔍

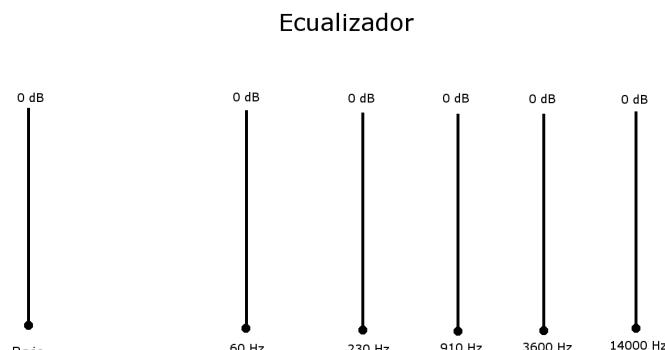
Iniciar sesión

[Entrar](#)[¿No estás registrado? Crear cuenta](#)

- Música
- Podcasts
- Listas de reproducción
- Amigos
- Perfil
- Ecuualizador

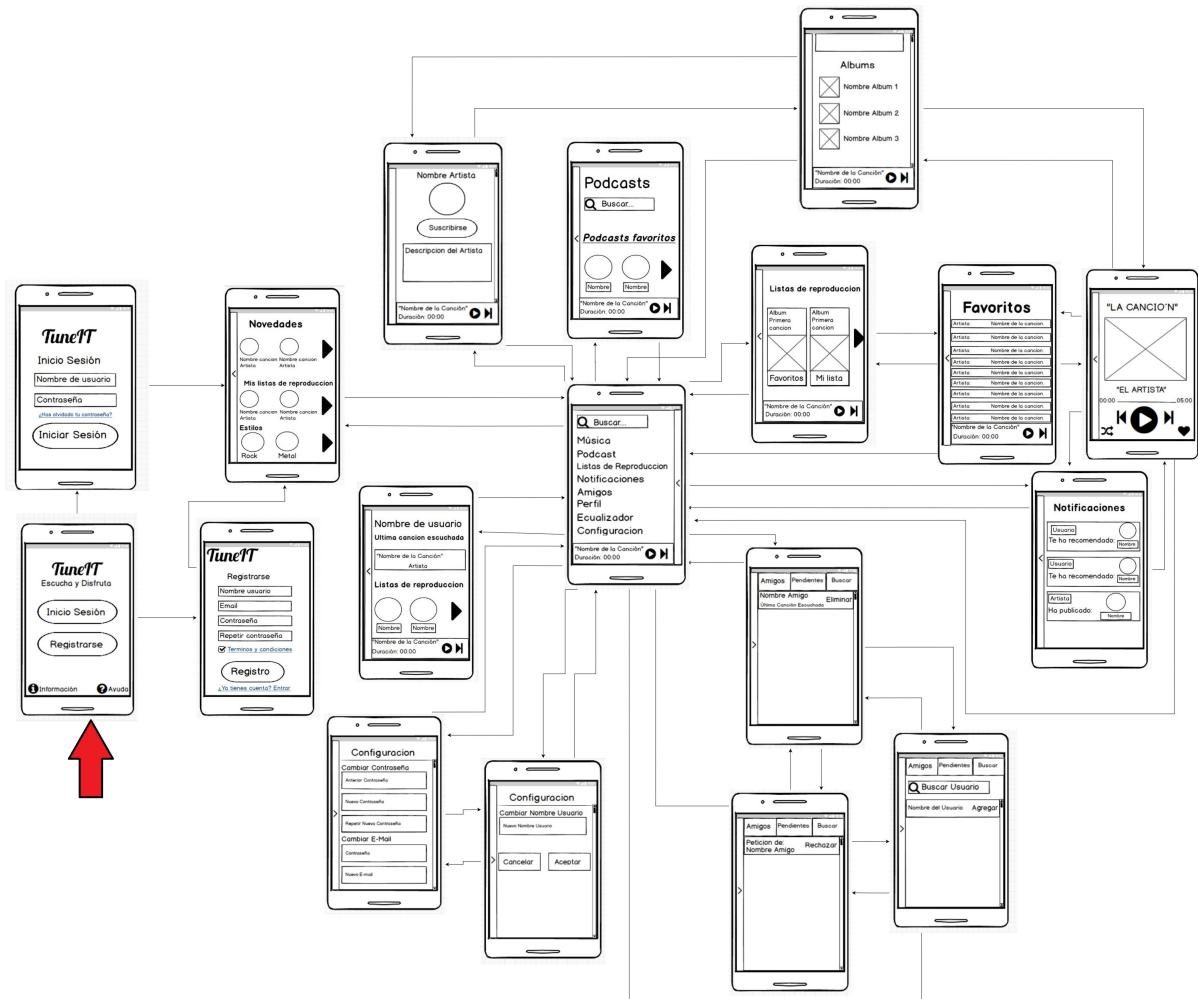
@2020 CodeMusic

Canción
Artista

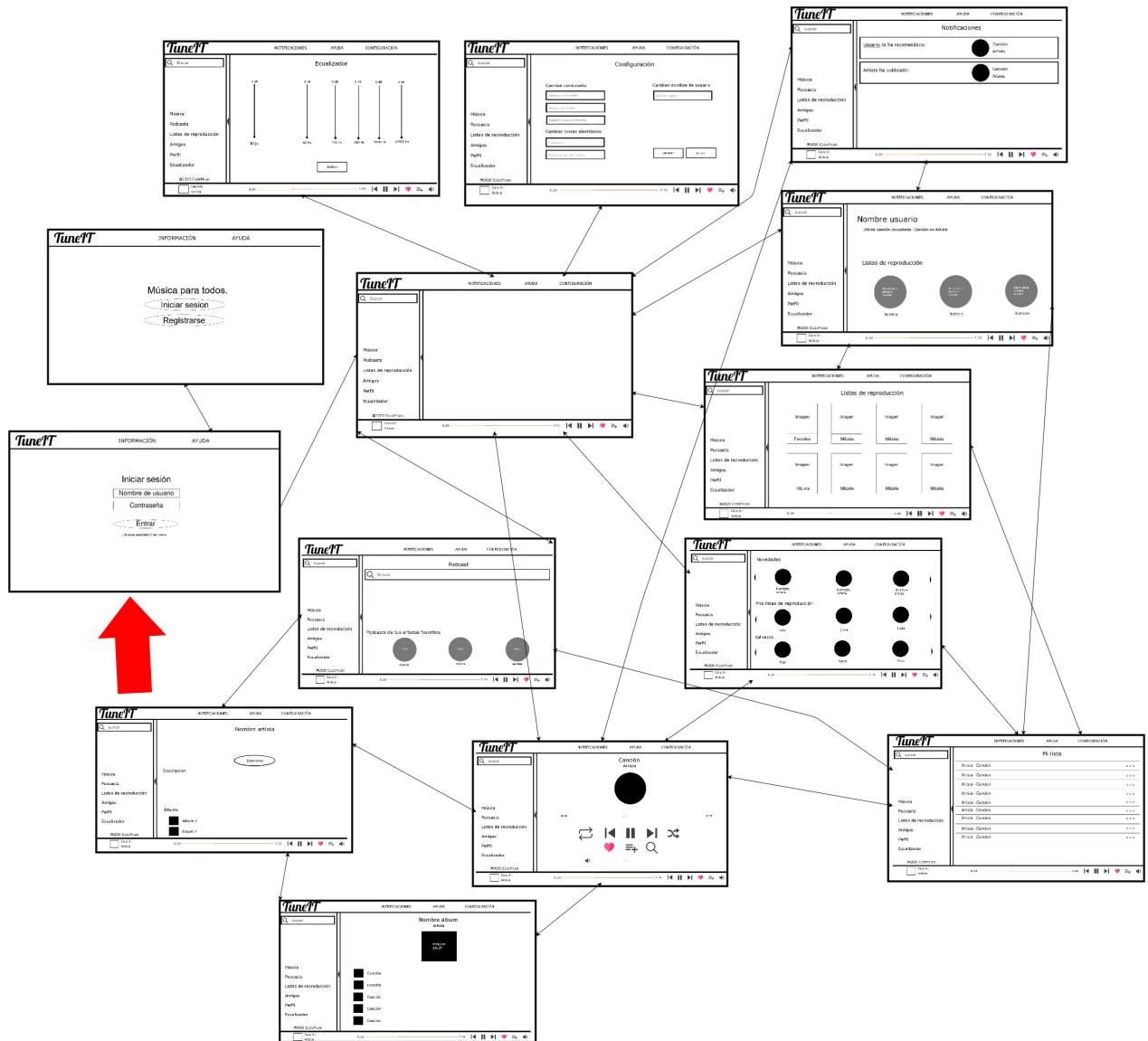
[Activar](#)

0:59 —————— 3:25 | ⟲ ⟳ ⟴ 🔍 🔍+ 🔊

Mapa de navegación de Android:



Mapa de navegación de Web:



5. Memoria del proyecto

En este capítulo se describe cómo se ha llevado a cabo el proyecto, qué cambios se han hecho respecto a la versión inicial, imprevistos surgidos, etc.

5.1. Inicio del proyecto

En el inicio del proyecto se organizaron dos grupos, uno de backend y otro de frontend. Tras la reunión con el profesor se siguió el consejo de dividir frontend en dos: web y android. Rápidamente se eligieron los frameworks con los que se iba a trabajar consultando a otros alumnos y tras la primera reunión con el profesor. De esta forma se inició rápidamente la primera etapa de documentación mediante tutoriales y la documentación oficial de los frameworks elegidos. La etapa de formación duró aproximadamente dos semanas, aunque se necesitó seguir formándose a lo largo del proyecto ya que casi todas las tecnologías eran nuevas o no se habían utilizado con los objetivos establecidos en este proyecto.

La división de los equipos permitió organizar a los miembros de una mejor forma reduciendo complejidad al trabajar como pequeños equipos. Se permitía que los coordinadores de cada uno se reuniesen entre ellos o con el director general sin tener que estar todos los miembros.

La elección de trabajar en cloud permitió a todos los miembros disponer de los demás componentes del sistema en una época temprana (aún en pruebas) para poder conectarse e interactuar con backend y la base de datos.

5.2. Ejecución y control del proyecto

El reparto de las tareas en el grupo de Front End Android empezó mediante la elección de los miembros del grupo de una tarea de un conjunto de tareas asignada por el propio coordinador, una vez realizada la tarea y revisada por dicho coordinador, este asignó a cada uno de los miembros del grupo una serie de tareas relacionadas con la tarea anteriormente realizada. Mientras que en el reparto de tareas del grupo Front End Web se ha realizado mediante el reparto de las diferentes tareas según la dificultad, las tareas que tienen que ver con algo realizado por alguno de los miembros del grupo son asignadas a estos, ya que están experimentados en la materia o están familiarizados con dicha tarea. Por último, en el grupo de Back End, uno de los miembros se ha centrado más en el funcionamiento del servidor mientras que el otro miembro del grupo se ha centrado más en el despliegue del mismo y la base de datos.

La comunicación interna entre los distintos miembros de los diferentes equipos se ha realizado según el apartado de Procesos de ejecución y control del proyecto [3.1.2], en el cual, explica las diferentes vías de comunicación que han utilizado dichos miembros de los grupos. El progreso se ha medido conforme al cumplimiento de los requisitos tanto en los grupos de Front End como Back End, los requisitos se deciden si se cumplen o no durante las reuniones entre todos los miembros del grupo, además de que cada requisito tiene asociado una puntuación entre 1 y 3, que fue decidida por todos los miembros del grupo durante una votación. Los trabajos se saben si han sido realizado mediante los “Issues” en la plataforma Github, explicado anteriormente en el apartado de Procesos de ejecución y control del proyecto [3.1.2].

Se han realizado divergencias frente al calendario inicial si más de la mitad de los miembros del proyecto no han realizado su parte correspondiente, dando como máximo 1 día más.

Durante el desarrollo de integración y de despliegue, los miembros del grupo de Back-End no sabían cómo realizar dichas acciones, esto se ha resuelto mediante la adquisición de conocimiento sobre Heroku y Docker, además la mínimos conocimientos de las tecnologías utilizadas en backend y frontend.

Durante las pruebas del software, se han cumplido las ideas que se tenían al respecto mediante el cumplimiento de los requisitos de la aplicación.

5.3. Cierre del proyecto

Para cerrar el proyecto, se van a llevar a cabo una serie de análisis sobre aspectos importantes, como es una comparación entre las estimaciones iniciales planteadas y los resultados finales, un resumen de las lecciones aprendidas sobre las herramientas y tecnologías empleadas y una recopilación del trabajo realizado por cada uno de los miembros.

En la propuesta técnica y económica del proyecto se plantearon unas estimaciones sobre las horas de esfuerzo enfocadas a realizar las tareas establecidas y el coste económico que éstas implicarían. Tal y como aparece se presupuestaron **760 horas** con un costo total de 14.060 € (18.5 € por hora), a lo que hay que sumar un incremento de 2810 € por gestión y 1.040 € por costes adicionales convirtiéndolos en **17.910 € sin IVA**. Se calculó un segundo presupuesto que venía acompañado de funcionalidades opcionales que se estimó de 2330 € más, esto es **20.240 € sin IVA**.

A día 10 de abril, se han invertido un total de **271** horas en desarrollo y documentación, lo que significa aproximadamente 200 horas mensuales. Además hay que tener en cuenta las reuniones que suman alrededor de 20 horas por miembro, por lo que se podría decir que hasta la fecha el total de horas invertidas es de 411.

A este ritmo se llegaría a la entrega final del 8 de junio con un total de **800 horas**, es decir, 14800 € contabilizando únicamente las horas de trabajo. Los resultados finales que se esperan son superiores a los estimados inicialmente, ya que han surgido numerosos imprevistos que han requerido de muchas horas para solucionarlos.

También es cierto que las estimaciones iniciales se calcularon sin dejar un margen de tiempo extra (un error que no se puede cometer) planteando los tiempos de manera optimista. Como consecuencia, el presupuesto final puede ser algo mayor que el estimado para el cliente.

Lecciones aprendidas sobre herramientas y tecnologías:

- Respecto a **SQLAlchemy**, se ha visto que proporciona un ORM (Object Relational Mapper) el cual permite crear y manipular una base de datos como si fuera POO (Programación orientada a objetos) lo cual es una ventaja dada nuestra experiencia con bases de datos relacionales comunes y POO.
- En el servidor **Heroku**, la curva de aprendizaje ha sido lenta y costosa debido a que no se sabía cómo se iba a desplegar el proyecto, ya sea con contenedores, con manifiestos que indiquen cómo descargar heroku, o haciendo push desde los repositorios de git... Otro factor que contribuyó a este lento aprendizaje fue que en la documentación no aparecen casos de error y sus posibles causas o soluciones por lo que se tenía que investigar cada pequeño fallo. A partir de esos 2 factores principales, la curva crece y se hace sencillo el manejo de Heroku y sus utilidades.
- En el lado de front end, se ha visto que el proyecto en la parte web, en **Angular**, se comenzó con los tutoriales y se aprendió los básicos de dicha tecnología de manera muy rápida. El problema comenzó cuando se tuvo que empezar a aplicar dichos conocimientos al proyecto. La curva de aprendizaje dejó de crecer y cada pequeña funcionalidad, costaba mucho más

tiempo del ideal debido a la diferencia entre la realidad y los tutoriales. El único momento donde se vió un aumento de la efectividad fue al realizar tareas semejantes ya que su “modus operandi” era muy parecido, por ejemplo, a la hora de crear un buscador de podcasts, se tardó 17 horas, muchas más de las esperadas, pero al crear un buscador de canciones apenas se tardaron 3 horas debido a que el proceso era muy similar.

- Por otra parte, desde el lado de móvil, se ha visto que la tecnología **Flutter** contiene un problema principal y es que está en constante desarrollo por lo que muchas cosas cambian y los tutoriales dejan de ser útiles, esto hace que la curva de aprendizaje sea muy pronunciada. También se ha visto que es muy útil para diseñar e implementar interfaces pero para implementar herramientas en concreto apenas tiene documentación y el equipo de desarrollo tiene que descender a Java para poder crear dicha herramienta, por ejemplo el ecualizador.
- Por último, en el uso de **Flask** se ha visto que es una herramienta muy sencilla, con una curva de aprendizaje muy pronunciada y muy bien documentada.

Recopilación de horas de trabajo y tareas de cada uno de los miembros del equipo:

| Miembro | Horas | Tareas |
|--------------------|-------|---|
| Alberto | 51,40 | Dirección de proyecto, coordinador de Documentación, encargado de integración y despliegue (recursos cloud) y administración de base de datos |
| Alejandro | 25,12 | Desarrollador Android |
| Álvaro | 53,58 | Coordinador de frontend web y desarrollador web |
| Germán | 40,67 | Desarrollador web y responsable actas |
| Luis | 53,50 | Coordinador de frontend Andrid y desarrollador Android |
| Óscar | 40,25 | Desarrollador Android |
| Saúl | 42,50 | Coordinador backend, desarrollador web y API |
| Total horas | | 307,02 |

6. Conclusiones

7. Anexo I. Glosario

8. Anexo II. Actas de todas las reuniones realizadas

9. Anexo III..Otros anexos que se consideren necesarios

- [1]: <https://developer.spotify.com/branding-guidelines/>
- [2]: <https://gcase.files.wordpress.com/2015/10/spotify-developer-brand-standards-guide-02jr.pdf>
- [3]: https://issuu.com/duncanriley/docs/01_sz_brand_guidelines_lite
- [4]: <https://marketplace.visualstudio.com/items?itemName=johnpapa.angular-essentials>
- [5]: <https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>
- [6]: <https://www.listennotes.com/es/api/>
- [7]: <https://prettier.io/>
- [8]: <https://www.npmjs.com/package/codelyzer>
- [9]: <https://dart.dev/tools/dartanalyzer>
- [10]: <https://www.freefonts.io/moderne-sans-free-typeface/>
- [11]: <http://cloc.sourceforge.net/>
- [12]: <https://developer.android.com/design>
- [13]: <https://www.w3.org/standards/>
- [14]: <https://www.heroku.com>
- [15]: <https://pypi.org/project/pylint-flask-sqlalchemy/>
- [16]: <https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM60jgkjFeUxCYH>
- [17]: <https://www.youtube.com/watch?v=PTZiDnuC86g>
- [18]: <https://angulararr.io/stat>
- [19]: <https://auth0.com/blog/amp/building-an-audio-player-app-with-angular-and-rxjs/>
- [20]: <https://developer.android.com/guide>
- [21]: <https://www.youtube.com/watch?v=1ukSR1GRtMU&list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ>