music.io

Trabajo práctico

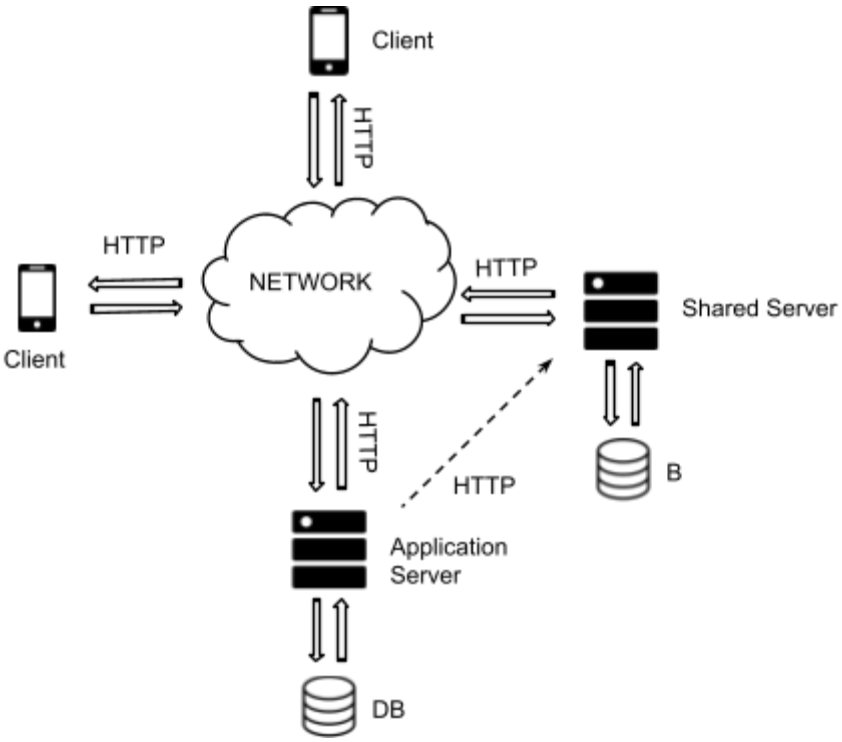
Objetivos	● Afianzar los conocimientos adquiridos durante la cursada. ● Poner en práctica la coordinación de tareas dentro de un grupo de trabajo. ● Realizar un aplicativo de complejidad media con niveles aceptables de calidad y usabilidad.
Instancias de Entrega	**Checkpoint 1:** clase 4 (dd/mm/aaaa). **Checkpoint 2:** clase 8 (dd/mm/aaaa). **Checkpoint 3:** clase 12 (dd/mm/aaaa). **Entrega:** clase 14 (dd/mm/aaaa).
Criterios de Evaluación	● Empleo de buenas prácticas de programación ● Empleo de test unitarios ● Coordinación de trabajo grupal. ● Planificación y distribución de tareas para cumplir con los plazos de entrega pautados. ● Cumplimiento de todos los requerimientos técnicos y funcionales. ● Facilidad de instalación y ejecución del sistema final. ● Calidad de la documentación técnica y manuales entregados. ● Buena presentación del trabajo práctico y cumplimiento de las normas de entrega establecidas por la cátedra

Introducción

Debido a la repercusión mediática de las aplicaciones para la reproducción de música la empresa AppMaker© nos ha encargado el desarrollo de una aplicación similar a fin de lograr participar de este negocio.
AppMaker busca posicionarse en el mercado de aplicaciones, busca conectar esta aplicación con sus otras aplicaciones que poseen gran cantidad de usuarios.

La aplicación constará de tres componentes:

- Un servidor (Application Server), el cual será el responsable del procesamiento, almacenamiento y el delivery de las solicitudes de los usuarios relacionados con la descarga de canciones.
- Un servidor (Shared Server), el cual será responsable de la administración de los información de uso general de la aplicación. Esta información consta de un catálogo de canciones, álbumes, géneros y artistas disponibles y un servicio de recomendaciones en base a la actividad del usuario en el sistema.
- Un cliente, el cual será responsable de visualizar y reproducir las canciones disponibles en el sistema.



Aplicaciones Requeridas

Application Server

Se trata de una aplicación linux por consola destinada a mantenerse en ejecución por períodos prolongados de tiempo. Esta aplicación debe brindar una interfaz para la comunicación de los diferentes clientes que se conecten a la misma. Para la interfaz de comunicación se utilizará Restful API [1], que definirá la forma de las solicitudes y respuestas de los diferentes servicios que brindará el servidor.

El objetivo principal de esta aplicación es la de administración del contenido multimedia del sistema.

Servicio de descarga

Este servicio permitirá al usuario descargar una canción para su posterior reproducción.

Log

El servidor debe constar con un sistema de log en donde se registren los eventos que se generen durante la ejecución del mismo. El sistema de log debe permitir configurar el nivel de los eventos que desean registrar. Estos niveles son:

Nivel	Condiciones
Error	Condición de falla catastrófica, el sistema no puede funcionar. (criterio de las 2 a.m.) Condición que haga que la aplicación no pueda ejecutar una funcionalidad. Ejemplo: No es posible conectarse con la base de datos
Warn	Cualquier condición anómala que afecte el funcionamiento del sistema, pero no impida la funcionalidad básica Ejemplos: Uso de APIs deprecadas Mal uso de APIs
Info	Cualquier acción correspondiente a un caso de uso iniciada por el usuario o el sistema. Información que permita trazar el historial de las entidades. Ejemplos: Conexión a la base de datos exitosa Conexión de nuevo cliente
Debug	Información de contexto que permita resolver un problema técnico. Debe ser útil incluso sin el código fuente Ejemplo: Datos de login para la DB

Instalación de aplicación

La instalación de la aplicación cliente debe ser un proceso simple y que ayude a un usuario inexperto a poder realizar la instalación de la misma sin mayores inconvenientes.
Para esto se deberá generar un paquete .deb siguiendo los lineamientos de empaquetado de debian. https://www.debian.org/doc/manuals/maint-guide/

Para generar el paquete .deb es posible realizarlo utilizando cpack https://cmake.org/Wiki/CMake:Packaging_With_CPack o realizarlo de forma manual.

Prueba sobre API REST

Para las pruebas funcionales sobre la API REST implementada se deberá utilizar python como lenguaje de scripting.

Shared Server

Dado que se requiere poseer varios servidores para la descentralización de la administración de datos de uso común de la aplicación, AppMaker© nos solicita que este servidor se encuentre disponible en la nube utilizando como plataforma Heroku (https://www.heroku.com/).
Se entiende por datos de uso común a:

- Listado de artistas disponibles
- Listado de canciones
- Listado de albumes
- Puntuaciones de canciones
- Sistema de recomendaciones

Este servidor deberá permitir administrar estos datos utilizando una interfaz gráfica (aplicación web) y a través de un una API para la administración de datos comunes (Restful API [1]). El sistema de recomendaciones no se administrará a través de la aplicación web.

Este servidor deberá ser implementado utilizando NodeJS (https://nodejs.org) para el desarrollo de la API y angular (https://angularjs.org/) para el desarrollo de la aplicación web. Además se deberá utilizar Postgresql como base de datos (http://www.postgresql.org.es/)

Para poder asegurar la interoperabilidad entre las diferentes implementaciones de Shared Server se deberá respetar una interfaz común a todos para la administración de datos a través de la API Restful.

Esto permitirá poder configurar desde el servidor de aplicaciones (Application Server) a cuál Shared Server conectarse.

Sistema de recomendación

En función de la actividad del mismo dentro del sistema se debe brindar recomendaciones de artistas y canciones a reproducir:

- Reproducciones realizadas

- Género musical de reproducciones
- Canciones marcadas
- Puntuaciones de canciones realizadas
- Artistas seguidos
- Información de los contactos del usuario

Almacenamiento de actividad de usuario

El servidor deberá almacenar la actividad del usuario para luego utilizarla para generar las sugerencias. Alguna de estas actividades son:

- Seguir artista
- Marcar canción
- Puntuación canciones
- Artistas seguidos por contactos

Especificación API

La especificación de la API se realiza utilizando OpenAPI-Specification. La especificación completa de la API se encuentra publicada en el siguiente link

Todos los servicios expuestos por la API deben poseer autenticación.

Autenticación

- **Generación de token para autenticación (POST /tokens):** genera un token para utilizar en los servicios del sistema (login)

Usuarios

- **Alta de usuario (POST /users)**
- **Listado de usuarios (GET /users?ids={userIds}):** obtiene el listado de usuarios del sistema
- **Obtener un usuario (GET /users/{userId}):** obtiene un usuario del sistema
- **Información del usuario (GET /users/me):** obtiene la información asociada al usuario (perfil)
- **Actualización de perfil (PUT /users/me):**
- **Actualización de foto de perfil (PUT /users/me/photo):**
- **Listado de contactos (GET /users/me/contacts):** obtiene el listado de contactos del usuario
- **Agregar un contacto (POST /me/users/contacts/{userId}):** obtiene un usuario del sistema
- **Modificar un usuario (PUT /users/{userId}):** realiza una modificación a un usuario existente
- **Eliminar un usuario (DELETE /users/{userId}):** eliminar un usuario

Canciones

- **Listado de canciones (GET /tracks?ids={trackIds})**: obtiene el listado de canciones indicadas
- **Obtener canción (GET /tracks/{trackId})**: Información completa sobre la canción
- **Listado de canciones recomendadas (GET /me/tracks/favorites):** obtiene el listado de canciones disponibles ordenadas por las preferencias del usuario
- **Dar de alta a una nueva canción (POST /tracks):** alta de una nueva canción
- **Modificar una canción (PUT /tracks/{trackId}):** realiza una modificación a una canción existente
- **Puntuar canción (POST /tracks/{trackId}/popularity):** indica una puntuación entre 1-5.
- **Obtener puntuación (GET /tracks/{trackId}/popularity)**: obtiene el puntaje (promedio de los realizados por el usuario).
- **Like a una canción (POST /tracks/{trackId}/like):** like a una canción
- **Dislike a una canción (DELETE /tracks/{trackId}/like):** dislike a una canción
- **Eliminar una canción (DELETE /tracks/{trackId}):** eliminar una canción

Album

- **Listado de álbumes (GET /albums?ids={albumIds})**: obtiene el listado de álbumes indicados
- **Dar de alta un nuevo álbum (POST /albums):** Alta de un nuevo álbum
- **Obtener álbum (GET /albums/{albumId})**: Información completa sobre el álbum
- **Modificar un álbum (PUT /albums/{albumId}):**realiza una modificación a un álbum existente
- **Eliminar un álbum (DELETE /albums/{albumId})**
- **Agregar una canción a un álbum (PUT /albums/{albumId}/track/{trackId})**
- **Eliminar una canción de un álbum (DELETE /albums/{albumId}/track/{trackId})**

Artistas

- **Listado de artistas (GET /artists?ids={artistIds}):**obtiene el listado de artistas indicados
- **Listado de artistas recomendadas (GET /users/me/artists/favorites):** obtiene el listado de artistas disponibles ordenadas por las preferencias del usuario
- **Seguir artista (POST /users/me/artists/{artistId}/follow)**
- **Dejar de seguir artista (DELETE /users/me/artists/{artistId}/follow)**
- **Listado de canciones por artistas (GET /artists/{artistId}/tracks)**
- **Dar de alta un nuevo artista (POST /artists)**
- **Modificar un artísta (PUT /artists/{artistId})**
- **Obtener artísta (GET /artists/{artistId})**
- **Eliminar artísta (DELETE /artists/{artistId})**

Playlists

- **Listado de playlists (GET /playlists?ids={playListIds}):**obtiene el listado de playlists indicadas
- **Listado de canciones de una playlists (GET /playlists/{playListId}/tracks)**
- **Listado de discos de una playlists (GET /playlists/{playListId}/albums)**
- **Dar de alta una nueva playlist (POST /playlists)**
- **Obtener playlist (GET /playlists/{playListId})**
- **Modificar una playlist (PUT /playlists/{playListId})**
- **Agregar una canción a una playlist (PUT /playlists/{playListId}/tracks/{trackId})**
- **Agregar un album a una playlist (PUT /playlists/{playListId}/albums/{albumId})**
- **Eliminar una playlist (DELETE /playlists/{playListId})**

Virtualización del server

Debido a la limitación de requests periódicos que se pueden realizar en la plataforma Heroku, se deberá armar una imagen docker con nodejs que pueda levantar el servidor de forma local, de forma que se puedan correr pruebas sobre el mismo. Se sugiere utilizar una imagen aparte para levantar un servidor Postgresql

Log

El servidor debe constar con un sistema de log en donde se registren los eventos que se generen durante la ejecución del mismo. El sistema de log debe permitir configurar el nivel de los eventos que desean registrar. Estos niveles son:

Nivel	Condiciones
Error	Condición de falla catastrófica, el sistema no puede funcionar. (criterio de las 2 a.m.) Condición que haga que la aplicación no pueda ejecutar una funcionalidad. Ejemplo: No es posible conectarse con la base de datos
Warn	Cualquier condición anómala que afecte el funcionamiento del sistema, pero no impida la funcionalidad básica Ejemplos: Uso de APIs deprecadas Mal uso de APIs
Info	Cualquier acción correspondiente a un caso de uso iniciada por el usuario o el sistema. Información que permita trazar el historial de las entidades. Ejemplos: Conexión a la base de datos exitosa Conexión de nuevo cliente
Debug	Información de contexto que permita resolver un problema técnico. Debe ser útil incluso sin el código fuente Ejemplo:

	Datos de login para la DB

Documentación

Es condición necesaria para la aprobación del trabajo práctico la entrega de una documentación formal de carácter profesional.

Pruebas

El desarrollo de la aplicación se deberá adaptar a los estándares de calidad utilizados por AppMaker©. Dentro de estos estándares para el servidor se encuentran:

- Pruebas unitarias [2]
- Métricas: code coverage debe ser mayor a 75% [3]
- Respetar estándar para estilo de codificación https://github.com/google/styleguide
- Pruebas de integración utilizando las herramientas que parezcan adecuadas (por ej. gtest, Karma,mocha,jasmine)
- Todas aquellas que se consideren convenientes para garantizar la calidad de las aplicaciones desarrolladas.

Integración continúa

Para poder realizar una integración de todos los componentes del servidor (Shared Server) debe utilizarse un servidor de integración continua para poder realizar una integración automática de estos componentes. En caso de utilizarse GitHub puede utilizarse Travis CI (https://travis-ci.org/)

Client

El sistema cliente posee una interfaz gráfica que permitirá visualizar el almacenamiento remoto del usuario que se encuentra utilizando la aplicación.
Para el diseño de la interfaz deben respetarse las guías de diseño propuesto por Google (https://www.google.com/design/spec/material-design/introduction.html).

Autentificación

Para que el cliente pueda enviar y recibir conversaciones el mismo debe autentificarse con el servidor. Para esto el cliente debe permitir al usuario ingresar usuario y contraseña. Se deberá diseñar una pantalla de ingreso a la aplicación.
Cuando la autenticación es exitosa, se almacenará un token de autenticación para realizar las operaciones con el servidor.

Registro

En caso de que la persona que desee utilizar la aplicación no tenga un usuario registrado la aplicación debe permitir poder registrar un usuario nuevo. Este registro deberá poder ser integrado con Facebook https://developers.facebook.com/docs/facebook-login/android, permitiendo ingresar las credenciales de facebook y utilizar los datos asociados de la cuenta para completar el registro del usuario dentro del sistema.

Interfaz gráfica

- Visualización de perfil
- Visualización de contactos de usuario
- Visualización de biblioteca de música
- Visualización de conversaciones
- Visualización de reproducción

Visualización de perfil
La aplicación debe permitir visualizar la información completa del perfil, visualizando:

- Nombre
- Foto de perfil
- Ubicación actual
- Cantidad de contactos
- Actividad

Además permitirá que el usuario que posea el perfil pueda editar de forma completa sus datos.

Visualización de contactos de usuario
La aplicación debe permitir visualizar la red de contactos que posee un usuario.

Biblioteca de música
La aplicación debe permitir realizar una búsqueda sobre los profesionales que se encuentran dentro del sistema. Esta búsqueda debe ser configurable y debe permitir filtrar la lista de profesionales, filtrando por skills o posiciones de trabajo.

Visualización de conversaciones (chat)
La aplicación debe permitir que un usuario pueda enviar un mensaje y mantener una conversación con un contacto que se encuentre registrado en el sistema. El chat debe implementarse con Firebase [9] (sin ninguna necesidad de crear una base de datos local en Android)

Visualización de reproducción
La aplicación debe permitir que un usuario pueda reproducir y visualizar la reproducción de una canción. Además debe permitir al usuario pausar la reproducción y manejar el volumen de la misma.

Notificaciones

Para el envío y recepción de notificaciones del sistema (ej. Solicitud de agregar contacto o recepción de nuevo mensaje) se deberá utilizar Google cloud message https://developers.google.com/cloud-messaging/.

Log

El cliente deberá utilizar el sistema de log por defecto http://developer.android.com/intl/es/reference/android/util/Log.html

Despliegue servidor

Para poder realizar una correcta administración de las dependencias que requiera la aplicación servidor (Application Server), y para poder simplificar el proceso de despliegue (deploy) del mismo. Se deberá utilizar Docker http://docker.io/ como herramienta.

Documentación

Es condición necesaria para la aprobación del trabajo práctico la entrega de una documentación formal de carácter profesional.

Pruebas

El desarrollo de la aplicación se deberá adaptar a los estándares de calidad utilizados por AppMaker©. Dentro de estos estándares se encuentran:

- Pruebas unitarias [2]
- Métricas: code coverage debe ser mayor a 75% [3]
- Respetar estándar para estilo de codificación https://github.com/google/styleguide
- Pruebas de integración utilizando python como lenguaje de scripting.
- Todas aquellas que se consideren convenientes para garantizar la calidad de las aplicaciones desarrolladas.

Integración continúa

Para poder realizar una integración de todos los componentes del servidor (Application Server) debe utilizarse un servidor de integración continua para poder realizar una integración automática de estos componentes. En caso de utilizarse GitHub puede utilizarse Travis CI (https://travis-ci.org/)

Restricciones

- El servidor (application server) debe ser desarrollado en C/C++. El servidor debe soportar múltiples conexiones en simultáneo.
- Para la compilación de la aplicación servidor se deberá utilizar CMake [4]
- La aplicación debe desarrollarse en Java utilizando el SDK de Android [5]
- Para la documentación técnica del trabajo práctico se utilizará Sphinx [6]
- Para el almacenamiento de la información del servidor se deberá utilizar mongodb. [8]

Referencias

- [1] http://en.wikipedia.org/wiki/Representational_state_transfer
- [2] http://en.wikipedia.org/wiki/Unit_testing
- [3] http://en.wikipedia.org/wiki/Code_coverage
- [4] http://www.cmake.org/
- [5] http://developer.android.com/sdk/index.html
- [6] http://sphinx-doc.org/
- [8] https://www.mongodb.com
- [9] https://firebase.google.com/