ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN PROTOTIPO EN ANDROID, PARA EL STREAMING Y REPRODUCCIÓN DE MEDIOS DE UNA RADIO, E INTEGRACIÓN CON CHROMECAST.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JUAN ANDRÉS QUINTUÑA SILVA

DIRECTOR: Ms. FRANKLIN LEONEL SÁNCHEZ CATOTA

Quito, julio 2021

AVAL

Demuestro que este trabajo fue desarrollado por Juan Andrés Quintuña Silva bajo mi dirección.

MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Juan Andrés Quintuña Silva prometo que el trabajo aquí descrito es de mi autoría; que no ha sido anteriormente presentado en ningún grado; y, que he investigado las referencias bibliográficas que se incluyen en este documento.

Por la presente declaro que la Escuela Politécnica Nacional podrá hacer uso del actual trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

JUAN ANDRÉS QUINTUÑA SILVA

DEDICATORIA

A mi Dios por darme la inteligencia necesaria, y ser mi apoyo incondicional.

A mi mama Gardenia Silva por alentarme y ser mi ejemplo de perseverancia

A mi esposa Diana Portilla por motivarme a finalizar esta fase de mi vida.

AGRADECIMIENTO

A mi familia por creer en mí y motivarme a culminar.

A mi tutor de tesis MSc. Franklin Sánchez por su ayuda y tiempo dedicado a la culminación de este proyecto.

CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE CÓDIGOS

RESUMEN

En el proyecto actual de titulación se desarrolla una aplicación web prototipo, para escuchar radio en vivo y proyectar podcast de la radio Nuevo Tiempo en la tv usando un dispositivo Chromecast.

El desarrollo de este trabajo se basó en la metodología Kanban, una arquitectura modular, el entorno Node.js y MongoDB. En la capa de datos se emplea Mongoose, la capa de presentación comprende un cliente web y la capa de negocio es un conjunto de paquetes npm y funciones JavaScript utilizando Node.js. Esta aplicación recibe información de documentos en formato Excel, también recibe archivos en formato mp4 para ser almacenados desde una página realizada con HTML5.

En el capítulo uno se revisa la fundamentación teórica necesaria para el desarrollo del proyecto, como la metodología Kanban, AndroidDX, bases de datos no relacionales, Node.js, Heroku, Github, y Google Cast.

En el capítulo dos se diseña el prototipo de acuerdo con el análisis de requerimientos, se presentan los diagramas del prototipo, se describe la implementación del proyecto presentando la codificación de módulos del sistema. Se explica la implementación del servidor node.js, la implementación de la base de datos usa Mongoose, los módulos del prototipo web y las tecnologías usadas.

En el capítulo tres se presentan los resultados de acuerdo con las pruebas de funcionalidad y encuestas de usuarios.

Por último, en el capítulo cuatro se presentan las conclusiones y recomendaciones alcanzadas como resultado del análisis e implementación del presente plan de titulación.

PALABRAS CLAVE: Node.js, JavaScript, GitHub, Heroku, Mongoose, Google Cast

ABSTRACT

In the current degree project, a prototype web application is being developed to listen to live radio and project podcasts of the radio Nuevo Tiempo on TV using a Chromecast device.

The development of this work was based on the Kanban methodology, a modular architecture, the Node.js environment and MongoDB. Mongoose is used in the data layer; the presentation layer comprises a web client and the business layer is a set of npm packages and JavaScript functions using Node.js. This application receives information from documents in Excel format, it also receives files in mp4 format to be stored from a page made with HTML5.

In chapter one the theoretical foundation necessary for the development of the project is reviewed, such as the Kanban methodology, AndroidDX, non-relational databases, Node.js, Heroku, Github, and Google Cast.

In chapter two the prototype is designed according to the requirements analysis, the prototype diagrams are presented, the implementation of the project is described, presenting the coding of the system modules. The node.js server implementation is explained, the database implementation uses Mongoose, the web prototype modules and the technologies used.

In chapter three the results are presented according to the functionality tests and user surveys.

Finally, chapter four presents the conclusions and recommendations reached because of the analysis and implementation of this degree plan.

KEYWORDS: Node.js, JavaScript, GitHub, Heroku, Mongoose, Google Cast

1. INTRODUCCIÓN

En la actualidad las radios están migrando sus servicios de transmisión tradicional, a una difusión de su contenido por medio de internet, levantando su propio servicio con una aplicación móvil [1]. Al principio, las transmisiones de la radio Nuevo Tiempo se realizaban a través de satélites NSS806, que cubre las Tres Américas y Europa. Actualmente la TV Nuevo Tiempo también tiene el satélite IS 10 que envía la señal a Angola, Mozambique y Santo Tomé. Por medio de la señal de antenas se alcanza un promedio de 45 millones de personas, en cambio por medio del uso de Internet, se espera que la cobertura sea mundial. [2].

Desde un principio la aplicación tendría una utilidad implementada. Esta utilidad es la facultad de emitir contenido a un dispositivo Chromecast. Se escogió este dispositivo por la facilidad de su uso, ya que su configuración es muy sencilla, además de su bajo costo y la eficaz incorporación que tiene con los dispositivos móviles Android [3].

Por lo mencionado, el enfoque de este proyecto Técnico es desarrollar una aplicación para Android, que permita reproducir la transmisión de la radio en vivo, visualizar la programación y transmitir la señal en vivo de la radio “Nuevo Tiempo” a un dispositivo Chromecast.

1.1 OBJETIVOS

El objetivo general de este Plan Técnico es elaborar una aplicación prototipo en Android, para el Streaming y Reproducción de medios de una radio, e integración con Chromecast.

Los objetivos específicos del Proyecto Técnico son:

- Analizar los conceptos fundamentales necesarios para la elaboración de este proyecto de titulación.
- Diseñar los módulos de la aplicación considerando los requisitos que debe cumplir el prototipo.
- Implementar los módulos del sistema según el diseño realizado.
- Analizar los resultados obtenidos en las pruebas realizadas.

1.2 ALCANCE

Este trabajo de titulación propuso la elaboración de una aplicación modelo en Android, que permita reproducir la transmisión de la radio en vivo, visualizar la programación y transmitir la señal en vivo de la radio “Nuevo Tiempo” a un dispositivo Chromecast.

La radio difunde su programación durante todo el día las 24 horas del día, los 7 días de la semana.

Además, se previó que la aplicación brinde información de parrilla de programación de la radio y reproduzca la radio en vivo, la parrilla de programación está alojada en un servidor Node.js [7], y será accedida por medio de una API (Application Programming Interface).

Toda la información de la parrilla de programación de la radio se obtiene desde el servidor ya mencionado, a su vez el servidor usa una base de datos que fue implementada en MongoDB [8] La arquitectura de la aplicación se muestra en la **Figura** 1.1

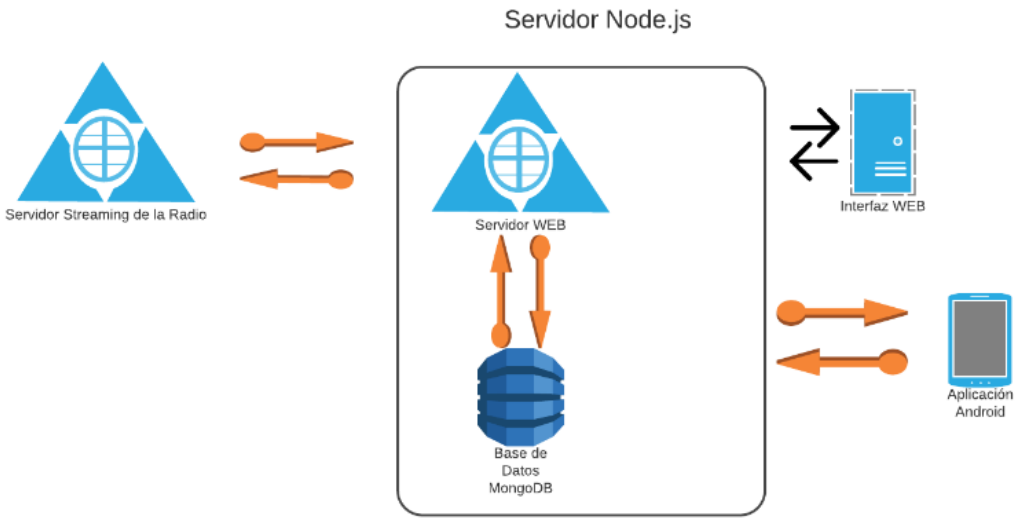


Figura 1.1 Arquitectura del sistema

La aplicación Android es la responsable de consumir los datos que el Servicio Web y el Servidor de Streaming de la Radio Nuevo Tiempo ofrecen, para poder mostrar los podcasts y escuchar la radio en vivo; y a través del servidor Web, enviar datos requeridos por la aplicación a la base de datos MongoDB.

Para actualizar la información de la aplicación Android es necesario contar con una cuenta de administrador que gestionará los videos que se mostrarán en la aplicación, este rol administrador tiene la posibilidad de crear y eliminar los videos podcast, así como, actualizar la parrilla de programación de la radio.

Para actualizar la parrilla de programación se tienen que definir ciertos parámetros de un formato predefinido como son: identificador, título del programa, descripción del programa, imagen de dicho programa; dichos parámetros son ingresados desde un documento Excel y enviados a la base de datos. Para crear o eliminar un podcast se lo realiza directamente en la base de datos a través de una interfaz web.

Al terminar el ingreso de algún podcast nuevo o la actualización de la parrilla de programación el administrador puede verificar la información en la aplicación móvil, o a su vez en el documento en formato JSON que se genera automáticamente al terminar el proceso mencionado.

El sistema fue implementado en un servidor web accesible desde Internet. El sistema está dividido en varios módulos que se especifican a continuación:

1. El primer módulo es el módulo Programación donde el usuario de la aplicación va a poder visualizar la parrilla mensual de programación de la radio, además, muestra información referente a la hora inicial y final de un programa en particular, y también una reseña de dicho programa.
2. El siguiente módulo es Radio en donde se muestra una interfaz con la facilidad para que el usuario pueda reproducir la emisora Nuevo Tiempo, en esta interfaz también se tiene la opción de enviar la trasmisión al dispositivo Chromecast.
3. El tercer módulo es Administración donde por medio de una interfaz web el administrador con sus credenciales, correo y contraseña, se puede entrar al servidor y enviar la parrilla de programación mensual de la radio en un formato predefinido en Excel, así mismo, la instauración, eliminación de podcasts en la base de datos.
4. El último módulo es Compartir que permite a los usuarios compartir el enlace de la radio en vivo a través de redes sociales incluyendo mensajería de WhatsApp.

1.3 MARCO TEÓRICO

Para la implementación de este trabajo de titulación se usaron varias herramientas tecnológicas que se describen a continuación.

1.3.1 NODE.JS

Para comprender cómo funciona Node.js, primero se deben comprender algunas características clave de JavaScript, que lo hacen adecuado para el desarrollo del lado del servidor. Este es un lenguaje simple y flexible que agrega interactividad a un sitio web.

JavaScript es un lenguaje de secuencias de comandos interpretados, es un lenguaje por naturaleza asíncrono. Es asíncrono porque se cargan los elementos de forma independiente. El entorno del servidor es el encargado de proporcionar acceso a todos los objetos necesarios para ejecutar un código JavaScript. Un ejemplo importante de JavaScript es la capacidad de agregar interactividad a un sitio web. Esto funciona porque el intérprete de código está integrado en un navegador web, por lo que no necesita un software adicional para esta tarea.

En el lado del cliente, se pueden agregar niveles de interactividad como por ejemplo responder a clics de los botones y validar el contenido de un formulario [4].

1.3.1.1 Características de Javascript

1.3.1.1.1 Métodos de JavaScript.

Una función incluye un conjunto de declaraciones. Las funciones son la unidad modular fundamental de JavaScript. Se utilizan para la reutilización de código, ocultación de información y la composición de bloques de código. Estas son usadas para especificar el comportamiento de los objetos, generalmente en programación se resume un conjunto de requisitos en un conjunto de funciones, datos y estructuras de código. Las funciones en JavaScript son objetos, y los objetos son colecciones de pares *nombre/valor.* Cada función tiene dos propiedades principales: un espacio de memoria pequeño que permitirá ejecutar dicha función y el conjunto de instrucciones que determinan la tarea a realizar por la función en cuestión, como se muestra en el Código 1.1

```
/* Función que convierte los ficheros xlsx de la parrilla a json para luego tratarlos */
function xlsToJSON(filename, res) {
    const path = 'public/xls/' + filename
    xlsxj({
        input: path,
        output: path.replace(".xlsx", ".json"),
        lowerCaseHeaders: true
    }, function (err, result) {
        if (err) {
            console.error(err)
            res.send({ saved: false, error: err });
        } else {
            JSONtoGrill(result, filename, res);
        }
```

Código 1.1 Ejemplo de declaración de funciones JavaScript

1.3.1.1.2. Sintaxis de JavaScript

Javascript es un lenguaje de escritura impreciso, por lo que los compiladores de JavaScript no pueden detectar errores de escritura de código esta característica puede ser alarmante para programadores que viene de otros lenguajes donde hay un intérprete que señala errores de escritura de código como Java. En JavaScript muchos de los errores no son los de escritura de código, pues hay una flexibilidad para escribir código; por ejemplo, no se necesitan formar jerarquías de clases complejas, o no hay que indagar con el tipo de datos de una variable para almacenar sin errores los datos adecuados, sino más bien errores que tiene que ver con el uso inadecuado de palabras clave, por ejemplo `null` que es un una palabra clave que se le asigna a una variable normalmente en otros lenguajes, pero en Javascript `null` es un objeto y no puede ser asignado a una variable.

Como se mencionó JavaScript fue ideado para ser usado en páginas web, Node.js es un entorno que facilita la ejecución de secuencias de comandos JavaScript.

El punto central de Node.js es la creación de aplicaciones de red, y como es un entorno de ejecución facilita la ejecución de JavaScript en el lado del servidor, lo cual era imposible para JavaScript por sí solo.

Node.js es orientado a eventos asíncronos, característica propia de JavaScript, es así como maneja multiprocesos, a diferencia de otros entornos modernos en donde se usan múltiples hilos para los diferentes procesos. El entorno Node.js maneja una cola para solicitudes bloqueantes, si llega una petición bloqueante al servidor node.js la coloca en una cola de espera, de esta manera mantiene su naturaleza no bloqueante. Por ejemplo, cuando se carga una página web y un cliente hace clic en un botón, esto desencadena una respuesta a esa llamada (callback), así el cliente podría seguir interactuando con la página web, ya que, la naturaleza de JavaScript permite registrar manejadores de eventos. A pesar de esto, Node.js incluye funciones síncronas, estas funciones son útiles cuando se realiza un proceso y desee esperar un tiempo antes de proceder a un siguiente proceso; por ejemplo, en el presente proyecto se usan funciones síncronas junto con asíncronas cuando se guarda un archivo multimedia a la base de datos, ya que este proceso toma un tiempo considerable para su completa ejecución. Diariamente node.js va mejorando la manera de manejar funciones asíncronas es por ello que se incluyen menos funciones de este tipo porque disminuyen el rendimiento del servidor, y en lugar de ello se usan promises o callbacks [5]. JavaScript junto a Node.js son usados en el lado del servidor de RNT (Radio

Nuevo Tiempo) que envía las parrillas de programación mensual al dispositivo Android. Más adelante se detallará las funciones adicionales de dicho servidor.

1.3.2. EXPRESS.JS

Para facilitar el desarrollo del servidor usando Node.js, ha sido creado el paquete Express.js. Dicho paquete es una infraestructura basada en el módulo HTTP que añade complementos que facilitan el trabajo a la hora de realizar peticiones al servidor. Express.js permite crear APIs (Application Programming Interfaces) y aplicaciones web, fácilmente provee un conjunto de características como manejo de rutas uso de motor de diferentes plantillas, etc.

Los principales métodos para manejar rutas con que aporta Express.js son los mismos que tiene HTTP, entre los más conocidos: GET, POST, PUT, y DELETE.

1.3.2.1. Funcionamiento de express.js

Express.Js es un paquete dependiente de la aplicación. Esto significa que cada proyecto que usa Express necesita los archivos dependientes de Express de forma local en el proyecto. Vamos a hacer una vista general de la estructura de una aplicación que usa Express [6].

1.3.2.1.1. Dependencia de terceros

La definición de la dependencia de terceros es como una importación de bibliotecas e incluye la librería en este caso de Express como se muestra en Código 1.2

```
const express = require('express');
```

Código 1.2 Ejemplo dependencia de librería Express

1.3.2.1.2. Instancias

Para usar el paquete Express se lo debe instanciar como se muestra en el Código 1.3

```
const app = express()
```

Código 1.3 Ejemplo de instanciación de Express

1.3.3.1.3 Ajustes de la librería express

La configuración del paquete Express consiste en establecer algunos valores por medio de `app.set()`, para ser usados por el servidor en el momento que este se inicia. Por ejemplo, si usamos un HTML tipo Pug, usamos 'view engine' para decirle a Express que debe buscar por archivos *.pug como se muestra en el Código 1.4

```
app.set('view engine', 'pug')
```

Código 1.4 Ejemplo de ajuste de paquete Express

1.3.3.1.4. Definición de middleware

Middleware es una función especial que permite una mejor organización y reutilización del código. Algunos Middleware son empaquetados como módulos de terceros (NPM) y solo se los descarga y se los usa como dependencia de terceros, otras veces, podemos codificar nuestros propios Middlewares, en ambos casos la sintaxis en `app.use ()` es como se muestra en el Código 1.5

```
app.use(cookieParser());
```

Código 1.5 Ejemplo de uso del middleware cookieParser

1.3.3.1.5. Definición de rutas

Las rutas son puntos de acceso (endpoints) donde el cliente puede acceder para comunicarse con el servidor. La sintaxis para declarar una ruta: empleamos router.VERBO(), donde VERBO () es un método HTTP como `GET, POST, DELETE, PUT, OPTIONS` o `PATCH`. Por ejemplo, logramos definir las rutas de la página de inicio de nuestro servidor RNT como se observa en el Código 1.6

```
router.get('/', auth, function (req, res, next) {
    res.render('admin.pug', { title: 'Radio Nuevo Tiempo' });
})
```

Código 1.6 Ejemplo de ruta GET para el paquete Express

1.3.3 POSTMAN

Postman es una plataforma para realizar pruebas de API (Application Programming Interface). Postman permite, en pocas palabras, suplantar a un cliente para probar la funcionalidad del servidor. Postman permite realizar peticiones HTTP como GET, DELETE, POST, etc., con la finalidad de monitorizar, automatizar y depurar una API.

Figura 1.2 Llamada al endpoint localhost:3000

Como observamos en la Figura 1.2 Postman posibilita realizar pruebas a los "*endpoints*" del servidor que se encuentra en desarrollo. Estas pruebas pueden ser guardadas para ser usadas en caso de ser necesario para futuras correcciones. De esta forma se mantiene un control de cambios hechos en el servidor de manera sencilla, ahorrándonos así implementar un cliente para el efecto.

Por último, se destaca la posibilidad que ofrece de trabajar en red con otros colaboradores de desarrollo, y compartir con ellos si fuere necesario las pruebas realizadas.

1.3.4 npm

Cada plataforma debe enfrentar problemas de modularización y redistribución de código, es decir, la reutilización de código escrito en diferentes programas. Este es un problema que tiene Java y que se lo solucionó con Maven, o Ruby que se lo solucionó con Gems, así también fue un problema de Node.js que se lo solucionó con npm (Node Package Manager). Alguien que desee usar librerías, o lo que se conoce como paquetes o módulos Node en Node.js, debería tener la posibilidad de instalar dichos paquetes, aquí es donde npm entra en acción.

npm es un gestor de paquetes por defecto para Node.js, es una biblioteca de paquetes creados por la comunidad de Node.js que ayudan para el desarrollo de aplicaciones.

npm permite registrar paquetes con un nombre tal que, puedan ser importados o exportados usando un nombre de registro. Este registro npm es un servicio web que almacena o que actúa como host de dichos paquetes. El registro npm de dichos paquetes se encuentran en el host *https://npmjs.org*. Este host es accesible a través de internet y mantenido por Joyent,

organización sponsor de node.js. Por ejemplo, el paquete express que mencionamos anteriormente en esta sección, esta publicado en el host con el nombre de registro *http://npmjs.org/package/express*. Esta manera de distribución de código o librerías en internet para desarrolladores es la razón por la cual npm existe.

Una parte integral de npm es un archivo llamado package.json. Este archivo es de vital importancia para usar los paquetes npm creados por la comunidad Node.js. Para poder crear este archivo, npm cuenta con dos comandos principales: el primero `npm init` el cual crea el archivo package.json, y el segundo `npm install <nombre del paquete>` que sirve para instalar en un proyecto el paquete npm[14].

Posteriormente, se describen los paquetes primordiales usados en la creación del servidor.

1.3.4.1. Iconv-Lite

Es un paquete usado para convertir diferentes tipos de codificaciones usando JavaScript. Por ejemplo, se puede usar para transformar un “String” a iso-8859-1 de la familia ISO (International Organization for Standardization) que define caracteres especiales y la letra ñ. Soporta gran variedad de codificaciones, además decodifica/codifica a partir de un Streaming de datos, ya sean de un servidor o de un archivo local, para luego ser almacenados en un nuevo archivo [15].

1.3.4.2. Multer

Es un paquete middleware que sirve para el manejo de datos provenientes de un formulario, se utiliza principalmente para cargar archivos. Multer permite personalizar varias opciones, por ejemplo, `dest` que le dice a Multer donde cargar los archivos, en caso de omitir esta función los archivos se guardarán en memoria y nunca se escribirán en el disco. Por defecto Multer cambiará el nombre de los archivos para evitar conflictos de nombres, esta opción puede también ser personalizada [16].

1.3.4.3. Nodemon

Es una herramienta que ayuda a desarrollar aplicaciones basadas en Node.js, al reiniciar automáticamente la aplicación en desarrollo cuando se detectan cambios en el código de dicha aplicación.

Nodemon tiene algunas opciones configurables, una importante es `ignore` , por defecto Nodemon reinicia todo un directorio/archivo, pero a veces se tienen archivos que se

requiere que no se reinicien antes de verificar otros documentos es aquí donde `ignore` ayuda a controlar este reinicio [17] .

1.3.4.4 Pugjs

Es un motor de plantillas que es implementado con JavaScript para Node.js, se destaca por su facilidad para integrar funciones Javascript y también su facilidad para realizar tareas complejas con simples líneas de código. Las plantillas con extensión pug pueden trabajar en conjunto con plantillas HTML, lo que permite que sea compatible con cualquier tipo de navegador.

1.3.4.5. Mongoose

Paquete que se usa para modelar los objetos MongoDB, de modo que funcionen de manera asíncrona. Esta herramienta se usa en este proyecto para almacenar los podcasts, por su flexibilidad y su sencillez en instanciar y guardar en la base de datos [18] .

1.3.4.6. Xlsx-to-Json

Por último, el paquete usado en la creación del servidor es xlsx-to-json. Este paquete transforma un formato Excel a un documento Json, mapeando columnas de hojas a claves de objetos. Este paquete permite configurar la clave del objeto a ser establecida a partir de la celda de la hoja de Excel, es así como, en el presente proyecto se usó este paquete para transformar la parrilla de programación de la radio Nuevo Tiempo.

1.3.10 BASE DE DATOS

1.3.10.1. SQL

Un sistema gestor de base de datos (SGBD) es un conglomerado de programas relacionados entre que permiten el acceso y la manipulación de datos en cuestión. El lenguaje usado para un sistema SGBD es SQL (Sequel Query Language). Estos sistemas son adecuados para datos estructurados almacenados en comunas y filas, que se pueden acceder usando SQL. Estos sistemas también se basan en el concepto de transacciones ACID siglas de atómicas, consistentes, aisladas y duraderas, donde:

- **Atómica** implica que todos los cambios de una transacción se aplican por completo o no se aplican en absoluto.
- **Consistentes** significa que después de que se da una transacción los datos se mantienen en consistencia, es decir al realizar una consulta de un dato en particular el resultado será el mismo de antes de la transacción.

- **Aislado** significa que las transacciones que se aplican al mismo conjunto de datos son independientes entre sí. Por lo tanto, una transacción no interfiere con otra.
- **Durable** significa que los cambios son permanentes en este sistema y que no se perderán debido a alguna falla.

1.3.10.2. NoSQL

NoSQL es un término que se usa para base de datos no relacionales, es decir no se basan en los principios de un sistema SGBD, estas bases de datos en su mayoría se usan para manejar grandes conjuntos de datos como por ejemplo cuando hablamos de Big data, o cuando hablamos de compras en internet donde un usuario entra a una tienda online y desea un producto en particular que otro usuario también requiere, aquí un sistema SGBD bloquearía parte de la base de datos en donde se aloja dicho producto hasta que el primer usuario realice la compra, mientras el segundo usuario tendría que esperar, por lo cual la experiencia de usuario se deteriora en este escenario bajo el sistema SGBD, una alternativa para la solución de este problema es una base de datos distribuida o NoSQL. Una definición como tal para el conjunto de bases de datos NoSQL no existe, pero si se la define sería una base de datos que no sigue los principios de un sistema SGBD [19].

En contraste con el enfoque ACID que acabamos de mencionar de los sistemas SGBD, NoSQL tiene un enfoque que se conoce como BASE. Antes de abordar este concepto debemos conocer el teorema cap.

1.3.10.2.1. Cap

Es un teorema descrito por Eric Brewer en el 2000, en donde se plantea que al diseñar una aplicación distribuida existen tres requisitos básicos a saber: consistencia, disponibilidad y tolerancia de partición [19].

- **Consistencia** significa que los datos permanecen consistentes después que se realiza cualquier operación o cambios sobre los mismos, y que cualquier usuario que accede a la aplicación puede ver los mismos datos actualizados.
- **Disponibilidad** significa que el sistema siempre está disponible
- **Tolerancia de partición** significa que el sistema continúa funcionando en escenarios donde ciertos servidores no pueden comunicarse entre ellos.

1.3.10.2.2. Base

Este acrónimo se define de la siguiente manera:

- **Básicamente Disponible (Basically Available)** significa que el sistema estará disponible en términos del teorema cap
- **Estado Suave** indica que incluso si no hay alguna entrada al sistema este cambia con el tiempo. Esto va de la mano con la siguiente definición (Consistencia Eventual).
- **Consistencia Eventual** significa que el sistema será consistente al largo plazo, siempre que no haya alguna entrada al sistema en ese tiempo.

En la Tabla 1.1 se muestra una comparación entre ambos sistemas SQL y NoSQL

Tabla 1.1 Comparación SQL y NoSQL [19]

	Base de datos SQL	**Base de datos NoSQL**
Tipos	Todo tipo que soporta estándar SQL	Existen varios tipos, como almacenamiento de documentos, valores de claves, columnas, bases de datos, etc
Ejemplos	SQL server, Oracle, MySQL.	MongoDB, Hbase, Cassandra
Modelo de almacenamiento de datos	Se almacenan en filas y columnas en una tabla donde cada columna es un tipo específico.	El modelo de datos depende de la base de datos, es decir si los valores son valores-clave entonces se almacenan de esta manera, por otro lado, si son tipo documentos se almacenan como tal.
Esquemas	Esquema fijo por lo que cualquier cambio de	Esquema dinámico, las estructuras pueden ser modificadas por dos

	esquema cambia la base de datos	maneras, expansión de la base o alteración de la misma. Se pueden agregar campos de forma dinámica.
Escalabilidad	Es un enfoque de ampliación, es decir entre más volumen de datos, más costoso el servidor para los mismos.	Enfoque de escalamiento horizontal, es decir, a medida que aumenta el volumen de datos se distribuye la carga en servidores de bajo costo.
Consistencia	Consistencia fuerte.	Depende del dato, algunos son de consistencia fuerte otras de consistencia eventual.
Capacidad de Consulta	Disponibles a través de una GUI fácil de usar.	La consulta puede requerir de conocimientos de programación. El enfoque se centra en la funcionalidad y la interfaz de programación.

1.3.10.2.3 Mongodb

Es una base datos NoSQL que almacena datos en formato BJSON (Json Binarios). Los datos no se almacenan en forma de columnas y filas en tablas, sino en documentos, los mismos que pueden tener diferentes esquemas, significa que puede cambiar a medida que la aplicación evoluciona. MongoDB está pensado para ser escalable, de alto rendimiento y alta disponibilidad.

MongoDB tiene todas las características mencionadas en la sección anterior. Una característica adicional propia de MongoDB es que usa documentos en formato JSON binario (BJSON). Esta fue una característica por la cual se usó esta base de datos para el desarrollo de este proyecto, debido a que Android descarga de manera simple archivos JSON para su respectivo tratamiento. Otra característica por la cual se optó por usar MongoDB es su escalabilidad, en este proyecto se almacenan archivos multimedia

pesados, que MongoDB no tiene problema en guardar. Finalmente, la última característica por la cual se optó por MongoDB, fue su compatibilidad con Node.js.

1.3.5 GOOGLE CAST

Este es uno de los recursos más importantes usados en el desarrollo de la aplicación. En la Figura 1.3 se muestra la consola de desarrollador, esta es la página donde se debe registrar el dispositivo Chromecast para poder acceder a la aplicación web Receiver antes de ser publicada. Una vez que se publique la aplicación estará disponible para todos los dispositivos Chromecast.

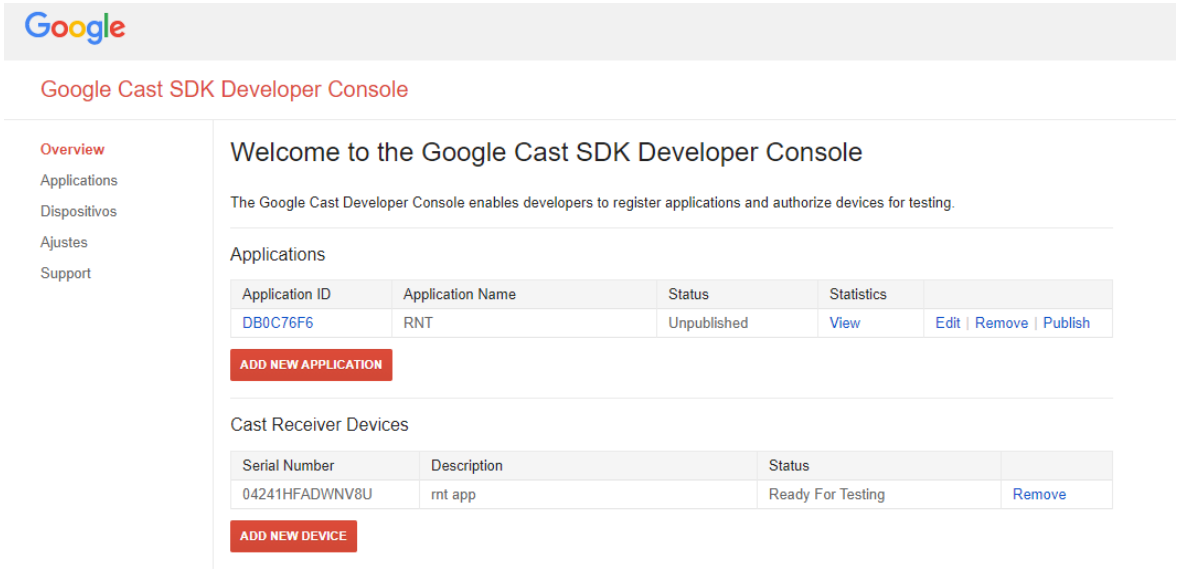


Figura 1.3 Google Cast SDK

Para completar con el proceso de publicación de la aplicación para dispositivos Chromecast de debe tener una cuenta de correo Gmail, y pagar una tasa de 5$. Una vez completado este proceso la aplicación estará disponible en el sitio chromecast.com/apps, donde los usuarios pueden descargarla.

Para instaurar una aplicación que sea funcional en un dispositivo Chromecast se tiene que pulsar el botón agregar que se muestra en la Figura 1.3, esta acción permite configurar, por ejemplo, el tipo de pantalla a mostrar en el dispositivo Chromecast, puede ser con un estilo personalizado o simplemente por defecto; también permite configurar nombre, apariencia, etc.

1.3.6. HEROKU

Heroku es una solución basada en Plataform as a Service (PaaS), PaaS simplifica el desarrollo de software. Bajo este modelo de servicio se construye un sistema usando la infraestructura (servidores, almacenamiento, redes), un middleware, herramientas de desarrollo, servicios de inteligencia, etc. de un proveedor de servicios en la nube. PasS elimina el gasto y la complejidad que suponen la compra y la administración de equipos hardware y licencias de software entre muchos otros más.

El diseño de la plataforma Heroku está estructurado para ejecutar aplicaciones implementadas en cualquier lenguaje de programación, esto es gracias a que maneja una capa de abstracción llamada `buildpacks`, que son bibliotecas que trasforman el código ingresado en la nube Heroku en pequeñas piezas de código para luego ser distribuidos horizontalmente en la arquitectura de Heroku. Al día de hoy Heroku tiene varios buildpacks que soportan las siguientes tecnologías: Ruby, Python, Node.js, Grails, Java, Scala, Clojure, Gradle, Play [21].

Un aspecto importante en Heroku es su fase de despliegue, la cual es sencilla e intuitiva, esto facilita el trabajo del desarrollador de una aplicación en su fase de prueba, así como su fase de producción. Heroku maneja un repositorio Git con el código de la aplicación. Cada vez que un desarrollador añade código en el repositorio remoto se inicia un proceso que se conoce como *despliegue de aplicación*. Heroku divide los procesos de una aplicación, así asigna procesos con tareas singulares. Todos estos procesos son creados las veces necesarias para adaptarse a las solicitudes, esto se conoce como escalamiento horizontal [22].

En este proyecto se usó la herramienta Git para enviar el código a un repositorio de código GitHub mencionado anteriormente, para luego enviarlo al servidor Heroku. Para este proceso se usó el Código 1.7 que se muestra a continuación:

```
git init
git add .
git commit -m "commit inicial"

git remote add origin https://github.com/vichuco/test.git
git branch -M main
git push -u origin main

heroku config
git push heroku main
```

Código 1.7 Comandos para despliegue del servidor

Cabe mencionar que para usar Heroku se debe crear una cuenta en GitHub y en Heroku.

En el Código 1.7 se observa que las seis primeras líneas de código son las encargadas de permitirle a la plataforma Heroku que use un control de versiones en el proyecto a través de Git, código que estará alojado en GitHub, la cual va a interactuar con Heroku para su correcto funcionamiento. En las dos últimas líneas se sube el proyecto a la plataforma Heroku, lo cual nos mostrará por consola que se ha realizado con éxito, así como, la dirección url por la cual podremos acceder a nuestro proyecto indefinidamente.

1.3.7 REST

REST (Representational State Transfer) es un tipo de arquitectura basado en un conjunto de principios para diseñar arquitecturas de software basadas en la red. A continuación, se describen los 4 principios para el estilo de arquitectura REST

1.3.7.1 Identificación de los Recursos

Los componentes básicos de la web se denominan recursos. Un recurso es cualquier cosa que pueda ser nombrado como un objeto de hipertexto (por ejemplo, un archivo, un script, una colección de recursos), cada recurso se identifica con un URI (Uniform Resource Identifier) como se muestra en la Figura 1.4

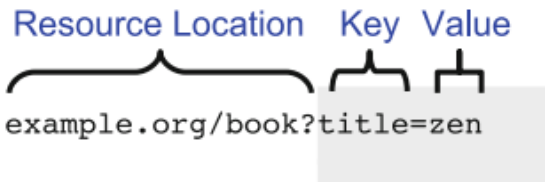


Figura 1.4 URI de un recurso **[23]**

En una arquitectura REST se puede acceder a los recursos de un servidor usando una URI. Las peticiones al servidor se hacen a través de métodos HTTP como son `POST`, `GET`, `PUT` y `DELETE`. En la Figura 1.4 se observa que se realizan peticiones en la ubicación example.org/book y la búsqueda en esta localidad es de tipo llave=valor. En este caso, se está buscando un libro de título zen; por el estilo del recurso solo acepta peticiones de lectura del recurso, por tanto, el método que se puede aplicar en este recurso es `GET`.

1.3.7.2 Representaciones

En respuesta a una solicitud de un recurso, el cliente recibe una representación de dicho recurso, en un formato diferente a la representación del mismo recurso en el servidor. Los

recursos admiten múltiples representaciones y pueden producir respuestas en diferentes formatos de datos. En http, el cliente especifica su formato en la cabecera de la petición HTTP, y al menos dos formatos son comúnmente usados XML y JSON.

1.3.7.3 Mensajes auto descriptivos

Los recursos se manipulan a través de mensajes que son estándar, es decir auto descriptivos en la web; estos mensajes son los métodos HTTP. REST usa mensajes auto descriptivos o estándar para que equipos intermediarios como proxis, Gateway, etc., también puedan entender y ayudar al sistema en su intercambio mensajes. Aunque HTTP/1.1 define ocho métodos, solo dos de ellos, `GET` y `POST` se han usado ampliamente en la web, en parte porque estos eran los únicos métodos admitidos por los primeros navegadores web.

1.3.7.4 Hateoas

Como último principio de una arquitectura de estilo REST tenemos HATEOAS (Hypermedia as the engine of application state), estas siglas significan que ni el cliente ni el servidor necesitan mantener el estado de un intercambio en una sesión, porque toda la información necesaria se almacena en los mensajes HTTP (en el encabezado y cuerpo HTTP). La definición de enlaces autónomos es fundamental para servicios web RESTful, porque estos enlaces permiten recorrer, descubrir y conectarse a otros servicios y aplicaciones.

1.3.8 ANDROIDX

Androidx es una mejora a las bibliotecas de compatibilidad de Android original [24]. Es decir, cuando se desarrolla aplicaciones que permitan varias versiones de API (Application Programming Interface), las bibliotecas de compatibilidad es un modelo para proporcionar nuevas funciones a referencias previas de Android o recurrir a funciones iguales. Además, las bibliotecas de compatibilidad tienen clases con propiedades que no están hábiles en la API del ámbito de trabajo, que facilitan la creación y la consistencia en más dispositivos, como por ejemplo `Fragment,` que posibilitan utilizar fragmentos en sistemas que ejecutan referencias anteriores a Android 3.0 (API nivel 11). Algunas funciones de Androidx son:

- Todos los paquetes de Androidx están alojados en un espacio de nombres coherente que comienza con `Androidx.`

- A diferencia de la biblioteca de compatibilidad común, los paquetes Androidx se mantienen actualizados por separado, y usan un control semántico de versiones a partir de la versión 1.0.0.

1.3.9 KANBAN

David Anderson fue el precursor en la fundación de conceptos para el uso de Kanban visto desde el desarrollo de software, en 2004 con la pauta de Reinertsen usó el tablero Kanban en un proyecto TI de la empresa Microsoft. El objetivo principal en Kanban es establecer tareas por hacer y modificar su preeminencia de acuerdo al desarrollo de las actividades, también con Kanban se visualiza para todo el equipo la línea de trabajo, y en el supuesto de que exista algún retraso es más sencillo identificarlo, estos preámbulos fueron tomados en cuenta por Toyota y hoy por hoy son de gran ayuda en el desarrollo de software [25].

Normas

Las tres principales normas de Kanban son las siguientes:

1. **Plasmar el flujo de trabajo.**

   Segmentar el trabajo por hacer en partes, describirlas en unas llamadas tarjetas del tablero y ubicarlas en la columna correspondiente, estas columnas o grupos de tarjetas pueden ser tantas como el trabajo lo amerite. El punto clave de esta norma es que el trabajo a ser hecho este claro para todos los miembros, y que cada miembro tenga una actividad asignada en el trabajo respetando la precedencia de cada tarea.

2. **Establecer el límite del WIP (Trabajo en Curso).**

   Definir un límite de tareas que se puedan realizar en cada columna del tablero, sea cual sea la extensión del trabajo se puede establecer un límite adecuado. La finalidad de esta norma es detectar los obstáculos y encontrar la solución, una de las soluciones puede ser la participación de los miembros del equipo de trabajo en la aceptación de nuevas tareas.

3. **Regular el tiempo para finalizar una tarea (Lead time).**

   Este tiempo está contemplado desde que inicia la solicitud hasta la finalización del proyecto, en otras palabras, este índice mide el rendimiento del proceso. Son importantes estas métricas para la regulación y mejora progresiva. Si se anhela calcular

el Rendimiento de trabajo traducido como la cantidad de tareas que un equipo puede realizar se debería dividir entre el Tiempo de ciclo y el WIP.

2. METODOLOGÍA

En este apartado se detallan 3 procesos fundamentales: diseño del sistema, codificación de los módulos y el despliegue del prototipo.

2.1. DISEÑO DEL PROTOTIPO

2.1.1 HISTORIAS DE USUARIO

Las historias de usuario son usadas para registrar, desde la perspectiva del usuario, una funcionalidad o una nueva capacidad que tendrá el sistema. Los campos que componen las historias son [20]:

- ID: identidad de la historia de usuario, empieza con HU (Historia Usuario) continuo de un número con patrón de dos dígitos
- Rol: indica el usuario al que está dirigida la funcionalidad
- Criterios de aceptación: indica la manera en el que el usuario confirma que se ha cumplido el objetivo de la historia de usuario
- Nombre de historia de usuario: título representativo de la historia de usuario
- Descripción: detalla el requerimiento del usuario
- Prioridad: indica la precedencia que figura la historia de usuario

El formato de las historias de usuario se muestra en la Tabla 2.1

Tabla 2.1 Formato de la historia de usuario

Historia de Usuario		
ID:	**PRIORIDAD:**	**ROL:**
NOMBRE		
DESCRIPCIÓN:		
CRITERIOS DE ACEPTACIÓN:		

2.1.1.1 Detalle de historias de usuario

A continuación, se presentan las Historias de usuario (HU), en concordancia con el levantamiento de requisitos. Se presentan ejemplos de HU más representativas de cada tipo de usuario, en el Anexo 1 se encuentran todas las HU realizadas.

En la Tabla 2.2 se presenta un ejemplo de HU para un usuario *administrador*.

Tabla 2.2 Historia de usuario inicio de sesión

Historia de Usuario		
ID: HU01	**PRIORIDAD:** Alta	**ROL:** *Administrador*
NOMBRE: Inicio sesión		
DESCRIPCIÓN: El *administrador* para iniciar sesión debe ingresar su correo electrónico y su contraseña		
CRITERIOS DE ACEPTACIÓN: El sistema permite autenticarse y acceder al perfil de *administrador*		

En la Tabla 2.3 se presenta un ejemplo de Historia de Usuario para un usuario *cliente*:

Tabla 2.3 Historia de usuario visualización de contenido

Historia de Usuario		
ID: HU13	**PRIORIDAD:** Alta	**ROL:** Usuario
NOMBRE: Visualización del contenido del podcast		
DESCRIPCIÓN: El usuario *cliente Android* tendrá la posibilidad de observar todo el contenido de los podcasts a través, de una opción para este efecto.		
CRITERIOS DE ACEPTACIÓN: El sistema permite los podcasts a través de una ruta		

Las historias de usuarios resumidas se especifican en la Tabla 2.4

Tabla 2.4 Historias de usuario

ID	Título	Descripción
HU01	Inicio sesión	El *administrador* para iniciar sesión debe ingresar su correo electrónico y su contraseña
HU02	Registro	El *administrador* para iniciar sesión debe ingresar su correo electrónico y su contraseña
HU03	Restablecimiento de cuenta de *administrador*	El *administrador* podrá recuperar su cuenta usando el método `post`, para actualizar ya sea el correo o la contraseña
HU04	Visualización de la página principal	El *administrador* al iniciar sesión visualizará la pantalla de la página principal, a partir de esta tendrá acceso a los menús y botones de acción
HU05	Administración de podcasts	El *administrador* podrá subir y eliminar podcasts del servidor
HU06	Estado de los podcasts	Los podcasts por defecto están inactivos, pero se los activará haciendo clic sobre el video.
HU07	Visualización de los podcasts	El *administrador* podrá visualizar los videos en la misma página donde se los sube.
HU08	Administración de la parrilla de programación	El *administrador* podrá elegir un archivo para subir en la ruta `upload` del servidor.
HU09	Comprobación del formato del archivo de la parrilla de programación	El *administrador* deberá ubicar la información de la programación en un formato predefinido en una plantilla Excel

HU10	Notificación de archivo subido con éxito	El *administrador*, después de subir el archivo Excel recibirá como respuesta una página donde se le notificará que el archivo ha sido exitosamente subido.
HU11	Salir del sistema	El *administrador* tendrá la opción de salir de todo el sistema usando un botón para esta acción
HU12	Visualización de guía para la API	El *administrador* en caso de requerir información del uso de `get` y `post` y el nombre de las rutas a las cuales debe acceder para obtener o enviar información hacia el servidor lo podrá realizar a través de una guía en la página principal.
HU13	Visualización del contenido de los podcasts	El usuario *cliente Android* tendrá la posibilidad de observar todo el contenido de los podcasts a través, de una opción para este efecto.
HU14	Visualización de página principal de la aplicación	El usuario cliente Android, apenas entre en la aplicación visualizará la radio donde podrá reproducir la misma
HU15	Envió de contenido multimedia	El usuario *cliente Android* podrá elegir entre dos opciones, reproducir localmente el video o audio, o a su vez, reproducir en un dispositivo Chromecast presente en su red.
HU16	Configuración de la aplicación	El usuario *cliente Android* podrá configurar el mando del volumen

		si bien la aplicación o la tv a través del Chromecast.
HU17	Compartir aplicación	El usuario *cliente Android* podrá compartir la aplicación a través, de sus redes sociales.
HU18	Parrilla mensual	El usuario *cliente Android*, tendrá la opción en el menú principal para visualizar la programación mensual de la radio, el título del programa, hora y una descripción.

2.1.2. REQUERIMIENTOS DE USUARIO

Además de la redacción de las historias de usuario, para el análisis de requerimientos del presente proyecto se efectuó una encuesta a 5 usuarios de la radio Nuevo Tiempo. La encuesta realizada consta de diez preguntas abiertas, las primeras tres preguntas contienen información sobre; nombre del entrevistado, rol que desempeña y su responsabilidad dentro del equipo de radio. Las cuatro preguntas siguientes permiten abordar la problemática que se busca solucionar con el desarrollo del prototipo, además de las necesidades que debe satisfacer la aplicación. Finalmente, las ultimas preguntas están dirigidas a averiguar si se ha intentado realizar un software similar, definición de usuarios que van a usar la aplicación y la limitante que estos pudieran tener en el uso de la aplicación. Un resumen de la encuesta se muestra en la Tabla 2.5. La encuesta se encuentra en el Anexo 2 y ayudó a determinar los siguientes requerimientos:

- Permitir al *Administrador* crear y eliminar podcasts del servidor RNT
- Permitir al usuario *cliente Android* compartir desde el aplicativo por cualquier red social la página de la radio Nuevo Tiempo.
- Permitir a cualquier usuario a través del aplicativo enviar contenido multimedia a un dispositivo Chromecast
- Permitir al Administrador la creación, modificación y visualización de la parrilla de programación.

Tabla 2.5 Resumen de Encuestas de requerimientos

Encuesta para determinar requerimientos de usuario		
Pregunta	**Respuesta**	**Porcentaje**
¿Desearía usted conocer la programación de la radio en el aplicativo móvil?	Si	100
	No	0
¿Desearía usted que la aplicación móvil tenga una opción para enviar los podcasts de la radio a su Smart tv usando un dispositivo Chromecast?	Si	100
	No	0
¿Cree usted útil compartir la aplicación con sus amigos a través de redes sociales?	Si	100
	no	0

2.1.2.1 Lista de requerimientos

La lista de requerimientos se definió basándose en las historias de usuario y la encuesta realizada.

2.1.2.1.1 Requerimientos funcionales

Los requerimientos funcionales se definieron en base a las encuestas y las historias de usuarios. A continuación, se detallan los requerimientos funcionales:

- Existirán dos perfiles usuario radio escuchas y *administrador*
- Se le mostrará al usuario el horario de su programa favorito
- Se le mostrará al usuario los podcasts subidos a través del servidor
- Se le mostrará al usuario una descripción del programa que elija
- Se le permitirá compartir con sus amigos la aplicación móvil
- El usuario tendrá la opción de escuchar la radio en vivo

- El sistema permitirá subir una programación mensual de la radio
- El usuario podrá subir/bajar el volumen de la radio usando botones para el efecto
- La radio se seguirá reproduciendo así sea que el usuario salga de la aplicación.
- El sistema mostrará todos los perfiles de *administrador* creador, a través de una ruta para el efecto
- El sistema tendrá la posibilidad de eliminar cuentas de *administrador*.
- Los usuarios podrán saber que el contenido multimedia se está reproduciendo en algún dispositivo Chromecast.
- EL dispositivo Chromecast mostrará el nombre de la radio en la pantalla principal de la televisión.
- Las reproducciones de los archivos multimedia podrán ser reproducidos o pausados, ya sea a través, de la aplicación o del dispositivo Chromecast.

2.1.2.1.2 Requerimientos no funcionales

Los requerimientos no funcionales se refieren principalmente a aspectos de comportamiento de un sistema y las restricciones externas en las cuales debe operar. Los atributos como pueden ser: interfaces de usuario, seguridad, flexibilidad y desempeño que forman parte del proyecto de titulación. A continuación, se detallan los requerimientos no funcionales:

- La contraseña se cifrará antes de almacenarla en la base de datos
- Todos los podcasts estarán almacenados en la base de datos no relacional
- Para el ingreso al sistema se utilizará un nombre de usuario y contraseña
- No se podrá registrar usuarios *administrador*es con cuentas duplicadas
- La aplicación está diseñada para funcionar en sistemas Android.

- La cuenta de administradores se almacenará en una base de datos no relacional.
- Para enviar el documento que contiene la información de la parrilla de programación hacia el servidor, se permitirá únicamente el documento con el formato especificado en la página de administración.

2.1.3 MÓDULOS

Los módulos se definieron en base a las funcionalidades ya mencionadas en las historias de usuarios y requerimientos del sistema. A continuación, se detallan los módulos de la aplicación:

- Programación: permite al usuario visualizar la parrilla de programación mensual de la radio.
- Radio: Es el módulo que se muestra al usuario apenas ingresa a la aplicación, en este módulo el usuario puede escuchar la radio en vivo, tendrá los controles de volumen y también la opción de transmitir el contenido multimedia a un dispositivo Chromecast.
- Administración: permite al *administrador* de la aplicación subir podcasts y la parrilla mensual de programación de la radio en un formato Excel prestablecido.
- Compartir: permite a los usuarios de la aplicación compartir a través, de redes sociales la aplicación de la radio.

En la Tabla 2.6 se indican las historias de usuario correspondiente a cada módulo.

Tabla 2.6 Módulos del sistema

Módulos del sistema	ID	Título
Administración	HU01	Registro
	HU02	Inicio sesión

	HU03	Restablecimiento de cuenta de *administrador*
	HU04	Visualización de la página principal
	HU05	Administración de podcasts
	HU06	Estado de los podcasts
	HU07	Visualización de los podcasts
	HU08	Administración de la parrilla de programación
	HU09	Comprobación del formato del archivo de la parrilla de programación
	HU10	Notificación de archivo subido con éxito
	HU11	Salir del sistema
	HU12	Visualización de guía para la api (*application programming interface)*
Radio	HU13	Visualización del contenido de los podcasts
	HU14	Visualización de página principal de la aplicación
	HU15	Envió de contenido multimedia

	HU16	Configuración de la aplicación
Compartir	HU17	Compartir aplicación
Programación	HU18	Parrilla mensual

2.1.4 TABLERO KANBAN

Para desarrollar el prototipo se ha definido el tablero Kanban dividiendo el trabajo en 3 columnas: tareas por hacer, tareas en progreso y tareas completadas.

La primera columna contiene tareas del diseño, implementación y pruebas del prototipo. Las tareas de diseño se las realizarán en un tiempo estimado de 4 semanas. Las tareas por hacer tanto en el servidor como en la aplicación se detallan a continuación: Entrevista con usuarios de la radio, definición de requerimientos, diseño de la capa de datos, diseño de la capa de negocio, generación de diagrama de clases.

Al iniciar el desarrollo de estas actividades pasarán a la columna “En progreso” y se las irán moviendo a la columna “Completado” conforme se finalicen. Figura 2.1 muestra la organización del tablero Kanban para el inicio del proyecto.

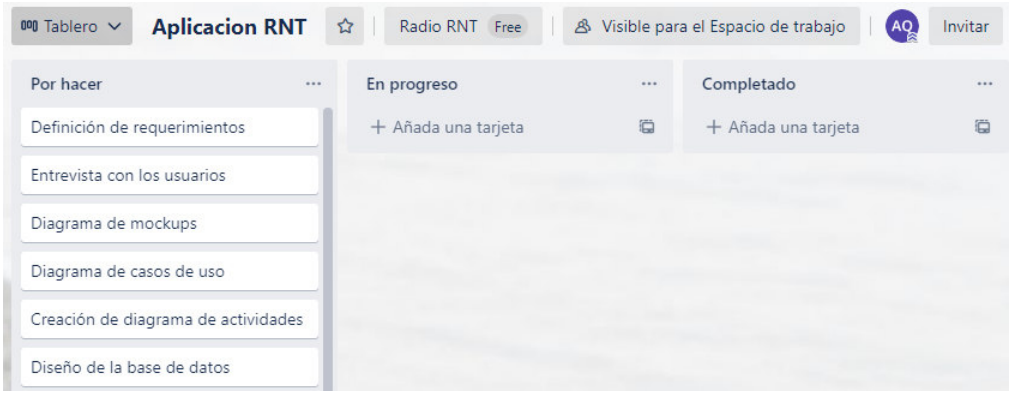


Figura 2.1 Planificación tablero Kanban

2.1.5 DIAGRAMAS UML

A continuación, se presentan los diagramas de caso de uso de los módulos que componen el sistema.

2.1.5.1 Diagramas de caso de uso

Un diagrama de caso de uso describe la funcionalidad que se espera del sistema. Abarca una serie de funciones que se ejecutan cuando se usa dicho sistema [26].

En este sistema se han identificado dos actores, que interactúan con la aplicación RNT, estos son: usuario *Cliente Android* y usuario *Administrador*. El primero usará el modelo de la aplicación móvil con el fin de escuchar la radio y visualizar los podcasts de la radio nuevo tiempo. El segundo actor empleará la interfaz de administración con la finalidad de administrar los podcasts y la parrilla de programación mensual que se mostrarán en la aplicación. El diagrama de casos de uso se realizará por módulos. El diagrama de casos de uso del módulo Administración se muestra en la Figura 2.2 El diagrama de casos de uso del módulo Radio cuenta con el actor cliente Android. En la Figura 2.3 se muestra que el *cliente Android* interactúa con el servidor de Streaming usando la aplicación móvil, en la que se encuentran los controladores para reproducir y pausar la señal de la radio en vivo.

El diagrama de casos de uso del módulo Programación cuenta con el actor *cliente Android*. En la Figura 2.4 se muestra que el *cliente Android* interactúa con el sistema para visualizar toda la información como: el contenido, tipo de dato, calidad, formato, etc. de cada uno de los programas de la radio.

El diagrama de casos de uso del módulo Compartir cuenta con el actor *cliente Android*. En la Figura 2.5 se muestra que el *cliente Android* interactúa con el sistema para elegir la red social y compartir el enlace de la Radio Nuevo Tiempo.

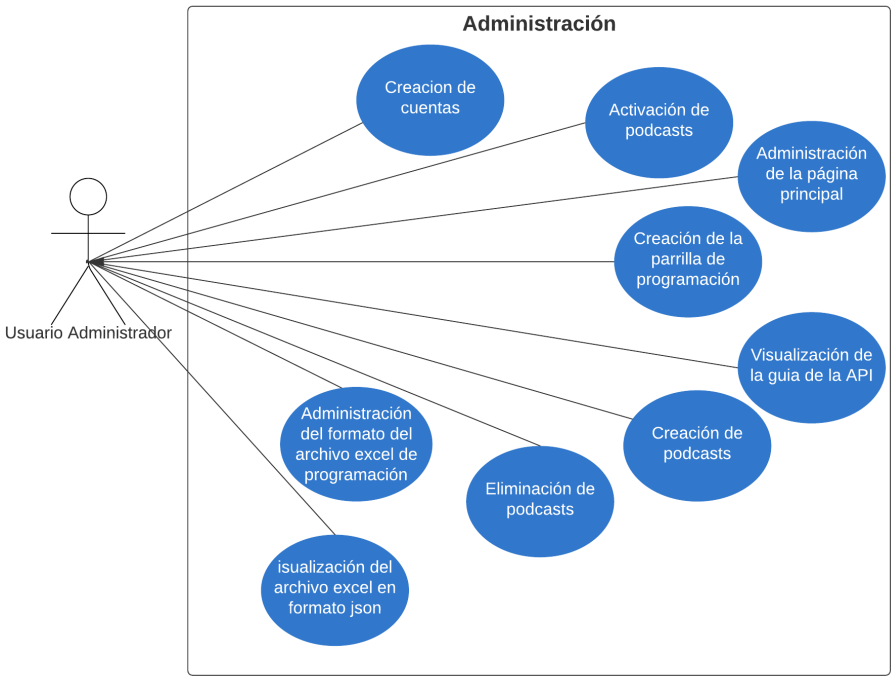


Figura 2.2 Diagrama de casos de uso del módulo Administración

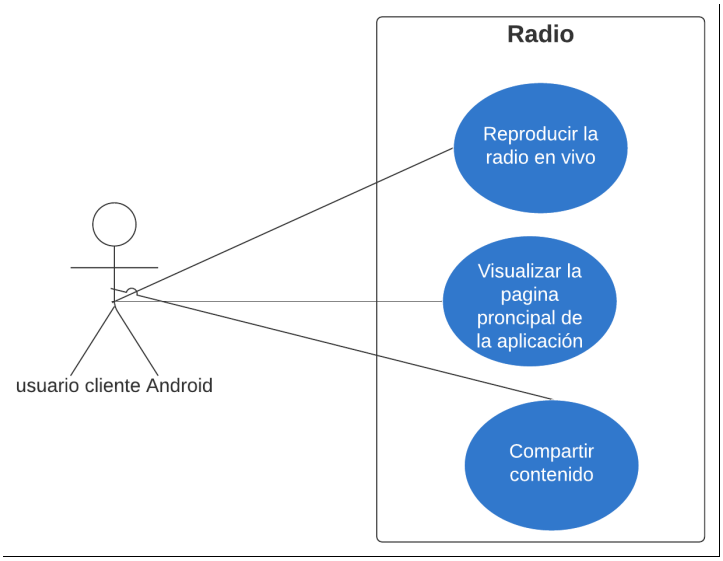


Figura 2.3 Diagrama de casos de uso módulo Radio

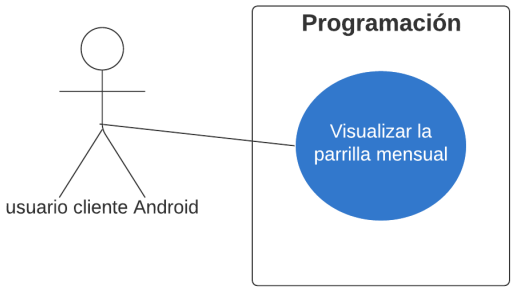


Figura 2.4 Diagrama de casos de uso módulo Programación

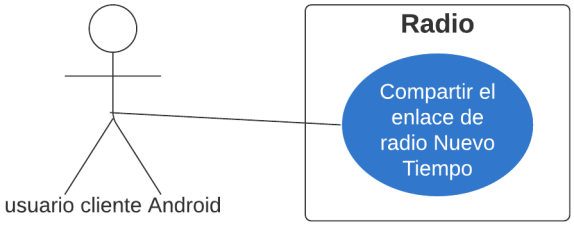


Figura 2.5 Diagrama de casos de uso módulo compartir

2.1.6 DIAGRAMA DE ACTIVIDADES

Los diagramas de actividades posibilitan explicar los procesos que actúan en los casos de uso [20]. A continuación, se muestran los diagramas de actividades de los procesos considerados más importantes: autenticación de *administrador*, reproducción de radio en vivo, envió de podcasts al dispositivo Chromecast y compartición de la aplicación por redes sociales.

2.1.6.1 Autenticación de usuario administrador

El sistema permitirá al *administrador* ingresar sus credenciales para acceder al sistema. La Figura 2.6 describe el proceso de autenticación del *administrador* y es como sigue: cuando el usuario *administrador* desee ingresar al sistema tendrá que escribir su correo y su contraseña, si es que no ha ingresado al sistema anteriormente o si no ha cerrado sesión, luego de haber ingresado correctamente el correo y la contraseña el sistema lo redirigirá a la página de administración principal, caso contrario se le mostrará un mensaje indicando que tendrá que volver a ingresar las credenciales correctamente.

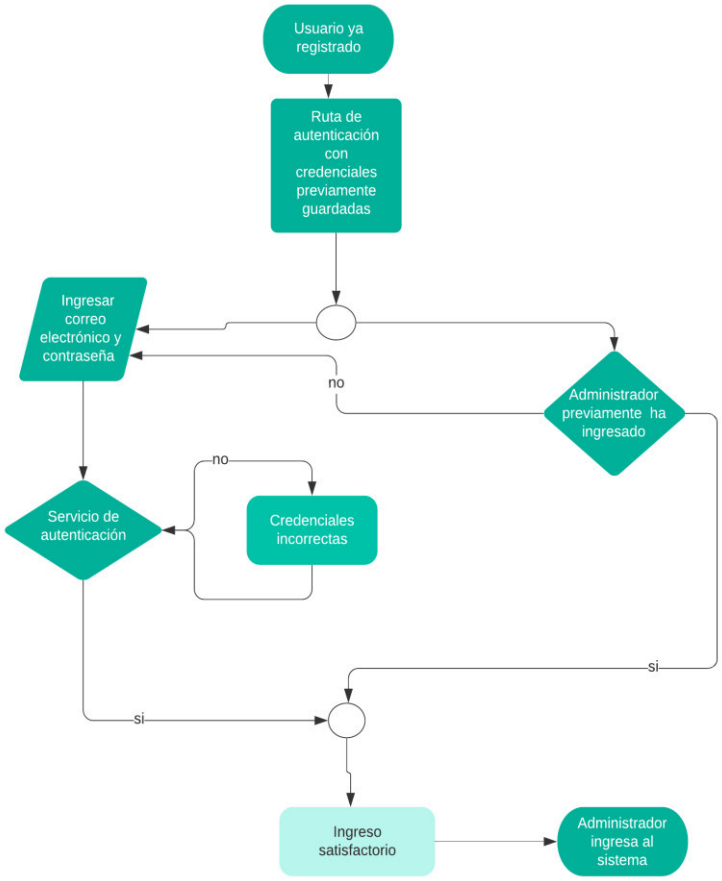


Figura 2.6 Diagrama de actividades para autenticar Administrador

2.1.6.2. Reproducción de radio en vivo

El usuario *cliente Android* en la página principal de la aplicación tiene los controles para reproducir la radio y pausarla. En la Figura 2.7 se muestra el proceso desde cuando el usuario ingresa a la aplicación y presiona el botón play, el sistema llama al método encargado de conectar el servicio con el servidor de streaming, y a su vez recepta los datos enviados por dicho servidor hacia la aplicación RNT (Radio Nuevo Tiempo).

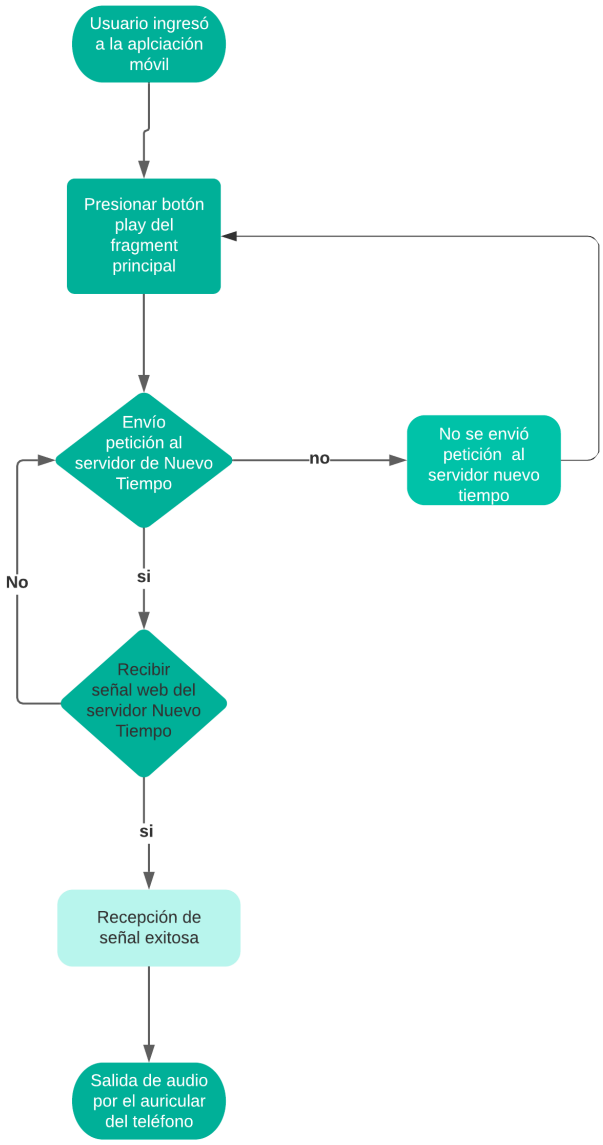


Figura 2.7 Diagrama de actividades para reproducir radio en vivo

2.1.6.3 Envió de podcasts al dispositivo Chromecast

El usuario *cliente Android* en el módulo Podcast tiene los controles para enviar contenido a un dispositivo Chromecast previamente conectado a la red wifi, en la Figura 2.8 se muestra el proceso desde que el usuario posee un equipo Chromecast vinculado a la misma red de la aplicación, primeramente la aplicación detecta un dispositivo Chromecast conectado y se le da al usuario la opción de enviar el contenido a dicho dispositivo, si decide enviar se envía el contenido al Chromecast usando un menú desplegable para esta función. Una vez que el Chromecast recibe la señal empieza a reproducir en la tv dicho contenido.

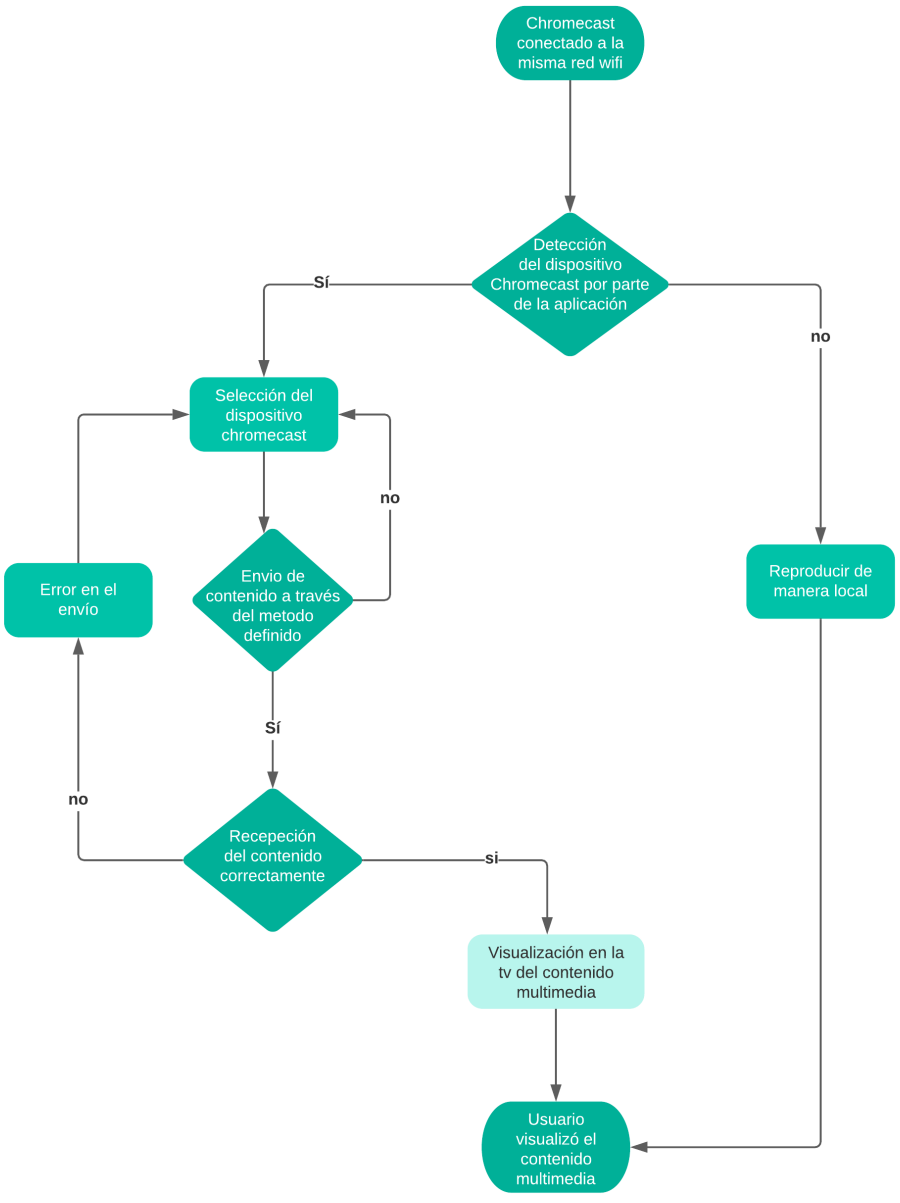


Figura 2.8 Diagrama de actividades para reproducir videos en dispositivo Chromecast

2.1.6.4 Compartición de la aplicación por redes sociales

El usuario *cliente Android* en el menú de opciones tiene la opción para compartir la aplicación por redes sociales. En la Figura 2.9 se muestra el proceso desde que el usuario escoge la opción compartir, primeramente, se le despliega un submenú con las opciones de redes sociales disponibles, luego escoge una de las opciones para luego ser enviada un enlace de la aplicación por aquella red social.

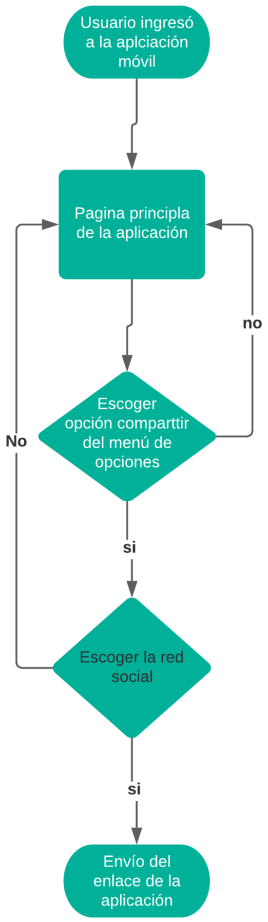


Figura 2.9 Diagrama de actividades para compartir aplicación por redes sociales

2.1.7 DIAGRAMA DE CLASES DEL SERVIDOR RNT

En la Figura 2.10 se presentan los atributos y métodos de las clases: `Auth`, `Admin`, `Api (application Programming) interface`, `App`, `Login`, `Upload`. Estas clases son un middleware del servicio Express que permiten la autenticación y la lógica de navegación en el sistema.

En la Figura 2.10 se muestran las clases `User` e `Index`, que son las clases que heredan las clases middleware para completar el funcionamiento del sistema RNT.

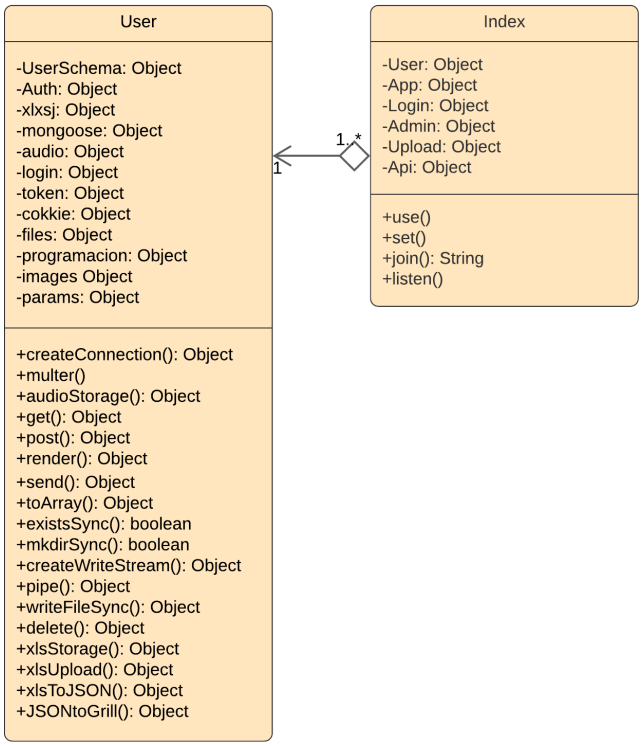


Figura 2.10 Diagrama de clases del servicio RNT

2.1.8 DIAGRAMA DE CLASES DE LA APLICACIÓN ANDROID

En esta sección se muestran las descripciones de la aplicación representadas por el diagrama de clases. En la Figura 2.11 se observan las clases usadas para representar las respuestas a solicitudes HTTP. La clase VideoProvider autorizará instaurar un grupo de variables tipo String que almacenará la información provista en un formato JSON desde el servidor, para luego mostrar esa información en la aplicación.

La mayoría de clases derivan de `Fragment` que es una clase que representa una parte de una interfaz para ser reusada en diferentes vistas, por lo cual se implanta el método `onCreateView` con la meta de inicializar los componentes y variables útiles para el funcionamiento del fragmento. Asimismo, implementan el método itemClicked con la finalidad de manejar un controlador para cada evento click del usuario en la aplicación, por ejemplo, llamar a un segundo fragmento.

En Figura 2.11 se presentan las clases empleadas para recibir la señal de la radio en vivo desde el servidor de Streaming, así mismo, se presentan las clases empleadas para transmitir los podcasts al dispositivo Chromecast, por último, se presentan las clases que van a manejar y mostrar los datos JSON enviados por el servidor RNT, en resumen, son las clases que manejarán la parrilla de programación.

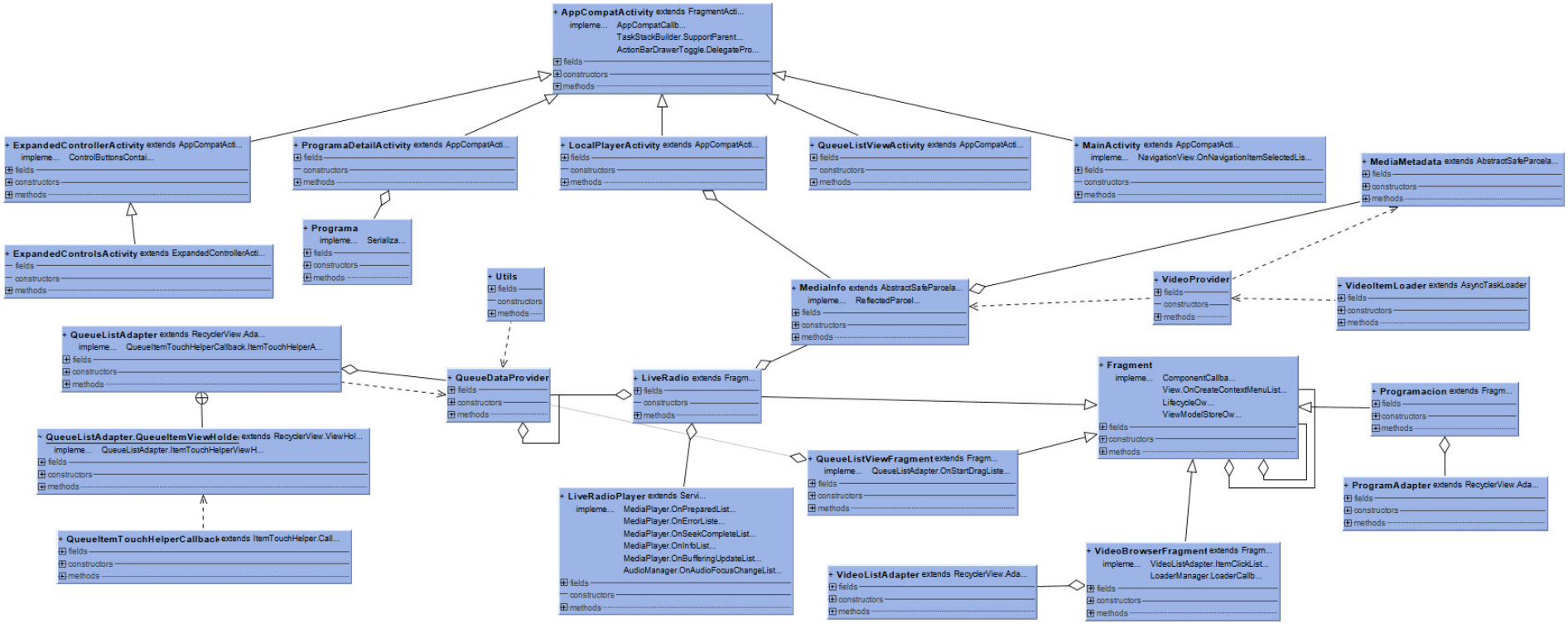


Figura 2.11 Diagrama de clases Aplicación Android

2.1.9 INTERFACES DE LA APLICACIÓN MÓVIL

Los bosquejos se elaboraron tomando en consideración la descripción de los requerimientos funcionales y permitirán diseñar las interfaces de usuario (UI) de la aplicación móvil. Posteriormente, se exponen los primordiales mockups.

En la Figura 2.12 se presenta el mockup de la página principal del usuario, se visualiza el diseño de la página principal de la Aplicación. Se trata de `LiveRadio` que en conjunto con `LiveRadioPlayer` son las clases encargadas de reproducir el contenido en vivo de la radio, así también, realizan una carga del contenido en un hilo principal por lo que la carga se balancea usando una propiedad asincrónica; con esto se consigue que la aplicación no se quede colgada.

En la Figura 2.13 se muestra el mockup del menú principal del usuario. Se trata de un `NavigationDrawer` que se define al crear la actividad principal de la aplicación de manera que se inicialice junto con la barra de herramientas. Como se muestra en la Figura 2.13 se instancia un botón en el panel izquierdo de la barra de herramientas que cuando es presionado se muestra el menú que permite navegar entre los fragmentos que constituyen la aplicación. Entre las opciones se encuentra: Radio que es la actividad principal que permite escuchar la señal en vivo de Nuevo Tiempo, Programación que permite visualizar la parrilla de la radio, Podcasts que permite visualizar el contenido multimedia de la radio y reproducirlo, Configuración que permite configurar opciones generales en presencia de un dispositivo Chromecast, Compartir que permite enviar por redes sociales el enlace web de la radio Nuevo tiempo.

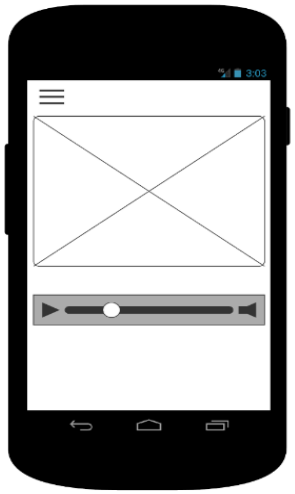


Figura 2.12 Mockup de la UI de la página principal del usuario

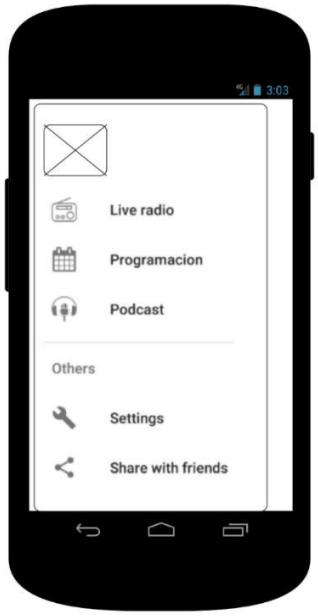


Figura 2.13 Mockup del UI del menú principal del usuario

2.2. ARQUITECTURA

El proyecto está diseñado de acuerdo con las capas definidas en el patrón de arquitectura modular, que consta de: capa interfaz o capa presentación, capa de negocio y capa de datos, como se muestra en la Figura 2.14 Cada capa de la arquitectura tiene una tarea específica y se comunicarán bilateralmente entre todas.

- Capa presentación: Es el nivel en que los usuarios interactúan con la aplicación, este nivel puede ser construido con lenguajes como HTML5, CSS entre otros, para el caso del desarrollo de este proyecto de titulación se usó la arquitectura Android y como lenguaje de etiquetas HTML5, CSS y JavaScript permitiendo la creación de interfaces finales de usuario.

- Capa de negocio: En esta capa se desarrolla toda la lógica del negocio, maneja lenguajes como java, PHP entre otros. La capa de negocio funciona como puente entre la capa de datos y la capa presentación. Para la implementación de esta capa en el actual proyecto de titulación se usó el lenguaje de etiquetas JavaScript.

- Capa de datos: esta capa consta de la base de datos, además de una herramienta para su administración de accesos, puede hospedarse en servidores locales como

también en la nube. En la construcción de la capa de datos se usará Mongoose como motor de base de datos.

La Figura 2.14 muestra la arquitectura en tres capas utilizada para el desarrollo del presente proyecto. El usuario interactúa con la interfaz gráfica, por ejemplo, presionando un botón de play en alguno de los videos, el servidor Node.js gestiona el evento, seguidamente accede a la capa de datos para obtenerlos, finalmente la capa de negocio envía el contenido a la capa presentación de acuerdo con la acción realizada por el cliente.

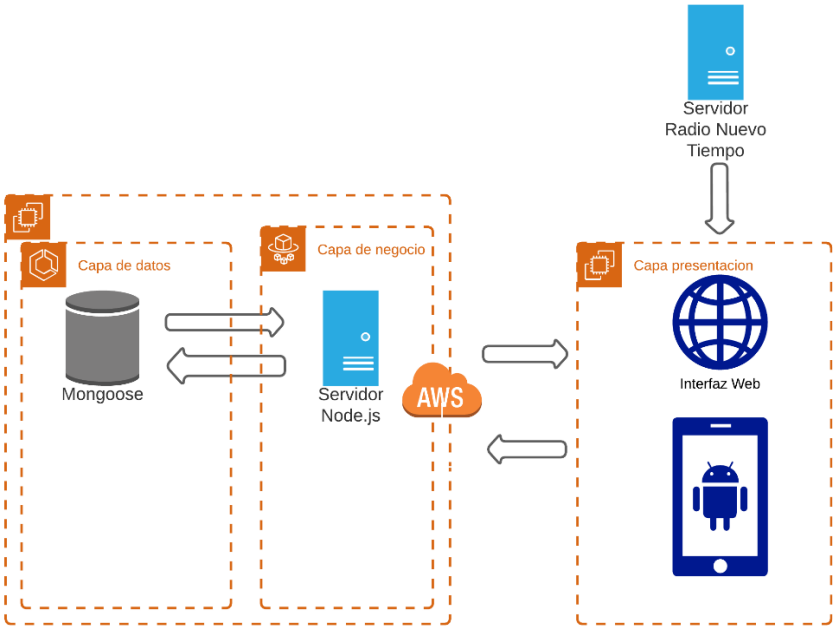


Figura 2.14 Arquitectura

2.3 IMPLEMENTACIÓN

2.3.1 ACTUALIZACIÓN DEL TABLERO KANBAN

La actualización del tablero Kanban para esta fase se muestra en la Figura 2.15 donde se contempla la columna “por hacer” , que se refiere a las tareas que son necesarias para implementar el servidor RNT, estas actividades son: creación de la base de datos, Subida de plantilla mensual de programación, instalación de paquetes npm para el servidor node.js, codificación del servidor node.js, creación del archivo Javascript para generar el prototipo , `GET/POST` a las rutas de servicio usando Postman, definición e implementación de las rutas

de servicio en el servidor, conexión entre base de datos y servidor node.js, codificación de express para node.js, uso de Mongoose para node.js, creación del servidor Node.js.

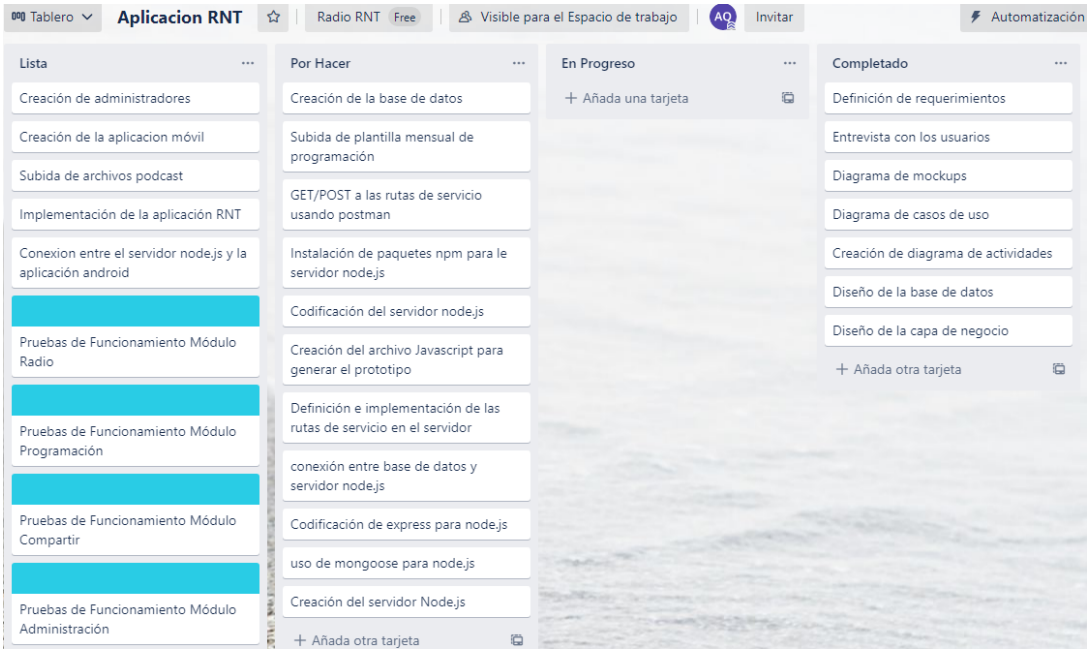


Figura 2.15 Actualización del tablero Kanban

2.3.2 APLICACIÓN ANDROID

Para la creación de esta aplicación se ha utilizado un código modelo creado por Google que implementa la funcionalidad de enviar contenido a través de un dispositivo Chromecast. La mencionada aplicación es de carácter público y está disponible para modificarla o ser usada en la plataforma de GitHub [27]. A continuación, se presentan las clases creadas para cumplir con lo esperado en este proyecto.

2.3.2.1 Actividad principal

A continuación, se explicará la estructura de las carpetas y las clases que se ejecutan al inicializar la aplicación. Cabe mencionar que se ha creado un menú de navegación que sirve para iniciar diferentes ventanas de la aplicación, dichas opciones usan a su vez los recursos alojados dentro de una carpeta `res`.

Dentro de la actividad principal se encuentra la clase `LiveRadio` que contiene el funcionamiento de Fragment en donde se encuentran los elementos multimedia que sirven para reproducir la radio, así mismo esta actividad es la encargada del manejo de eventos

que se dan cuando el usuario efectúa alguna acción sobre la misma. La actividad principal (`fragment_live_radio`) es lo primero que se muestra al abrir la aplicación. Al ejecutar esta actividad se crean varios elementos de la aplicación que se detallan a continuación: en primer lugar, se crea la vista que se va a mostrar con todos sus elementos gráficos que se encuentran alojados dentro de la carpeta `res` del directorio del proyecto, como se muestra en la Figura 2.16

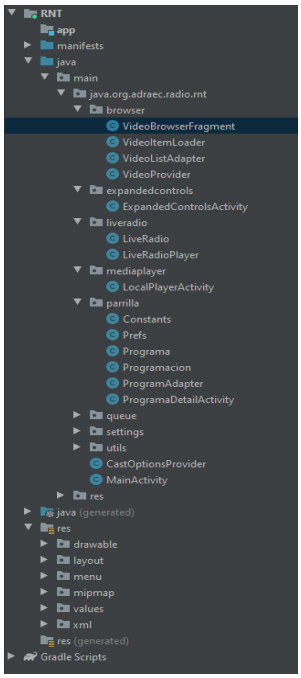


Figura 2.16 Directorio del proyecto

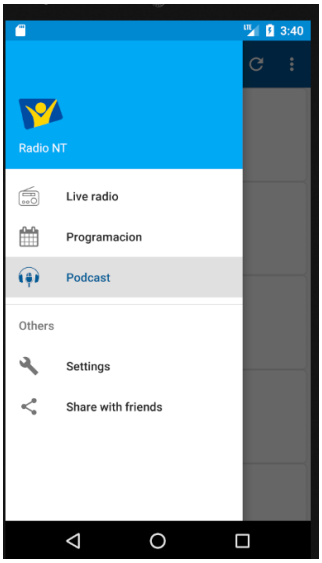


Figura 2.17 Menú de opciones

Después se crea, la barra de herramientas de la aplicación seguido de un menú de opciones que se muestra en la Figura 2.17 finalmente se crea un `CastStateListener`, clase que crea un icono de búsqueda de un dispositivo Chromecast dentro de la red WiFi. Al pulsar sobre alguna opción dentro del menú, se crean o se destruyen los fragmentos que componen la aplicación, como se muestra en el Código 2.1, y que dan la apariencia de navegación a través de la aplicación móvil. Como se aprecia en el Código 2.1, se cuenta con varios condicionales que ejecutan las cinco opciones del menú, por ejemplo, si se elige la opción “Compartir con amigos” se crea un `Intent` con una acción `SEND` , que le indica al sistema operativo Android que la aplicación va a compartir contenido por alguna red social que elija el usuario dentro de un submenú desplegable que se crea al elegir esta opción, como se muestra en la Figura 2.18

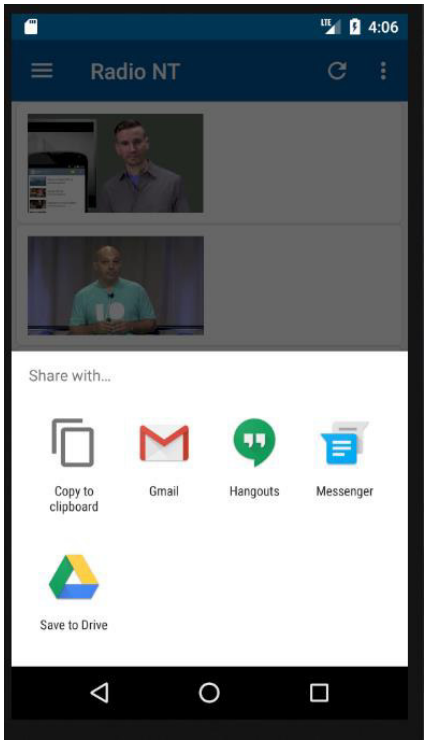


Figura 2.18 Barra de herramientas

```
try {
    if(fragmentClass != null) {
        fragment = (Fragment) fragmentClass.newInstance();
        for (int i = 0; i < fragmentList.size(); i++) {
            if(fragmentList.get(i).getId() != R.id.cast_mini_controller) {
                fragmentManager.beginTransaction().remove(fragmentList.get(i));
                fragmentList.get(i).onDestroy();
            }
        }
        fragmentManager.beginTransaction()
                .replace(R.id.browse, fragment)
                .commit();
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

Código 2.1 Transacción de fragmentos

2.3.2.2 Fragmento de navegador podcast

La aplicación RNT cuenta con fragmentos, que son vistas intermedias entre la Activity y el contenedor de la vista. Se pueden remplazar fácilmente sin necesidad de cambiar de Activity [28]. Los fragmentos generalmente son usados cuando existe interacción con la aplicación a través de un menú, solo cambiando la parte interior que corresponde a la opción seleccionada en el menú, como se muestra en Figura 2.19

El fragmento Video Browser está constituido por un RecyclerView, clase que fue creada por Google para mostrar una lista de elementos, pero de manera más eficaz que un `ListView` por ejemplo. Uno de los servicios de `RecyclerView` es que solo muestra los elementos de alguna lista que se muestran en la interfaz en el momento actual, para proporcionar un uso de memoria eficiente y visualización de grandes colecciones de datos. RecyclerView se basa en un modelo más flexible que sus predecesores como AdapterView en cuanto a la ubicación de los componentes de la vista secundaria, en donde se delega a una instancia de LayoutManager para esta tarea [29].

Para poder mostrar algún contenido usando `RecyclerView` son necesarias por lo menos dos clases, la una que sirve para descargar el contenido del formato JSON, y la otra que se encarga de adaptar el contenido al widget (RecyclerView) para que se pueda visualizar de una manera organizada en la lista.

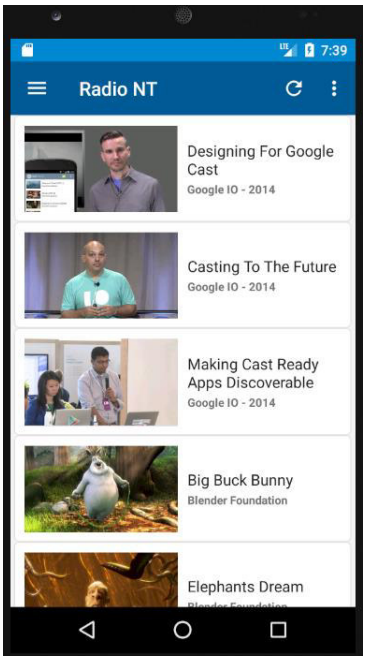


Figura 2.19 Actividad Podcast RNT usando Google

2.3.2.3 Proveedor de podcast

VideoProvider es la clase a la cual se hace referencia en este punto, a través de esta clase se obtiene el contenido del fichero JSON que se encuentra alojado en nuestro servidor, este archivo se lo obtiene a través de una dirección url, este archivo contiene los datos necesarios para crear un archivo de audio o de video. Para lograr este fin, se crea una lista de objetos en Java usando la clase `MediaInfo` que se muestra en el Código 2.2 Estos objetos contienen la información necesaria para reproducir un video como es la url donde se aloja el video, título del video, duración en milisegundos del video, entre otras opciones extras que se pueden añadir.

```java
private static MediaInfo buildMediaInfo(String title, String studio, String subTitle,
                                        int duration, String url, String mimeType, String imgUrl, String bigImageUrl,
                                        List<MediaTrack> tracks) {
    MediaMetadata movieMetadata = new MediaMetadata(MediaMetadata.MEDIA_TYPE_MOVIE);

    movieMetadata.putString(MediaMetadata.KEY_SUBTITLE, studio);
    movieMetadata.putString(MediaMetadata.KEY_TITLE, title);
    movieMetadata.addImage(new WebImage(Uri.parse(imgUrl)));
    movieMetadata.addImage(new WebImage(Uri.parse(bigImageUrl)));
    JSONObject jsonObj = null;
    try {
        jsonObj = new JSONObject();
        jsonObj.put(KEY_DESCRIPTION, subTitle);
    } catch (JSONException e) {
        Log.e(TAG,  msg: "Failed to add description to the json object", e);
    }

    return new MediaInfo.Builder(url)
            .setStreamType(MediaInfo.STREAM_TYPE_BUFFERED)
            .setContentType(mimeType)
            .setMetadata(movieMetadata)
            .setMediaTracks(tracks)
            .setStreamDuration(duration )
            .setCustomData(jsonObj)
            .build();
}
```

Código 2.2 Objeto MediaInfo

2.3.2.4 Adaptador de lista de podcast

Como ultima clase de las ya citadas, que se usan para construir el `RecyclerView`, se tiene a VideoListAdapter. Esta clase actúa como un adaptador, donde se acoge la lista Java con los datos de los podcasts a mostrar en un `cardView`, un elemento visual de Android para mostrar contenido multimedia.

Para lograr antes lo mencionado, `RecyclerView` instancia todos los elementos de forma personalizada renderizando cada uno de ellos de acuerdo como lo solicite la aplicación. Para cada uno de ellos se inserta la información contenida en el objeto `MediaInfo`, el cual contiene entre otras cosas la miniatura, titulo, estudio, etc. En el Código 2.3 se presenta la creación de estos elementos y su vinculación con los widgets. Finalmente, esta clase crea un `ClickEventListener` en cada ítem del `RecyclerView` para que al presionar sobre uno de ellos se abra la interfaz local, mostrando el contenido asociado del podcast.

```
private void loadViews() {
    mVideoView = (VideoView) findViewById(R.id.videoView1);
    mTitleView = (TextView) findViewById(R.id.titleTextView);
    mDescriptionView = (TextView) findViewById(R.id.descriptionTextView);
    mDescriptionView.setMovementMethod(new ScrollingMovementMethod());
```

Código 2.3 Referencia a Elementos visuales

2.3.2.5 Actividad de reproductor local

La clase `LocalPlayerActivity` es una de las clases que ya existían en la aplicación modelo de Google para el casting, esta clase es la encargada de reproducir el podcast seleccionado en la actividad `Programacion`, donde se alojan todos los podcasts disponibles. El funcionamiento de este Activity es como sigue; cuando el usuario elije uno de los podcasts disponibles se crea un `Intent` que contienen la información del podcast. Esta clase `Intent` tiene un método que se llama `putExtra` y extrae toda la información referente al podcast, como se muestra en la Figura 2.20 Actividad LocalPlayerActivity

Como primer punto, se cargan todos los elementos visuales de la actividad para guardar sus referencias, como se muestra en el Código 2.4 y luego puedan ser pobladas de acuerdo con la información que viene embebida en el `Intent`. Después de este proceso se verifica que no hay un dispositivo Chromecast conectado a la red wifi, si no fuera el caso se reproduciría en el dispositivo y no en el reproductor local.

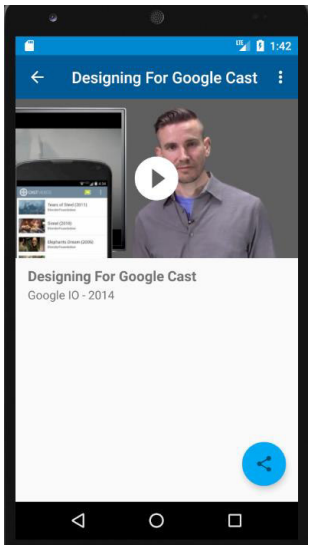


Figura 2.20 Actividad LocalPlayerActivity

```
private void loadRemoteMedia(int position, boolean autoPlay) {
    if (mCastSession == null) {
        return;
    }
    final RemoteMediaClient remoteMediaClient = mCastSession.getRemoteMediaClient();
    if (remoteMediaClient == null) {
        return;
    }
    remoteMediaClient.registerCallback(onStatusUpdated() → {
            Intent intent = new Intent( packageContext: LocalPlayerActivity.this, ExpandedControlsActivity.class);
            startActivity(intent);
            remoteMediaClient.unregisterCallback(this);
    });
    remoteMediaClient.load(new MediaLoadRequestData.Builder()
            .setMediaInfo(mSelectedMedia)
            .setAutoplay(autoPlay)
            .setCurrentTime(position).build());
}
```

Código 2.4 Verificación de presencia de un dispositivo Chromecast

Finalmente, se obtiene la información contenida en el objeto `MediaInfo` que ha sido remitido en el `Intent`, esta información es por ejemplo el título, subtítulo, dirección del Streaming, etc.

2.3.2.6 Actividad de radio en vivo

Para el correcto funcionamiento de la radio en vivo se han creado dos clases, `LiveRadio` y `LiveRadioPlayer`, la primera clase compone el Fragment y controla la interacción del usuario con esta vista como es: la reproducción de radio, subir el volumen, pausar la radio, como se puede apreciar en la Figura 2.21; la segunda clase es un servicio de Android que se ejecuta en segundo plano en un hilo diferente del hilo principal, esto para que al cargar el contenido de la radio en vivo no se vea afectado el rendimiento por la carga del contenido.

El funcionamiento de la actividad `LiveRadio` empieza comprobando si existe un dispositivo Chromecast conectado a la red, para darle la opción al usuario de enviar el contenido de la radio al dispositivo. Después se instancia un `Intent` que contiene la información para remitir a la clase `LiveRadioPlayer` el comando para reproducir la radio, al crearse este intent se crea un enlace entre la clase que contiene el Fragmento y la clase que contiene el servicio de reproducción de la radio . A merced de este solo enlace que se crea, se evita que se generen difusiones sincrónicas de la radio lo cual implicaría una deficiente experiencia de usuario. Justo después de haberse creado este enlace se lo guarda en una variable tipo `LiveRadioPlayer`, que se puede apreciar en él Código 2.5, con el fin de que cuando se salga de la aplicación o se cambie de vista dentro del menú,

este enlace se destruya y no existan retardos como se muestra en el Código 2.6, dando como resultado en un mal funcionamiento de la aplicación.

```
public boolean onNavigationItemSelected(MenuItem item) {
    int id = item.getItemId();
    Intent intent;
    Class fragmentClass = null;
    Fragment fragment;
    FragmentManager fragmentManager = getSupportFragmentManager();
    List<Fragment> fragmentList = fragmentManager.getFragments();;

    if (id == R.id.nav_live_radio) {
        currentFragment = "LIVERADIO";
        fragmentClass = LiveRadio.class;
        invalidateOptionsMenu();
    }
    else if (id == R.id.nav_podcasts) {
        currentFragment = "PODCASTS";
        fragmentClass = VideoBrowserFragment.class;
        invalidateOptionsMenu();

    } else if (id == R.id.nav_programacion) {
        currentFragment = "PROGRAMACION";
        fragmentClass = Programacion.class;
        invalidateOptionsMenu();
    }

     else if (id == R.id.nav_settings) {
            intent = new Intent( packageContext: MainActivity.this, CastPreference.class);
            startActivity(intent);
            invalidateOptionsMenu();
        }
     else if (id == R.id.nav_share) {
        Intent myIntent = new Intent(Intent.ACTION_SEND);
        myIntent.setType("text/plain");
        String shareBody = "I'm listening to Radio Nuevo Tiempo, listen here:" + " https://www.nuevotiempo.org/en-vivo-radio-ecuade
        String shareSub = "Radio Nuevo Tiempo";
        myIntent.putExtra(Intent.EXTRA_SUBJECT, shareSub);
        myIntent.putExtra(Intent.EXTRA_TEXT, shareBody);
        startActivity(Intent.createChooser(myIntent, "Share with…"));
    }
```

Código 2.5 Opciones de menú

```
@Override
public void onDestroy() {
    //Log.d(TAG, "Ondestroy: ");
    //hideNotification();
    if (mediaPlayer != null) {
        stopMedia();
        mediaPlayer.release();
        mediaPlayer = null;
    }
    removeAudioFocus();
    super.onDestroy();
}

boolean isPlaying() {
    if(mediaPlayer != null)
        return mediaPlayer.isPlaying();
    return false;
}
```

Código 2.6 Destrucción del enlace Fragment – servicio

Figura 2.21 Actividad LiveRadio

2.3.3 SERVIDOR NODE.JS

En esta sección se explica la implementación del servidor RNT_Server creado para alimentar a la aplicación RNT.

2.3.3.1 Vista inicial de login

Como primer punto se detalla la vista inicial del servidor RNT. Como se muestra en la Figura 2.22 se definió un método `POST` apuntando a la ruta `/login` la cual es usada por parte del *administrador* del servidor para comprobar sus credenciales, y si son correctas, iniciar sesión.

Cuando el servidor recibe el email y la contraseña, contrasta esta información con la información guardada en la base de datos, la base de datos contiene el email del *administrador* y hash de la contraseña, si ambos campos coinciden con los campos almacenados en la base de datos se le permite el ingreso a la siguiente vista, caso contrario se le envía un mensaje indicando que las credenciales son incorrectas como se muestra en el Código 2.7. Para evitar que algún *administrador* ingrese al sistema sin iniciar sesión, se ha instaurado un método ubicado en el fichero `index.js`, en este punto es donde se

inicia el servidor; el *administrador* tiene iniciada sesión, si no ha iniciado sesión lo redirige a la página principal, como se puede apreciar en el Código 2.8

```
router.post('/login', urlencodedParser, async (req, res) => {
    try {
        const user = await User.findByCredentials(req.body.email, req.body.password)
        req.session.userInfo = user
        const token = await user.generateAuthToken()
        res.cookie('authcookie', token, { maxAge: 900000, httpOnly: true })
        //res.send({ user, token })
        /*res.send({valid: true})*/
        //res.render('admin', { title: 'Radio Nuevo Tiempo'})
        //res.send({ user, token })
        // res.setHeader('Authorization', 'Bearer '+ token)
        //req.session.userInfo = ({ token  })

        gfs.files.find().toArray((err, files) => {
            // Check if files
            if (!files || files.length === 0) {
                res.render('audio.ejs', { files: false });
            } else {
                files.map(file => {
                    if (
                        file.contentType === 'video/mp4'
                    ) {
                        file.isImage = true;
                    } else {
                        file.isImage = false;
                    }
                });
                res.render('audio.ejs', { files: files });
            }
        })
    } catch (e) {
        res.status(400).send("usuario o contrasena incorrectas")
    }
})
```

Código 2.7 Ruta Login para iniciar sesión

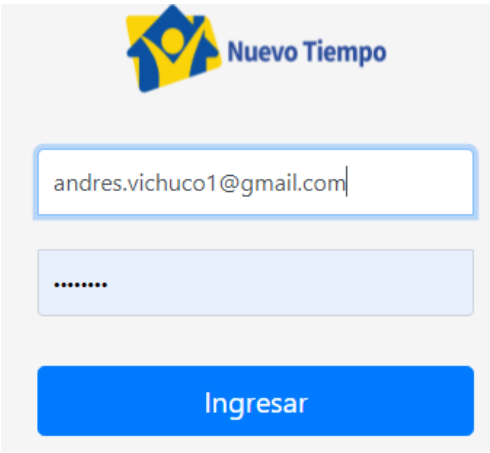


Figura 2.22 Vista de login del servidor RNT

```
app.use(function (req, res, next) {
    if(!req.session.userInfo && (req.path === '/upload'  || req.path === '/login'   ) && req.method === 'GET') {
        res.redirect('/');
    }
    else if(req.path === '/logout' && req.method === 'POST'){
        res.redirect('/');
    }
    //else if(req.session.userInfo && (req.path === '/login' ), req.method === 'POST' ) {
      //  res.redirect('/upload');
    //}
    else {
        next();
    }
});
```

Código 2.8 Middleware para controlar sesiones

2.3.2.2 Página de podcast

Una vez que el *administrador* ha iniciado sesión satisfactoriamente, ingresa a la interfaz podcast que se muestra en la Figura 2.23

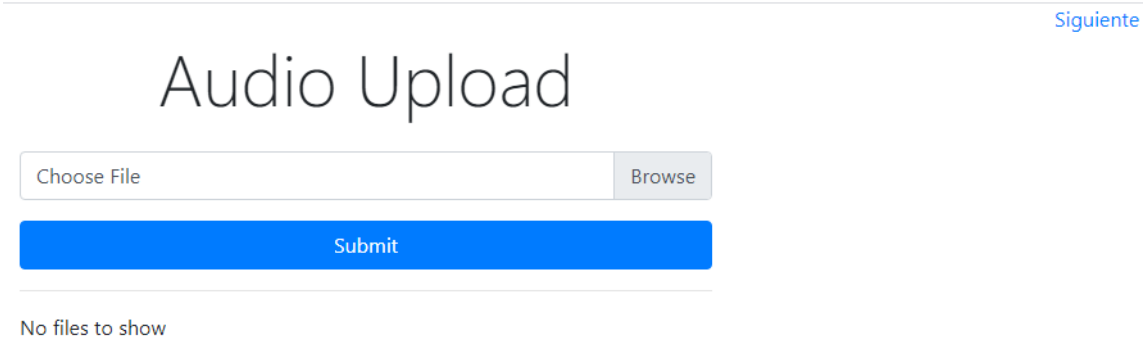


Figura 2.23 Pagina de podcasts

En primer lugar, se encuentra la opción para subir un podcast al servidor, el podcast se mostrará en esta página contenedora (`audio upload`). Luego se tiene una pestaña la cual servirá en caso de que el *administrador* no decida subir un podcast, y le mostrará la página de administración de la parrilla de programación.

Se debe mencionar que todos los archivos multimedia subidos al servidor deben ser de tipo mp4, el servidor fue pensado, diseñado e implementado para aceptar únicamente este tipo de formato. Los archivos multimedia se almacenan en la base de datos, estos son la base para crear el documento JSON que luego se utiliza a través de la aplicación. Este proceso

se lo codificó bajo la ruta `/upload` como se puede apreciar en el Código 2.9

```
router.post('/upload', auth, upload.single('file'), (req, res) => {
    xlsActual = false
    xlsSiguiente = false
    const path = "public/xls/"
    // Comprobamos si hay parrilla de semana actual y semana siguiente guardadas en el servidor para luego mostrarlo en el front
    if (fs.existsSync(path + "semana_actual.xlsx")) xlsActual = true
    if (fs.existsSync(path + "semana_siguiente.xlsx")) xlsSiguiente = true
    gfs.files.find().toArray((err, files) => {
        // Check if files
        if (!files || files.length === 0) {
            res.render('audio.ejs', { files: false });

        } else {
            files.map(file => {
                if (
                    file.contentType === 'video/mp4'
                ) {
                    file.isImage = true;
                } else {
                    file.isImage = false;
                }
            })

            const audios = "public/audios/"
            if (!fs.existsSync(audios)) {
                fs.mkdirSync(audios);
            }
```

Código 2.9 Función que realiza el almacenamiento de podcasts

La función que se muestra en el Código 2.9 tiene un middleware que comprueba si se ha iniciado sesión o si la sesión ya ha caducado para poder continuar con el proceso, luego se crea una carpeta para almacenar el fichero json que se creará a partir del archivo multimedia que se suba.

2.3.2.3 Pagina de administración

Después que el *administrador* ha subido el/los podcasts, tiene la opción de acceder a la interfaz de administración que se visualiza en la Figura 2.24

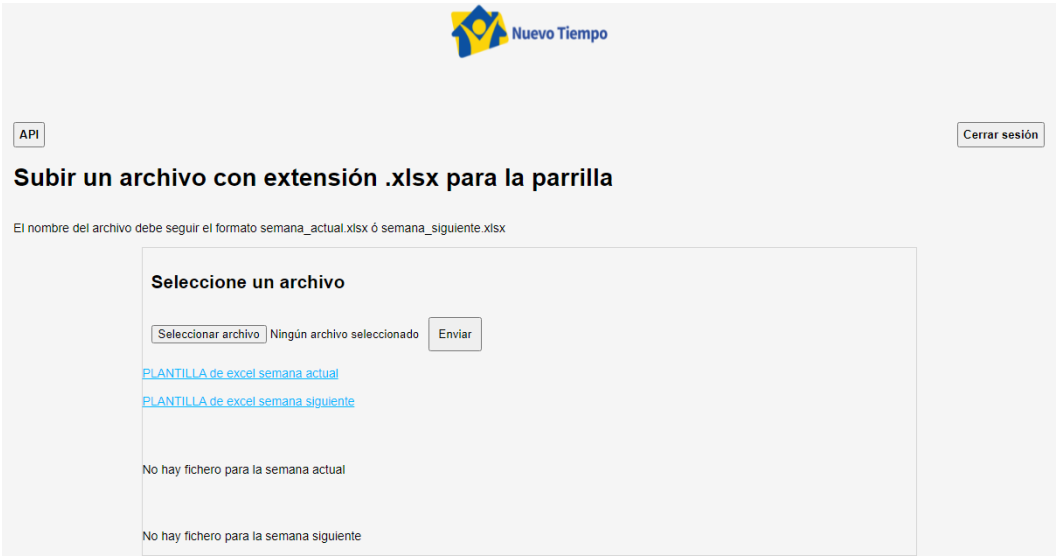
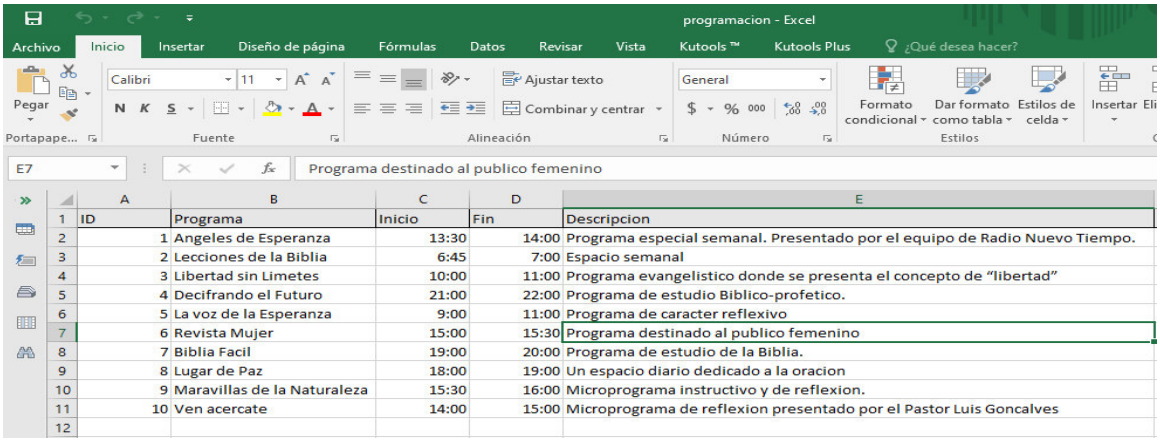


Figura 2.24 Página de administración

En Figura 2.25 se puede apreciar un botón que tiene por título API (application programming interface) que nos lleva a una referencia de cómo usar esta aplicación. Después tenemos el botón cerrar sesión que nos lleva nuevamente a la página de login. En el centro de la página encontramos un botón para subir la parrilla de programación y luego un botón para enviar dicha parrilla al servidor. Después encontramos un modelo de la parrilla de programación que puede ser descargada en formato xlsx para que el *administrador* entienda el formato en el cual llenar la parrilla de programación de la semana vigente, al igual que de la semana posterior, para esto se ha creado un formato para la parrilla de programación, como se ve en la Figura 2.25 en la cual se observa el nombre del programa, la hora de inicio - fin del programa y la descripción que se va a mostrar en la aplicación.



	A	B	C	D	E
1	ID	Programa	Inicio	Fin	Descripcion
2	1	Angeles de Esperanza	13:30	14:00	Programa especial semanal. Presentado por el equipo de Radio Nuevo Tiempo.
3	2	Lecciones de la Biblia	6:45	7:00	Espacio semanal
4	3	Libertad sin Limetes	10:00	11:00	Programa evangelistico donde se presenta el concepto de “libertad”
5	4	Decifrando el Futuro	21:00	22:00	Programa de estudio Biblico-profetico.
6	5	La voz de la Esperanza	9:00	11:00	Programa de caracter reflexivo
7	6	Revista Mujer	15:00	15:30	Programa destinado al publico femenino
8	7	Biblia Facil	19:00	20:00	Programa de estudio de la Biblia.
9	8	Lugar de Paz	18:00	19:00	Un espacio diario dedicado a la oracion
10	9	Maravillas de la Naturaleza	15:30	16:00	Microprograma instructivo y de reflexion.
11	10	Ven acercate	14:00	15:00	Microprograma de reflexion presentado por el Pastor Luis Goncalves
12					

Figura 2.25 Fichero Excel de la parrilla

Para subir el archivo excel se usa la ruta `/addgrill` que se muestra en el Código 2.10. Como primer paso se usa el paquete `Multer` mencionado anteriormente, que sirve para guardar el archivo Excel dentro de la carpeta xlsx, esta carpeta se crea antes de subir el archivo de manera automática. Luego, como siguiente paso, se llama a la función `xlsToJSON` que será la que transforma el formato xlsx a un formato JSON.

```
router.post('/addgrill', xlsUpload.single('file'), function (req, res) {
    xlsToJSON(req.file.filename, res); // Llamada a función que transforma el xlsx a json
})
```

Código 2.10 Método post para subir la parrilla de programación

En el Código 2.11 se muestra la función `xlsToJSON`, esta función como primer punto define la ruta en donde se almacenará el archivo JSON. La función `xlsToJSON` transforma todo el contenido del formato xlsx, pero no es lo que espera recibir la aplicación, por este motivo hay algunos parámetros como: el archivo en formato xlsx, la ruta donde almacenar el archivo en su nuevo formato, y la función que se usará para el nuevo archivo con formato json que deben ser añadidos durante este proceso.

```
function xlsToJSON(filename, res) {
    const path = 'public/xls/' + filename
    xlsxj({
        input: path,
        output: path.replace(".xlsx", ".json"),
        lowerCaseHeaders: true
    }, function (err, result) {
        if (err) {
            console.error(err)
            res.send({ saved: false, error: err });
        } else {
            JSONtoGrill(result, filename, res);
        }
    })
}
```

Código 2.11 Función para transformar xlsx a JSON

En la Figura 2.26 se muestra el archivo JSON que se envía a la aplicación móvil. Continuando con la creación de este fichero se llama a una función `JSONtoGrill`, que se muestra en el Código 2.12. En el código se puede ver la definición del modelo clave – valor que es la parte inicial de la parrilla. Se define un objeto con el nombre `categories`, que tiene los podcasts subidos por el *administrador*, luego se tiene el estudio de la radio que será el mismo para todos los podcasts, por último, se tiene un array vacío donde se añadirán los videos o podcast que constituirán la parrilla.


```
"categories": [
  {
    "name": "videos",
    "mp4": "http://192.168.0.109:3000/",
    "images": "http://192.168.0.109:3000/",
    "videos": [
      {
        "subtitle": "Fusce id nisi turpis. Praesent viverra bibendum semper. Donec tristique, orci sed semper lacinia,
rhoncus massa, non congue tellus est quis tellus",
        "sources": [
          {
            "type": "mp4",
            "mime": "videos/mp4",
            "url": "sound/e38d1f65362a374ee0a0be13b371da5e.mp4"
          }
        ],
        "thumb": "thumbnails/headphones-480x270.png",
        "image-480x270": "thumbnails/headphones-480x270.png",
```


Figura 2.26 Fichero JSON de la parrilla final


```
        "duration": 19965059
      }
```


```
function JSONtoGrill(json, filename, res) {
    let saved = true;
    const grill = {
        "categories": [
            {
                "name": "programas",
                //"images": "https://www.nuevotiempo.org/radio/",
                "studio": "Radio Nuevo Tiempo",
                "files": []
            }
        ]
    }
```

Código 2.12 Función para crear JSON de la parrilla final (parte 1)

El Código 2.13 muestra la continuación del código que crea el documento JSON creado con la información necesaria para ser enviada a la aplicación. En primer lugar, se recorre dentro del array que se obtuvo como resultado de la función `xlsToJSON` y se comprueba que no haya elementos dentro de esta colección de valor nulo que comprende desde la línea 405 a la línea 440. Después se crea una variable para almacenar los metadatos que

se obtienen de cada uno de los elementos de la función mencionada, y se los almacena en una variable del tipo clave-valor esto se observa desde la línea 408 a la línea 414. Luego de este proceso, usando el método push de Javascript, se inserta dentro de la lista categorías toda la información extraída de la base de datos, hay una condición para insertar los datos dentro la lista categorías y esta es que se verifica que exista el directorio en primer lugar, lo mencionado comprende desde la línea 421 a la línea 426; en caso de que no exista un directorio se crea uno, a partir de aquí se codifica la lista usando el paquete npm iso-8859-1 que permite el uso de ñ y signos de acentuación dentro del fichero JSON, esto comprende desde la línea 428 a la línea 431. Si se produce algún error durante el proceso de guardar el archivo JSON se envía una respuesta al cliente de parte del servidor, dando a entender que no se guardó el archivo exitosamente esto se codifica desde la línea 433 a la línea 438, en caso contrario también se indica al cliente que si se realizó la operación satisfactoriamente, ese proceso esta detallado desde la línea 442 a la línea 445.

```
405     json.forEach(element => {
406         if (element.ID !== '') {
407             const mp4Json = {
408                 "id": element.ID,
409                 "titulo": element.Programa,
410                 "inicio": element.Inicio,
411                 "fin": element.Fin,
412                 "descripcion": element.Descripcion,
413                 "poster": element.Poster,
414                 "video": "http://192.168.100.7:3000/programas/bibliaFacil.mp4",
415             }
416             grill.categories[0].files.push(mp4Json);
417             const path = 'public/jsons/' + filename.replace(".xlsx", ".json");
418             const path2 = 'public/jsons/' + element.Programa.replace(/\s+/g, '') + ".json";
419             try {
420                 // Si aun no existe el directorio /jsons debe crearse antes de guardar el archivo de
421                 if (!fs.existsSync('public/jsons/')) {
422                     fs.mkdirSync('public/jsons/');
423                     const str2 = iconvlite.encode(JSON.stringify(mp4Json), 'iso-8859-1'); // Se codi
424                     const str = iconvlite.encode(JSON.stringify(grill), 'iso-8859-1'); // Se codifi
425                     fs.writeFileSync(path, str);
426                     fs.writeFileSync(path2, str2);
427                 } else {
428                     const str = iconvlite.encode(JSON.stringify(grill), 'iso-8859-1'); // Se codifi
429                     fs.writeFileSync(path, str);
430                     const str2 = iconvlite.encode(JSON.stringify(mp4Json), 'iso-8859-1'); // Se codi
431                     fs.writeFileSync(path2, str2);
432                 }
433             } catch (e) {
434                 console.log(e);
435                 // Si se produce algún error se notifica al front para que muestre una alerta
436                 saved = false;
437                 res.send({ saved: false, error: e });
438             }
439         }
440     })
441     // En caso de que se haya guardado correctamente el archivo se notifica al front para que muestr
442     if (saved) {
443         res.send({ saved: true });
444         // res.redirect('/login')
445     }
```

Código 2.13 Función para crear JSON de la parrilla final (parte 2)

2.3.3 BASE DE DATOS

La base de datos fue creada usando el paquete npm `Mongoose`. Se modeló la base de datos asociado a la tarea del servidor como se muestra en Código 2.14(formato JSON). Se codifico la conexión a la base de datos como se muestra en el Código 2.15 en donde, en

primer lugar, se importa el paquete Mongoose, luego usando la función `connect` se le pasa la url de conexión que se encuentra almacenada dentro de la carpeta `config` en un archivo dev.env, como se muestra en el Código 2.16 Después se declara un driver `useNewUrlParser` del paquete Mongoose, que se muestra en la línea 4, que verifica la validez de la cadena de conexión, luego se define una opción `useCreateIndex` mostrado en la línea 5, que crea índices en la base de datos de manera automática, lo cual es útil al momento de guardar los archivos multimedia; por último, se establece en falso la opción `useFindAndModify` mostrada en la línea 6, la cual por defecto viene con valor true, si no cambiamos esta opción afecta el comportamiento del método `find()` que estamos usando para buscar un elemento de la base de datos y eliminarlo.

En el Código 2.14 Formato JSON de los datos se muestra el modelo de datos asociado a la tarea del servidor (formato JSON)

```
name: {
    type: String,
    required: true,
    trim: true
},
email: {
    type: String,
    unique: true,
    required: true,
    trim: true,
    lowercase: true,
    validate(value) {
        if (!validator.isEmail(value)) {
            throw new Error('Email is invalid')
        }
    }
},
password: {
    type: String,
    required: true,
    minlength: 7,
    trim: true,
    validate(value) {
        if (value.toLowerCase().includes('password')) {
            throw new Error('Password cannot contain "password"')
        }
    }
},
tokens: [{
    token: {
        type: String,
        required: true
    }
```

Código 2.14 Formato JSON de los datos

```
1  const mongoose = require('mongoose')
2
3  mongoose.connect(process.env.MONGODB_URL, {
4      useNewUrlParser: true,
5      useCreateIndex: true,
6      useFindAndModify: false
7  })
```

Código 2.15 Conexión a Mongoose

```
PORT=3000
MONGODB_URL=mongodb://127.0.0.1:27017/radio-nt-api
JWT_SECRET=thisisasecretformyapp
```

Código 2.16 Cadena de conexión a Mongoose

Cabe mencionar que para el correcto almacenamiento de los podcasts en la base de datos se usó el paquete npm Multer junto con el paquete GridsStorage, como se observa en el Código 2.17. Este último paquete permite guardar grandes cantidades de datos sin poner un límite en la base de datos, si no se usara tuviera un límite de 16 MB (Megabytes), este límite asegura que un archivo o documento no use una cantidad de RAM o un ancho de banda excesivo [30] . En el Código 2.17 se observa que se define un nombre hexadecimal tomando como referencia el nombre original del archivo multimedia, esto se realizó con el objetivo de establecer como clave primaria el nombre del archivo multimedia y de esta manera conseguir que no se repita el mismo nombre en ningún otro archivo. Como segundo punto, se puede observar que se crea una colección en la base de datos que tiene por nombre `uploads`, esta colección contendrá todos los podcasts subidos a través de la interfaz del *administrador*.

```
const storage = new GridFsStorage({
    url: process.env.MONGODB_URL,
    file: (req, file) => {
        return new Promise((resolve, reject) => {
            crypto.randomBytes(16, (err, buf) => {
                if (err) {
                    return reject(err);
                }
                const filename = buf.toString('hex') + path.extname(file.originalname);
                const fileInfo = {
                    filename: filename,
                    bucketName: 'uploads'
                };
                resolve(fileInfo);
            });
        });
    }
});
const upload = multer({ storage });
```

Código 2.17 Función para almacenar los podcasts

Como último punto, se creó un esquema o modelo de la base de datos con diferentes campos que se pueden ver en el Código 2.18, podemos observar que en primer lugar se definió un nombre para el *administrador* de tipo `String` y que es de tipo obligatorio, también se definió un email para el *administrador* el cual va a servir para iniciar sesión en el servidor, en este campo se usa un paquete npm `validator` que nos ayuda a verificar la validez de un correo electrónico, es decir si tiene los caracteres que componen un correo electrónico como el signo de @, este campo es de tipo `String` y tiene un valor único para que no se repita en la base de datos. Después se estableció el campo password que, más adelante se codifica usando el paquete npm `bcryptjs` que se mencionó anteriormente. Por último, se tiene el campo tokens que sirve para mantener la sesión del *administrador* iniciada mientras no expire o se cierre la sesión a través del botón cerrar sesión.

```
const mongoose = require('mongoose')
const validator = require('validator')
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    email: {
        type: String,
        unique: true,
        required: true,
        trim: true,
        lowercase: true,
        validate(value) {
            if (!validator.isEmail(value)) {
                throw new Error('Email is invalid')
            }
        }
    },
    password: {
        type: String,
        required: true,
        minlength: 7,
        trim: true,
        validate(value) {
            if (value.toLowerCase().includes('password')) {
                throw new Error('Password cannot contain "password"')
            }
        }
    },
    tokens: [{
        token: {
            type: String,
            required: true
        }
    }],
```

Código 2.18 Modelo de la base de datos administrador

3. RESULTADOS Y DISCUSIÓN

Para verificar la funcionalidad del prototipo se realizaron pruebas de funcionamiento y validación. Las pruebas de funcionamiento se hicieron para verificar el cumplimiento de las historias de usuario. Para la validación se realizó una encuesta orientada al uso de aplicación a 5 personas usuarios de la radio diferentes de los que fueron encuestados en la etapa de requerimientos.

Luego se procedió a la corrección de errores en caso de ser necesario en algún modulo del sistema.

3.1 ACTUALIZACIÓN DEL TABLERO KANBAN

La actualización del Tablero Kanban para esta fase se muestra en la Figura 3.1, en donde se muestran las actividades pendientes en la columna “Por Hacer”. Esta columna está definida por las tareas: pruebas de funcionamiento Módulo Radio, pruebas de funcionamiento Módulo Programación, pruebas de funcionamiento Módulo Compartir, pruebas de funcionamiento Módulo Administración.

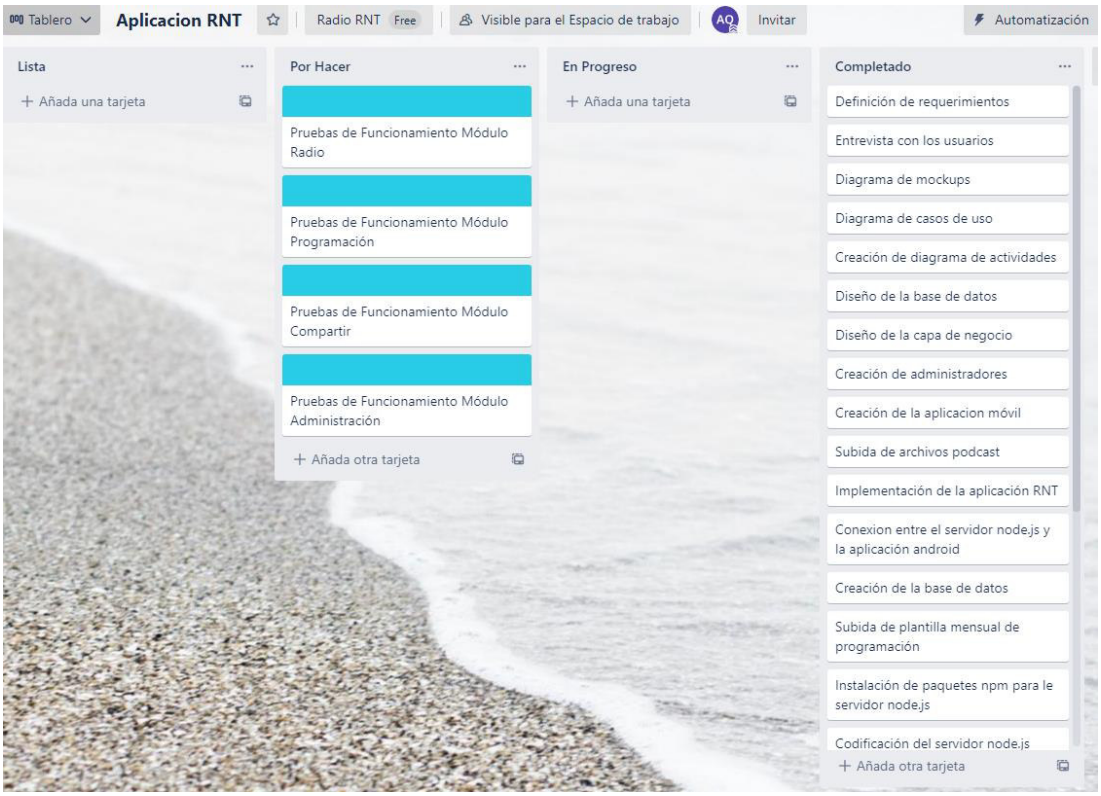


Figura 3.1 Actualización del tablero Kanban

3.2 PRUEBAS DE FUNCIONAMIENTO DE LOS MÓDULOS DEL PROTOTIPO

En esta sección se valida el correcto funcionamiento de los requerimientos de los módulos que componen la aplicación los cuales son: programación, radio, administración y compartir. En primer lugar, se detallarán los nombres de las historias de usuario de cada módulo y luego se realizará una descripción del resultado de cada prueba.

3.2.1 MÓDULO ADMINISTRACIÓN

Para validar el módulo administración un usuario *administrador* de la radio realizó las siguientes pruebas que se describen a continuación:

Prueba 1: Registro

Procedimiento:

1. Ingresar los datos imprescindibles para la inscripción de un nuevo administrador (ver Figura 3.2)
2. Presionar botón `SEND`

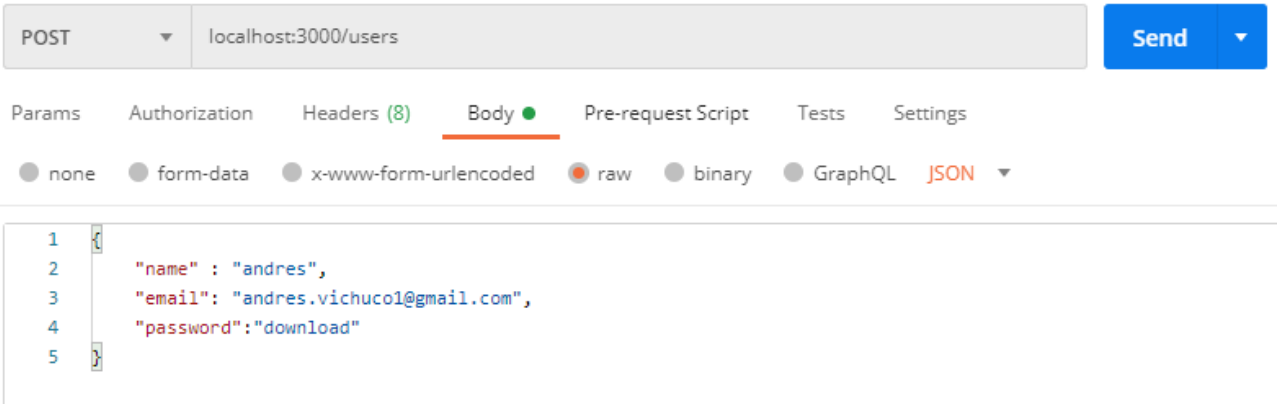


Figura 3.2 Ejemplo de registro de un administrador

Resultado: Como resultado de la prueba se pudo registrar el administrador en el sistema. Los datos ingresados se ven reflejados en la base de datos con su contraseña cifrada como se observa en la Figura 3.3.

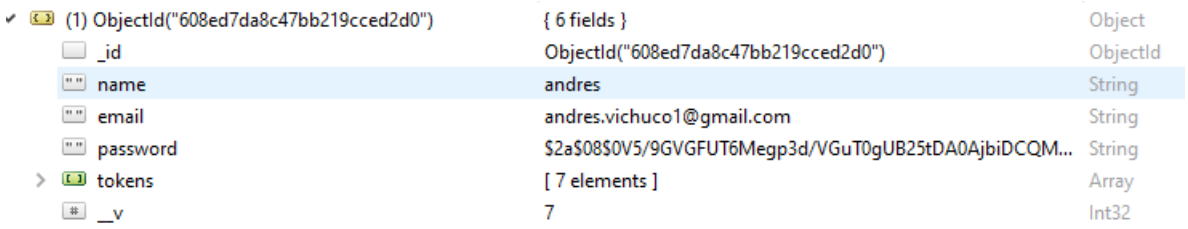


Figura 3.3 Registro de datos de un administrador en el sistema

Prueba 2: inicio de sesión

Procedimiento:

1. Ingresar el correo electrónico y la contraseña asociada de un administrador registrado (ver Figura 3.4)
2. Presionar botón `ingresar`

Figura 3.4 Ejemplo de autenticación

Resultado: Como resultado el administrador pudo iniciar sesión exitosamente y accedió a la página de administración de podcasts como se ve en la Figura 3.5

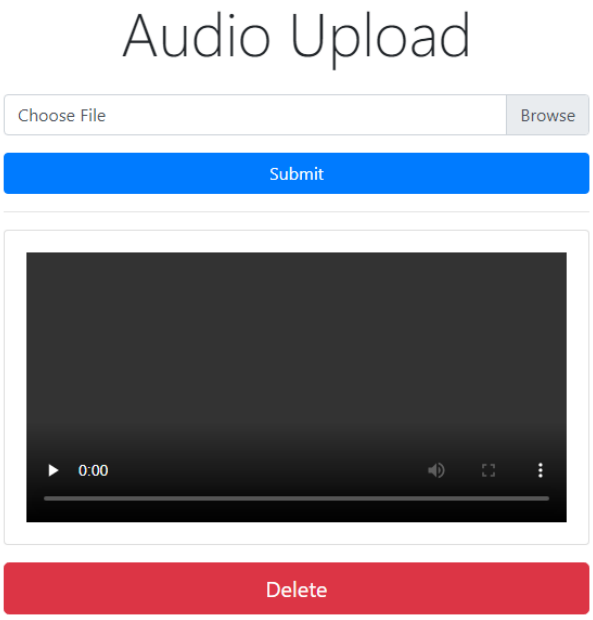


Figura 3.5 Ejemplo de podcast en la nube

Prueba 3: Inicio de sesión

Procedimiento:

1. Ingresar ya sea un correo o una contraseña incorrectos.
2. Presionar el botón `ingresar`

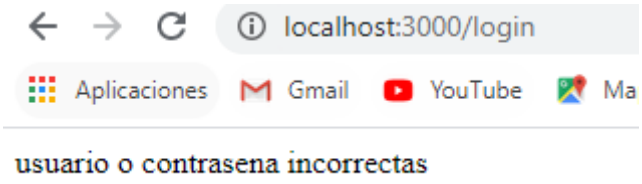


Figura 3.6 Ejemplo de inicio de sesión con datos no registrados

Resultado: Como resultado el administrador no pudo iniciar sesión y se le mostró un mensaje de error como se ve en la Figura 3.6

Prueba 4: Restablecimiento de cuenta de *administrador*

Procedimiento:

1. El administrador puede ingresar en los campos respectivos las nuevas credenciales que desee actualizar

2. Presiona el botón enviar (ver Figura 3.7)

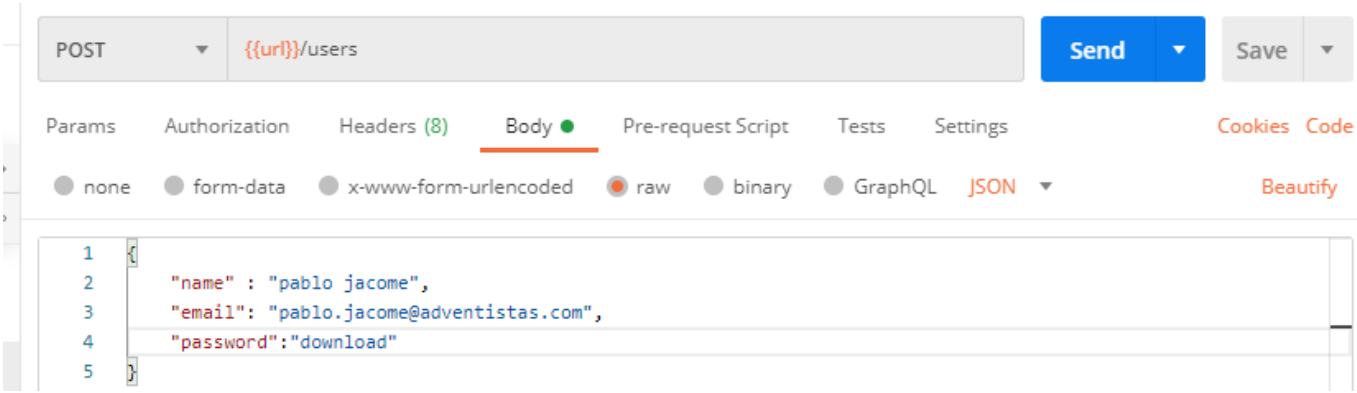


Figura 3.7 Ejemplo de restablecimiento de la cuenta de administrador

Resultado: Como resultado la cuenta se actualizó correctamente, ahora el sistema permite ingresar con las nuevas credenciales. En la Figura 3.8 se puede observar el password cifrado y un token asignado a la cuenta.

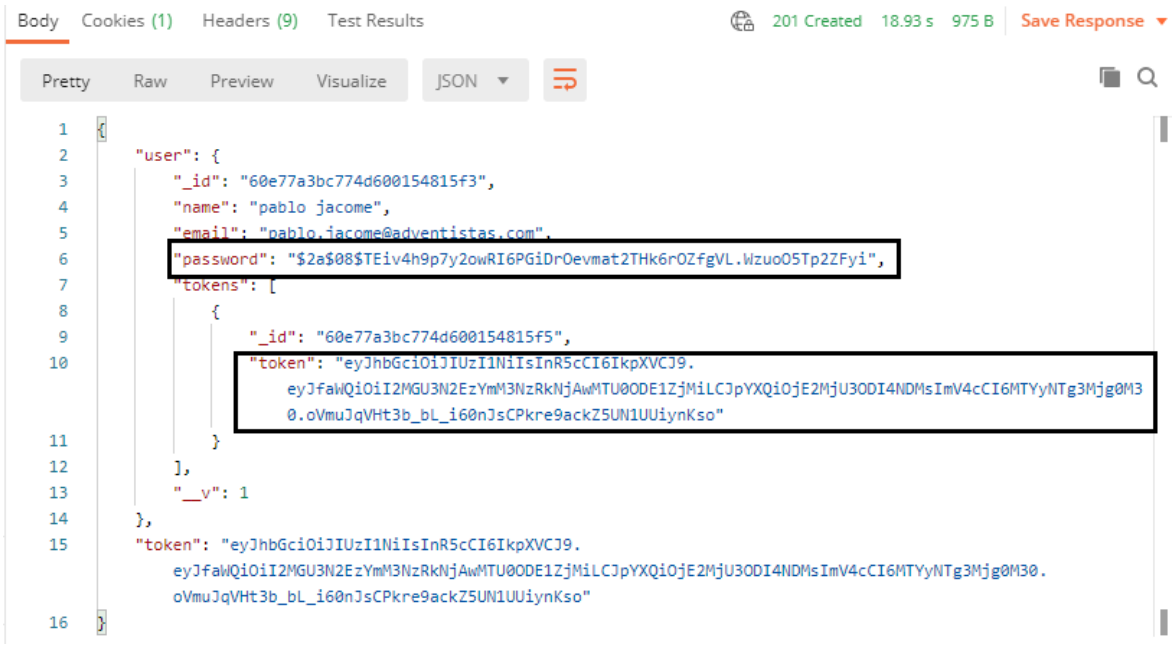


Figura 3.8 Ejemplo de cuenta de administrador actualizada y creada

Prueba 5: Administración de podcasts

Procedimiento:

1. Elegir el podcast a eliminar (ver Figura 3.9) o subir (Figura 3.10)

2. Presionar el botón `Delete` (ver Figura 3.9) o el botón `Submit` (ver Figura 3.10) según el caso correspondiente

Resultado: Como resultado el podcast ya no se visualiza en la página de multimedia (ver Figura 3.11) , o si se eligió subir el podcast se lo visualizará en la página principal de podcasts (ver Figura 3.12)

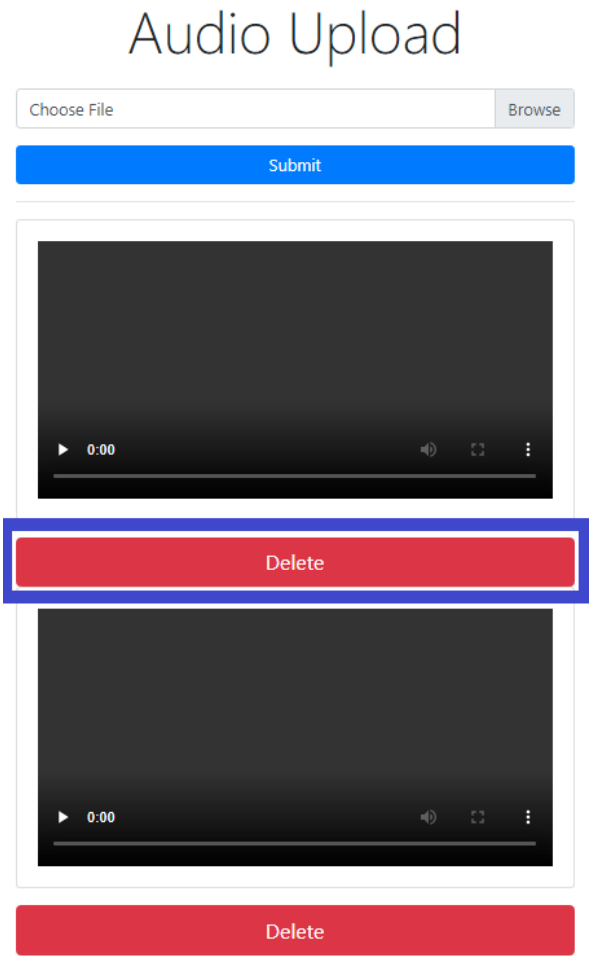


Figura 3.9 Ejemplo de elección de podcast a eliminar

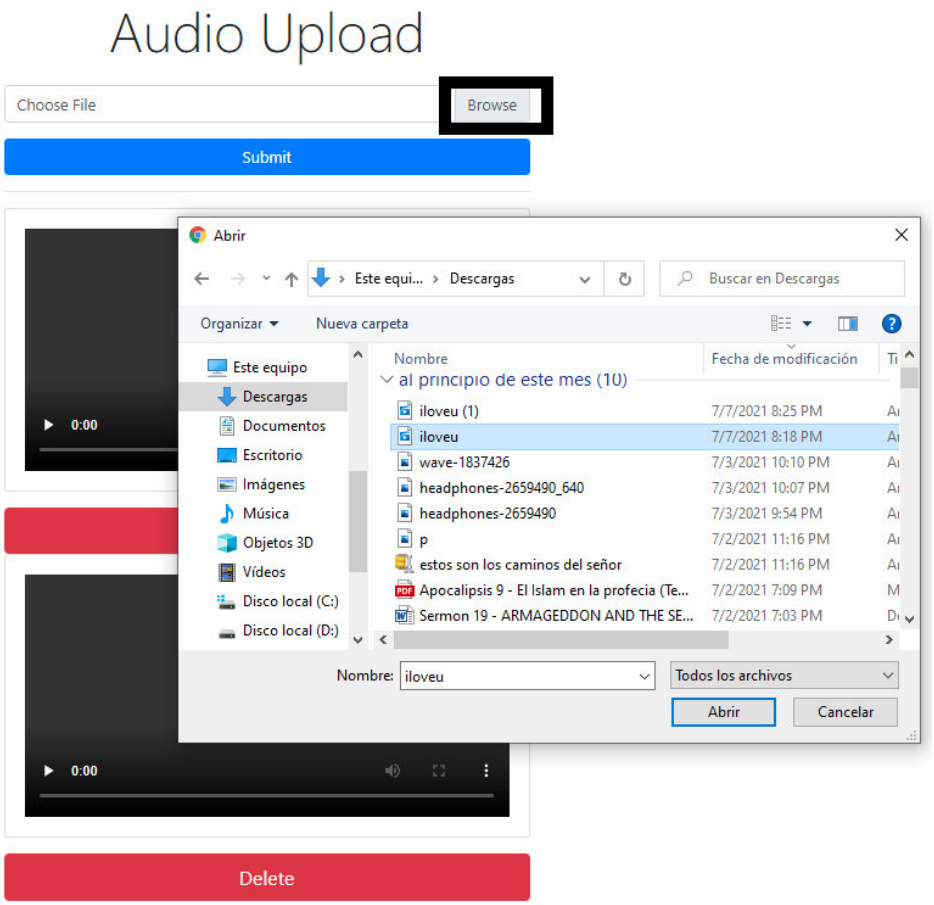


Figura 3.10 Ejemplo de elección de podcast a subir

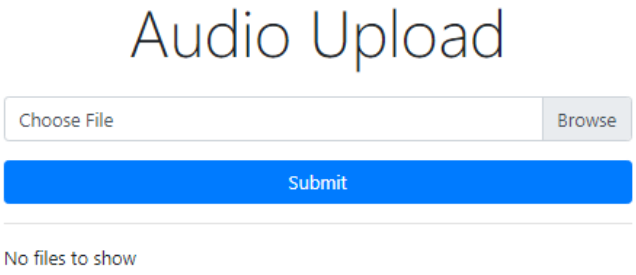


Figura 3.11 Ejemplo de podcasts eliminados

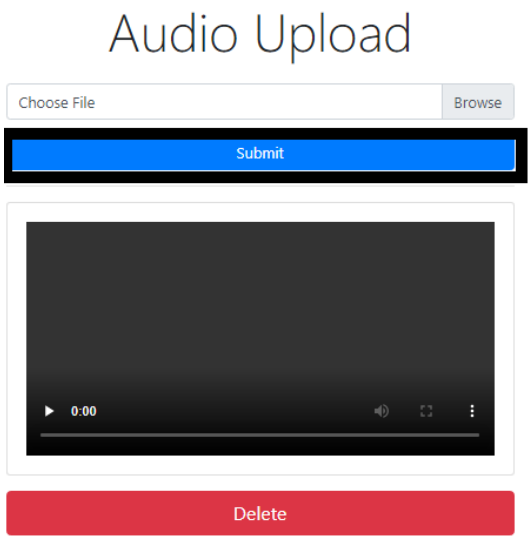


Figura 3.12 Ejemplo de subir un archivo hacia la base de datos

Prueba 6: Estado de los podcasts

Procedimiento:

1. Hago click en cualquier podcast para activarlo (ver Figura 3.13)

Resultado: Como resultado el podcast se puede reproducir.

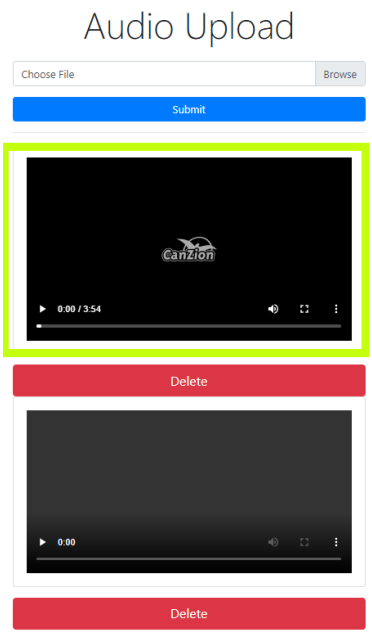


Figura 3.13 Ejemplo de activación de podcast

Prueba 7: Visualización de los podcasts

Procedimiento:

1. Dar clic sobre el botón `play` del podcast a reproducir (ver Figura 3.14)

Resultado: Como resultado se pudo reproducir el podcast que se elija.

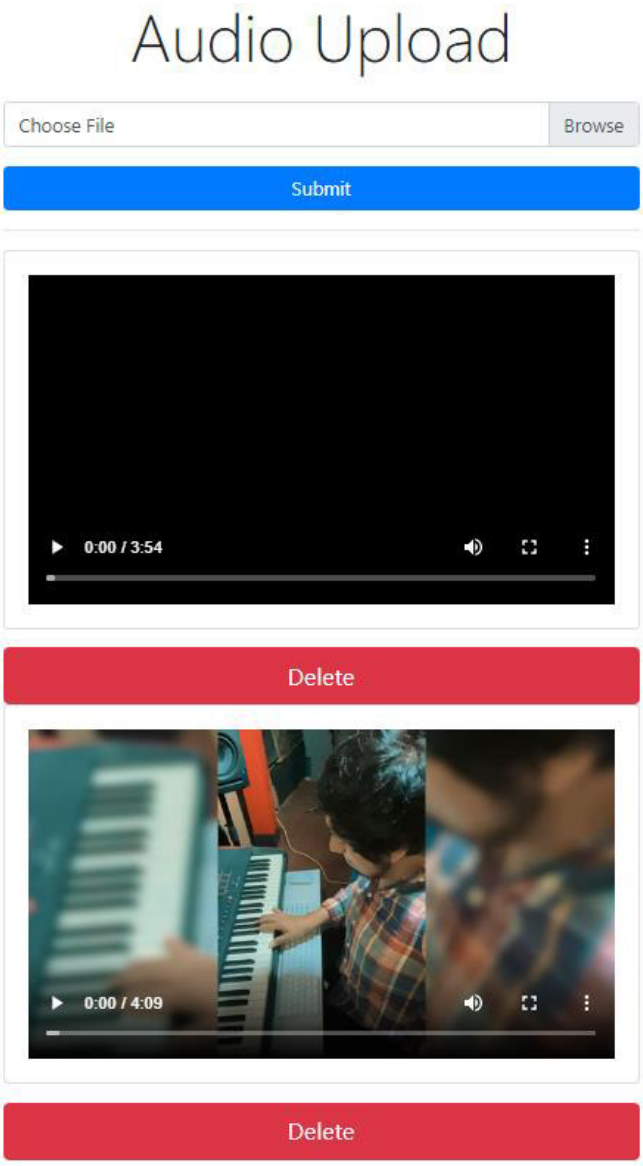


Figura 3.14 Ejemplo de visualización de podcasts

Prueba 8: Administración de la parrilla de programación

Procedimiento:

1. Ir a la página de administración de la parrilla para elegir el archivo Excel a subir
2. Elegir el documento y hacer clic en enviar (ver Figura 3.15)

Resultado: Como resultado el sistema permitió elegir el archivo a subir en una nueva ventana al hacer clic en `seleccionar archivo` (ver Figura 3.16)

Figura 3.15 Ejemplo de post de la parrilla de programación

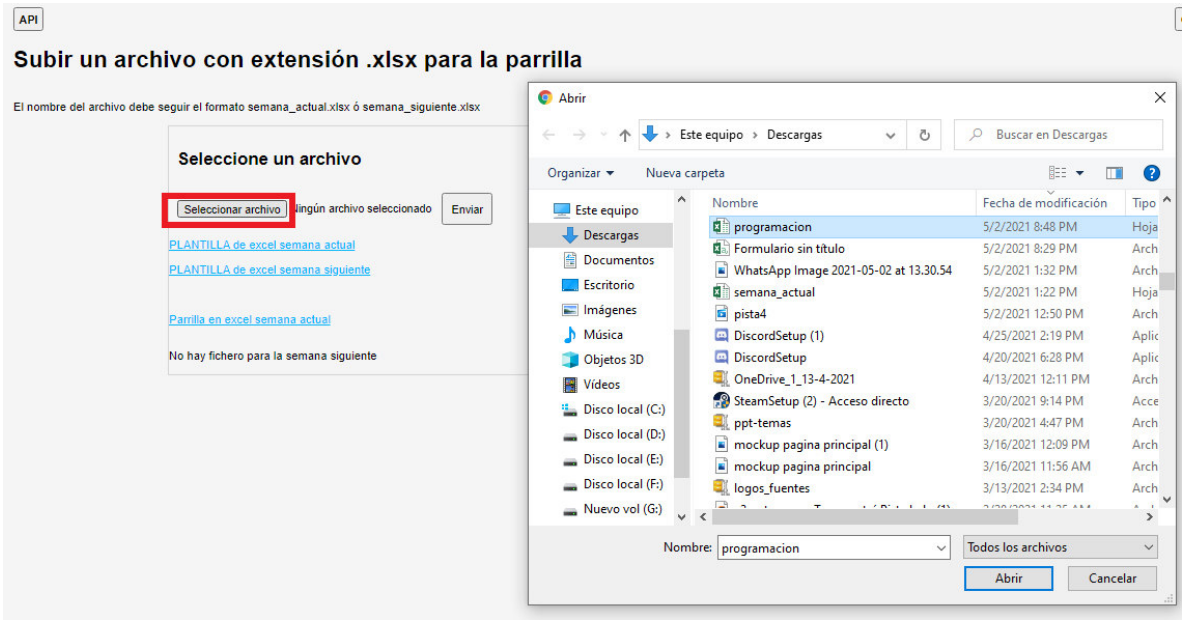


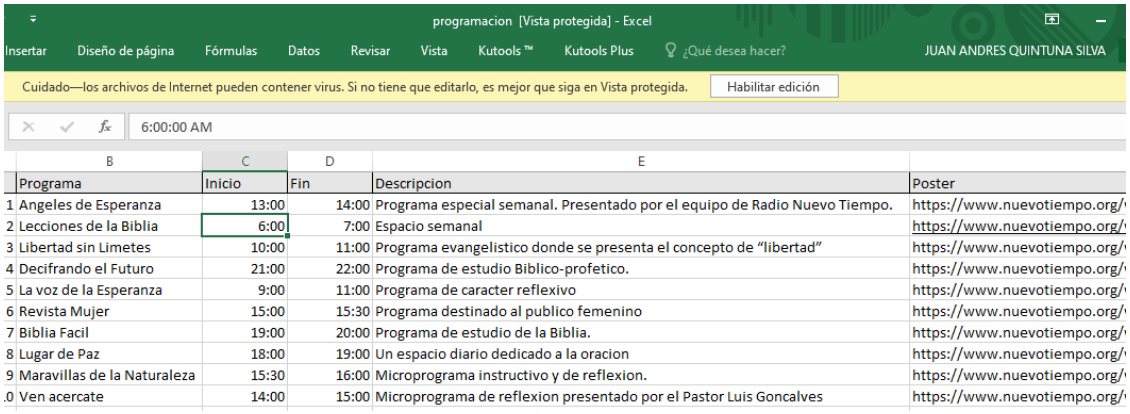
Figura 3.16 Ejemplo de subir archivo Excel al servidor

Prueba 9: Comprobación del formato del archivo de la parrilla de programación

Procedimiento:

1. Elegir un archivo Excel llenando el formato preestablecido y llenar la información requerida (Figura 3.17)

Resultado: El sistema aceptó solo archivo con formato Excel caso contrario muestra un error interno del sistema (Figura 3.18)



	B	C	D	E	
	Programa	Inicio	Fin	Descripcion	Poster
1	Angeles de Esperanza	13:00	14:00	Programa especial semanal. Presentado por el equipo de Radio Nuevo Tiempo.	https://www.nuevotiempo.org/
2	Lecciones de la Biblia	6:00	7:00	Espacio semanal	https://www.nuevotiempo.org/
3	Libertad sin Limetes	10:00	11:00	Programa evangelistico donde se presenta el concepto de "libertad"	https://www.nuevotiempo.org/
4	Decifrando el Futuro	21:00	22:00	Programa de estudio Biblico-profetico.	https://www.nuevotiempo.org/
5	La voz de la Esperanza	9:00	11:00	Programa de caracter reflexivo	https://www.nuevotiempo.org/
6	Revista Mujer	15:00	15:30	Programa destinado al publico femenino	https://www.nuevotiempo.org/
7	Biblia Facil	19:00	20:00	Programa de estudio de la Biblia.	https://www.nuevotiempo.org/
8	Lugar de Paz	18:00	19:00	Un espacio diario dedicado a la oracion	https://www.nuevotiempo.org/
9	Maravillas de la Naturaleza	15:30	16:00	Microprograma instructivo y de reflexion.	https://www.nuevotiempo.org/
10	Ven acercate	14:00	15:00	Microprograma de reflexion presentado por el Pastor Luis Goncalves	https://www.nuevotiempo.org/

Figura 3.17 Ejemplo de parrilla con formato en Excel

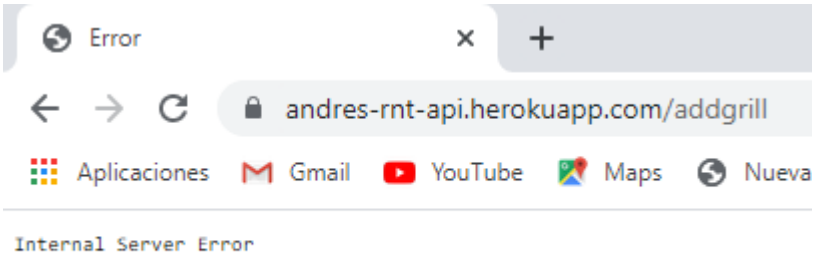


Figura 3.18 Ejemplo de error de formato de archivo

Prueba 10: Notificación de archivo subido con éxito

Procedimiento:

1. Seleccionar el archivo a subir en el formato correcto
2. Presionar botón `Enviar` (ver Figura 3.19)

Resultado: Como resultado se subió el archivo con éxito y se puede comprobar por el mensaje mostrado en la misma página de administración de parrilla de programación (ver Figura 3.20)

Figura 3.19 Ejemplo de subir archivo Excel al servidor

Figura 3.20 Ejemplo de subida de archivo exitosa

Prueba 11: Salir del sistema

Procedimiento:

1. Seleccionar botón de `cerrar sesión` (ver Figura 3.21)

Resultado: El sistema redirige al administrador a la página de login (ver Figura 3.22)

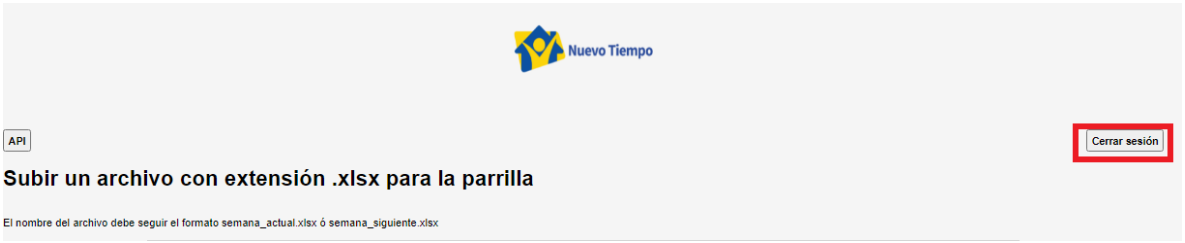


Figura 3.21 Ejemplo de cerrar sesión

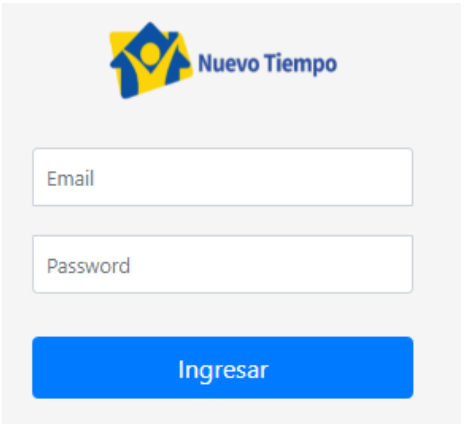


Figura 3.22 Ejemplo de página de login después de cerrar sesión

Prueba: Visualización de guía para el uso de la aplicación (api)

Procedimiento:

1. Seleccionar botón `API` (ver Figura 3.23)

Resultado: El sistema redirige al administrador a la guía del sistema (ver Figura 3.24)

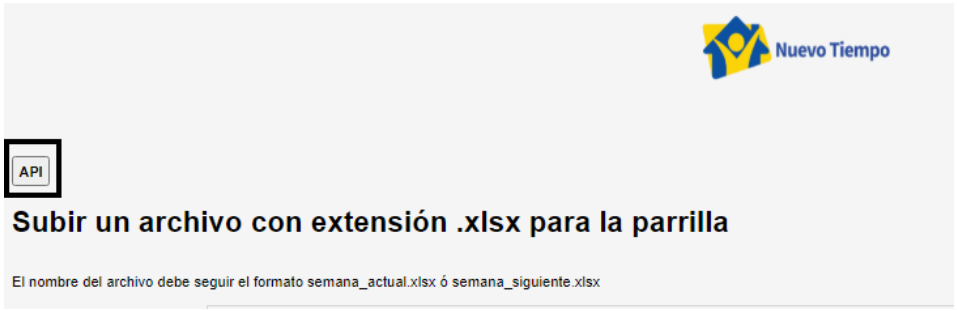


Figura 3.23 Ejemplo de guía API

Figura 3.24 Ejemplo de guía API

3.2.2 MÓDULO RADIO

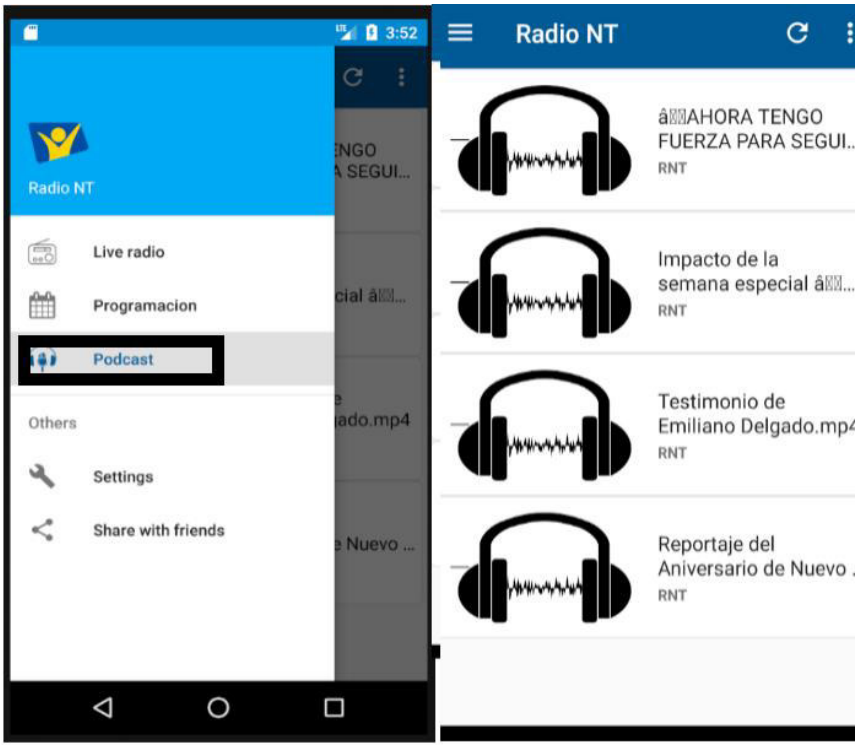
Para validar el funcionamiento de este módulo un usuario *cliente* realizó las siguientes pruebas:

Prueba 1: Visualización del contenido del podcast

Procedimiento:

1. Ingresa a la aplicación
2. Selecciono el menú `podcast` (ver Figura 3.25 opción a)

Resultado: La aplicación permite visualizar los podcasts (Figura 3.25 opción b)



a) b)

Figura 3.25 Ejemplo de opción podcast en la aplicación móvil a) selección de opción podcast b) lista de podcasts disponibles

Prueba 2: Visualización de página principal de la aplicación

Procedimiento:

1. Ingresa a la aplicación

Resultado: Como resultado la aplicación muestra la pantalla principal (Figura 3.26)

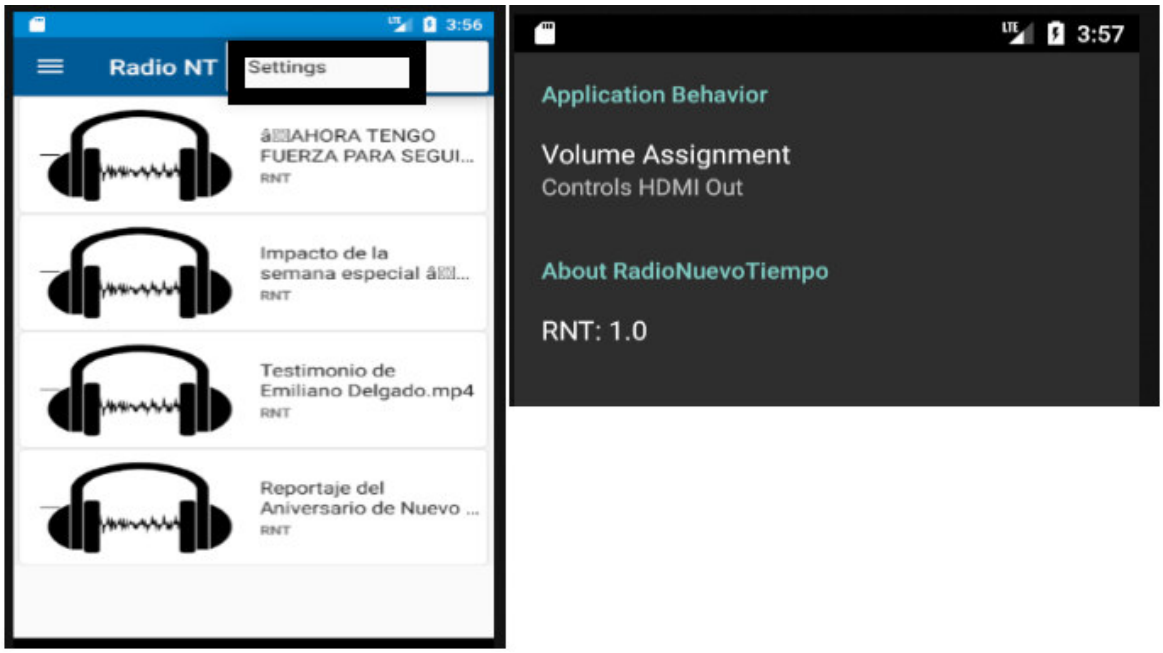
Figura 3.26 Página principal de la aplicación RNT

Prueba 3: Configuración de la aplicación

Procedimiento:

1. Ingresa a la aplicación y selecciona el menú de configuración en la parte superior (ver Figura 3.27 opción a)

Resultado: La aplicación muestra un menú en donde se puede elegir que dispositivo tendrá el control de volumen (ver Figura 3.27 opción b)



a) b)

Figura 3.27 Menú de configuraciones a) selección de opción configuración b) menú desplegable para seleccionar el mando del volumen

3.2.3 MODULO COMPARTIR

Para validar este módulo se realizó la siguiente prueba:

Prueba 1: Compartir aplicación

Procedimiento:

1. Seleccionar la opción compartir con amigos en el menú principal (ver Figura 3.28)
2. Seleccionar el medio a compartir (Figura 3.29)

Resultados:

1. Se pudo ingresar al módulo compartir
2. Se pudo enviar el enlace de la página de la radio por medio de Messenger

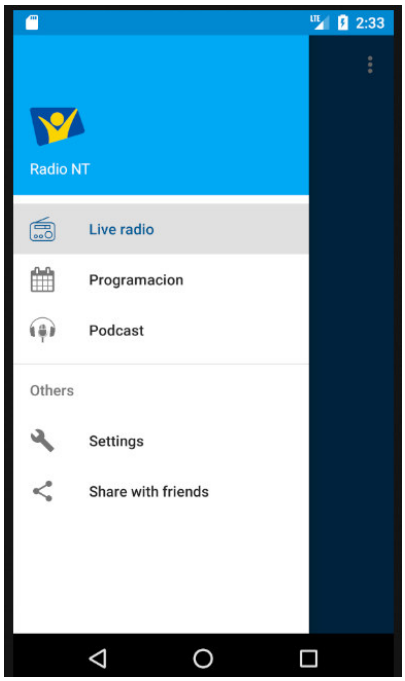


Figura 3.28 Ejemplo de la opción compartir con amigos

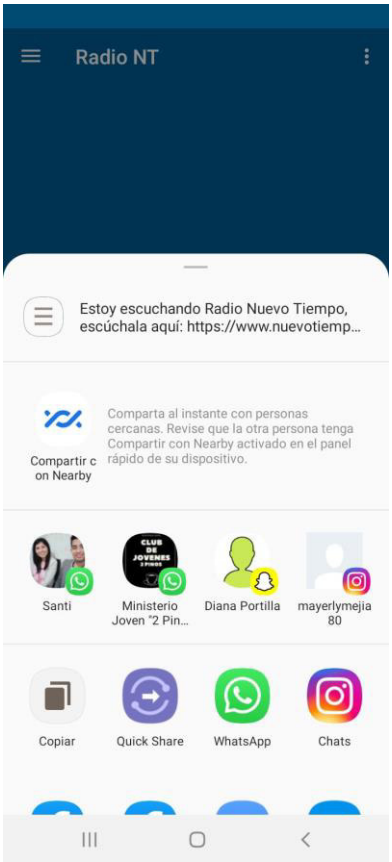


Figura 3.29 Ejemplo de selección de red social por la cual compartir

3.2.4 MÓDULO PROGRAMACIÓN

Para validar este módulo se realizó la siguiente prueba:

Prueba 1:

1. Visualizar la parrilla mensual de programación
2. Visualizar los detalles de un programa en particular

Procedimiento:

1. Seleccionar la opción `programación` en el menú principal (ver Figura 3.30)
2. Elegir algún programa de las opciones disponibles (ver Figura 3.31)
3. Visualizar los detalles del programa seleccionado (ver Figura 3.32)

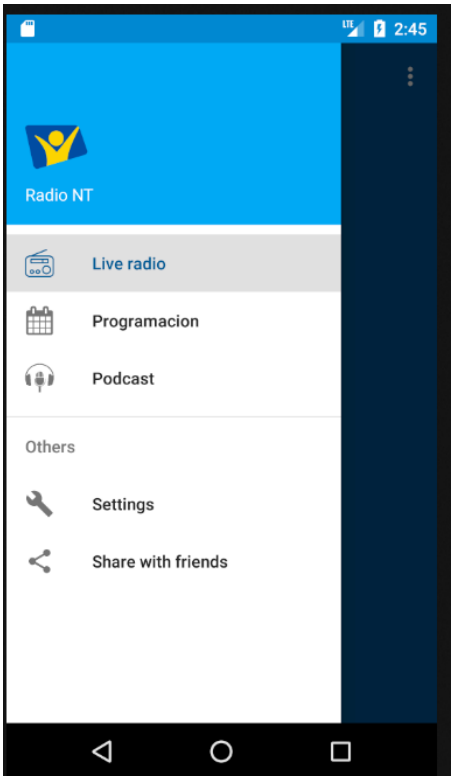


Figura 3.30 Ejemplo de menú principal

Figura 3.31 Ejemplo de parrilla de programación

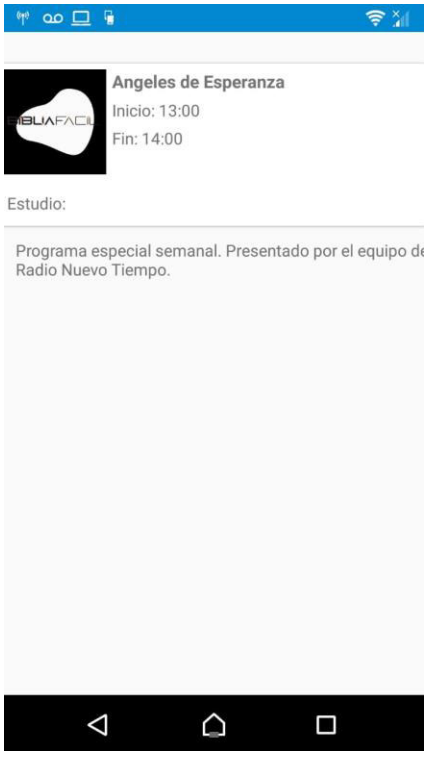


Figura 3.32 Ejemplo del detalle de un programa de la radio Nuevo Tiempo

3.3 RESULTADOS DE LAS ENCUESTAS Y EXPERIENCIA DE USO

Las pruebas de funcionamiento del prototipo se realizaron a 10 usuarios, siendo estos 9 radioescuchas de la radio y 1 administrador. La cuenta del administrador que fue el primer usuario creado en la base de datos probó las funciones de administración de podcasts y parrilla de programación del sistema. Finalmente, los *usuarios clientes* realizaron el proceso de escuchar la radio en vivo, reproducción de un podcast en un dispositivo Chromecast, compartición de la aplicación por redes sociales y visualización de la parrilla de programación.

Una vez realizadas las pruebas de funcionamiento, los usuarios llenaron las encuestas de funcionalidad y experiencia de uso de acuerdo con el rol que desempeñaron en el sistema. El resultado de las encuestas, presentado en el Anexo 2, fue determinante para conocer acerca de la utilidad y funcionamiento del prototipo, debido a que las encuestas fueron orientadas a determinar el comportamiento de los módulos del sistema de acuerdo con las historias de usuario realizadas en la etapa de levantamiento de requisitos.

Posteriormente, se hizo las pruebas de envío a un dispositivo Chromecast por parte de un usuario *administrador*, el resultado fue que en un 50% es medianamente fácil el proceso de envío de contenido multimedia y un 30% fácil, lo cual indica que se tiene un nivel de aceptación de un 80%.

El envío de contenido a un dispositivo Chromecast es importante por esta razón, una de las preguntas al respecto arrojó los siguientes resultados: 50% de los usuarios encontraron medianamente fácil el proceso de enviar contenido al dispositivo, 30 % fácil y el 20% difícil debido a que no estaban familiarizados con el dispositivo Chromecast.

La recomendación del administrador en cuanto al sistema es muy importante, por esta razón una de las preguntas al respecto arrojó los siguientes resultados: 100% recomendaría el sistema, con lo que podemos concluir que el prototipo web para administrador ha cumplido con lo esperado en este trabajo.

Al analizar los resultados obtenidos de las encuestas de usuarios *clientes* se pudo determinar:

- El módulo radio tuvo un cien por ciento de funcionalidad dado que todos los usuarios pudieron escuchar la radio sin ningún inconveniente.

- Todos los usuarios con dispositivos Chromecast en sus hogares pudieron reproducir los podcasts en su Smart tv.
- Todos los usuarios pudieron visualizar la parrilla mensual de programación de la radio y sus respectivos programas y horarios.
- Las opciones de configurar y compartir por redes sociales se ejecutaron correctamente en su totalidad.

Al analizar los resultados obtenidos de la encuesta de usuario *administrador* se pudo determinar:

- El proceso de autenticación tuvo un cien por ciento de funcionalidad dado a que los usuarios administradores pudieron iniciar sesión, siendo identificados y direccionados a la página propia del rol.
- La administración de podcast, la subida y eliminación de podcasts, se realizó en un cien por ciento.
- El administrador pudo subir la parrilla de programación correctamente.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El objetivo principal de este proyecto era la realización de una aplicación que pueda enviar contenido multimedia a un dispositivo Chromecast, en resumen, una aplicación para la radio Nuevo Tiempo. Tras hacer el análisis se llegó a la conclusión que no sería posible realizar la aplicación sin crear un servidor nuevo al ya existente de la radio, es por esto que se añadió un servidor que permita crear las parrillas de programación mensual para la radio, dando la posibilidad de que los administradores de la radio subieran esta información de una manera sencilla.

- Los requisitos funcionales se cumplieron de tal manera que el sistema permita la creación o eliminación de cuentas de administrador, visualizar los podcasts de la radio y reproducirlos en la aplicación móvil o a través del Chromecast en la tv, escuchar la radio en vivo y visualizar la parrilla de programación mensual de la radio.

- Para la construcción de las interfaces finales de usuario se usó los paquetes ejs, y pug como lenguaje de etiquetas HTML. El aspecto visual de las vistas HTML se mejoró al hacer uso de CCS3 para dar estilo y las bibliotecas Bootstrap que brindan una mejor estructura y adaptabilidad a las interfaces del administrador.

- Para el despliegue del proyecto se usaron los servicios de computación en la nube provistos por Heroku, lo que facilitó en gran medida este proceso debido a las características como comandos por consola, soporte, abundante documentación en internet, gratuidad y una versión de producción para el presente proyecto.

- Para la lógica de la base de datos se usó mongoDB Atlas, servicio para las bases de datos no relacionales cuya principal característica es la facilidad de conexión con el entorno Node.js y su escalabilidad para el almacenamiento de datos multimedia.

- La aplicación web prototipo permite tener un servidor de videos para la radio en donde se puede alojar los podcasts producidos por la radio nuevo tiempo, mejorando la experiencia de radio escuchas que de acuerdo a las encuestas realizadas un ochenta por ciento de los usuarios calificaron de fácil el uso de la aplicación, y un setenta por ciento calificaron de sencillo el proceso de envió de contenido multimedia a un dispositivo Chromecast lo cual indica un índice de aprobación favorable.

4.1 RECOMENDACIONES

- Desarrollar un módulo en el cual se realice donaciones a la radio de manera regular a través de un sistema seguro de transacciones como PayPal, lo cual brindaría al equipo de radio nuevo tiempo información de sus ingresos mensuales a través de este medio para la proyección financiera de la misma.

- Crear una funcionalidad a la aplicación que permita valorar el contenido que observan los usuarios, de esta manera se mostraría a través de gráficos las calificaciones de los usuarios acerca del contenido propuesto por la radio, información que probablemente se utilizaría para toma de decisiones por parte del equipo de la radio nuevo tiempo.

- Programar la misma aplicación para un entorno IOS usando React Native que es un entorno de trabajo la cual permite el desarrollo en diferentes sistemas operativos móviles, con lo cual se llegaría a más usuarios y no solamente a los usuarios con dispositivos Android como está desarrollado actualmente.

- Optar por la posibilidad de obtener una base de datos pagada usando la misma opción de atlas la cual ofrece un servicio pagado a través de Amazon, con lo cual se almacenaría mayor cantidad de podcasts sin límite de espacio en disco como se lo maneja actualmente. La radio almacenaría programas enteros de más de una hora de duración sin ningún inconveniente.

5. REFERENCIAS

[1] K. Kakar, «Online Radio Streaming Apps: Benefits and Opportunities». https://insights.daffodilsw.com/blog/online-radio-streaming-apps-benefits-and-opportunities (accedido abr. 13, 2021).

[2] «Nuestra Historia - Nuevo Tiempo». https://www.nuevotiempo.org/historia/ (accedido ene. 13, 2021).

[3] S. Weber, «Chromecast Users Manual | Guide books». https://dl.acm.org/doi/book/10.5555/2655335 (accedido ene. 13, 2021).

[4] R. Ferguson, *Beginning JavaScript: The Ultimate Guide to Modern JavaScript Development*. Berkeley, CA: Apress, 2019. doi: 10.1007/978-1-4842-4395-4.

[5] S. Tilkov y S. Vinoski, «Node.js: Using JavaScript to Build High-Performance Network Programs», *IEEE Internet Comput.*, vol. 14, n.º 6, pp. 80-83, nov. 2010, doi: 10.1109/MIC.2010.145.

[6] A. Mardan, *Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development*. Apress, 2014. doi: 10.1007/978-1-4842-0037-7.

[7] W. Jackson, *Pro Android Wearables*. Berkeley, CA: Apress, 2015. doi: 10.1007/978-1-4302-6551-1.

[8] X. Zhao y D. Tian, «The Architecture Design of Streaming Media Applications for Android OS», p. 4.

[9] *microsoft/vscode*. Microsoft, 2021. Accedido: ene. 17, 2021. [En línea]. Disponible en: https://github.com/microsoft/vscode

[10] J. M. de la Rosa-Trevín *et al.*, «Scipion: A software framework toward integration, reproducibility and validation in 3D electron microscopy», *Journal of Structural Biology*, vol. 195, n.º 1, pp. 93-99, jul. 2016, doi: 10.1016/j.jsb.2016.04.010.

[11] A. Del Sole, *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*. Berkeley, CA: Apress, 2019. doi: 10.1007/978-1-4842-4224-7.

[12] M. Tsitoara, *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress, 2020. doi: 10.1007/978-1-4842-5313-7.

[13] «GitHub: Where the world builds software», *GitHub*. https://github.com/ (accedido ene. 18, 2021).

[14] J. Ali, *Instant Node Package Manager*. Packt Publishing Ltd, 2013.

[15] A. Shtuchkin, *ashtuchkin/iconv-lite*. 2021. Accedido: ene. 25, 2021. [En línea]. Disponible en: https://github.com/ashtuchkin/iconv-lite

[16] «GitHub - expressjs/multer: Node.js middleware for handling `multipart/form-data`.» https://github.com/expressjs/multer#readme (accedido ene. 25, 2021).

[17] «nodemon». https://nodemon.io/ (accedido ene. 25, 2021).

[18] *Automattic/mongoose*. Automattic, 2021. Accedido: ene. 25, 2021. [En línea]. Disponible en: https://github.com/Automattic/mongoose

[19] S. G. Edward y N. Sabharwal, *Practical MongoDB*. Berkeley, CA: Apress, 2015. doi: 10.1007/978-1-4842-0647-8.

[20] T. Baar, A. Strohmeier, y A. Moreira, Eds., *UML 2004 - The Unified Modeling Language: Modeling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings*. Berlin Heidelberg: Springer-Verlag, 2004. doi: 10.1007/b101232.

[21] «All articles | Heroku Dev Center». https://devcenter.heroku.com/articles (accedido feb. 04, 2021).

[22] A. Mardan, *Practical Node.js: Building Real-World Scalable Web Apps*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3039-8.

[23] E. Wilde y C. Pautasso, Eds., *REST: From Research to Practice*. New York, NY: Springer New York, 2011. doi: 10.1007/978-1-4419-8303-9.
[24] «Descripción general de AndroidX | Desarrolladores de Android», *Android Developers*. https://developer.android.com/jetpack/androidx?hl=es (accedido abr. 25, 2021).
[25] Y. Monden, *Toyota Production System*. Boston, MA: Springer US, 1993. doi: 10.1007/978-1-4615-9714-8.
[26] M. Seidl, M. Scholz, C. Huemer, y G. Kappel, *UML @ Classroom: An Introduction to Object-Oriented Modeling*, 1st ed. 2015. Cham: Springer International Publishing : Imprint: Springer, 2015. doi: 10.1007/978-3-319-12742-2.
[27] *googlecast/CastVideos-android*. Google Cast, 2021. Accedido: ene. 25, 2021. [En línea]. Disponible en: https://github.com/googlecast/CastVideos-android
[28] G. Allen, «Android Fragments», en *Beginning Android*, Berkeley, CA: Apress, 2015, pp. 181-195. doi: 10.1007/978-1-4302-4687-9_11.
[29] D. Smith y E. Hellman, *Android Recipes*. Berkeley, CA: Apress, 2016. doi: 10.1007/978-1-4842-2259-1.
[30] «MongoDB Limits and Thresholds — MongoDB Manual», *https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt*. https://docs.mongodb.com/manual/reference/limits/ (accedido jun. 08, 2021).

6. ANEXOS

Los anexos se incluyen en el CD adjunto al presente documento.

ANEXO 1. Historias de Usuario

ANEXO 2. Requerimientos de Usuario