**ESCUELA SUPERIOR DE INGENIERÍA**

**INGENIERÍA INFORMÁTICA**

**jUCAbox**

**jukebox Social**

AUTOR: Sergio Ruiz Piulestán

30 de septiembre de 2017

**ESCUELA SUPERIOR DE INGENIERÍA**

**INGENIERÍA INFORMÁTICA**

**jUCAbox**

**jukebox Social**

AUTOR: Sergio Ruiz Piulestán DIRECTOR: Iván Ruiz Rube

Cádiz, 30 de septiembre de 2017

***Agradecimientos***

Quisier agradecer a mis padres, amigos, familia, y sobre todo a Iván Ruiz, mi tutor del proyecto, por el tiempo y apoyo dedicados para poder realizar el proyecto.

Índice general

[Parte I – Prolegómeno 1](#_Toc487217008)

[Capítulo 1 – Introducción 2](#_Toc487217010)

[1.1. Motivación 3](#_Toc487217012)

[1.2. Alcance 3](#_Toc487217013)

[1.3. Glosario de Términos 3](#_Toc487217014)

[1.4. Organización del documento 4](#_Toc487217015)

[Capítulo 2 – Planificación 6](#_Toc487217016)

[2.1. Metodología de desarrollo 6](#_Toc487217018)

[2.2. Planificación del proyecto 7](#_Toc487217019)

[2.3. Organización 10](#_Toc487217020)

[2.4. Costes 11](#_Toc487217021)

[2.5. Riesgos 12](#_Toc487217022)

[2.6. Aseguramiento de calidad 13](#_Toc487217023)

[Parte II – Desarrollo 14](#_Toc487217024)

[Capítulo 3 – Requisitos del Sistema 13](#_Toc487217026)

[3.1. Situación actual 13](#_Toc487217028)

[3.1.1. Procesos de Negocio 13](#_Toc487217029)

[3.1.2. Entorno Tecnológico 13](#_Toc487217030)

[3.2. Necesidades de Negocio 14](#_Toc487217031)

[3.2.1. Objetivos de Negocio 14](#_Toc487217032)

[3.3. Objetivos del Sistema 14](#_Toc487217033)

[3.4. Catálogo de Requisitos 14](#_Toc487217034)

[3.4.1. Requisitos funcionales 14](#_Toc487217035)

[3.4.2. Requisitos no funcionales 15](#_Toc487217036)

[3.4.3. Reglas de negocio 16](#_Toc487217037)

[3.4.4. Requisitos de información 16](#_Toc487217038)

[3.5. Alternativas de Solución 17](#_Toc487217039)

[3.6. Solución Propuesta 17](#_Toc487217040)

[Capítulo 4 – Análisis del Sistema 18](#_Toc487217041)

[4.1. Modelo Conceptual 18](#_Toc487217043)

[4.2. Modelo de Casos de Uso 20](#_Toc487217044)

[4.2.1. Actores 20](#_Toc487217045)

[4.2.2. Casos de Uso 21](#_Toc487217046)

[4.3. Modelo de Comportamiento 31](#_Toc487217048)

[4.3.1. Acceso y salida del sistema 32](#_Toc487217050)

[4.3.2. Búsqueda y envío de canciones 33](#_Toc487217051)

[4.3.3. Gestión de usuarios 34](#_Toc487217052)

[4.3.4. Gestión de lugares 36](#_Toc487217053)

[4.3.5. Contrato de operaciones 38](#_Toc487217054)

[4.4. Modelo de Interfaz de Usuario 43](#_Toc487217055)

[Capítulo 5 – Diseño del Sistema 51](#_Toc487217056)

[5.1. Arquitectura del Sistema 51](#_Toc487217058)

[5.1.1. Arquitectura Física 51](#_Toc487217059)

[5.1.2. Arquitectura Lógica 51](#_Toc487217060)

[5.2. Diseño físico de datos 52](#_Toc487217061)

[5.3. Diseño detallado de Componentes 58](#_Toc487217062)

[5.4. Diseño detallado de la Interfaz de Usuario 72](#_Toc487217063)

[Capítulo 6 73](#_Toc487217064)

[Construcción del Sistema 73](#_Toc487217065)

[6.1. Entorno de Construcción 73](#_Toc487217066)

[6.2. Código Fuente 74](#_Toc487217067)

[6.2.1. Estructura de ficheros 74](#_Toc487217072)

[6.3. Scripts de Base de datos 79](#_Toc487217073)

[Capítulo 7 80](#_Toc487217074)

[Pruebas del Sistema 80](#_Toc487217075)

[7.1. Estrategia 80](#_Toc487217076)

[7.2. Entorno de Pruebas 80](#_Toc487217077)

[7.3. Roles 80](#_Toc487217078)

[7.4. Niveles de Pruebas 81](#_Toc487217079)

[7.4.1. Pruebas Unitarias 81](#_Toc487217080)

[7.4.2. Pruebas de Integración 81](#_Toc487217081)

[7.4.3. Pruebas de Sistema 81](#_Toc487217082)

[Pruebas funcionales 81](#_Toc487217083)

[Pruebas no funcionales 82](#_Toc487217084)

[7.4.4. Pruebas de Aceptación 85](#_Toc487217085)

[Parte III 86](#_Toc487217086)

[Epílogo 86](#_Toc487217087)

[Capítulo 8 29](#_Toc487217088)

[Manual de implantación y explotación 29](#_Toc487217089)

[8.1. Introducción 29](#_Toc487217090)

[8.2. Requisitos previos 29](#_Toc487217091)

[8.3. Inventario de componentes 29](#_Toc487217092)

[8.4. Procedimientos de instalación 30](#_Toc487217093)

[8.5. Pruebas de implantación 40](#_Toc487217094)

[8.6. Procedimientos de operación y nivel de servicio 40](#_Toc487217095)

[Capítulo 9 41](#_Toc487217096)

[Manual de usuario 41](#_Toc487217097)

[9.1. Introducción 41](#_Toc487217098)

[9.2. Instalación 41](#_Toc487217099)

[9.3. Uso del sistema 41](#_Toc487217100)

[Capítulo 10 43](#_Toc487217101)

[Conclusiones 43](#_Toc487217102)

[10.1. Objetivos alcanzados 43](#_Toc487217103)

[10.2. Lecciones aprendidas 43](#_Toc487217104)

[10.3. Trabajo futuro 43](#_Toc487217105)

[Bibliografía 45](#_Toc487217106)

Índice de figuras

Índice de tablas

# Parte I

# Prolegómeno

# Capítulo 1

# Introducción

El presente documento detalla el proceso que se ha seguido para la creación de la herramienta jUCAbox.

En la actualidad, muchos locales tienen sistemas de reproducción musical. Cuando un cliente quiere solicitar alguna canción especial, tiene que ir al encargado de la música y decirle el nombre de la canción y el artista que desea. Conllevaría la interrupción de la labor desempeñada por el encargado de la música, así como la dificultad por el ruido y el ambiente en el local de poder transmitir que canción desea.

Por otro lado, está creciendo la necesidad de solicitar canciones antes de realizar un evento social, como una boda, una comunión, o simplemente una fiesta entre amigos. Por lo que el organizador, debería pedir, mediante correos, mensajes y/o personalmente, las canciones que le gustaría que sonaran en el evento.

La herramienta **jUCAbox** es un Jukebox social, una aplicación pensada para pubs, discotecas y eventos sociales.

Con **jUCAbox**, se pretende paliar estos problemas. El organizador del evento, o el encargado del sistema musical del local, registrarían su evento/local en la herramienta y automaticamente esta disponible para que los usuarios puedan solicitar/registrar las canciones deseadas. Evitando el tráfico de información que pueda ser susceptible de perderse.

Con **jUCAbox** se podrá interactuar con el DJ en tiempo real para solicitar canciones, intercambiar mensajes con otros usuarios de la herramienta, e incluso, el propio local podrá añadir promociones para favorecer a que el cliente use la herramienta.

A su vez, se podrán enviar canciones a una lista de reproducción futura, no teniendo que ser en el momento del evento, para su posterior validación. Este listado de canciones, son todas las disponibles en el ámbito de Spotify, ya que la herramienta se conecta a través de un API proporcionado por la misma.

**jUCAbox** se compone de 3 partes:

* **Búsqueda de canciones**: El usuario podrá buscar todas las canciones disponibles en Spotify, para luego poder enviarla a un evento de entre los registrados en la herramienta. Esta funcionalidad no requiere que el usuario esté registrado en jUCAbox. Si quiere guardar información de que canciones ha enviado, o guardar lugares y artistas favoritos, entonces si deberá registrarse.
* **Búsqueda de lugares**: El usuario podrá buscar todos los lugares dados de alta en la herramienta, para poder ver una descripción más detallada del mismo y canciones enviadas y top de canciones entre rangos de fechas. El usuario podrá crear lugares y administrarlos, para controlar las canciones que le envien y postear mensajes a los usuarios que tengan al lugar como favorito.
* **Usuario**: Tendrá un apartado para ver los datos referente a su usuario. Log de actvidad, lugares y artistas favoritos, y asi como los amigos que tenga agregados para empezar un chat si están en linea.

## **Motivación**

Las motivaciones para realizar esta aplicación han sido diversas.

Conocer nuevos frameworks de desarrollo web, nuevos modelos de datos no convencionales, como son los modelos de base de datos no relaciones, y poder explotar toda esa información mediante API Rest full.

El sistema utilizado se define como SPA, Single Page Application, por el cual la navegación es muy fluida y la actualizacion de los datos es en tiempo real. Por el cual es muy importante para el contexto en el que se basa la herramienta. Todo sistema con interacción social requiere de un tiempo de respuesta muy corto, uno de los puntos fuertes de los frameworks utilizados.

La principal motivación de este proyecto es poner en práctica los conocimientos adquiridos a lo largo de las asignaturas de la Titulación de Ingeniería en Informática así como conocer nuevos lenguajes de programación que nos permita ampliar nuestros conocimientos dentro del abanico de posibilidades que nos ofrece la informática hoy en día y aprender a desenvolverse en situaciones desconocidas hasta el momento por uno mismo.

## **Alcance**

El objetivo principal de este proyecto es que el usuario pueda solicitar una canción de manera autónoma y que el administrador del local/evento pueda llevar un control de las canciones enviadas y a su vez poder utilizarlo como portal de promociones hacia los usuarios registrados, pudiendo promover el uso de la aplicación.

De modo que se obtendrán estadísticas de las canciones más solicitadas, para poder adaptar las canciones futuras a los gustos de las personas que frecuenten el local.

## **Glosario de Términos**

A continuación mostramos algunas definiciones, acrónimos y abreviaturas que irán apareciendo a lo largo de la memoria, para así facilitar la comprensión de algunos conceptos.

**Angular**: Framework javascript para desarrollo de páginas Web basado en el modelo MVC. La primera version de este Framework, fue llamada AngularJS, fue creada por google y se desarrollaba utilizando javascript. El 15 de Septiembre de 2016, aparecería la version 2.0, no siendo un simple evolutivo de la primera version, si no que dejaba de lado javascript, para utilizar typescript, como lenguaje base para su implementación. Los sistemas desarrollados por Angular se denominan SPA, en el cual se pasa la lógica de negocio al cliente en vez de al servidor. Mejorando tiempos de espera y latencia con la aplicación.

**TypScript**: Es un lenguaje de programación de código abierto, desarrollado por Microsoft. Se basa en javascript, añadiendole tipado a las variables y clases. Es un lenguaje, en el cual su salida compilada es lenguaje javascript.

**MongoDB**: Es un sistema de gestión de base de datos no relacionales basadas en JSON, orientado a documentos. Es una base de datos de código abierto nacida en 2007.

**Express**: Es una infraestructura para Node.js en el cual proporciana una serie de caracteristicas para conectar el servidor Node con la base de datos. Se suele utilizar con MongoDB, mendiante mongoose, una librería javascript para facilitar la integración.

**Node**.**js**: Es un intérprete de javascript en el lado del servidor basado en eventos basado en el motor V8 desarrollado por Google. Con lo cual permite al usuario desarrollar en el servidor con el mismo lenguaje de programación del cliente, siendo su curva de aprendizaje minima.

**QlikView**: Es una plataforma de Business Intelligence que sirve, para mostrar y analizar datos de manera inmediata. Se utiliza como cuadro de mandos, para poder reporter datos a gran escala, y que cada usuario pueda filtrar y ordenar la información como desee.

**Git**: Repositorio de código utilizado para la gestión de versiones. Fue creado pensando en la eficiencia y confiabilidad del mantenimiento de aplicaciones, cuando estas tienen un número elevado de ficheros fuente.

**Spotify**: Es una herramienta multiplataforma utilizada para la reproducción de música vía Streaming.

**Api Rest**: Es un sistema por el cual los programadores pueden interactuar con una aplicación y obtener y enviar datos de manera ordenada y planificada. Se basa en obtener y enviar datos mediante JSON y peticiones HTTP.

## **Organización del documento**

El documento está divido en once capítulos en los cuales se irán describiendo las distintas etapas que se han seguido a la hora de desarrollar el proyecto.

En este primer capítulo, ofrece una visión algo general de lo que este proyecto abarcará. Definiremos los límites que esperamos alcanzar en la aplicación a desarrollar al final del documento.

En el segundo de los capítulos indicaremos la metodología utilizada para el desarrollo de la herramienta, asi como la planificación y organización del proyecto. Abarcaremos los coste y riesgos que conllevarían su implantación.

En los cinco capítulos siguientes se abordarán las distintas fases que comprenden cualquier desarrollo de software bajo el modelo de desarrollo lineal secuencial, que son:

Requisitos del sistema, Análisis, Diseño, Construcción del sistema y Pruebas.

**Requisitos del sistema** Se defininen los requisitos necesarios para poder realizar el sistema, asi como las necesidades de negocio y studio de las solición propuesta

**Análisis**. Fase en la cual se estudia la información que deberá manejar el sistema así como los procesos a desarrollar sobre estos y la interconexión entre ambos.

**Diseño**. Etapa consistente en la estructuración de los datos y en el diseño de los algoritmos que tratarán dichas estructuras de información. En esta fase, se obtiene ya una representación del software previa a la implementación que podrá dar una idea de la calidad que tendrá este.

**Construcción del sistema e Implementación**. Paso en el que se transformará la representación del software generada en el paso anterior a un lenguaje entendible por la máquina.

**Pruebas**. Previa a la implantación del software se harán necesarias unas pruebas que garanticen la calidad del sistema. Esta fase se prorrogará incluso estando ya el software en funcionamiento, ya que no hay mejor prueba que un entorno real capaz de simular todas y cada una de las situaciones existentes.

En el octavo capítulo describiremos un completo manual para la implantación del sistema, donde explicamos como realizar el proceso de instalación.

En el noveno capítulo describiremos un completo manual para el usuario, mostrando de manera detallada e ilustrada como realizar las distintas funcionalidades.

En el capítulo diez detallaremos el manual del desarrollador, modelo de trabajo utilizado, consideraciones a la hora de desarrollar e indicaremos las instrucciones para el despliegue del sistema.

En el capítulo once analizaremos como ha ido el desarrollo completo del sistema, si se han cumplido las expectativas y los distintos problemas que nos hemos encontrado.

Y por ultimo describiremos el software usado para la realización de todo el proyecto, bibliografía usada y la licencia aplicada a la aplicación.

# Capítulo 2

# Planificación

## **Metodología de desarrollo**

La metodología utilizada para la elaboración del proyecto se basa en el Racional Unified Process (RUP), que traducido es proceso racional unificado. Este proceso de desarrollo software unido al lenguaje unificado de modelado (UML) componen la metodología habitual estándar para el análisis, implementación y documentación de sistemas orientados a objetos.

El proceso unificado racional (RUP) se utiliza como en nuestro caso para desarrollar sistemas orientados a objetos, basado originalmente en el Proceso unificado (UP) y que utiliza Lenguaje unificado de modelado (UML) para su notación.

Consta de cuatro fases:

* **Inicio**: Se plantea y determina las características del proyecto. Se define el alcance del proyecto, se identifican los riesgos y donde se propone una visión general del proyecto.
* **Elaboración**: Donde se realiza el diseño, requisitos, análisis. Se seleccionan los casos de uso que permitan definir la arquitectura del sistema, en el cual se diseña una solución preliminar.
* **Desarrollo**: Donde se realiza la implementación. Se le llama también fase de Construcción. Su objetivo es completar la funcionalidad del sistema de acuerdo a las evaluaciones hechas por los usuarios.
* **Transición**: El objetivo de esta fase es asegurar que la herramienta este disponible a los usuarios finales, verificando que el producto cumpla con los requisitos tomados.

Como hemos mencionamos anteriormente, en el desarrollo de nuestro proyecto seguimos un enfoque orientado a objetos. La programación orientada a objetos es un paradigma de la ingeniería del software que basa la arquitectura de los sistemas en los objetos y sus interacciones. Estos objetos representan las entidades físicas y conceptuales del mundo real y están estrechamente relacionados con la aplicación a construir.

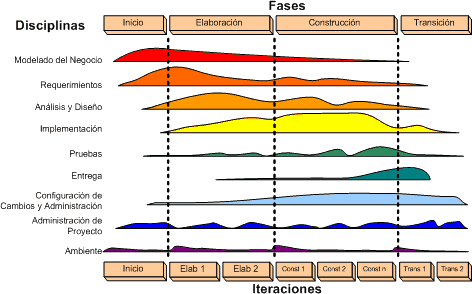


Figura 2.1 – Fase RUP

Basado en el desarrollo incremental, donde todas los roles o actores se involucran más en el proyecto. Haciendo entregas parciales del software, evaluándose como partes individuales, pudiendo reconstruirse partes anteriores conforme se elaboran las nuevas.

De esta forma se determinan las funcionalidades importantes y las comprobaciones son constantes a lo largo del desarrollo, permite una implementación con refinamientos sucesivos y con cada incremento se agrega nueva funcionalidad, se cubren nuevos requisitos o bien se mejora la versión previamente implementada del sistema.

## **Planificación del proyecto**

A continuación se detallan las iteraciones que se ha seguido a lo largo del proyecto. Despues de cada iteración se procede a realizar la documentación de cada una de ellas.

Iteración 0 – **Elicitación de requisitos y prototipado**

* Se realiza un estudio de los requisitos necesarios para elaborar el sistema. Se realizan varias reuniones con clientes y usuarios potenciales, elaborando una serie de condiciones para que la herramienta sea lo más funcional posible.
* Se realiza un prototipo de la herramienta, organizándose en bloques para su consiguiente desarrollo.

Iteración 1 – **Usuarios/Roles**

* Se desarrolla un sistema de autenticación y perfilado de la herramienta. Asignando una función a cada uno de los perfiles dados de alta.

Iteración 2 – **Búsqueda de Canciones**

* Se crea la conexión con la API de Spotify. Pudiendo realizar conexiones y extraer y parsear la información para que este disponible en la herramienta en tiempo real.

Iteración 3 – **Lugares**

* Se crea un menú de administración para crear y editar distintos lugares, locales y eventos para que estén disponible en la herramienta.
* Se crea el módulo de búsqueda de lugares, habilitando la posibilidad de que envíen canciones ya buscadas a un lugar concreto.
* Se crea el módulo de favoritos, añadiendo los usuarios con un local en concreto.

Iteración 4 – **Envío y Reproducción de canciones**

* Se crea la conexión entre la canción y el lugar, habilitando la validación de las mismas.
* Se crea la conexión de Spotify con el local. Añadiendo las canciones en tiempo real a Spotify
* Se desarrolla un módulo para la reproducción de canciones en Spotify directamente desde la herramienta con actualizaciones en tiempo real.

Iteración 5 – **Mensajería**

* Se crea un módulo para publicar mensajes en el tablón. Dando la opción al administrador del lugar enviar un mensaje a los usuarios que tienen agregado al local como favorito.

Iteración 6 – **Reporting**

* Se desarrollan una serie de informes para que el administrador del local pueda tener constancia de las peticiones que recibe, así como el género de las mismas, para adecuar la música al gusto de los clientes.

A continuación se detallan las actividades:

Actividad 1 - **Análisis**:

* Tecnología a usar
* Formación
* Visión Global
* Definición alcance total

Actividad 2 - **Diseño**:

* Modelo de datos
* Fase 1 – Usuarios / roles
* Fase 2 – Buscar Canciones
* Fase 3 – Lugares
* Fase 4 – Envío y Reproducción de canciones
* Fase 5 – Mensajería
* Fase 6 – Reporting

Actividad 3 - **Implementación**:

* Fase 1 – Usuarios / roles
* Fase 2 – Buscar Canciones
* Fase 3 – Lugares
* Fase 4 – Envío y Reproducción de canciones
* Fase 5 – Mensajería
* Fase 6 – Reporting

Actividad 4 - **Pruebas**:

* Fase 1 – Usuarios / roles
* Fase 2 – Buscar Canciones
* Fase 3 – Lugares
* Fase 4 – Envío y Reproducción de canciones
* Fase 5 – Mensajería
* Fase 6 – Reporting

Actividad 5 – **Documentación**.

Se muestra el diagrama de Gantt, basado en las actividades descritas.

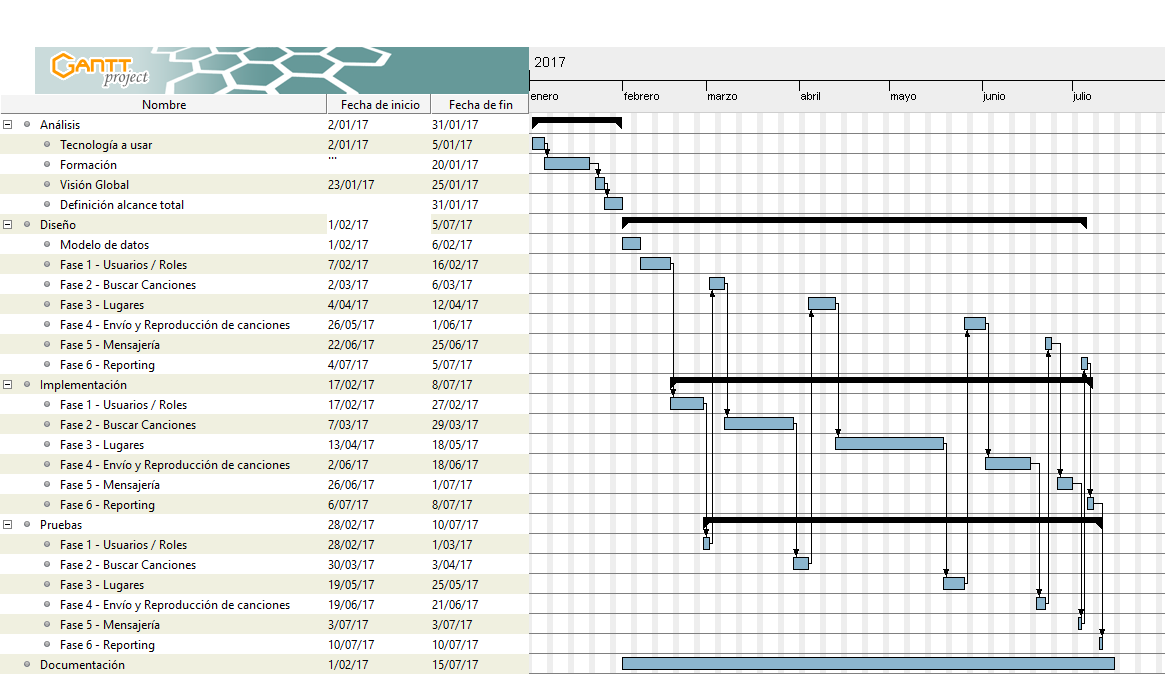


Figura 2.2. Diagramas de Gantt

Se realiza una planiﬁcación de los tiempos de realización de cada una de las fases del proyecto. Algunas etapas, por problemas con los sitemas fuente o por mayor complejidad de lo esperado, superan el tiempo estimado, otras, sin embargo, una vez empezado el desarrollo, se realizaron en un tiempo menor del esperado.

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteración** | **Actividad** | **Tiempo Estimado horas** | **Tiempo Real horas** |
| Iteración 0 | Análisis | 176 | 208 |
|  | Diseño | 32 | 31 |
| Iteración 1 | Diseño | 64 | 74 |
|  | Implementación | 56 | 54 |
|  | Pruebas | 16 | 18 |
|  | Documentación | 8 | 8 |
| Iteración 2 | Diseño | 24 | 25 |
|  | Implementación | 136 | 144 |
|  | Pruebas | 24 | 29 |
|  | Documentación | 10 | 10 |
| Iteración 3 | Diseño | 56 | 57 |
|  | Implementación | 208 | 220 |
|  | Pruebas | 40 | 47 |
|  | Documentación | 10 | 10 |
| Iteración 4 | Diseño | 40 | 46 |
|  | Implementación | 88 | 105 |
|  | Pruebas | 24 | 27 |
|  | Documentación | 8 | 8 |
| Iteración 5 | Diseño | 16 | 18 |
|  | Implementación | 40 | 48 |
|  | Pruebas | 8 | 8 |
|  | Documentación | 10 | 12 |
| Iteración 6 | Diseño | 16 | 19 |
|  | Implementación | 16 | 18 |
|  | Pruebas | 8 | 9 |
|  | Documentación | 4 | 4 |
| Total |  | 1.138 | 1.258 |

Tabla 2.1 – Tiempo de realización del proyecto

## **Organización**

Para la realización de la aplicación, han estado involucrados los distintos roles,:

* Analista:
  + Encargado de obtener los requisitos funcionales necesarios para la elaboración de la herramienta
* Desarrollador:
  + Encargado de llevar a cabo los requisitos obtenidos por el analísta
* Encargado de pruebas
  + Encargado de evaluar la herramienta, tanto de fallos técnicos, como requisitos que no se hayan desarrollado de forma correcta.

Puesto que el proyecto no ha sido colaborativo, los roles de analista, desarrollo y encargado de pruebas técnicas, han sido asumidos por el autor del proyecto.

Para poder desarrollar la herramient jUCAbox se han utilizado los frameworks propuestos por el fork MEAN stack.

* **MongoDB:** Sistema de base de datos no relacionales basada en JSON. Utiliza una version propia llamada BSON.
* **Express:** Framework encargado de conectar MongoDB con el servidor WEB
* **Angular2:** Framework basado en MVC utilizado para la parte Cliente de la aplicación WEB. Se utiliza para el desarrollo de páginas SPA (Single Page Application). Se desarrolla con typescript, una versión libre de javascript creada por Microsoft.
* **Node.js:** Framework utilizado como servidor WEB. Se utiliza javascript para el desarrollo de las reglas del servidor.

A su vez, se ha empleado otra sistema de gestión de base de datos, para el envío en tiempo real de los mensajes entre usuarios. FireBase, un sistema de dase de datos no relacionales basada en JSON. La gran diferencia que tiene con MongoDB es que se trata de un sistema en tiempo real.

A su vez, se han desarrollado una serie de informes, mediante la herramienta QlikView. Una herramienta capaz de transformar la información en tiempo real y que se encuentra alojada en la nube.

El hardware utilizado para el desarrollo de esta herramienta ha sido un portatil Lenovo ideapad 100 con un procesador intel i5 con 8gb Ram y disco duro de 256 GB.

## **Costes**

El coste final del producto, sería la suma del coste de los recursos humanos, con los recursos materiales. El cálculo se realiza con los tiempos estimados.

Los **recursos materiales** se dividen en dos:

* Recursos software
  + Fork utilizado para el desarrollo Mean Stack. Todo lo utilizado es software libre, por lo tanto el coste de los recursos software es 0€.
* Recursos hardware
  + Para el desarrollo de la herramienta se ha utilizado un portátil Lenovo ideapad con un precio de 585€.
  + Para el despliegue de la aplicación, se utilizaría un sistema en la nube basado en AWS de amazon. El precio para el alojamiento de la aplicación depende del uso de la herramienta, y el impacto social que pueda tener. Todos estos servidores son escalables, es decir, en el momento que se necesiten más recursos se pueden contratar, y automáticamente serían añadido. El modelo utilizado para el despliegue de esta herramienta sería Amazon Ec2 bajo demanda, con un almacenamiento estandar de 50TB/mes de uso de datos, y un servidor Linux t2.medium con unas especificaciones de 2 CPU virtuales y 4 GB de ram, serían 34.41€ al mes.

Los **recursos humanos** necesarios para el desarrollo serían:

* **Analista**: Sueldo medio anual de un analista funcional es de 45.000€. Siendo la hora a 23€
* **Desarrollador:** Sueldo medio anual de un desarrollador web es de 35.000€. Siendo la hora a 18€.
* **Encargado de Pruebas**:Sueldo medio anual de un especialista encargado de realizar pruebas técnicas y funcionales es de 30.000€. Siendo la hora a 15€.

Los costes de cada rol han sido obtenidos a través de la web http://espana.jobtonic.es

Coste desarrollo de **jUCAbox:**

|  |  |  |
| --- | --- | --- |
|  | Cantidad | Precio |
| Recursos materiales para el desarrollo | 3 Lenovo ideapad | 1.755 € |
| Recursos humanos necesarios | 1 Analista  474 horas | 10.902 € |
|  | 1 Desarrollador  544 horas | 9.792 € |
|  | 1 Encargado de pruebas  120 horas | 1.800 € |
| Total |  | 24.249 € |

Tabla 2.2 – Coste de desarrollo del proyecto

Coste anual de mantenimiento de **jUCAbox:**

|  |  |  |
| --- | --- | --- |
|  | Cantidad | Precio |
| Recursos materiales | Alojamiento | 412,92 € |
| Total |  | 412,92 € |

Tabla 2.3 – Coste de mantenimiento anual del proyecto

## **Riesgos**

La gestión de riesgos es un enfoque estructurado para manejar la incertidumbre de las diferentes amenazas existentes en el desarrollo y puesta en marcha de la aplicación. Para ello es necesario realizar una evaluación de riesgos, estrategias de desarrollo para manejarlos y mitigación del riesgo.

**Principales riesgos en el desarrollo del proyecto:**

* **Falta de experiencia en el desarrollo con el framework**
  + Al ser desarrollo con herramientas nuevas en el mercado, el conocimiento de las mismas no es tan elevado, por lo que podría suponer retrasos en el desarrollo. La exposición al riesgo serían de 2 días por cada etapa. La forma de mitigar este riesgo es mediante el estudio y análisis de los frameworks utilizados, buscando suficiente documentación técnica que sirva de apollo.
* **Cambio en la obtención de Api externa**
  + El recurso de datos principal del que se nutre nuestra aplicación es la API proporcionada por Spotify. Por la tanto uno de los principales riesgos de la herramienta es el posible cambio en la definición de las funciones proporcionadas por el API. Como medida, Spotify, anuncia los cambios que va a efectuar en su API y cuando serán efectivos. Por lo que se podrían reprogramar esas conexiones en nuestra herramienta, y hacer el cambio cuando sea posible.

**Principales riesgos técnicos:**

* **No disponibilidad del sistema por la caida del servidor**
  + Para mitigar este riego y no sufrir dependencia de un único servidor, se utiliza el balanceo de servidores. Su estructura se compone de 3 servidores. El primer servidor es el que se encarga de gestionar la comunicación de la base de datos con el cliente Web. El Segundo hace una replica en tiempo real de los datos del primero. Y el tercero se mantiene a la espera de posibles daños.
  + Cuando en el primer servidor se produce un fallo, comprometiendo la integridad de la aplicación, el Segundo servidor se pone como maestro y el tercero ocupa el puesto que este deja, siendo el que replica la información del servidor.
  + Cuando el servidor vuelve a la normalidad, se convierte en el tercero de ellos, esperando que se produzcan posibles fallos.
* **Pérdida de datos**
  + Para ello, como se ha comentado antes, en el grupo de servidores, el Segundo servidor no estaría esperando a que el primero falle para ocupar su lugar. Serviría también de almacen de copia de seguridad a corto plazo, pudiendo recuperar la información si fuera necesario.

## **Aseguramiento de calidad**

Para la verificar la calidad de realización de este proyecto, se ha han comparado los frameworks con más repercusión en la actualidad, llegando a la conclusión que el nuevo framework Angular, teniendo detrás el soporte de Google, es una gran apuesta de futuro, garantizando la continuidad del proyecto, gracias a su gran comunidad de desarrolladores. Además al realizarse una fase de pruebas, se garantiza la calidad funcional de la herramienta.

Para el maquetado y diseño de la web se utiliza HTML5 y CSS3, siguiendo los estándares propuestos por la World Wide Web Consortium. Por lo que se sigue con el estándar marcado para la realización de webs.

Para el Front se utiliza Angular 2, basándose en TypeScript, un framework creado por Microsoft. A su vez es un lenguaje que genera código compilado de JavaScript, siguiendo el estándar ECMA6. Con el cuál se ha desarrollado el proyecto, garantizando así su integridad.

Para el Back-end, pasa exactamente lo mismo, pues se ha desarrollado con Node.js, utilizando javascript como lenguaje de programación. Igualmente se ha seguido con el estándar ECMA6 con pilar de desarrollo.

# Parte II

# Desarrollo

# Capítulo 3

# Requisitos del Sistema

## **Situación actual**

Actualmente, cuando se va a celebrar un evento, o acudir a un local, existe la necesidad de introducir un ambiente musical. Cuando un cliente/invitado quiere escuchar alguna canción, tiene que acercarse al encargado de la música, y decirle el nombre de la canción, con lo que conllevaría la interrupción de la labor desempeñada por el encargado de la música, así como la dificultad de entendimiento por el ruido.

Por otro lado, está creciendo la necesidad de solicitar unas canciones antes de realizar un evento social, como una boda, una comunión, o simplemente una fiesta entre amigos. Por lo que el organizador, debería pedir, mediante correos, mensajes y/o personalmente, las canciones que le gustaría que sonaran en el evento. Dado que las fuentes de información son diversas, se corre el riesgo de que bien, se pierda esa información, o que la información sea la misma y se duplique el trabajo de organización de las canciones.

jUCAbox nace con la necesidad de crear un sistema autónomo de envio de canciones en lugares con ambiente musical. No se desarrolla para una entidad concreta, si no que tiene como objetivo crear una herramienta, en la que se centralice todos los locales y/o eventos, con el fin de tener la información centralizada.

### Procesos de Negocio

En la actualidad, no existe ningún proceso especificio, puesto que con la herramienta se va a dar una utilidad que anteriormente no tenían.

El proceso actual es el intercambio de información de manera verbal. El usuario/cliente pediría una canción a la persona encargada de la música. Esta petición no sería registrada en ningún sistema, por lo que se perdería.

Con la herramienta, se intenta emular esta situación, pero garantizando la recepción de la petición, y haciendo un seguimiento, y aportando un valor añadido al proceso.

### Entorno Tecnológico

Puesto que la herramienta no ha sido desarrollada para ninguna empresa en particular, no se han tenido en cuenta el entorno tecnológico de las mismas. Para poder utilizar el software, sería necesario un portátil y/o móvil con conexión a internet. El entorno anterior a la herramienta, será exactamente el mismo que después de la misma. Puesto que para reproducir música en un local sería necesario un ordenador con conexión a internet.

## **Necesidades de Negocio**

Se busca cumplir con los procesos de negocio descritos anteriormente. Emulando el ciclo de peticiones que tenían, y mejorando el flujo de información, pudiendo explotarla de la manera más sencilla posible.

### Objetivos de Negocio

El principal objetivo que se buscan conseguir es el de proporcionar al cliente la posibilidad de solicitar una canción sin tener que moverse por todo el local. Y a su vez, facilitar al responsable de la música los gustos y preferencia de las personas que se encuentren en el local.

Se intenta dar, a su vez, la posibilidad de crear listas de reproducción por un único canal, para que el organizador del evento sepa que música poner antes de la celebración del mismo.

Un objetivo colateral del resto, es hacer sentir al cliente escuchado, pudiendo tomar parte en la reproducción de canciones en el local.

## **Objetivos del Sistema**

Los objetivos que se pretenden conseguir con este nuevo sistema es cubrir todos los procesos de negocio descritos anteriormente en los que son necesarios cubrir el registro de lugares y eventos en la herramienta. Así como el registro de usuarios para la interacción con la misma.

Teniendo en cuenta otro de los objetivos a cumplir por nuestro sistema, es la conexión en tiempo real con la aplicación Spotify, siendo la comunicación bidireccional.

Se intentará crear un semired social, donde los usuarios podrán interactuar con otros usuarios de la misma, así como los locales podrán dejar mensaje en los tablones de la herramienta, para que los usuarios que los tengan como favorito poder leerlos.

## **Catálogo de Requisitos**

A continuación se describen los requisitos funcionales y no funcionales del sistema, para conocer las características que debe cumplir la aplicación una vez finalizada.

### Requisitos funcionales

En esta sección se van a describir los requisitos funcionales del sistema. Es necesario recalcar, que el perfil de usuario, se comportaría como administrador en el caso de gestionar algún lugar. Se puede acceder a la herramiento sin ser un usuario registrado. Siempre que se hable de usuario, se hablará de usuario registrado. Si el usuario no está registrado se indicará como invitado.

* Se deberá tener un apartado para el registro de usuarios
  + El invitado se podrá registrar con Google, Facebook, Twitter o registro interno
  + El usuario podrá buscar a usuarios y agregarlos como amigo
  + El usuario podrá ver los datos de otros usuarios si los agrega como amigos
  + El usuario podrá modificar sus datos
  + El usuario verá un log de actividad
  + El usuario podrá chatear con otros usarios si son amigos
* Se deberá tener un apartado para la gestión de lugares
  + El usuario podrá crear lugares. Pasará a ser administrador para ese lugar.
  + El usuario podrá ver lugares agregados como favoritos
  + El usuario/invitado podrá buscar lugares
  + El administrador podrá publicar mensajes de sus lugares
  + El administrador podrá validar o rechazar las canciones enviadas
  + El administrador podrá crear distintas playlist asociadas al lugar
  + El administrador podrá reproducir la lista de canciones desde la herramienta
* Se deberá tener un apartado de búsqueda de canciones y artistas
  + El usuario/invitado podrá buscar cualquier canción/artista de spotify
  + El usuario/invitado podrá enviar una canción a cualquier lugar registrado en la herramienta
  + El usuario podrá agregar como favorito a artistas

### Requisitos no funcionales

La aplicación deberá cumplir los requisitos no funcionales listados a continuación:

* **Disponibilidad**
  + La aplicación debe estar disponible las 24 horas del día, los 365 días del año. Para garantizar una mayor experiencia al usuario.
* **Usabilidad**
  + La herramienta debe ser de fácil uso. No restringiendo solo a personas con alto conocimiento tecnológico, sino de sencilla comprensión, y con funcionalidades intuitivas.
* **Portabilidad**
  + La aplicación debe poder usarse en cualquier dispositvo, independientemente del sistema operativo usado y/o el navegador.
* **Seguridad**
  + La herramienta ante todo tendrá implementada una capa de seguridad, en la que impediría a cualquier usuario externo acceder a datos internos. Será necesario evitar cualquier ataque a los datos, estableciendo unas medidas extremas en el lado del servidor.
* **Interfaz**
  + La interfaz de usuario debe ser sencilla a la vez que completa. Pudiendo permitir al usuario de forma agil y veloz acometer su acción de la forma más sencilla posible. Siendo, a su vez, una interfaz atractiva y correctamente consultable desde cualquier dispositivo.
* **Mantenebilidad**
  + La herramienta podrá ser mantenidad por cualquier desarrollador previamente formado en los frameworks utilizados. Por lo que es necesario mantener unas buenas prácticas a la hora de desarrollar la aplicación, evitando duplicidad de código o creando funcionalidades demasiado complejas, pudiendo desglosarse en varias.
* **Rendimiento**
  + Al ser una herramienta con conexión de datos con sistemas terceros, los tiempos de respuestas deben de ser cortos, implementando la transmisión asíncrona de datos siempre que sea posible.
* **Fiabilidad**
  + La aplicación debe de ser lo suficientemente robusta para evitar errores en su ejecución.

### Reglas de negocio

Se exponen unas reglas de negocio necesarias que debe cumplir la herramienta.

* Solo puede existir una persona que gestione un local
* Para poder chatear es necesario que los dos usuarios sean amigos
* Para gestionar un local es necesario estar registrado.

### Requisitos de información

En esta sección se describen los requistos de información que debe cumplir la herramienta.

* Se almacenará la dirección de los lugares a registrar. Para poder realizar búsquedas dependiendo de la ciudad o provincia.
* Se almacenarán los datos referentes a las canciones enviadas al local. Será necesario para la explotación a futuro de los datos, conocer la temática o autores más relevantes.
* Será necesario guardar en un log de actividad toda la actividad de los usuarios. En una siguiente fase se podrá hacer un estudio de mercado con la información obtenida.

## **Alternativas de Solución**

En una primera fase se analizó los requisitos que debería cumplir una aplicación para que pudiera enviar y recibir canciones.

En primera instancia se pensó en crear una NAS de alojamiento de ficheros de audio. Un catálogo de canciones para poder relacionar esos ficheros con el artista, nombre de la canción etc…

En segundo lugar, para no tener que litigiar con derechos de autor, se pensó en utilizar herramientas de mercado que dieran conexión a terceros. Entre las más utilizadas por la sociedad estaban, Spotify, Napster y SoundCloud.

Las herramientas ponen al servicio de terceros unas API de solicitud de información de datos que manejan.

Para el desarrollo de aplicación se pensó en realizar una herramienta móvil para la utilización del usuario y otra herramienta de escritorio para comunicar las canciones solicitadas.

La otra opción es realizar una Web responsive para el uso tanto de móvil como desde pc. Para el desarrollo de esta herramienta se barajo utilizar frameworks como Symphony2, basados en php, ASP .NET MVC o el fork de Mean Stack.

## **Solución Propuesta**

La solución propuesta finalmente fue realizar una herramienta con conexión a Spotify, para la obtención de datos a través de su API. Para el desarrollo de la herramienta se realizaría con el fork de herramientas MEAN stack, complveuesto por MongoDB y Express para los datos, Angular 2 para el Front y Node.js para el Backend, puesto que son frameworks que se preveen que van a tener una gran comunidad de desarrolladores detrás, facilitando su mantenimiento futuro.

Además este fork, ofrece de manera implícita, el desarrollo responsive, por lo que el desarrollo será más sencillo. En un futuro si se quiere implementar una aplicación móvil, existe un componente llamado IONIC 2, que transforma el código de Angular 2 en una aplicación nativa de móvil, tanto IOS como Android.

# Capítulo 4

# Análisis del Sistema

Esta sección va a contener en análisis del sistema, utilizando modelado UML para ello. El primer modelo mostrará las relaciones entre las entidades del sistema desarrollado.

A continuación se mostrarán los casos de uso de cada requisito funcional descrito en el capítulo anterior. En cada uno de ellos, se hara una descripción, conociendo los actores que interactúan en el requisito y las condiciones necesarias para ser cumplido, identificando los escenarios donde se producen.

Más adelante se mostraran los modelo de comportamiento, que describirán el comportamiento del sistema, ofreciendo una visión global de como se comportará con las interacciones de los distintos usuarios.

Y al final de esta sección, detallaremos los modelos de interfaz de usuario, pudiendo ver los diseños de las distintas funcionalidades de la herramienta y como interactúan entre ellas.

## **Modelo Conceptual**

A continuación se muestra el diagrama conceptual de clases basado en UML. Cabe recordar, que este sistema esta diseñado con bases de datos no relacionales, por lo que la relación entre clases no se comporta como un sistema relacional clásico. El diagrama se va a mostrar como si fuera un sistema relacional, y posteriormente se expondrán las diferencias.

Las entidades externas del diagrama siguiente corresponden a información obtenida a través del api de Spotify.

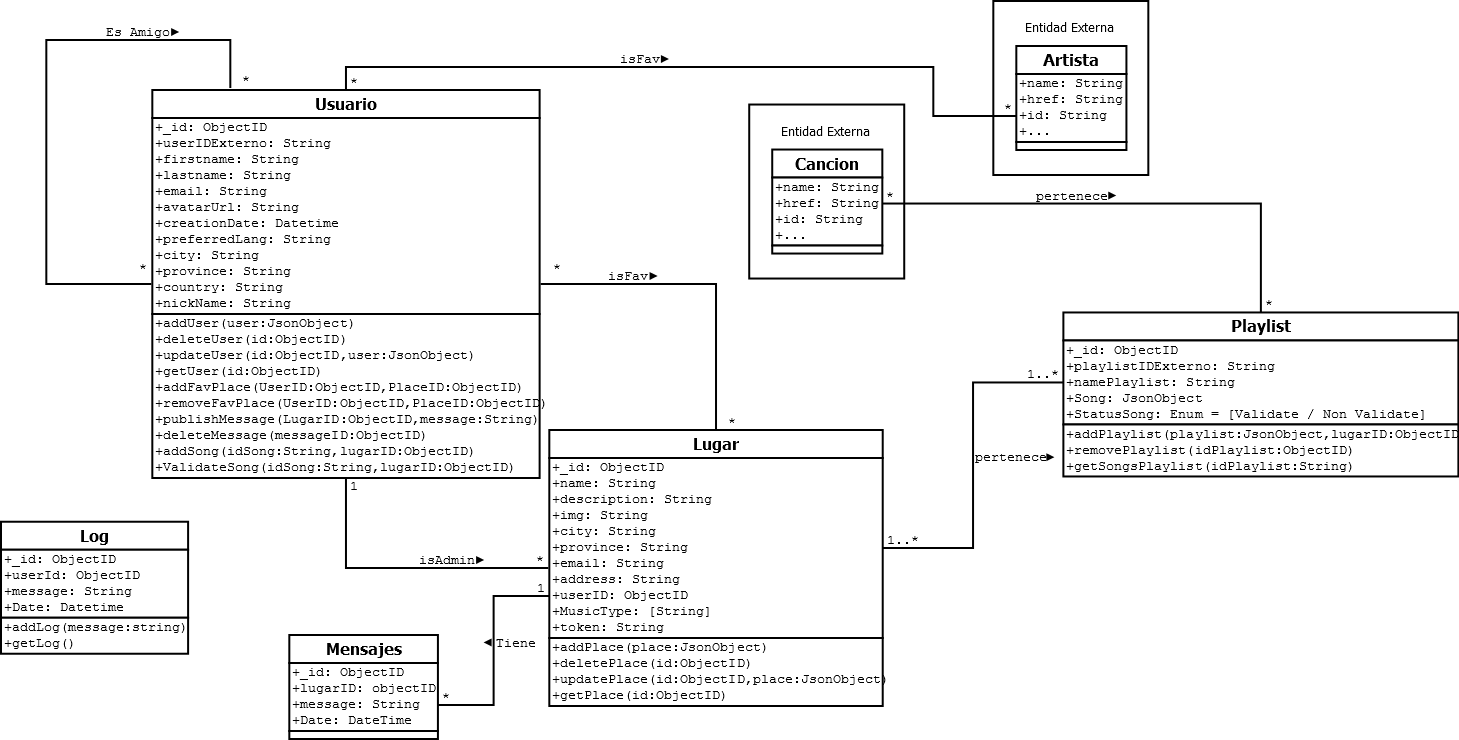


Figura 4.1 – Digrama relaciones UML

## **Modelo de Casos de Uso**

En el siguiente apartado, se describirán los actores involucrados en los requisitos funcionales, nombrados anteriormente. Posteriormente, se mostrarán cada uno de los casos de uso necesarios para los requisitos funcionales.

### Actores

En este apartado se describirán los diferentes actores involucrados en el sistema.

* **Invitado**:
  + Son las personas que acceden a la herramienta sin registro previo. Podrán utilizar la parte esencial de la aplicación, como es enviar canciones. Pero no podrán añadir como favoritos lugares ni artistas, ni tampoco conocer los mensajes publicados de los lugares. Podrá ver los lugares y canciones enviadas, a su vez podrá filtrar el top de canciones del lugar entre fechas.
* **Usuario**:
  + Son las personas que acceden a la herramienta habiéndose registrados. Podrán realizar todas las funciones descritas para el invitado. Además de añadir lugares y artistas como favoritos. Intercambiar mensajes con otros usuarios, ver mensajes publicados por los lugares que tenga añadido como favorito. Asi como crear nuevos lugares en la aplicación.
* **Administrador:**
  + Son las personas que acceden a la herramienta habiéndose registrado y que gestionan algún lugar. Podrá hacer las mismas gestiones que los usuarios además de validar las canciones enviadas y publicar mensajes en el tablón del lugar.

Los actores Administrador y Usuario no son excluyentes, es decir, para los lugares creados por el Usuario, será considerado como Administrador, para el resto seguirá siendo Usuario. A partir de este momento los perfiles de los usuarios solo se describirán el de menor rango. Siempre que se defina un caso de uso para un invitado, podrá realizarlo Usuarios y Administradores. Si un Caso de uso es realizado por un Usuario, podrá ser realizado por un Administrador. A excepción del registro en la herramienta, solo se podrá registrar un invitado.

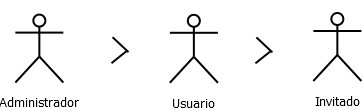


Figura 4.2 – Actores jUCAbox

### Casos de Uso

En esta sección se muestra el modelo con el paquete de casos de uso a alto nivel en el que se muestran todos los requisitos funcionales descritos anteriormente. Posteriormente se irán desglosando los requisitos en distintos casos de uso.

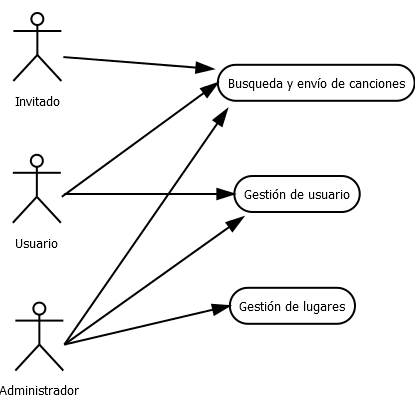


Figura 4.3 – Paquetes de casos de uso

#### 4.2.2.1. Búsqueda y envío de canciones

En el siguiente modelo, se va a mostrar el caso de uso de búsqueda y envío de canciones, de los requisitos funcionales vistos anteriormente.

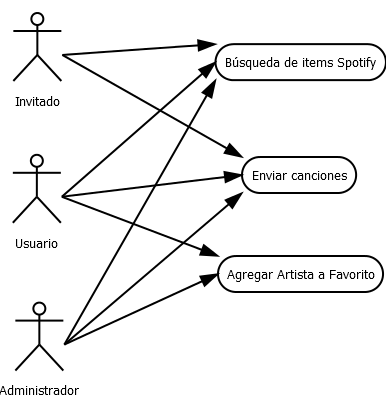


Figura 4.4 – Casos de uso Búsqueda y envío de canciones

* **Caso de uso**: **Búsqueda de ítems Spotify**
  + **Descripción**: El invitado buscará un ítem en jUCAbox perteneciente a Spotify. Este ítem puede ser una canción, un artistia, una lista de reproducción y/o un álbum.
  + **Actores**: Invitado
  + **Precondiciones**: Ninguna.
  + **Postcondiciones**: El sistema devolverá el ítem buscado.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el invitado decide buscar un ítem en la herramienta.
    2. El invitado elige el tipo de ítem a buscar.
    3. El invitado pulsa el botón buscar.
    4. El sistema comprueba que los datos introducidos correponde a algún dato de Spotify.
    5. El sistema devuelve los datos encontrados.
  + **Escenario alternativos**:

1. a. Los datos no existen en Spotify.
   * + 1. El sistema muestra el error diciendo que no ha encontrado datos.

* **Caso de uso**: **Enviar Canciones**
  + **Descripción**: El invitado seleccionara la canción que desea enviar, y elegirá a que lugar quiere hacer la petición.
  + **Actores**: Invitado,Usuario
  + **Precondiciones**: Si se desea elegir de una lista de lugares favoritos, el perfil necesario para el envio debe ser un Usuario registrado.
  + **Postcondiciones**: El sistema enviará la canción solicitada al lugar.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el invitado decide enviar una canción.
    2. El usuario busca una canción.
    3. El invitado elige la canción deseada.
    4. El invitado pulsa el botón enviar.
    5. El invitado elige del listado de lugares a cual quiere enviar la canción
    6. El invitado introduce el token para comprobar la validez del lugar
    7. El invitado pulsa enviar
    8. El sistema comprueba que los datos introducidos son correcto.
    9. El sistema envía la solicitud de la canción al lugar.
  + **Escenario alternativos**:

1. a. El token introducido es incorrecto.
   * + 1. El sistema muestra el error diciendo que el token es incorrecto.

* **Caso de uso**: **Añadir Artista Favorito**
  + **Descripción**: El usuario buscará un artista, y lo seleccionará como favorito.
  + **Actores**:Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema almacenará el artista como favorito.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide agregar un artista como favorito.
    2. El usuario busca el artista.
    3. El usuario accede al artista.
    4. El usuario selecciona añadir a favoritos
    5. El sistema añade el artista a favoritos del usuario.

* **Caso de uso**: **Eliminar Artista Favorito**
  + **Descripción**: El usuario buscará un artista, y lo eliminará como favorito.
  + **Actores**:Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El artista debe ser un favorito del usuario.
  + **Postcondiciones**: El sistema eliminará el artista como favorito.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide eliminar un artista como favorito.
    2. El usuario busca el artista desde la búsqueda de items.
    3. El usuario accede al artista.
    4. El usuario selecciona eliminar de favoritos
    5. El sistema elimina el artista a favoritos del usuario.
  + **Escenario alternativos**:

1. a. El usuario busca el artista desde el menú de usuario.
2. a. El usuario selecciona eliminar de favoritos
3. a. El sistema elimina el artista a favoritos del usuario.

#### 4.2.2.2. Gestión de Usuario

En el siguiente modelo, se va a mostrar el caso de uso de gestión de usuarios, de los requisitos funcionales vistos anteriormente.

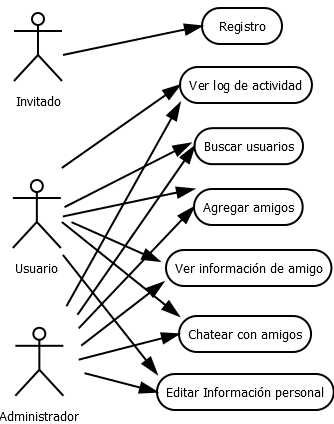


Figura 4.5 – Casos de uso Gestión de usuarios

* **Caso de uso**: **Registro en el sistema**
  + **Descripción**: El invitado se registrará en la herramienta.
  + **Actores**: Invitado
  + **Precondiciones**: Ninguna.
  + **Postcondiciones**: El sistema creará una cuenta de usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el invitado decide registrarse en la herramiento.
    2. El invitado elige el método de registro Google / Facebook / Twitter / Interno.
    3. El invitado introduce su correo y contraseña.
    4. El sistema crea un nuevo usuario.

* **Caso de uso**: **Ver log de actividad**
  + **Descripción**: El usuario verá el log de activad y peticiones de la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario deberá estar registrado y logeado en la herramienta.
  + **Postcondiciones**: El sistema devolverá el log de usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario desea ver su log de actividad.
    2. El usuario accede a su perfil.
    3. El usuario solicita su log de actividad.
    4. El sistema devuelve el log de actividad.

* **Caso de uso**: **Buscar usuarios**
  + **Descripción**: El usuario buscará a otros usuarios en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema devolverá los usuarios buscados.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide buscar a otro usuario.
    2. El usuario busca el nombre o apellidos del usuario desde su perfil.
    3. El sistema devuelve los usuarios que coincidan con la búsqueda.
  + **Escenario alternativos**:

1. a. El sistema no encuentra a los usuarios solicitados.

* **Caso de uso**: **Agregar nuevo amigo**
  + **Descripción**: El usuario agregará a otros usuarios como amigos en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema añadirá un nuevo amigo al usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide buscar a otro usuario y agregarlos.
    2. El usuario busca el nombre o apellidos del usuario desde su perfil.
    3. El sistema devuelve los usuarios que coincidan con la búsqueda.
    4. El usuario agrega al usuario que desea seguir.
    5. El sistema añade al usuario como amigo

* **Caso de uso**: **Eliminar amigo**
  + **Descripción**: El usuario desea eliminar a otros usuarios como amigos en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema eliminar el amigo del usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide buscar a otro usuario y eliminarlo como amigo.
    2. El usuario busca el nombre o apellidos del usuario desde su perfil.
    3. El sistema devuelve los usuarios que coincidan con la búsqueda.
    4. El usuario eliminará el usuario al que ya no desea seguir.
    5. El sistema eliminar el amigo del usuario.
* **Caso de uso**: **Ver información de amigo**
  + **Descripción**: El usuario desea ver la información de otros usuarios como amigos en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser amigo del usuario que desea consultar.
  + **Postcondiciones**: El sistema devolverá los datos del usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide buscar a otro usuario y ver si información.
    2. El usuario busca el nombre o apellidos del usuario desde su perfil.
    3. El sistema devuelve los usuarios que coincidan con la búsqueda.
    4. El usuario seleccionará el usuario que desea ver.
    5. El sistema devolverá los datos del amigo del usuario.
* **Caso de uso**: **Chatear con amigo**
  + **Descripción**: El usuario desea chatear con otros usuarios en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser amigo del usuario que desea consultar.
  + **Postcondiciones**: El sistema devolverá y enviará los mensajes intercambiados con el usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide buscar a otro usuario y empezar a chatear.
    2. El usuario busca el nombre o apellidos del usuario desde su perfil.
    3. El sistema devuelve los usuarios que coincidan con la búsqueda.
    4. El usuario seleccionará el usuario con el que desea chatear.
    5. El sistema devolverá y enviará los mensajes intercambiados con el usuario seleccionado.
  + **Escenario alternativos**:

1. a. El usuario no está conectado para poder chatear.
   * + 1. El usuario dejará escrito los mensajes y el sistema se lo mostrará al usuario cuando se conecte

* **Caso de uso**: **Editar Información personal**
  + **Descripción**: El usuario desea modificar sus datos en la herramienta.
  + **Actores**: Usuario
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema modificará los datos solicitados por el usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario modificar sus datos registrados en la herramienta.
    2. El usuario accede a su perfil.
    3. El usuario modifica los datos en su perfil.
    4. El sistema guarda los datos modificados por el usuario.
  + **Escenario alternativos**:

1. a. El usuario ha introducido incorrectamente los datos.
   * + 1. El sistema devuelve que campos son incorrectos sin actualizar los datos del usuario

#### 4.2.2.3. Gestionar Lugares

En el siguiente modelo, se va a mostrar el caso de uso de gestión de lugares, de los requisitos funcionales vistos anteriormente.



Figura 4.6 – Casos de uso Gestionar Lugares

* **Caso de uso**: **Buscar lugares**
  + **Descripción**: El invitado buscará lugares en la herramienta.
  + **Actores**: Invitado
  + **Precondiciones**: Ninguno.
  + **Postcondiciones**: El sistema devolverá los lugares buscados.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el invitado decide buscar un lugar.
    2. El invitado busca el nombre, ciudad o provincia desde el menú de búsqueda de lugares.
    3. El sistema devuelve los lugares que coincidan con la búsqueda.
  + **Escenario alternativos**:

1. a. El sistema no encuentra los lugares solicitados.

* **Caso de uso**: **Agregar lugar** **a favoritos**
  + **Descripción**: El usuario buscará lugares en la herramienta y los agregará como favoritos.
  + **Actores**: Usuario.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta..
  + **Postcondiciones**: El sistema añadirá el lugar a los lugares favoritos del usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide agregar un lugar a favoritos.
    2. El usuario busca el nombre, ciudad o provincia desde el menú de búsqueda de lugares.
    3. El usuario selecciona el lugar que quiere añadir a favoritos.
    4. El sistema añade el lugar a los lugares favoritos del usuario.
* **Caso de uso**: **Ver lugares favoritos**
  + **Descripción**: El usuario accederá a sus lugares favoritos.
  + **Actores**: Usuario.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario deberá tener agregado el lugar como favorito
  + **Postcondiciones**: El sistema mostrará los lugares favoritos del usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide agregar un lugar a favoritos.
    2. El usuario buscará el lugar favorito en su perfil.
    3. El sistema mostrará los lugares favoritos del usuario.
* **Caso de uso**: **Eliminar lugar de favoritos**
  + **Descripción**: El usuario eliminará un lugar de sus favoritos.
  + **Actores**: Usuario.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta..
  + **Postcondiciones**: El sistema eliminará el lugar de los lugares favoritos del usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide agregar un lugar a favoritos.
    2. El usuario buscará el lugar favorito en su perfil.
    3. El usuario selecciona el lugar que quiere eliminar de favoritos.
    4. El sistema eliminará el lugar de los lugares favoritos del usuario.
* **Caso de uso**: **Crear Lugar**
  + **Descripción**: El usuario creará un nuevo lugar en la herramienta.
  + **Actores**: Usuario.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta.
  + **Postcondiciones**: El sistema creará un lugar administrado por el usuario.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el usuario decide crear un nuevo lugar.
    2. El usuario accederá al menú de lugares.
    3. El usuario seleccionará crear un nuevo lugar.
    4. El usuario añadirá todos los campos necesarios para la creación del lugar.
    5. El usuario seleccionará crear nuevo lugar.
    6. El sistema añadirá un nuevo lugar.
* **Caso de uso**: **Publicar mensaje**
  + **Descripción**: El administrador publicará un nuevo mensaje en el tablón del lugar.
  + **Actores**: Administrador.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser administrador del lugar.
  + **Postcondiciones**: El sistema publicará un nuevo mensaje en el tablón del lugar.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el administrador decide añadir un nuevo mensaje al tablón del lugar.
    2. El administrador accederá al menú de lugares.
    3. El administrador seleccionará el lugar que desea administrar.
    4. El administrador añadirá todos los campos necesarios para la publicación del mensaje.
    5. El sistema añadirá un mensaje al tablón del lugar.
* **Caso de uso**: **Crear Playlist**
  + **Descripción**: El administrador creará una nueva playlist para un lugar en la herramienta.
  + **Actores**: Administrador.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser administrador del lugar. El administrador debe tener una cuenta Spotify y estar logeado como usuario Spotify.
  + **Postcondiciones**: El sistema creará una nueva playlist asociada al lugar.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el administrador decide crear una nueva playlist.
    2. El administrador accederá al menú de lugares.
    3. El administrador buscará el lugar que desa administrar.
    4. El administrador seleccionará administrar lugar.
    5. El administrador accederá al apartado de playlist.
    6. El administrador creará una nueva playlist.
    7. El sistema añadirá una nueva playlist al lugar.
* **Caso de uso**: **Validar/Rechazar canciones**
  + **Descripción**: El administrador validará o rechazará las canciones enviadas al lugar.
  + **Actores**: Administrador.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser administrador del lugar. El administrador debe tener una cuenta Spotify y estar logeado como usuario Spotify.
  + **Postcondiciones**: El sistema validará o rechazará la canción seleccionada.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el administrador validar una canción.
    2. El administrador accederá al menú de lugares.
    3. El administrador buscará el lugar que desa administrar.
    4. El administrador seleccionará administrar lugar.
    5. El administrador accederá al apartado de playlist.
    6. El administrador accede a la playlist que quiere utilizar.
    7. El administrador validará la canción seleccionada.
    8. El sistema envía a Spotify la canción validada.
    9. El sistema añadirá una nueva playlist al lugar.
  + **Escenario alternativos**:

1. a. El administrador rechaza la canción seleccionada.
2. a. El sistema elimina la canción rechazada.

* **Caso de uso**: **Reproducir canciones**
  + **Descripción**: El administrador reproducirá las canciones que tiene validada.
  + **Actores**: Administrador.
  + **Precondiciones**: El usuario debe estar registrado y logado en la herramienta. El usuario debe ser administrador del lugar. El administrador debe tener una cuenta Spotify y estar logeado como usuario Spotify.
  + **Postcondiciones**: El sistema reproducirá, a través de Spotify, la canción seleccionada.
* **Identiﬁcación de escenarios:**
  + **Escenario principal**:
    1. El caso de uso se inicia cuando el administrador decide reproducir una canción que tiene validada.
    2. El administrador accederá al menú de lugares.
    3. El administrador buscará el lugar que desa administrar.
    4. El administrador seleccionará administrar lugar.
    5. El administrador accederá al apartado de playlist.
    6. El administrador accede a la playlist que quiere utilizar.
    7. El administrador elige en que dispositivo quiere reproducir la canción.
    8. El administrador eligirá que canción quiere reproducir.
    9. El sistema reproducirá la canción en el dispositivo a través de Spotify.

## **Modelo de Comportamiento**

A partir de los casos de uso anteriores, se crea el modelo de comportamiento. Para ello, se rea- lizarán los diagramas de secuencia del sistema, donde se identificarán las operaciones o servicios del sistema. Luego, se detallará el contrato de las operaciones identificadas. Se realizarán los modelos de compartamientos más importantes de la herramienta, puesto que muchos son similares.



### Acceso y salida del sistema

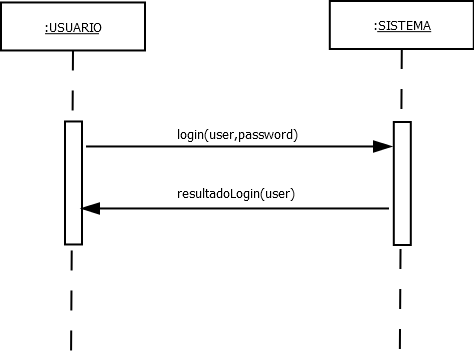


Figura 4.7 – Modelo comportamiento Login interno

A continuación se expone el modelo de comportamiento, cuando el sistema utiliza un sistam como google para iniciar sesión. El sistema externo devuelve Ok cuando el usuario inicia sesión, desconociendo el sistema la contraseña del usuario para el sistema externo. El sistema externo únicamente devuelve el usuario con el que se ha logado.

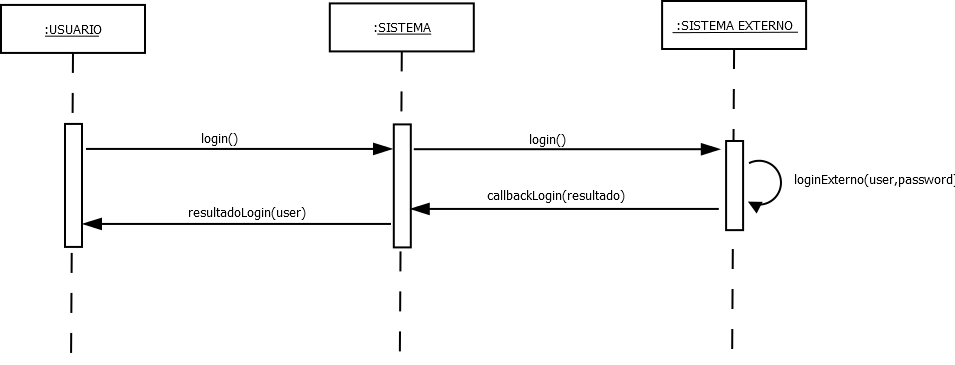


Figura 4.8 – Modelo comportamiento Login externo

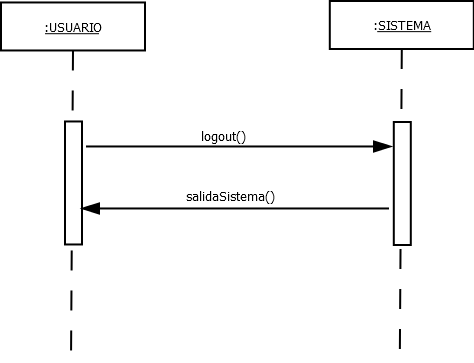


Figura 4.9 – Modelo comportamiento Logout

### Búsqueda y envío de canciones

Para la realización de búsquedas es necesario que el sistema le mande a Spotify un token, autenticando el sistema origen que hace la petición.

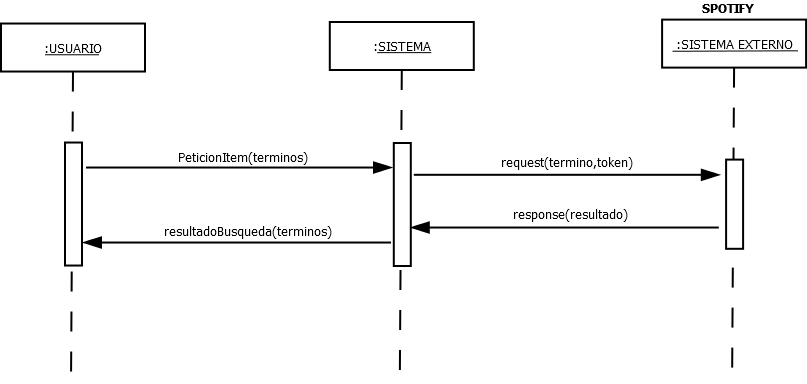


Figura 4.10 – Modelo comportamiento Búsqueda Item

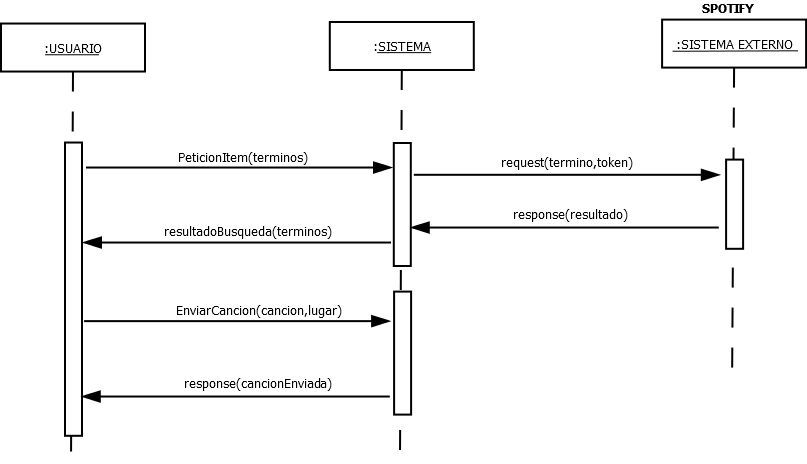


Figura 4.11 – Modelo comportamiento Enviar Canción

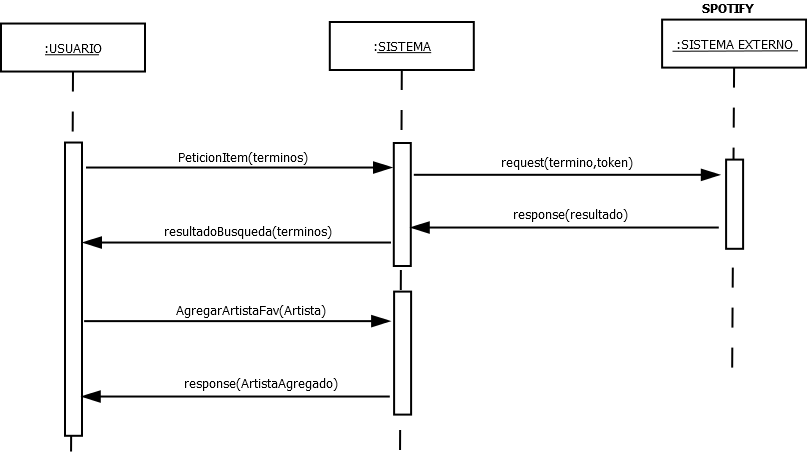


Figura 4.11 – Modelo comportamiento Agregar Artista Favorito

### Gestión de usuarios

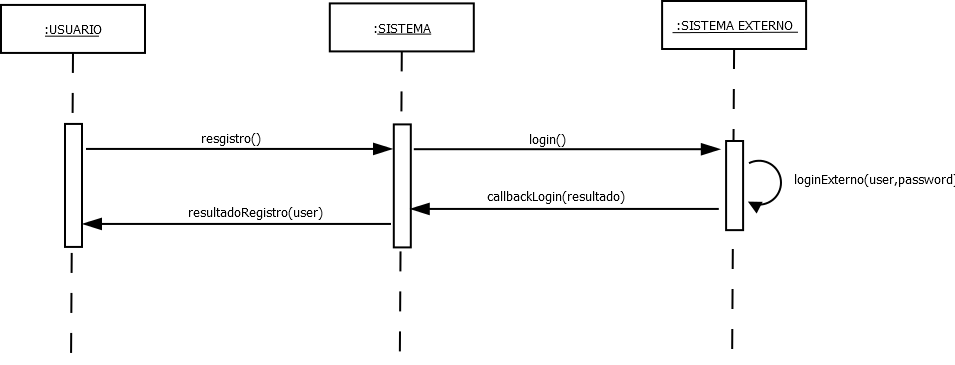


Figura 4.12 – Modelo comportamiento Registro usuario

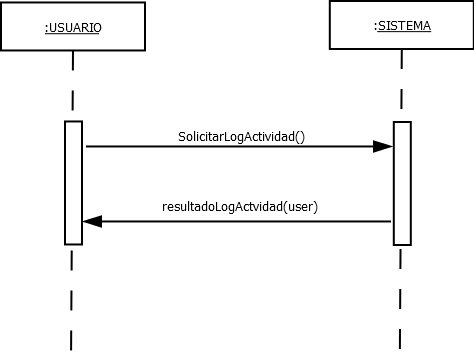


Figura 4.13 – Modelo comportamiento Log de Actvidad

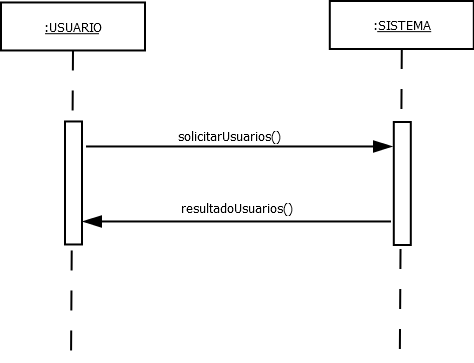


Figura 4.14 – Modelo comportamiento Búsqueda usuarios

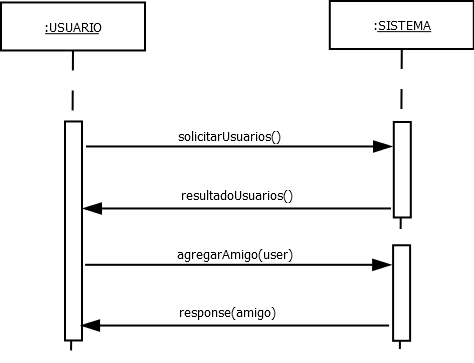


Figura 4.15 – Modelo comportamiento Agregar Amigo

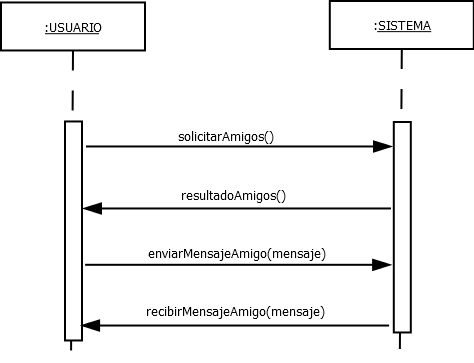


Figura 4.16 – Modelo comportamiento Chatear

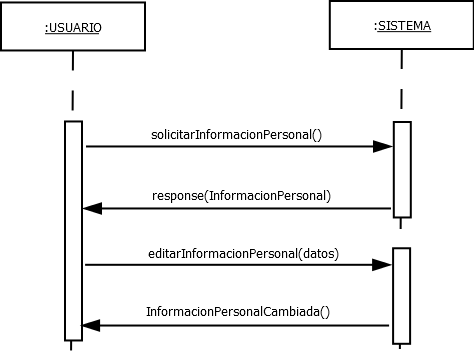


Figura 4.17 – Modelo comportamiento Editar Información Personal

### Gestión de lugares

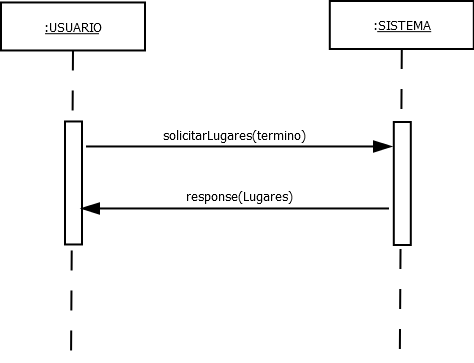


Figura 4.18 – Modelo comportamiento Búsqueda de Lugares

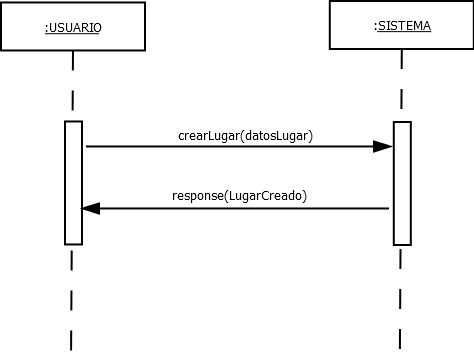


Figura 4.19 – Modelo comportamiento Crear Lugar

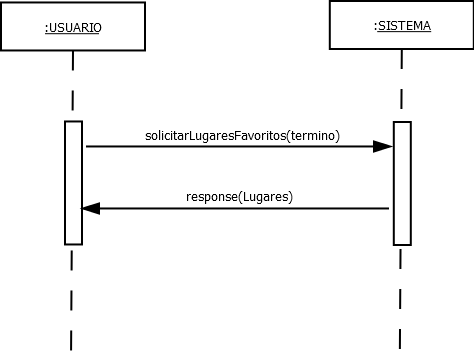


Figura 4.20 – Modelo comportamiento Ver Lugares Favoritos

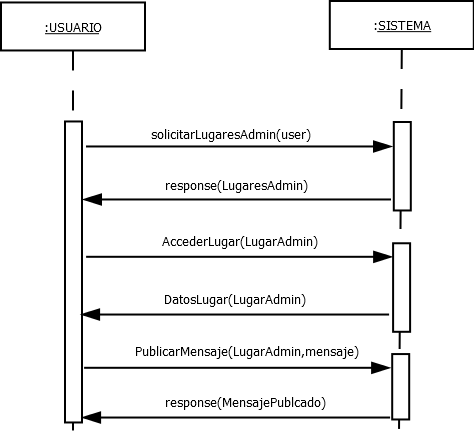


Figura 4.21 – Modelo comportamiento Publicar Mensaje

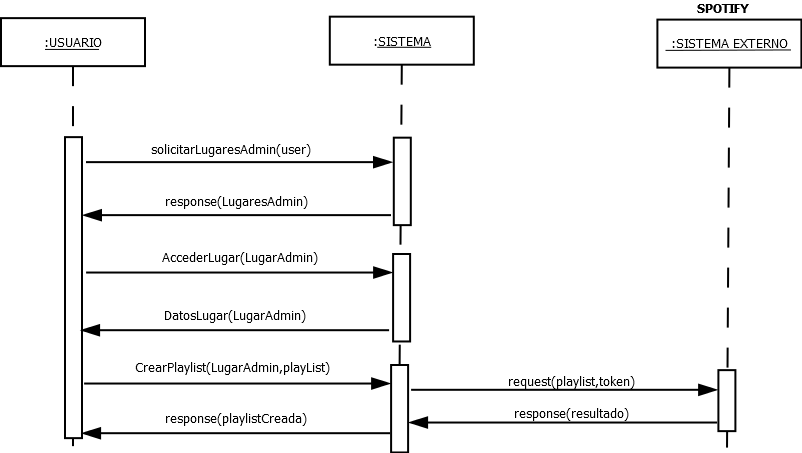


Figura 4.22 – Modelo comportamiento Crear Playlist

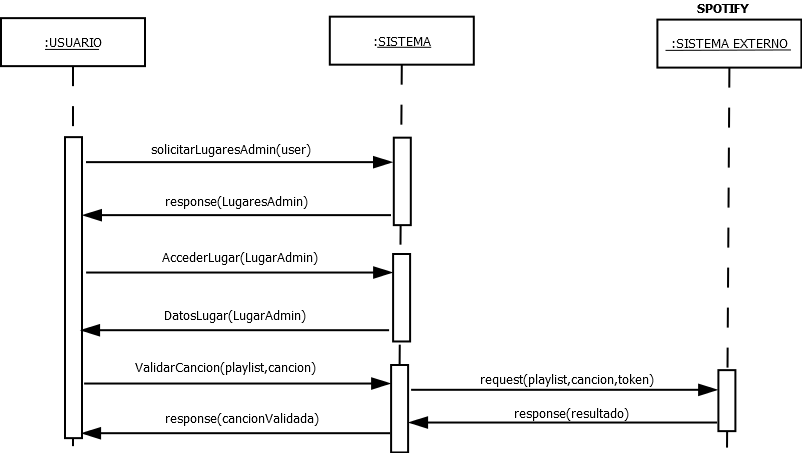


Figura 4.23 – Modelo comportamiento Validar Canción

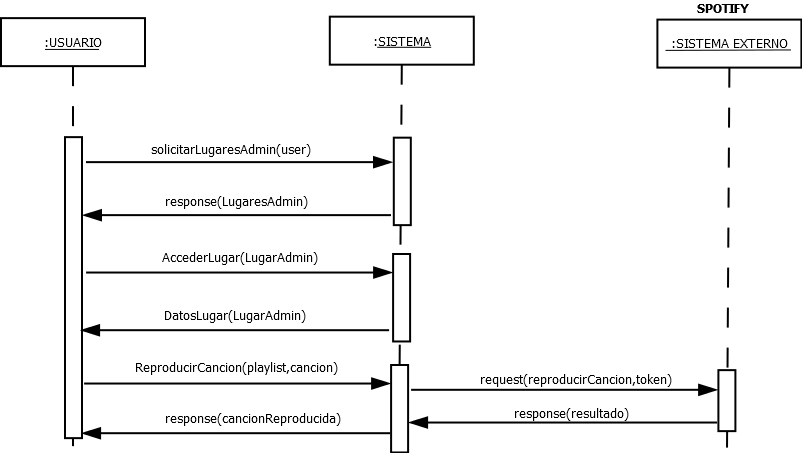


Figura 4.24 – Modelo comportamiento Reproducir Canción

### Contrato de operaciones

En esta sección describiremos las operaciones de los modelos de comportamiento. Solo describiremos las más relevantes. Se indicará que hacen, no como lo hacen. Para diferenciar las variables del sistema con las de la función, las connotaremos con una S al final de la variable.

|  |  |
| --- | --- |
| **Contrato**: Login Interno | |
| **Nombre** | Login(user,password) |
| **Responsabilidad** | Inicia sesión en la aplicación |
| **Excepciones** | Ninguna |
| **Salidas** | Datosusuario : user |
| **Precondiciones** | Debe existir:  userS = user  passwordS = password |
| **Poscondiciones** | Se actualiza que el usuario está conectado, y el sistema muestra la ventana de inicio de la herramienta |

Tabla 4.1 – Contrato: Login(user,password)

|  |  |
| --- | --- |
| **Contrato**: Logout | |
| **Nombre** | Logout() |
| **Responsabilidad** | Finaliza sesión en la aplicación |
| **Excepciones** | Si el usuario no está logado |
| **Salidas** | Ninguna |
| **Precondiciones** | El usuario debe estar logado |
| **Poscondiciones** | Se actualiza que el usuario está desconectado, y el sistema muestra la ventana de inicio de la herramienta sin usuario |

Tabla 4.2 – Contrato: Logout()

|  |  |
| --- | --- |
| **Contrato**: Búsqueda Item | |
| **Nombre** | PeticiónItem(termino) |
| **Responsabilidad** | El sistema devuelve una petición de un elemento buscado |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con el resultado de la petición |
| **Precondiciones** | Ninguna |
| **Poscondiciones** | Se muestra por pantalla los datos solicitados. Si no existe ningún elemento, se muestra una mensaje. |

Tabla 4.3 – Contrato: PeticiónItem(termino)

|  |  |
| --- | --- |
| **Contrato**: Enviar Canción | |
| **Nombre** | EnviarCanción(canción,lugar) |
| **Responsabilidad** | El sistema añade una nueva petición de una canción a un lugar. |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con la canción enviada |
| **Precondiciones** | Ninguna |
| **Poscondiciones** | Se añade a la lista de canciones pendientes de validar por parte del administrador del lugar. |

Tabla 4.4 – Contrato: EnviarCanción(canción,lugar)

|  |  |
| --- | --- |
| **Contrato**: Agregar artista a favorito | |
| **Nombre** | AgregarArtistaFav(artista) |
| **Responsabilidad** | El sistema añade un artista favorito al usuario que está logado |
| **Excepciones** | Si el artista ya está en la lista de favoritos |
| **Salidas** | Objeto Json con la artista añadido |
| **Precondiciones** | El usuario debe estar logado en la herramienta |
| **Poscondiciones** | Se añade a la lista de favoritos el usuario seleccionado. |

Tabla 4.5 – Contrato: AgregarArtistaFav(artista)

|  |  |
| --- | --- |
| **Contrato**: Solicitar log de actividad | |
| **Nombre** | SolicitarLogActividad() |
| **Responsabilidad** | El sistema devuelve el log de actividad del usuario conectado |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con el log de actividad |
| **Precondiciones** | El usuario debe estar logado en la herramienta |
| **Poscondiciones** | El sistema muestra la pantalla del log de actividad con los datos del mismo. |

Tabla 4.6 – Contrato: SolicitarLogActividad()

|  |  |
| --- | --- |
| **Contrato**: Solicitar usuarios | |
| **Nombre** | SolicitarUsuarios() |
| **Responsabilidad** | El sistema devuelve los usuarios registrados en la aplicación |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con los usuarios de la aplicación |
| **Precondiciones** | El usuario debe estar logado en la herramienta |
| **Poscondiciones** | El sistema muestra la pantalla de usuarios con los usuarios del sistema |

Tabla 4.7 – Contrato: SolicitarUsuarios()

|  |  |
| --- | --- |
| **Contrato**: Agregar Amigo | |
| **Nombre** | AgregarAmigo(user) |
| **Responsabilidad** | El sistema inserta un nuevo amigo al conjunto de amigos del usuario |
| **Excepciones** | El amigo no debe estar previamente en la lista de amigos |
| **Salidas** | Objeto Json con el amigo añadido |
| **Precondiciones** | El usuario debe estar logado en la herramienta |
| **Poscondiciones** | El sistema añade a user a la colección de amigos del usuario |

Tabla 4.8 – Contrato: AgregarAmigo(user)

|  |  |
| --- | --- |
| **Contrato**: solicitar lugares | |
| **Nombre** | solicitarLugares(termino) |
| **Responsabilidad** | El sistema devuelve los lugares que coincidan con el término |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con la lista de lugares |
| **Precondiciones** | Ninguna |
| **Poscondiciones** | El sistema muestra la pantalla de lugares, filtrada por el término. Si no existe ninguno el sistema mostrará un mensaje. |

Tabla 4.9 – Contrato: solicitarLugares(termino)

|  |  |
| --- | --- |
| **Contrato**: Crear Lugar | |
| **Nombre** | crearLugar(datosLugar) |
| **Responsabilidad** | El sistema añade un nuevo lugar |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con el lugar creado |
| **Precondiciones** | El usuario debe estar logado en la herramienta |
| **Poscondiciones** | El sistema crea el lugar con los datos proporcionados por el usuario.  El sistema muestra la pantalla de detalle del lugar.  El sistema asigna el rol de Administrador del lugar al usuario que lo ha creado. |

Tabla 4.10 – Contrato: crearLugar(datosLugar)

|  |  |
| --- | --- |
| **Contrato**: Publicar mensaje | |
| **Nombre** | publicarMensaje(lugarAdmin,mensaje) |
| **Responsabilidad** | El sistema inserta un nuevo mensaje en el lugarS = lugarAdmin |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con el mensaje insertado |
| **Precondiciones** | El usuario debe estar logado en la herramienta y ser el administrador del lugar |
| **Poscondiciones** | El sistema inserta el mensaje, en la lista de mensajes del lugar |

Tabla 4.11 – Contrato: publicarMensaje(lugarAdmin,mensaje)

|  |  |
| --- | --- |
| **Contrato**: Crear Playlist | |
| **Nombre** | crearPlaylist(lugarAdmin,playlist) |
| **Responsabilidad** | El sistema inserta una nueva playlist en el lugarS = lugarAdmin |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con la playlist creada |
| **Precondiciones** | El usuario debe estar logado en la herramienta, ser el administrador del lugar y estar logado con el usuario de Spotify. |
| **Poscondiciones** | El sistema inserta una nueva playlist, y la crea en Spotify.  El sistema muestra las canciones a validar con la playlist creada. |

Tabla 4.12 – Contrato: crearPlaylist(lugarAdmin,playlist)

|  |  |
| --- | --- |
| **Contrato**: Validar Canción | |
| **Nombre** | validarCancion(canción,playlist) |
| **Responsabilidad** | El sistema valida una canción previamente solicitada por usuarios. |
| **Excepciones** | Ninguna |
| **Salidas** | Objeto Json con la canción validada |
| **Precondiciones** | El usuario debe estar logado en la herramienta, ser el administrador del lugar y estar logado con el usuario de Spotify. |
| **Poscondiciones** | El sistema añade la canción a la playlist seleccionada.  El sistema añade al log que ha sido validada la canción. |

Tabla 4.13 – Contrato: validarCancion(canción,playlist)

## **Modelo de Interfaz de Usuario**

Estos prototipos será utilizado para el desarrollo de la aplicación más adelante, siendo lo más fieles posibles.

A continuación se muestran los prototipos de la aplicación web, mostrando las pantallas más relevantes y la estructura general de la misma.

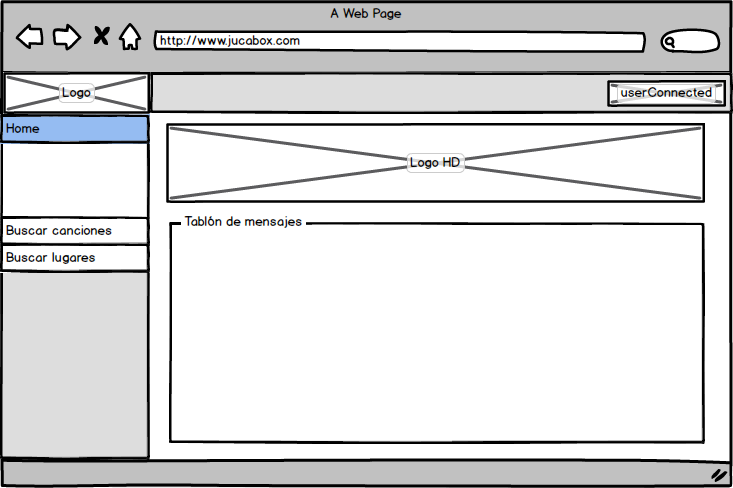


Figura 4.25 –Prototipado – Home

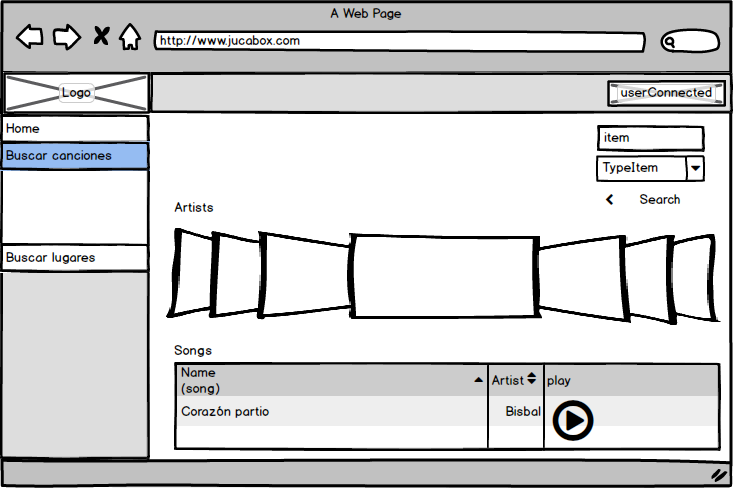


Figura 4.26 – Prototipado – Buscar Canción

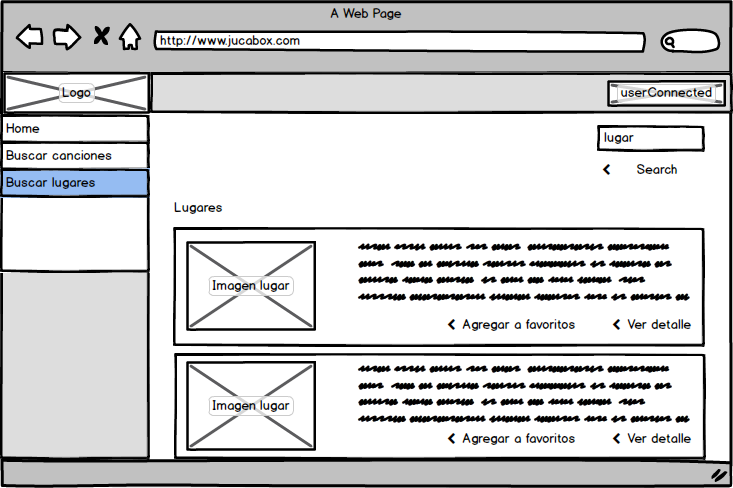


Figura 4.27 – Prototipado – Buscar lugar



Figura 4.28 – Prototipado – Datos lugar

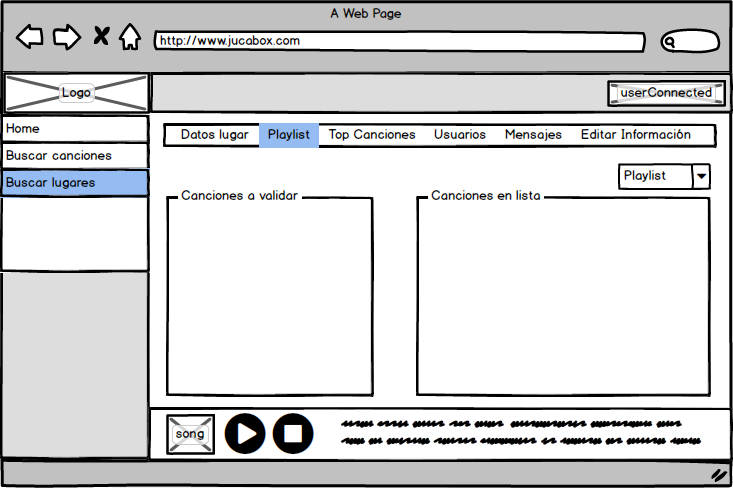


Figura 4.29 – Prototipado – Playlist

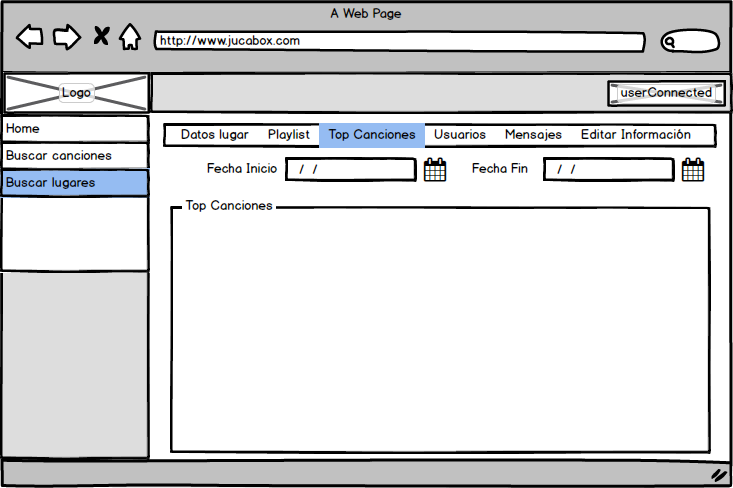


Figura 4.30 – Prototipado – Top canciones



Figura 4.31 – Prototipado – Usuarios Lugar

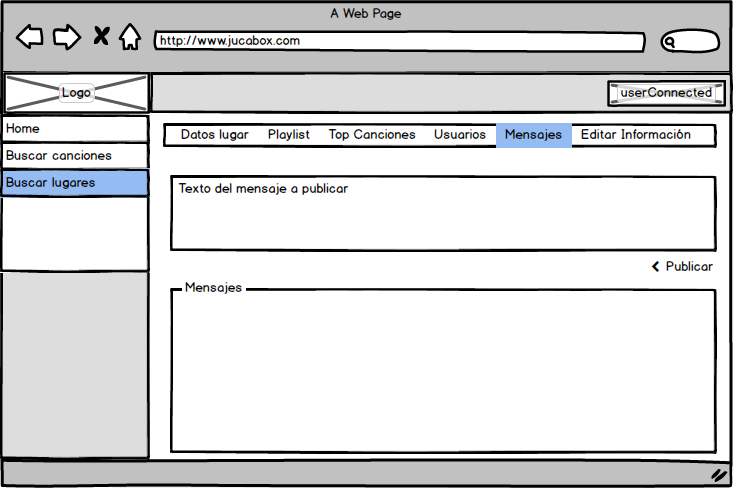


Figura 4.32 – Prototipado – Mensajes lugar

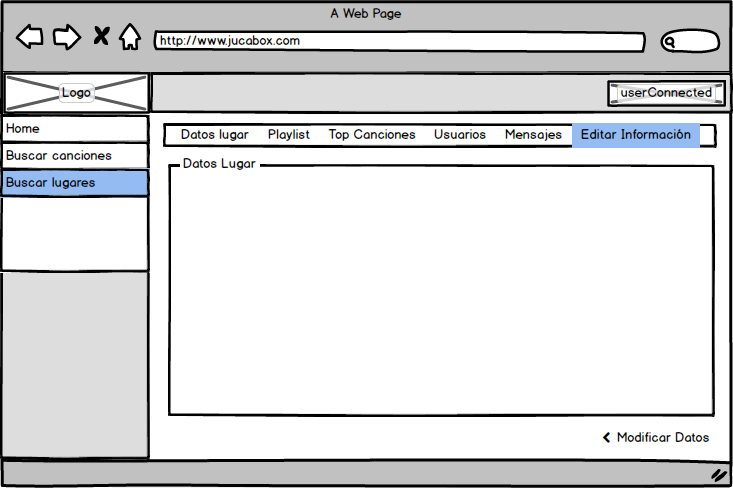


Figura 4.33 – Prototipado – Editar información Lugar

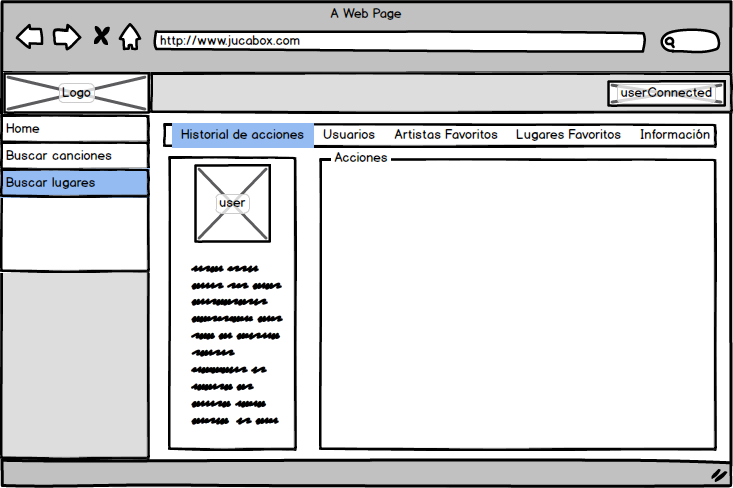


Figura 4.34 – Prototipado – Historial Acciones Usuario

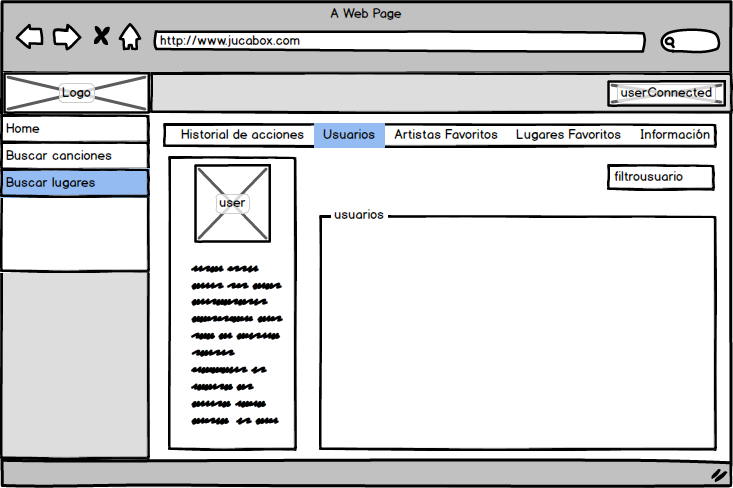


Figura 4.35 – Prototipado – Amigos de Usuario

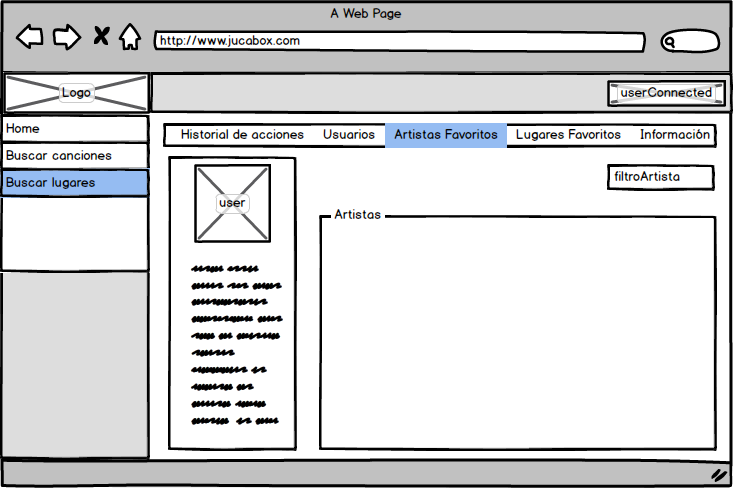


Figura 4.36 – Prototipado – Artistas Favoritos Usuario

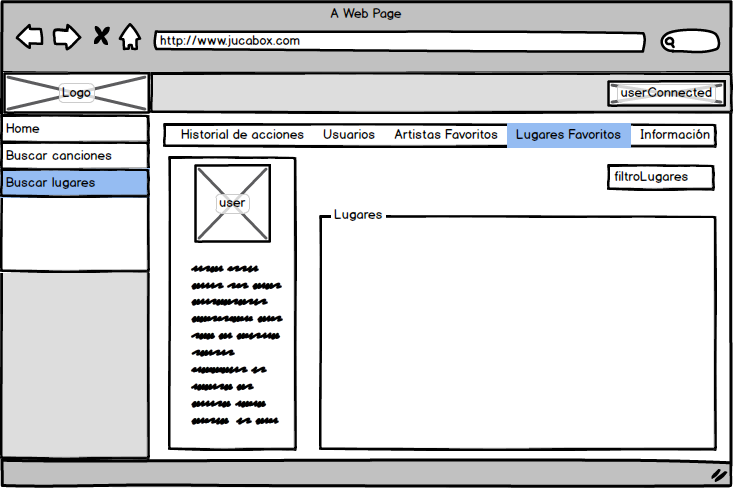


Figura 4.37 – Prototipado – Lugares Favoritos Usuario

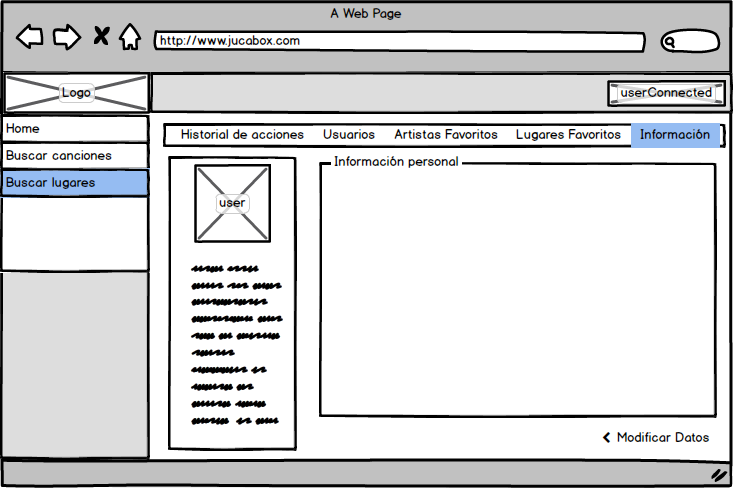


Figura 4.38 – Prototipado – Información Personal Usuario

A continuación se muestran los diagramas de interacción de la aplicación Web. A través de ellos se puede observar la navegación por el sistema.

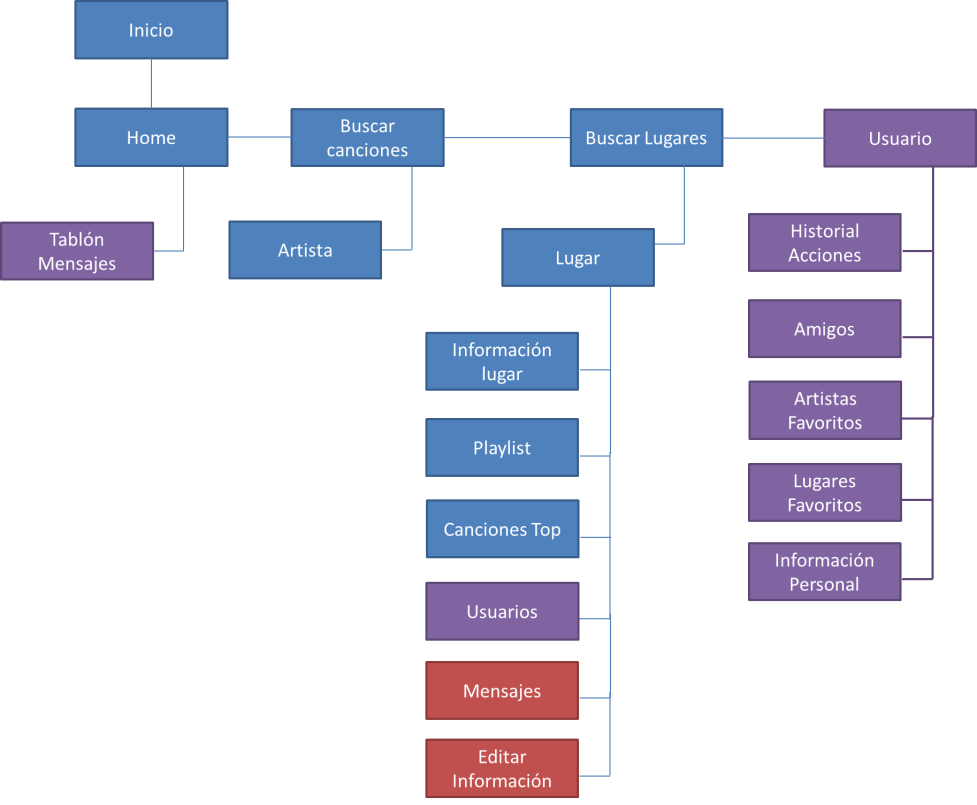


Figura 4.39 – Diagrama de interacción jUCAbox

# Capítulo 5

# Diseño del Sistema

En esta sección se recoge la arquitectura general del sistema de información, el diseño físico de datos, el diseño detallado de componentes software y el diseño detallado de la interfaz de usuario.

## **Arquitectura del Sistema**

En esta sección se define la arquitectura general del sistema de información, donde se especifica la infraestructura tecnológica necesaria para dar soporte al software y la estructura de los componentes que lo con forman.

### Arquitectura Física

En este apartado se va a deficinir la arquitectura utilizada para el desarrollo del sistema, tanto de la parte cliente como de la parte servidor.

En el lado del servidor, vamos a necesitar una infraestructura necesaria para albergar Mongodb 3.2.11, que será nuestro sistema de base de datos. A su vez debe tener instaladas las dependencias de Express 4.15.13 para poder comunicarse con Node.js 6.11.10 que a su vez incorpora el inyector de dependencias npm 3.10.10.

Para la ejecución del software, es necesario únicamente conexión a internet y un navegador web. No existen limitaciones sobre el navegador, ni el dispositivos desde el cual pueda visualizar la web.

### Arquitectura Lógica

En este apartado se va a describir la arquitectura lógica de la aplicación.

En la siguiente figura se muestra el diagrama de componentes y las dependencias existentes de la herramienta.

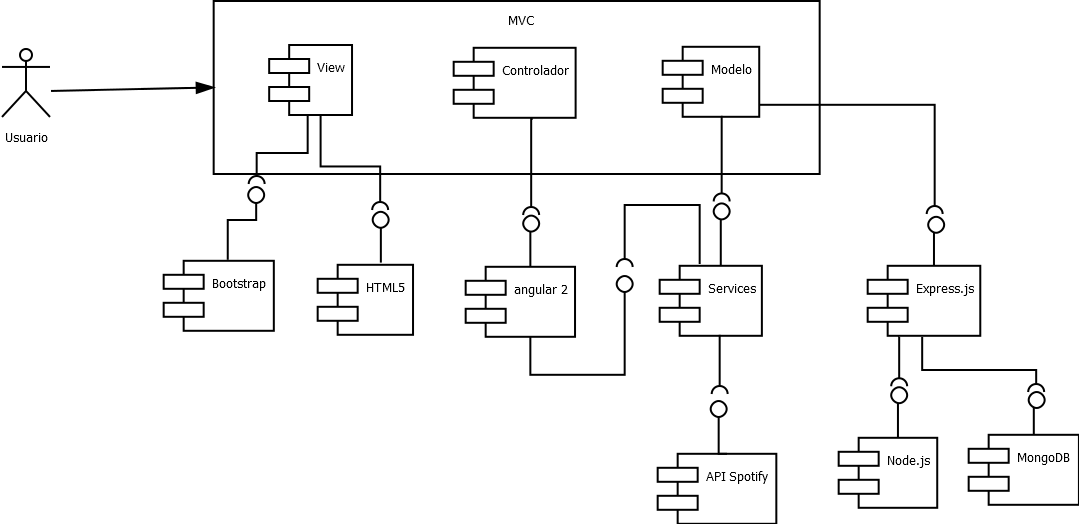


Figura 5.1 – Diagrama de componentes

Para la arquitecura de la aplicación web se utilizó el patrón de diseño MVC (Modelo – Vista – Controlador) pues es en el que se basa el fork MEAN Stack. En este patrón de diseño, se separa la lógica de negocio, de la lógica visual. El patrón de diseño se divide en tres partes anteriormente mencionadas:

* **Modelo**
  + Es el encargado de controlar los datos, asi como su acceso y modificación. En el cual se implementan los roles de acceso y permisos para dichas consultas. En nuestra aplicación el encargado es Mongodb, que a través de Express.js y Node.js, hacen que el acceso a los datos sean seguro.
* **Vista**
  + Es el encargado de renderizar los datos proporcionado por el controlador, es la capa visual que puede ver y utilizar el usuario. Es el que muestra la interfaz de usuario. En nuestra aplicación es controlado por Angular2, mediante el ViewModel.
* **Controlador**
  + Es el encargado de invocar peticiones y responde a eventos cuando se solicita una petición a través de la vista. Esta función la cubre también Angular2, a través de su interacción con los Templates de html5.

## **Diseño Físico de Datos**

En esta sección se define la estructura física de datos que utilizará el sistema, a partir del modelo de conceptual de clases, de manera que teniendo presente los requisitos establecidos para el sistema de información y las particularidades del entorno tecnológico, se consiga un acceso eficiente de los datos. La estructura física en este caso difiere a la de los sistemas relacionales. MongoDB al ser un sistema NoSQL, no contiene tablas ni registros, sino colecciones y documentos.

Este tipo de bases de datos son utilizadas para el conocido mundo BIG Data, con lo que pueden manejar gran cantidad de datos de manera muy eficiente. El modelo de datos se considera muy flexible, pues dos documentos de una misma colección podrían no contener los mismos campos.

Aún así, se establecen unos modelos básicos, donde se definen que campos puede tener un documento, no se consideran obligatorios, pero si deben de cumplir con el tipo de datos definido.

Estas bases de datos, se basan en documentos JSON, en el caso de Mongodb los llama BSON, son documentos de pares clave-valor.

Los tipos de datos soportado por cada clave-valor son los siguientes:

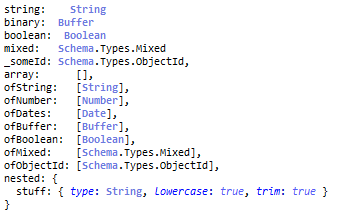


Figura 5.2 – Tipos de datos MongoDB

Cabe recordar que en Mongodb, existe un identificador autonumérico, que nunca se define y se crea por defecto que es el \_id. Siendo único en toda la colección, y de tipo Objectid.

En los sistemas relaciones, si existe una relación M:N se crearía una tabla intermedia. Por ejemplo, en el caso de Lugares Favoritos, se crearía una tabla de relación con Usuario-Lugares. Pero en las no relacionales directamente en el modelo Usuario, se añadiría un nuevo atributo de tipo Lugar, el cual, sería un array de lugares.

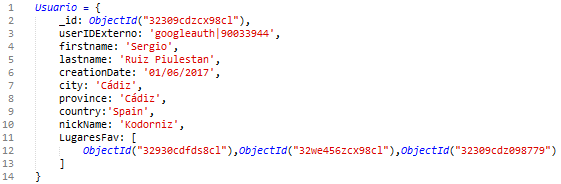


Figura 5.3 – Objeto Usuario

Mediante el método populate de Mongodb, se podría extrae los datos del lugar, puesto que en la definición del modelo Usuario, se ha indicado que LugaresFav es un array de lugares.

Cabe destacar que en los modelos no relacionales, los campos no son obligatorios.

A continuación se van a definir los modelos en los que se va a basar nuestra herramienta:

* Usuario

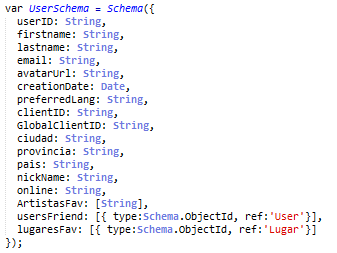


Figura 5.4 – Objeto de definición de la colección Usuario

Como se puede observar en UsersFriends, es un array de objectos de identificadores de usuario. En la base de datos solo se guardaría los identificadores que hace referencia, pero luego cuando se obtienen los datos se puede hacer una población (populate) de los mismos para extrar toda la información.

|  |  |  |
| --- | --- | --- |
| **Nombre de campo** | **Tipo de datos** | **Descripción** |
| **userID** | String | Id externo proporcionado por el plugin Auth0 |
| **firstname** | String | Nombre del usuario |
| **lastname** | String | Apellidos del usuario |
| **email** | String | Correo del usuario |
| **avatarUrl** | String | Dirección url de la imagen del usuario |
| **creationDate** | Date | Fecha de registro del usuario |
| **preferredLanguage** | String | Lenguaje de origen del usuario |
| **clientID** | String | Id externo proporcionado por el plugin Auth0 |
| **globalclientID** | String | Id externo proporcionado por el plugin Auth0 |
| **ciudad** | String | Ciudad del usuario |
| **provincia** | String | Provincia del usuario |
| **país** | String | Pais del usuario |
| **nickname** | String | Alias del usuario |
| **online** | String | Estado de conexión del usuario |
| **artistasFav** | Array de String | Artistas favoritos del usurio. Al no se una colección de jUCAbox, guardamos el objeto completo |
| **userFriends** | Array de ObjectID de users | Usuarios agregados como amigos |
| **lugaresFav** | Array de ObjectID de lugares | Lugares favoritos del usuario |

Tabla 5.1 – Campos entidad usuario

* Lugar



Figura 5.5 – Objeto de definición de la colección Lugar

|  |  |  |
| --- | --- | --- |
| **Nombre de campo** | **Tipo de datos** | **Descripción** |
| **nombre** | String | Nombre del lugar |
| **descripción** | String | Descripción del lugar |
| **img** | Array de String | Rutas de imágenes a mostrar |
| **provincia** | String | Provincia del lugar |
| **ciudad** | String | Ciudad del lugar |
| **email** | String | Correo del lugar |
| **direccion** | String | Dirección física del lugar para situarlo en google maps |
| **userAdmin** | ObjectID de usuario | Id del usuario administrador |
| **tipoMusica** | Array de objectos | Tipo de música del local |
| **token** | String | Clave por lo cual se comprueba para que el usuario pueda enviar una canción |
| **playlist** | Array de ObjectID de Playlist | Id de las playlists del lugar |

Tabla 5.2 – Campos entidad lugar

* Playlist

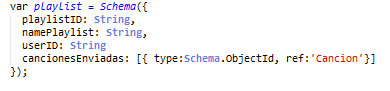


Figura 5.6 – Objeto de definición de la colección Playlist

|  |  |  |
| --- | --- | --- |
| **Nombre de campo** | **Tipo de datos** | **Descripción** |
| **playlistID** | String | Id de la playlist en Spotify |
| **namePlaylist** | String | Nombre de la playlist |
| **userID** | String | Usuario de Spotify owner de la playlist |
| **cancionesEnviadas** | Array de ObjectID de Canciones | Canciones enviadas a la playlist |

Tabla 5.3 – Campos entidad playlist

* Canción

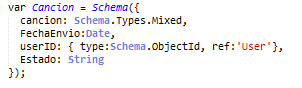


Figura 5.7 – Objeto de definición de la colección Canción

|  |  |  |
| --- | --- | --- |
| **Nombre de campo** | **Tipo de datos** | **Descripción** |
| **canción** | Objeto JSON | Objeto completo de la canción de Spotify |
| **Fecha Envío** | Date | Fecha de Envío de la canción a jUCAbox |
| **Estado** | String | Estado de la canción. Si está pendiente, validada o rechazada |
| **userID** | String | Usuario que ha solicitado la canción |

Tabla 5.4 – Campos entidad canción

* Log



Figura 5.8 – Objeto de definición de la colección Log

Log es una colección donde se almacenan los eventos de la aplicación.

|  |  |  |
| --- | --- | --- |
| **Nombre de campo** | **Tipo de datos** | **Descripción** |
| **userID** | ObjectID de usuario | Id del usuario que ha realizado la acción |
| **tipoMensaje** | String | La categoría del mensaje, si es envio canción, creación de lugar, validación de canción o agregar amigo |
| **mensaje** | String | Cuerpo del log |
| **FechaLog** | Date | Fecha cuando se realiza la acción |

Tabla 5.4 – Campos entidad log

Al no ser un sistema relacionale, el esquema relacional se basará en el modelo conceptual de datos ya descrito en capítulos anteriores.

## **Diseño detallado de Componentes**

A continuación vamos a describir algunos diagramas de secuencia de los módulos funcionales de la herramienta. Solo mostraremos los más importantes, debido a las similitudes con otros.

* **Caso de uso: Búsqueda de ítems Spotify**

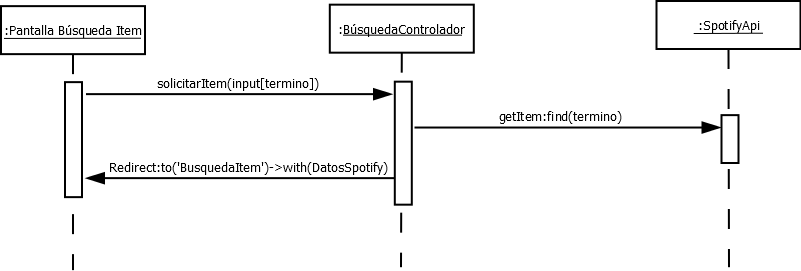
****

Figura 5.9 – Diagrama de secuencia – Caso de uso búsqueda de ítems

* **Caso de uso: Enviar Canción**

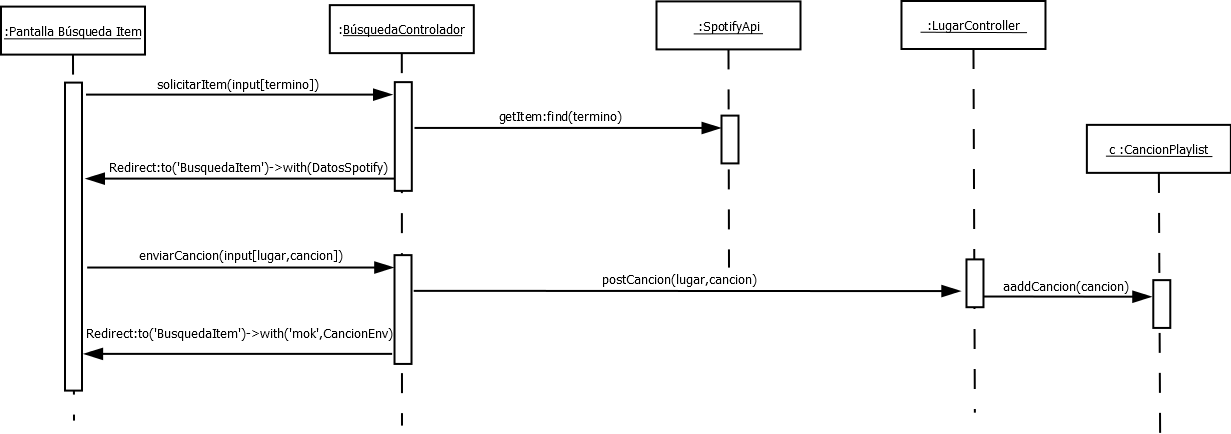


Figura 5.10 – Diagrama de secuencia – Caso de uso enviar canción

* **Caso de uso: Añadir artista a favoritos**

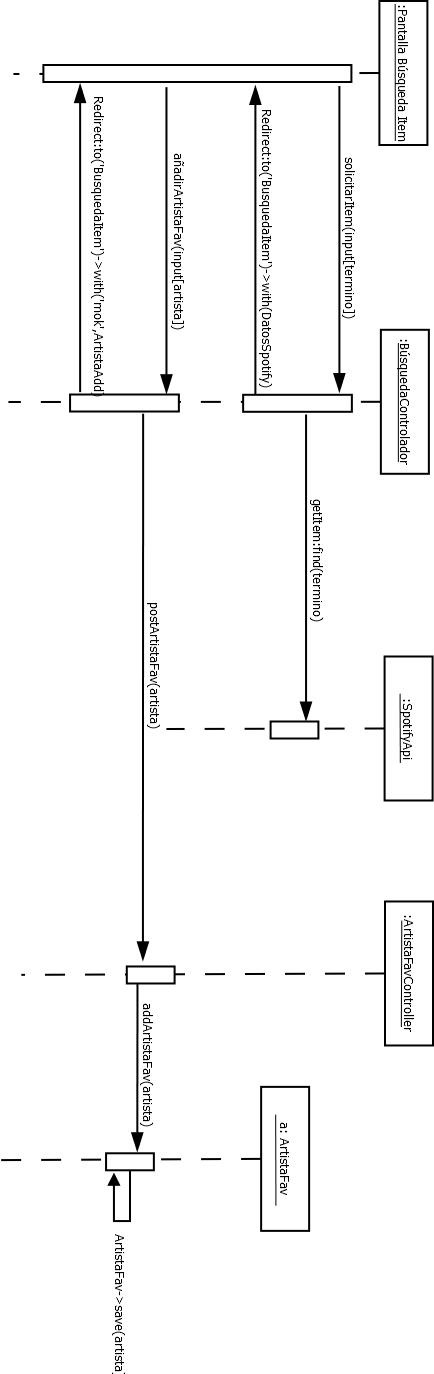


Figura 5.11 – Diagrama de secuencia – Caso de uso añadir artista a favoritos

* **Caso de uso: Eliminar artista de favoritos**

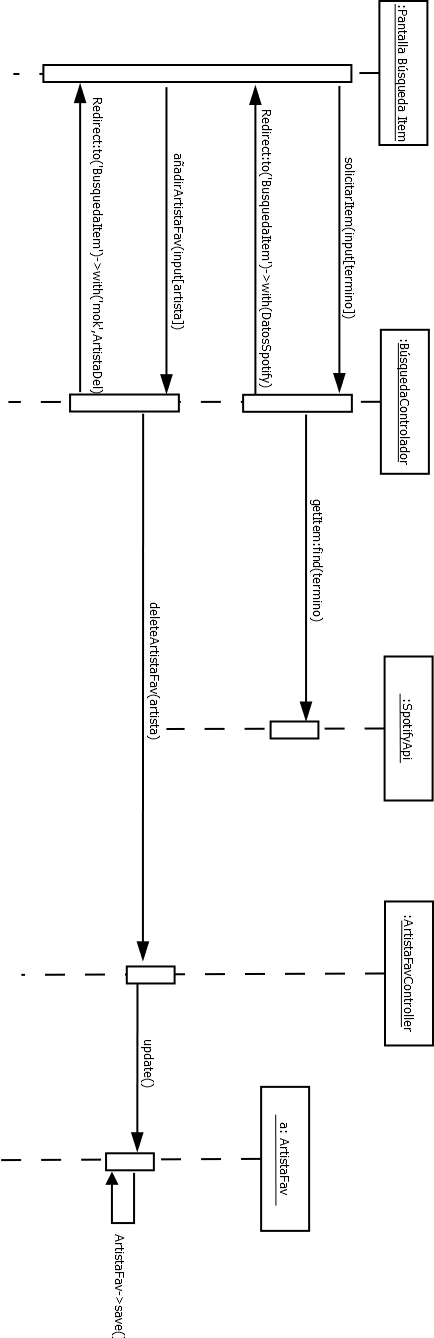


Figura 5.12 – Diagrama de secuencia – Caso de uso eliminar artista de favoritos

* **Caso de uso: Registro en el sistema**

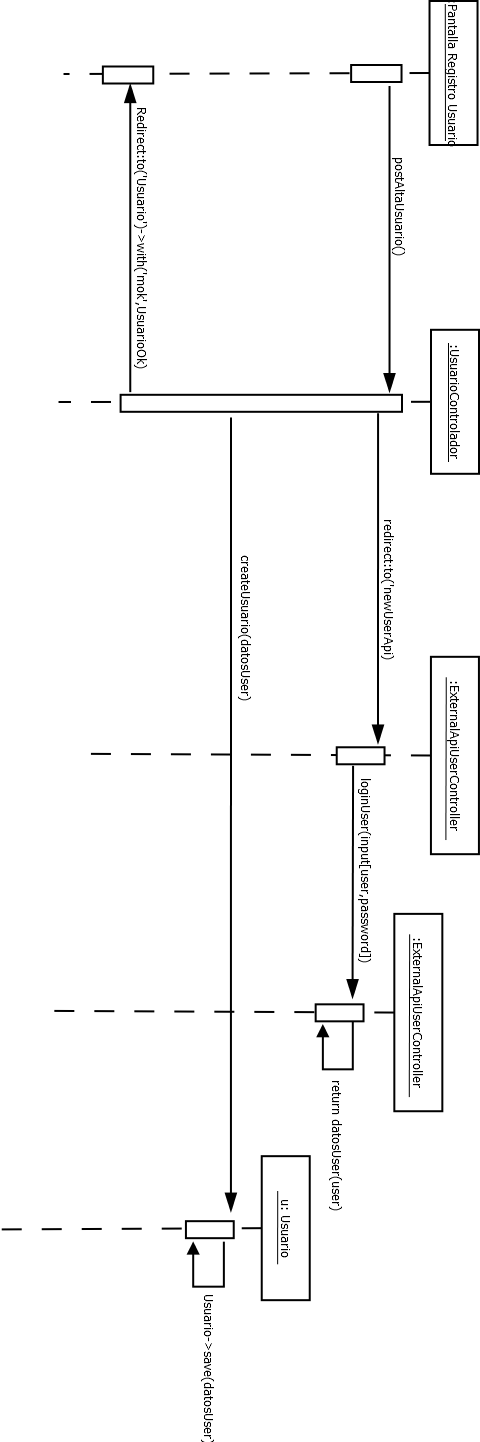


Figura 5.13 – Diagrama de secuencia – Caso de uso registro en el sistema

* **Caso de uso: Ver log actividad**

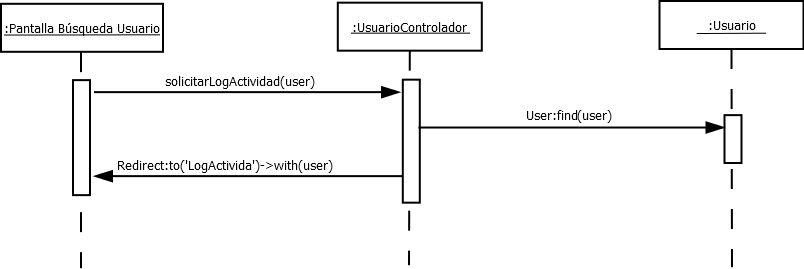


Figura 5.14 – Diagrama de secuencia – Caso de uso ver log actividad

* **Caso de uso: Buscar usuarios**

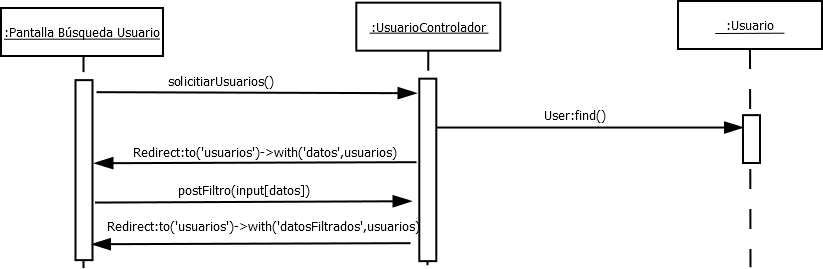


Figura 5.15 – Diagrama de secuencia – Caso de uso buscar usuarios

* **Caso de uso: Agregar nuevo amigo**

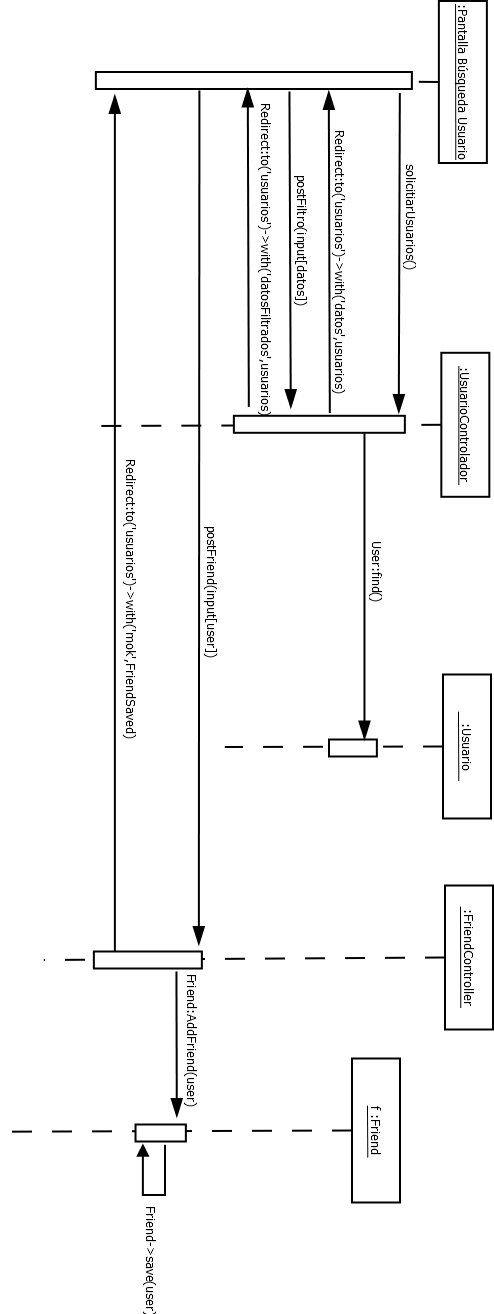


Figura 5.16 – Diagrama de secuencia – Caso de uso Agregar nuevo amigo

* **Caso de uso: Agregar nuevo amigo**

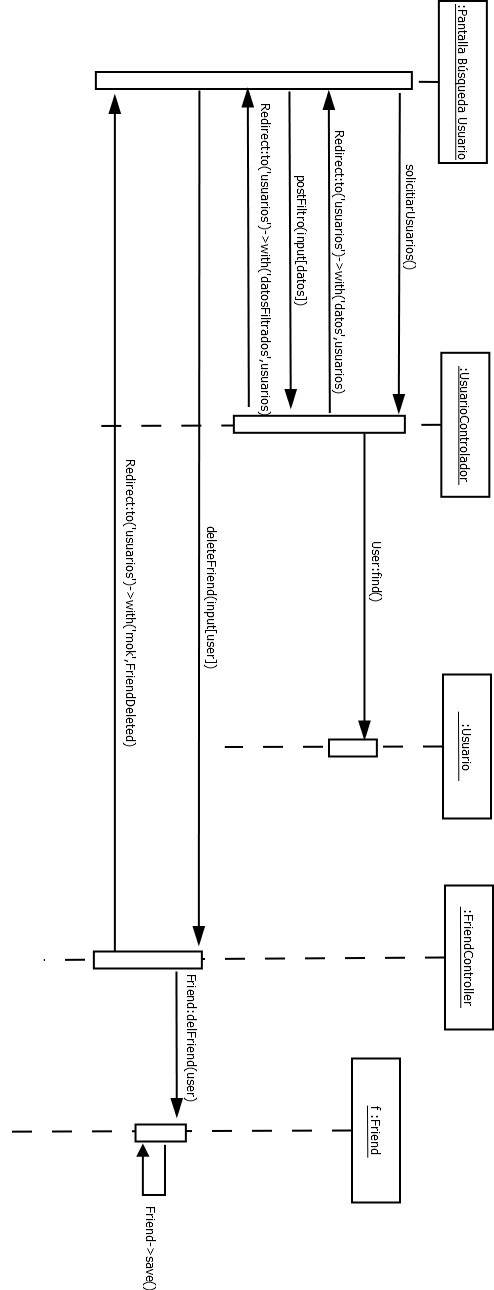


Figura 5.17 – Diagrama de secuencia – Caso de uso Agregar nuevo amigo

* **Caso de uso: Editar información personal**

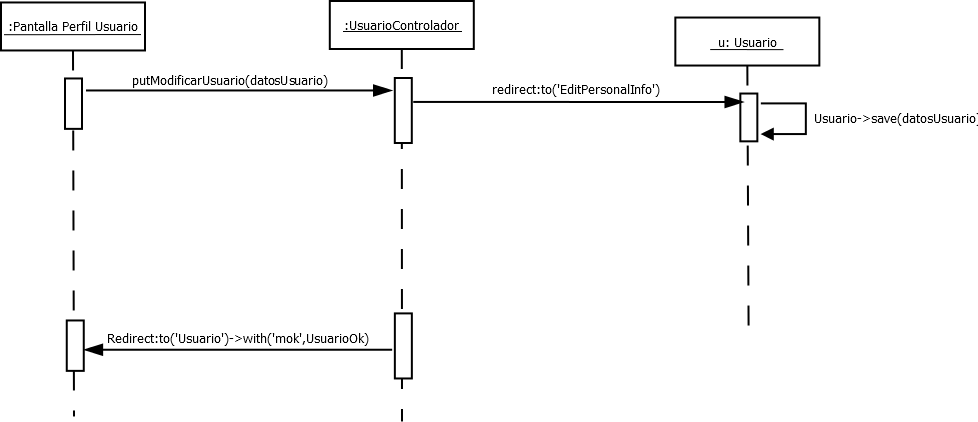


Figura 5.18 – Diagrama de secuencia – Caso de uso editar información personal

* **Caso de uso: Buscar lugares**

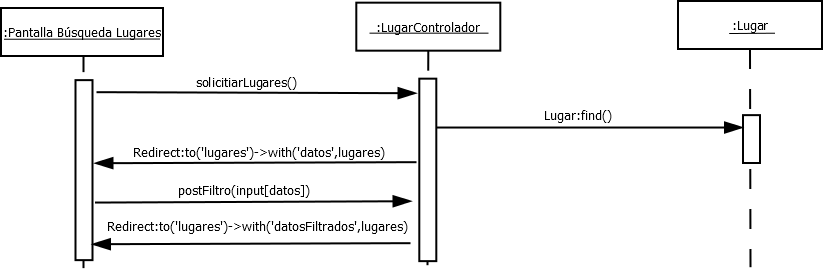


Figura 5.19 – Diagrama de secuencia – Caso de uso buscar lugares

* **Caso de uso: Añadir lugar a favoritos**



Figura 5.20 – Diagrama de secuencia – Caso de uso añadir lugar a favoritos

* **Caso de uso: Eliminar lugar de favoritos**

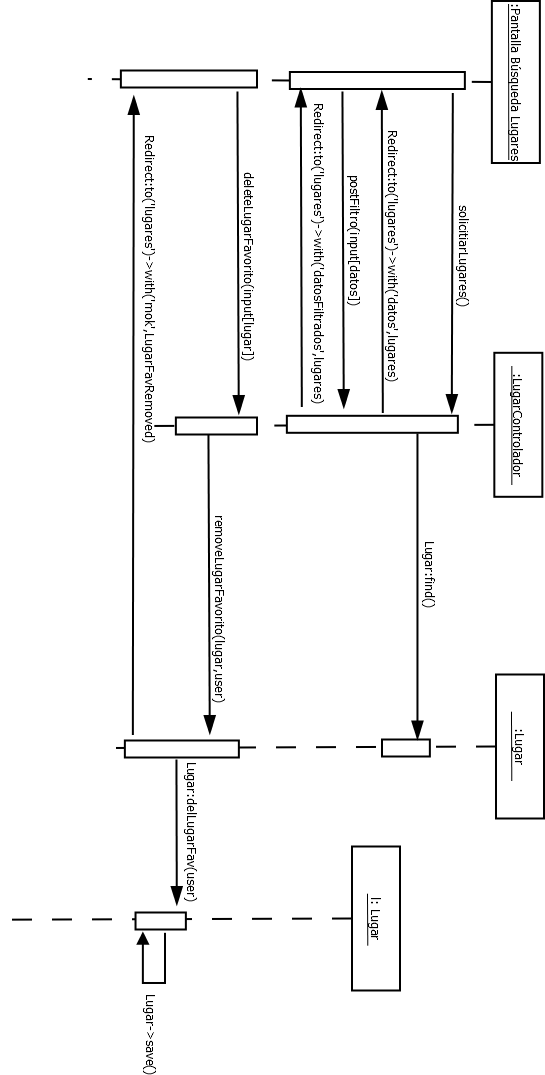


Figura 5.21 – Diagrama de secuencia – Caso de uso eliminar lugar de favoritos

* **Caso de uso: Crear lugar**

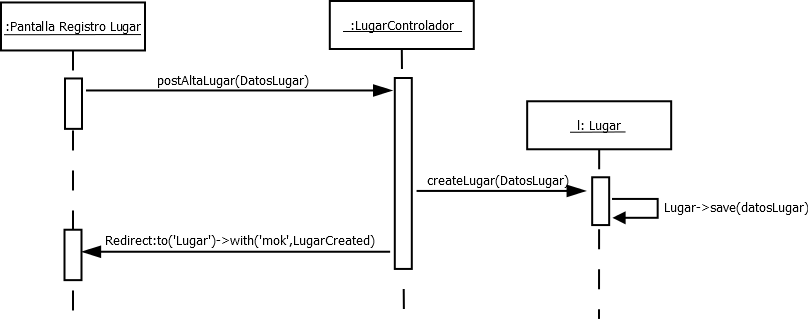


Figura 5.22 – Diagrama de secuencia – Caso de uso crear lugar

* **Caso de uso: Publicar mensaje**

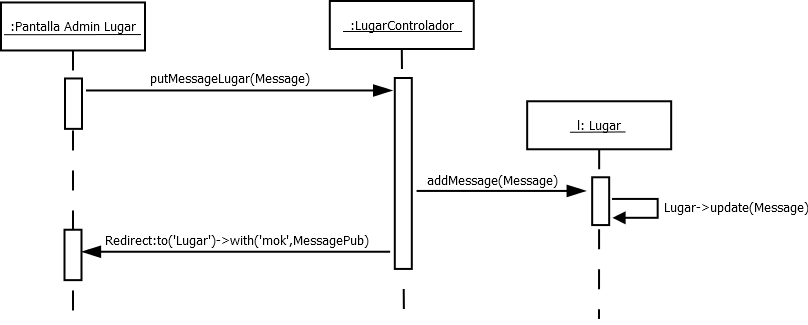


Figura 5.23 – Diagrama de secuencia – Caso de uso publicar mensaje

* **Caso de uso: Validar canciones**

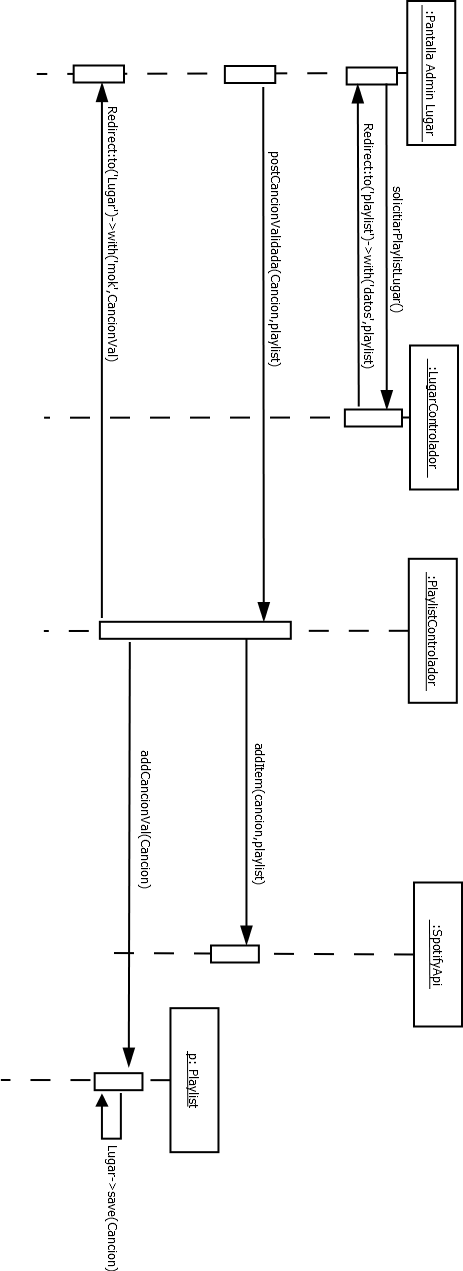


Figura 5.24 – Diagrama de secuencia – Caso de uso publicar mensaje

* **Caso de uso: Reproducir canciones**

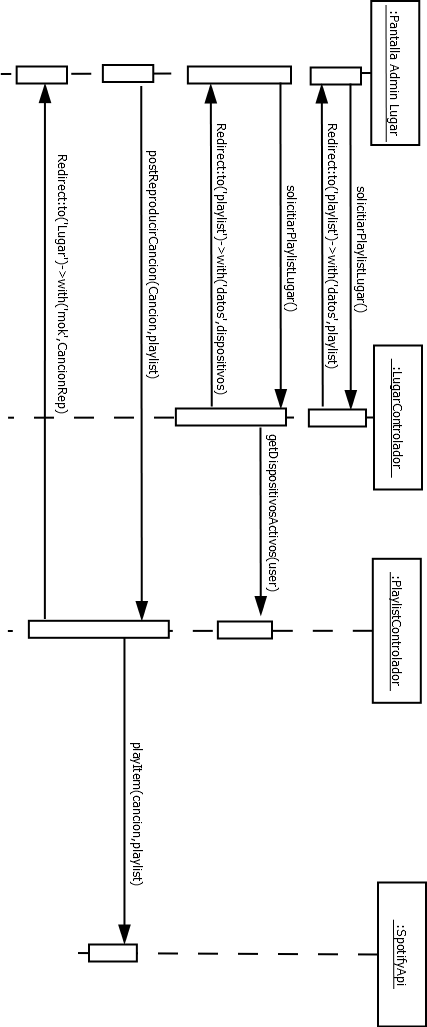


Figura 5.25 – Diagrama de secuencia – Caso de uso reproducir canciones

## **Diseño detallado de la Interfaz de Usuario**

En esta sección se detallará la interfaz de usuario de la aplicación web, para ello se mostrará un ejemplo de la pantalla principal y el aspecto que tendrá con sus diferentes partes.

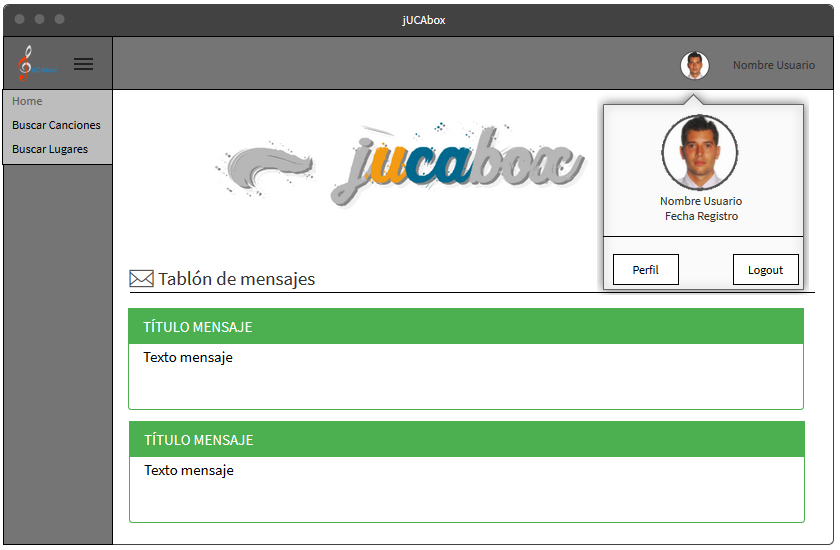


Figura 5.26 – Diseña de la pantalla principal jUCAbox

Se ha dividio la herramienta en 3 bloques diferentes. El bloque 1, muestra el menú principal de la aplicación. Este bloque es un menú lateral desarrollado de manera responsive, por lo que cuando se vea en una pantalla de menor tamaño se mostrará oculto. El bloque 2 muestra la información del usuario conectado y el bloque 3 muestra información sobre la página que estamos consultando.

En el capítulo del manual de usuario, se detallará con más profundidad todas las partes de la herramienta y su interacción con el resto de elementos.

# Capítulo 6

# Construcción del Sistema

Este capítulo trata sobre todos los aspectos relacionados con la implementación del sistema en código, haciendo uso de un determinado entorno tecnológico.

## **Entorno de Construcción**

Para el desarrollo de la herramienta se a utilizado el fork de herramientas Mean Stack, el cual utiliza un desarrollo ent-to-end basadas en JavaScript como lenguaje principial, tanto en el servidor como en el cliente. Las siglas MEAN significan:

* **MongoDB:** Sistema de base de datos no relacionales basada en JSON. Utiliza una version propia llamada BSON.
* **Express:** Framework encargado de conectar MongoDB con el servidor WEB. Es el responsable de gestionar la API Rest. Capaz de manejar las solicitude y peticiones Http.
* **Angular2:** Framework basado en MVC utilizado para la parte Cliente de la aplicación WEB. Se utiliza para el desarrollo de páginas SPA (Single Page Application). Se desarrolla con typescript, una versión libre de javascript creada por Microsoft.
* **Node.js:** Framework utilizado como servidor WEB. Se utiliza javascript para el desarrollo de las reglas del servidor.

En sus inicios MEAN comenzó con la primera versión de Angular, AngularJS, pero a raíz de la nueva versión de Angular publicada por Google, muchos programadores se decantaron por sustituirla por la nueva versión de Angular2. Son frameworks realtivamente nuevos, pero con una gran comunidad detrás dando soporte.

Angular2 incorpora la inyección de dependencias con NPM, impulsando su rendimiento de forma exponencial.

Angular2 es el único framework del Mean Stack que no está basado en JavaScript. Se desarrolla en TypeScript, que al fin y al cabo, es un lenguaje que enmascara el JavaScript, siendo este su salida final. Su principial diferencia es el tipado de datos y clases. Fue creado por Microsoft, para obtener un código de JavaScript más limpio y ordenado. Cumple con la especificación ECMA6.

Al ser código JavaScript, existen infinidad de IDEs para su desarrollo, desde Visual Studio Code, un IDE gratuito de Microsoft, a los conocidos editores de código fuente SublimeText y Notepad++.

Tras tiempo sopesando las alternativas, descrubrimos el nuevo producto del equipo de desarrollo de GitHub, el editor de código llamado Atom.io

Atom.io a través de su consola de comando, es bastante configurable. Pudiendo añadirle funcionales muy útiles, como la conexión directa con el repositorio GitHub, predictor de variables y funciones en tiempo real, y hasta Angular-Cli, una herramienta que facilita la generación de código para Angular2. Finalmente nos decantamos por Atom, por su simplicidad y potencia.

Para el repositorio de información utilizamos GitHub, en su versión gratuita. Como herramientas de apoyo al desarrollo principal se encuentran, Mongoose, que es el conector para Node.js de MongoDB, el inyector de dependencias NPM, la librería de iconos FontAwesome y como plantilla de HTML5 y CSS3 AdminLTE 2, que es un dashboard open source.

Para el desarrollo de la memoria, se han utilizado herramientas como DIA para los diagramas, Balsamiq para el prototipado y Microsoft Word para la redacción de la misma.

## **Código Fuente**

A continuación se describirá la estructura de los ficheros de desarrollo, así como la API Rest creada para el envío y recepción de peticiones con el servidor.



### Estructura de ficheros

La estructura de los ficheros es la recomendada por el estándar de desarrollo de Mean Stack. Iremos desde la estructura general, hasta profundizar en el resto. Centrandonos en las carpetas propias del desarrollo.

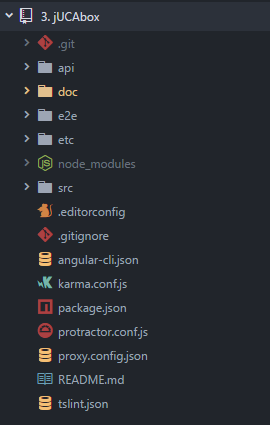


Figura 6.1 – Estructura de ficheros de jUCAbox

* **/.git**
  + Archivos de configuración del repositorio Github
* **/api**
  + Carpeta de desarrollo de la Api Rest utilizada en el proyecto, es la parte del desarrollo basada en Node.js , Express y MongoDB. Su arquitectura es la siguiente:

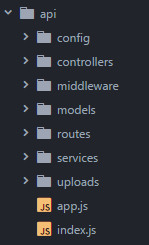


Figura 6.2 – Estructura de ficheros de la API de jUCAbox

* + **/api/config**
    - Archivos generales de configuración
  + **/api/controllers**
    - Archivos donde se indica la lógico interna de las peticiones al servidor. Como ejemplo se mostrará una petición GET para solicitar la información de un lugar.

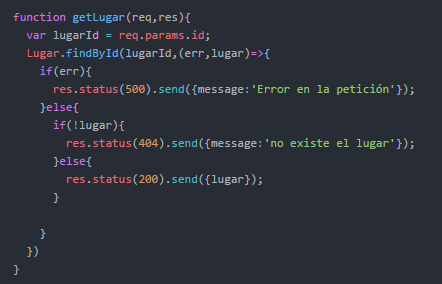


Figura 6.3 – Petición GET Node.js

* + **/api/middleware**
    - Son los archivos utilizados como pasarela de validación. La herramienta está configurada, para que cualquier operación que comprometa los datos, sea necesario indicarle un token de autorización proporcionado por la misma. Para estos tokens se utiliza un método llamado JWT, que codifica la información del usuario, para comprobar que la petición es correcta.
  + **/api/models**
    - Representa los archivos donde se indican que campos va a contener cada entidad.
  + **/api/routes**
    - Contiene los ficheros de configuración de rutas. Siguiente con el ejemplo anterior de obtener un lugar. Tendrías un fichero de rutas de lugares con la siguiente configuración. Donde se puede observa la instrucción api.get de getLugar

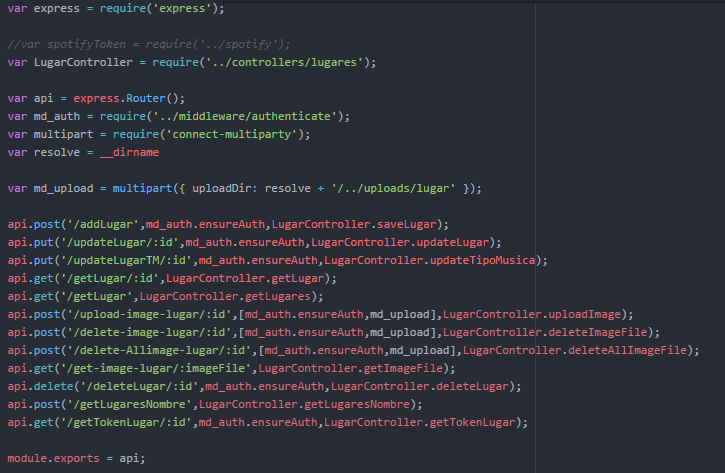


Figura 6.4 – Rutas lugares Node.js

* + **/api/services**
    - Contiene los ficheros para la generación del token JWT
  + **/api/uploads**
    - Carpeta donde se albergarán los ficheros que el usuario suba a la aplicación
  + **Apps.js e Index.js**
    - Son los ficheros principales de configuración de la Api.
* **/doc**
  + Todos los documentos de documentación de jUCAbox
* **/e2e**
  + Ficheros internos de configuración del framework
* **/etc**
  + Carpeta generada automáticamente por el framework, actualmente vacía.
* **/node\_modules**
  + Carpeta donde se encuentran las dependencias inyectadas a través de NPM
* **/src**
  + Archivos con el código del Front de la aplicación, sería la parte de Angular2.

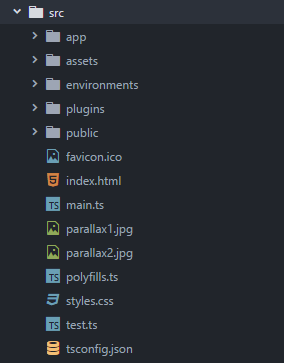


Figura 6.5 – Estructura de ficheros del Front de jUCAbox

* + **/src/app**
    - Relación de carpetas donde se incluye el código fuente propio de la herramienta, como son los controladores, modelos y servicios.
  + **/src/assets**
    - Carpeta que contiene todos los archivos relacionados con la parte visual de la herramienta, archivos css,js y fuentes entre otros.
  + **/src/environments**
    - Archivos sobre la configuración del entorno. Si es en pruebas o si está en producción la herramienta.
  + **/src/plugins**
    - Carpeta donde se incluirían plugins de terceros si se hicieran uso. En nuestro caso la carpeta está vacía.
  + **/src/public**
    - Ficheros públicos de la aplicación. Es la carpeta que utiliza fontawesome por defecto para dejar sus ficheros.
  + **/src/\*.\***
    - El resto de ficheros son propios de Angular2, siendo el index.html el archivo central de la misma
* **/\*.\***
  + El resto de los ficheros son propios del Mean Stack, donde se configuran las dependencias de manera automática a través de Angular-cli

## **Scripts de Base de datos**

Como ya hemos mencionado antes, la herramienta utiliza una base de datos NoSQL como es MongoDB. La configuración de base de datos se hace a través de los ficheros contenidos en la carpeta models de la API.

En ellos se describen el tipo de datos que va a contener la base de datos. En este caso, que información va a contener los documentos de las colecciones.

No existe un Script de creación de colecciones, puesto que cuando se inserta un documento en una colección que no existe, MongoDB interpreta ese documento y crea la colección automáticamente.

Como ejemplo, ya citado anteriormente, mostraremos uno de los ficheros modelo utilizado en la API como es el de usuario.

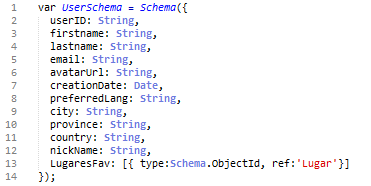


Figura 6.6 – Modelo de Usuario

# Capítulo 7

# Pruebas del Sistema

En este capítulo se presenta el plan de pruebas del sistema de información, incluyendo los dife- rentes tipos de pruebas que se han llevado a cabo.

## **Estrategia**

Para el testeo de la herramienta se ha seguido con un método de pruebas de manera manual. Pues la cantidad de funcionalidades de jUCAbox no requería el desarrollo de un módulo específico para pruebas.

Al ser una herramienta orientada al mercado, no a algún cliente en particular, se ha realizado pruebas de aceptación con distintos perfiles potenciales de la herramienta, desde un usuario básico, a un administrador de un local de música.

Para probar la dependencia entre componentes, se realizaron pruebas de regresión. Cuando se hacía alguna modificación en algún componente que interactuaba con otros, se comprobaban que cumplían con las necesidades propuestas. Cuando la herramienta ya estuvo lo suficientemene madura, las pruebas se realizaban en la capa visual de la herramienta. Era más sencillo realizar las pruebas conforme se iban desarrollando funcionalidades, que buscar los fallos según la dependencia de los componentes.

## **Entorno de Pruebas**

El entorno utilizado para la realización de las pruebas, es el mismo que se ha utilizado para el desarrollo de la aplicación. Para la parte hardware un portatil Lenovo ideapad 100 con un procesador intel i5 con 8gb Ram y disco duro de 256 GB. El entorno software en el cual se hicieron las pruebas fue en el servidor local proporcionado por Node.js.

## **Roles**

Para el testeo de la herramienta, fundamentalmente han interactuado dos roles. El propio desarrollador y usuario potencial de la herramienta. Al tener dos enfoques la herramienta, uno para la persona solicitante de canciones, y otropara el administrador de locales. Se ha contado con dos personas para este rol, cada una testeando las funcionalidades con las cuales se identificaba.

* **Desarrollador**
  + Las pruebas realizadas por el desarrollador, están condicionadas a la propia naturaleza de la herramienta. Es inherente, probar algo para lo que se ha diseñado, con esto no quiere decir que las pruebas sean inútiles, sino todo lo contrario, el desarrollador conoce la funcionalidad de la herramienta pudiendo cubrir un amplio espectro de las posibles interactuaciones del usuario con la misma. Aún así es necesario realizar una batería de pruebas de un rol ajeno al desarrollo de la aplicación
* **Usuario**
  + Las pruebas por parte de los usuarios se enfocaban en las necesidades de cada uno. Al ser bastante sencillo el poder realizar una petición, el usuario no estaba formado en el uso de la herramienta y las pruebas concluyeron con éxito.
  + Para el administrador de locales, puesto que tiene una visión más amplia del negocio, a la vez que iba realizando el testeo de la herramienta, iba aportando nuevos requisitos que podrían incorporse a la herramienta.

## **Niveles de Pruebas**

En esta sección se documentan los diferentes tipos de pruebas que se han llevado a cabo.

### Pruebas Unitarias

Para la realización de pruebas unitarias, como se ha comentado antes, se han realizado en la capa visual de la herramienta. Para cada apartado se comprobaba que la aparencia y respuesta de las acciones era la esperada.

Para la API Rest, se comprobaba si las peticiones hacía el servidor devolvían una respuesta satisfactoria. Para las pruebas de la API, se utilizó una herramienta gratuita llamada Postman\*, que gestiona las llamadas al servidor a través del protocolo http.

### Pruebas de Integración

Una vez comprobado que la aplicación Web y la API respondían correctamente por separado, se procedió a la integración de las mismas. Haciendo llamadas al servidor desde la herramienta Web. Para el cual se utilizo la herramienta RoboMongo, una aplicación de escritorio para gestionar de manera visual la base de datos MongoDB. Pudiendo así comprobar que la respuesta de las peticiones realizadas desde la parte Web eran correctas y coherentes.

### Pruebas de Sistema

En esta sección se realizan las pruebas de sistema de modo que se asegure que el sistema cumple con todos los requisitos establecidos.

### Pruebas funcionales

Para realizar las pruebas funcionales, se han utilizado los requisitos descritos en la sección 3.4.1, lo cual fue necesario probar tanto los flujos normales como los flujos alternativos.

Se comenzó realizando el acceso de los usuarios al sistema, ya que fue la parte inicial del proyecto. Se comprobó que el acceso era correcto, y al conectarse a través de inicio de sesión de terceros, además de que los datos que se obtenían eran correctos. Una vez realizada la comprobación, se proboó el correcto funcionamiento de la protección de rutas, ya que cada perfil puede acceder a diferentes funcionalidades.

Una vez realizada estas pruebas, se procede a realizar búsquedas de otros usuarios, así como la interactuación con los mismos. Se comprobó que los datos registrados, correspondían con los datos proporcionados por terceros.

Para el apartado de gestión de lugares, en primer lugar se comprobó la creación de los mismos. Si los datos introducidos se registraban correctamente. Una vez realizada la creación de lugares, se procedía a la búsqueda del mismo, desde el módulo especifico para ello.

Se realizó una búsqueda del lugar creado, y el usuario previamente logado, comprobaba si la inserción del lugar a la lista de favoritos era correcta. Una vez concluida con la prueba inicial de creación de lugares, se probó el resto de funciones del apartado. Como son la inserción de mensajes en el tablón y la creación de playlists para el mismo.

A raíz de este punto, se comprobó que la conexión con la API de spotify era satisfactoria, detetectando que los datos obtenidos, se correspondían con la información solicitada. Una vez comprobada que la conexión era correcta, se procedió a enviar una canción para el proceso de validación de la misma, y validar el flujo de información desde la herramienta a Spotify.

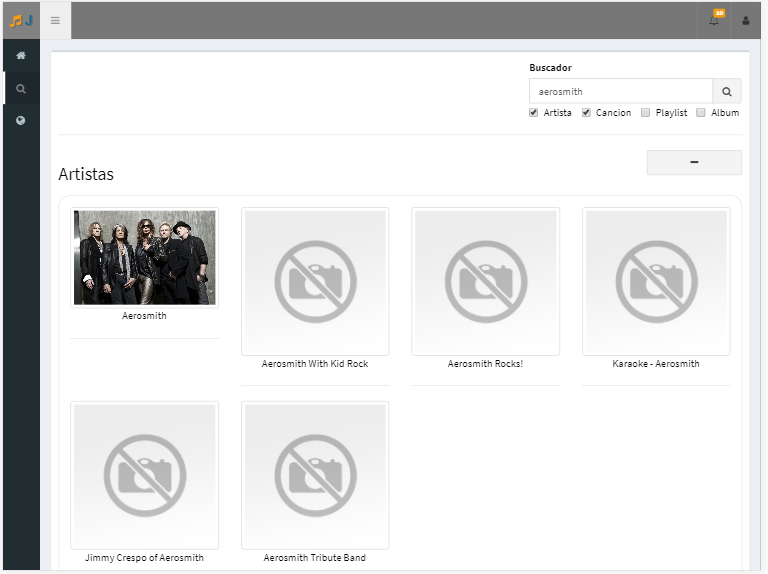
Para ello se procedió a comprobar la interacción de la herramienta con la API de spotify. Enviando y validando las canciones introducidas previamente por el usuario.

Por último se validaron que las estadísiticas proporcionadas por el reporte de QlikView eran correctas y que los datos mostrados eran reales.

### Pruebas no funcionales

A continuación se describen las pruebas funcionales realizadas en la aplicación Web.

* **Portabilidad**
  + A través de diferentes navegadores se comprueba la correcta visualización de los elementos, así como desde diferentes dispositivos móviles. Ya que el desarrollo de la herramienta se ha basado en un desarrollo responsive, para que pueda ser visualizada desde cualquier dispositivo con conexión a internet. A través de google Chrome, se realizaron varias visualizaciones en distintos dispositivos.



**Figura 7.1 –** jUCAbox Ipad

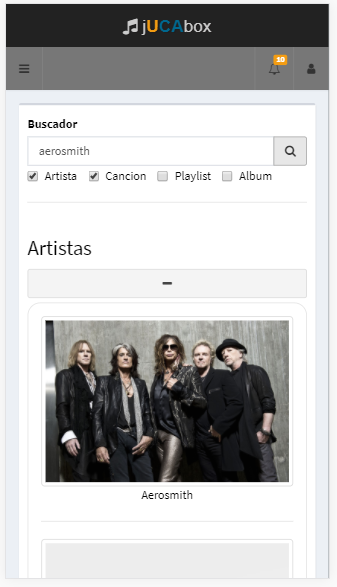


Figura 7.2 – jUCAbox Iphone 6

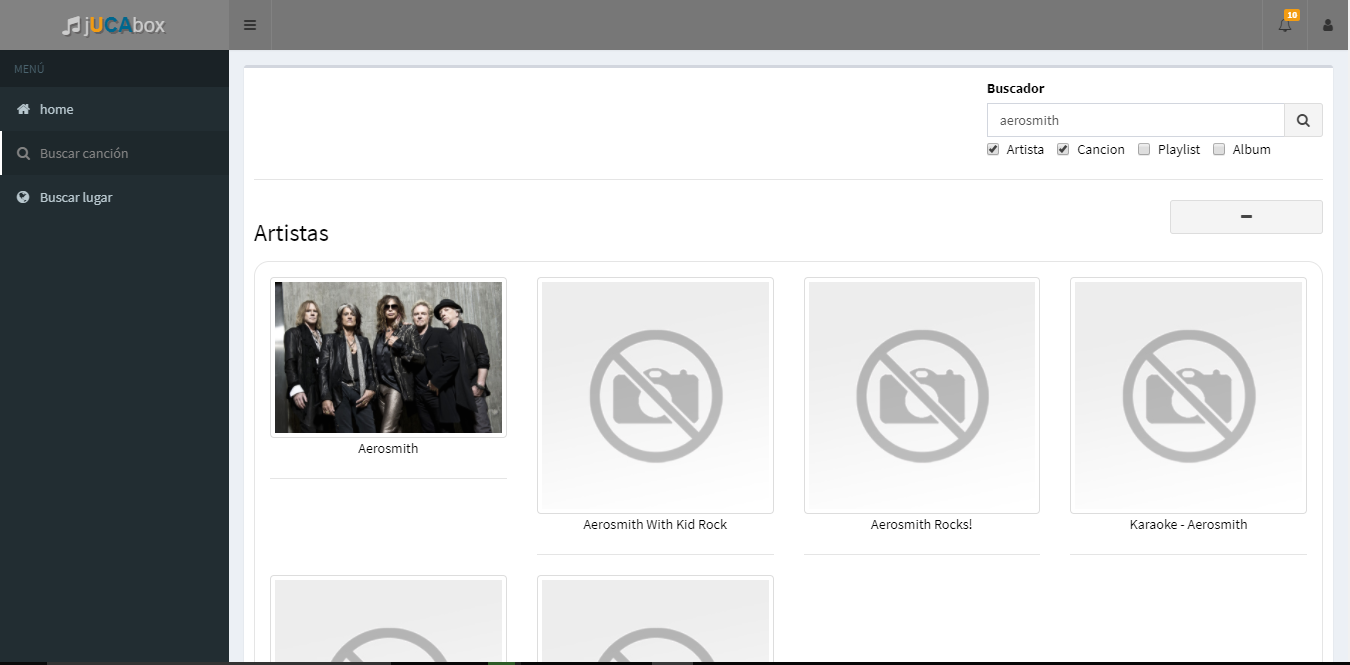


Figura 7.3 – Navegador Web

* **Mantenibilidad**
  + Al utilizar herramientas novedosas, siempre se corre el riesgo de que dejen de ser mantenidas. Pero en este caso el fork MEAN Stack, tienen detrás a una gran comunidad de usuarios, incluyendo el propio desarrollador de Angular, Google, ampliando cada vez más la funcionalidad del mismo.
* **Seguridad**
  + Las pruebas realizadas a la aplicación verifican que el sistema cumple con todos los requisitos de seguridad establecidos en la etapa de análisis.
* **Fiabilidad**
  + Después de realizar toda la batería de pruebas y usar sistemas de confianza, nos aseguran el correcto funcionamiento de la herramienta. Esto añadido a la confianza de elegir un servidor adecuado, como Amazon Web Service, garantiza que el sistema se encontrará en correcto funcionamiento.
* **Escalibilidad**
  + Para garantizar la escalabilidad se ha utilizado un sistema de bases de datos MongoDB, la cual permite particionar de manera transparente para el usuario, aumentando así el rendimiento de acceso y procesado de los datos.
* **Rendimiento**
  + Para garantizar el rendimiento de la aplicación se ha seguido con el estándar marcado por las herramientas utilizadas. Se ha utilizada un medidor de velocidad web llamado Performance-Analyser, la cuál arroja los siguientes resultados:

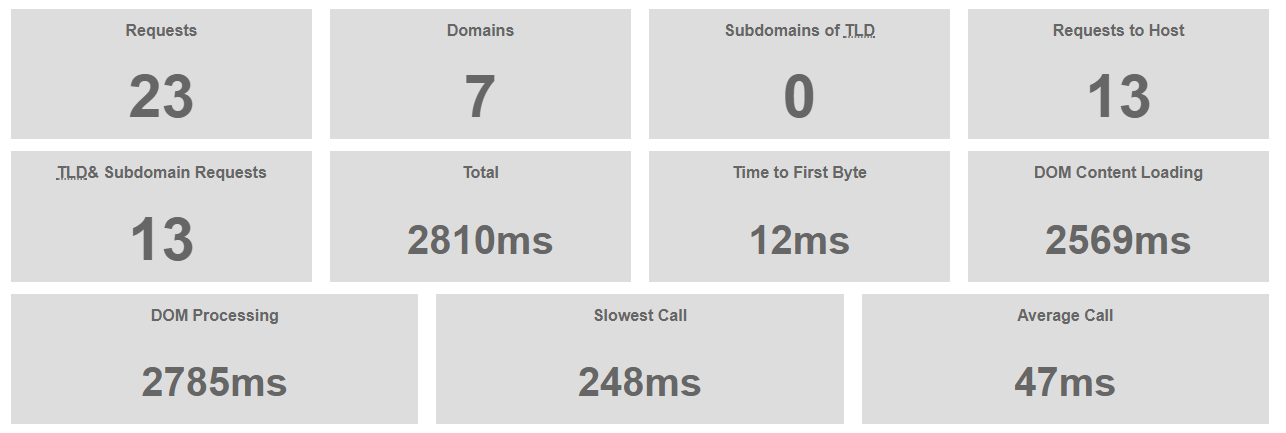


Figura 7.4 – Peticiones y Tiempos Performance-Analyser



Figura 7.5 – Tiempo de navegación Performance-Analyser

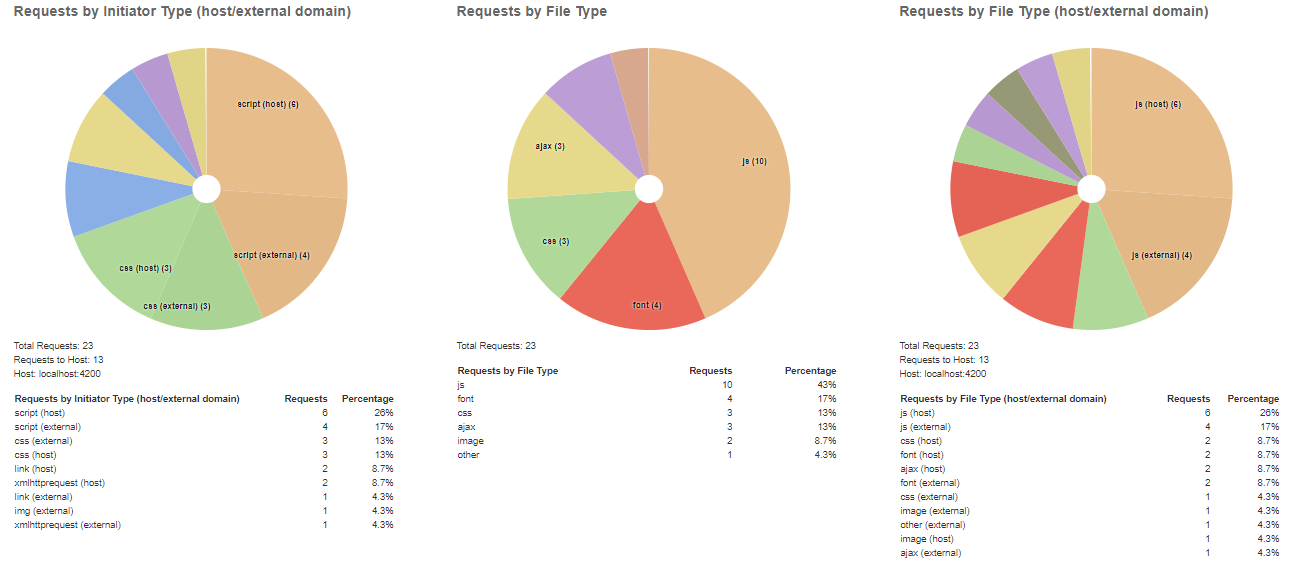


Figura 7.6 – Clasificación de peticiones Performance-Analyser

### Pruebas de Aceptación

Una vez realizadas todas las pruebas oportunas, se hicieron las pruebas finales de aceptación de la herramienta. Para estas pruebas son necesarias las personas integrantes en los roles mencionados anteriormente.

Básicamente, estas pruebas dejan el sistema a disposición del usuario, para que lo utilice el producto desarrollado. Perminitiendo obtener su feedback final de la herramienta.

Las pruebas se realizaron sin supervisión del desarrollador, esperando ver reacciones y comportamientos de usuarios finales potenciales.

# Parte III

# Epílogo

# Capítulo 8

# Manual de implantación y explotación

Las instrucciones de instalación y explotación del sistema se detallan a continuación.

## **Introducción**

La herramienta se ha desarrollado como canal de comunicación de canciones entre clientes de un local y el encargado de la música. Consiguiendo trazabilidad de la información y mayor control del flujo de datos. Pudiendo realizarse estudios y análisis de las peticiones obtenidas. A su vez tiene como objetivo potenciar las capacidades de un lugar a nivel musical, es decir, poderse dar a conocer, así como enviar mensajes a sus clientes potenciales.

La otra visión para la que está ideada la aplicación, es para eventos sociales, donde el organizador puede crear listas de reproducción a petición de sus invitados.

## **Requisitos previos**

Para poder instalar la herramienta en un servidor es necesario tener instalado:

* Node.js siendo su versión igual o superior a 7.4.0
* El inyector de dependencias NPM, siendo su versión igual o superior a 4.2.0
* Angular-cli, con versión igual o superior a 1.0.0-beta 28.3
* Express, con versión igual o superior a 4.15.3
* Y una versión de MongoDB de 3.4.2 o superios
* Además de las dependencias ya incluidas en el package.json de la herramienta

## **Inventario de componentes**

No es necesario ningún componente específico hardware, para el despliegue de la aplicación se ha utilizada una instancia de AWS con 1 CPU Virtual y 1 GB de memoria. Para el lado del cliente, no existe ninguna limitación de hardware.

Los componentes software se le proporcionará al usuario un fichero con la relación de carpetas ya mencionada anteriormente, donde estarán todos los ficheros necesarios para su implantación. Además se incluirá el fichero package.json, con el cual se realiza la instalación de dependencias.

Por la parte de la base de datos, no es necesario ningún archivo con el Scripts, puesto que al iniciarse el proyecto, automáticamente se creará la base de datos y las tablas necesarias a petición de la aplicación. En el fichero, index.js, de la carpeta api, se indicará puerto y ruta de la base de datos.

## **Procedimientos de instalación**

Para el despliegue de la aplicación hemos utilizado Amazon Web Service (AWS) y su servicio Amazon Elastic Compute Cloud ec2, que se basa en una colección de servidores remotos.

Suponiendo que ya tenemos creada una cuenta en AWS, nos dirigimos al apartado EC2



Figura 8.1 – Amazon Web Service Ec2

Antes de crear la instancia, es necesario crearnos un EC2 Key Pair, referidas al las claves publicas y privadas usadas para encriptar y desencriptar la información para logarnos en el servidor.

Para ello, en el menú de la izquierda, en el apartado NETWORK & SECURITY, elegimos Key Pairs.

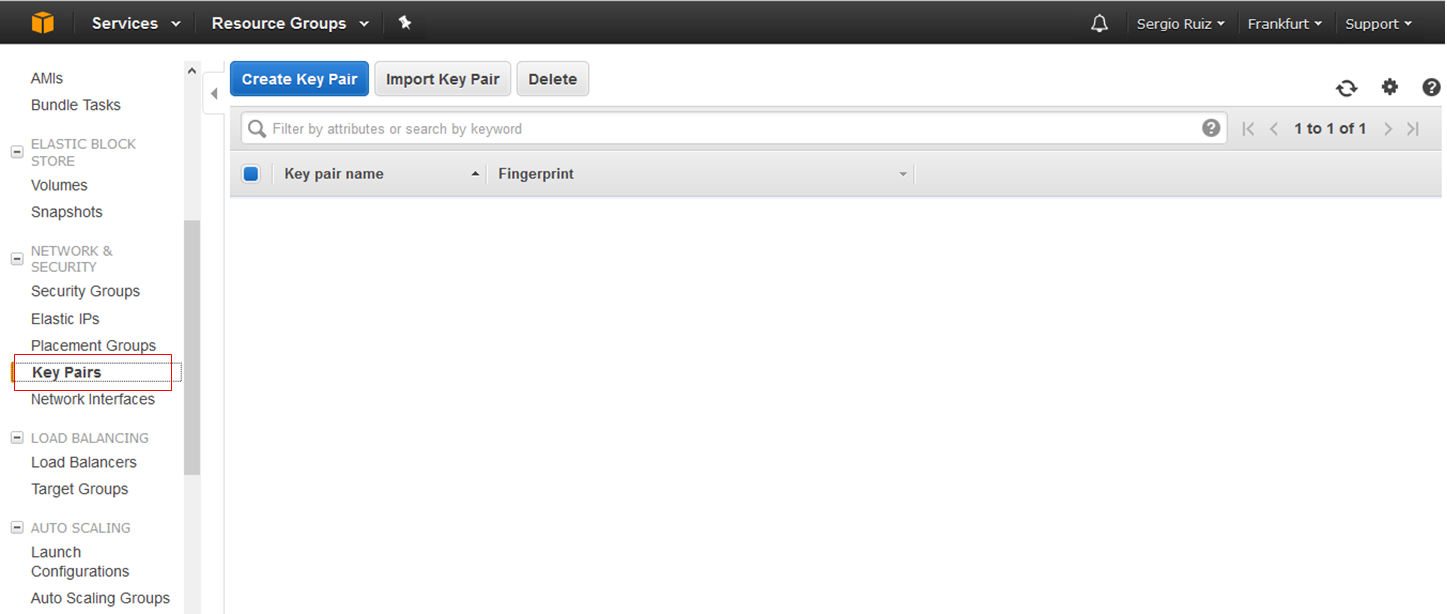


Figura 8.2 – Key Pairs AWS

Y hacemos click en Create Key Pair.

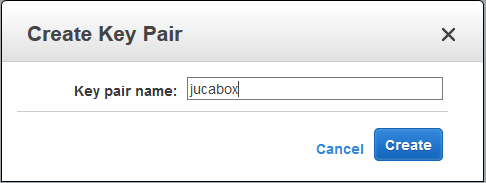


Figura 8.3 – Create Key Pairs AWS

Nos aparecerá la clave creada y se nos descargará un fichero llamada jucabox.pem , que será el fichero necesario para conectarnos a través de ssh al servidor.

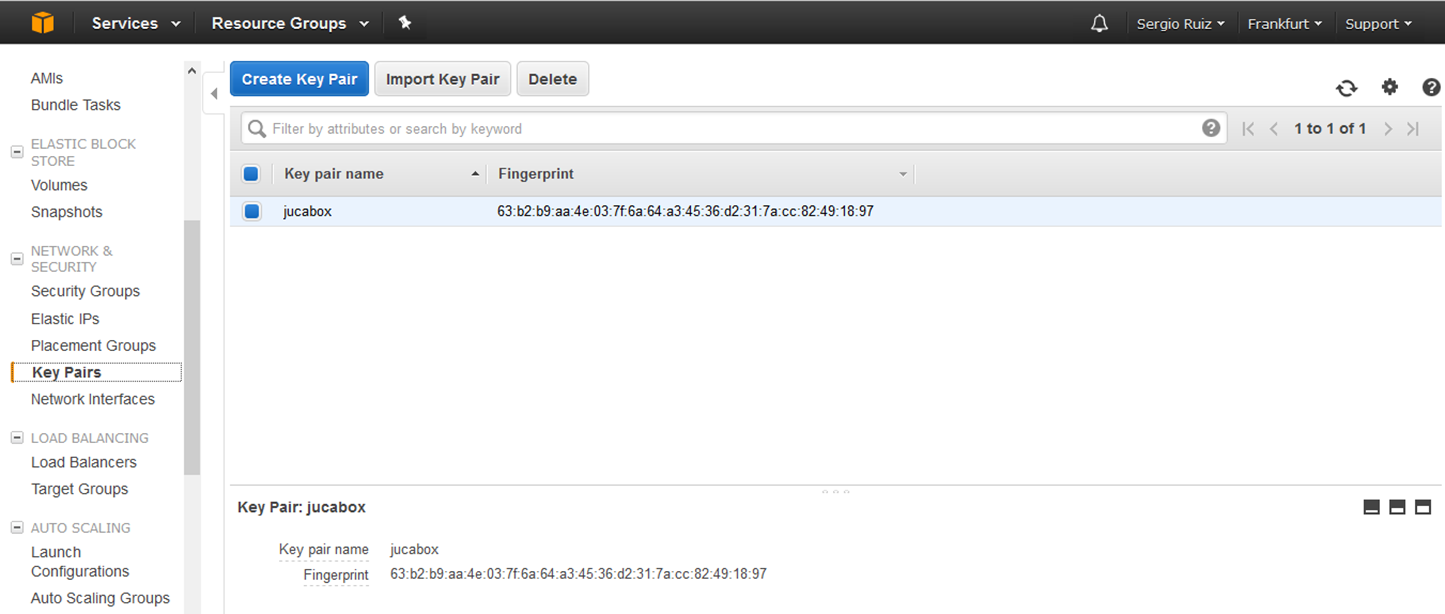


Figura 8.4 – Key Pairs Creada AWS

Una vez creado el par de claves, tendremos que dirigirnos al AWS Marketplace, el cual incluye imágenes preconfiguradas con herramientas. En nuestro caso vamos a buscar Bitnami MEAN image from BitRock, la cual incluye todas las herramientas necesarias para ejecutar aplicaciones basadas en MEAN Stack.

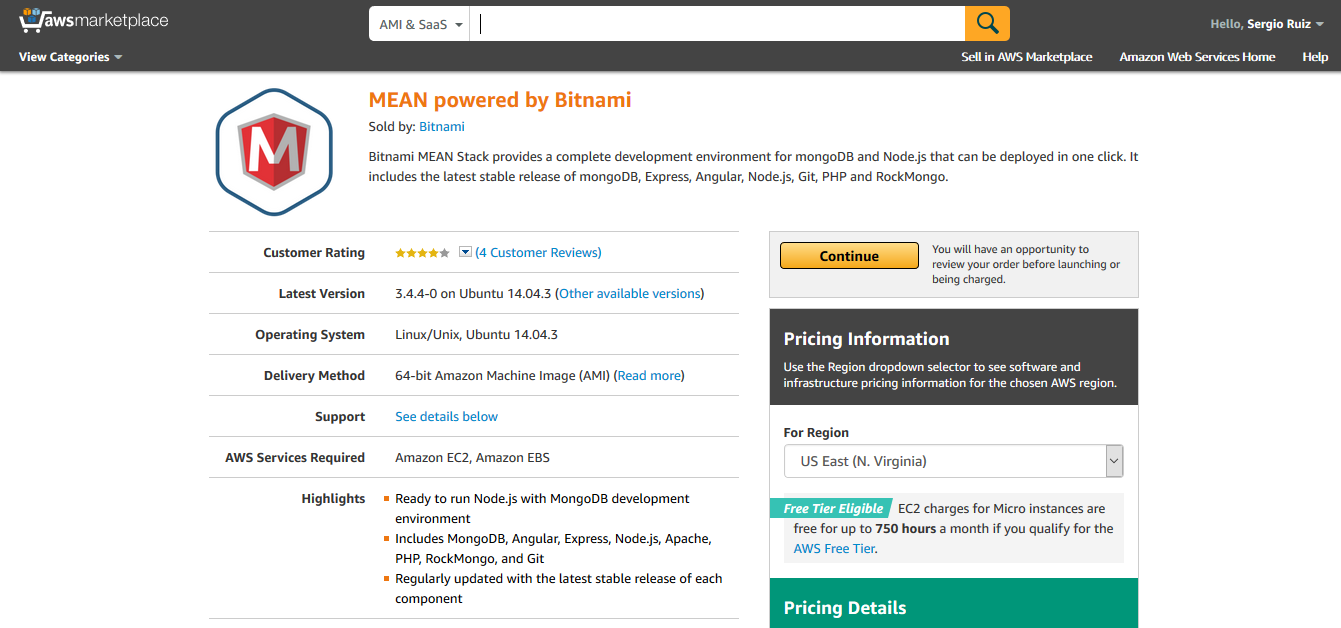


Figura 8.5 – Bitnami AWS Marketplace

Pulsamos Continue, y elegimos la instancia deseada, en este caso t2.micro y añadimos la Key Pair creada anteriormente.

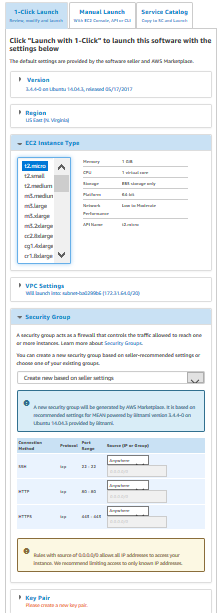


Figura 8.6 – Configuración Bitnami AWS Marketplace

Una vez instalado todo, volvemos a el Dashboard de configuración de AWS, y veremos que tenemos activa una instancia.

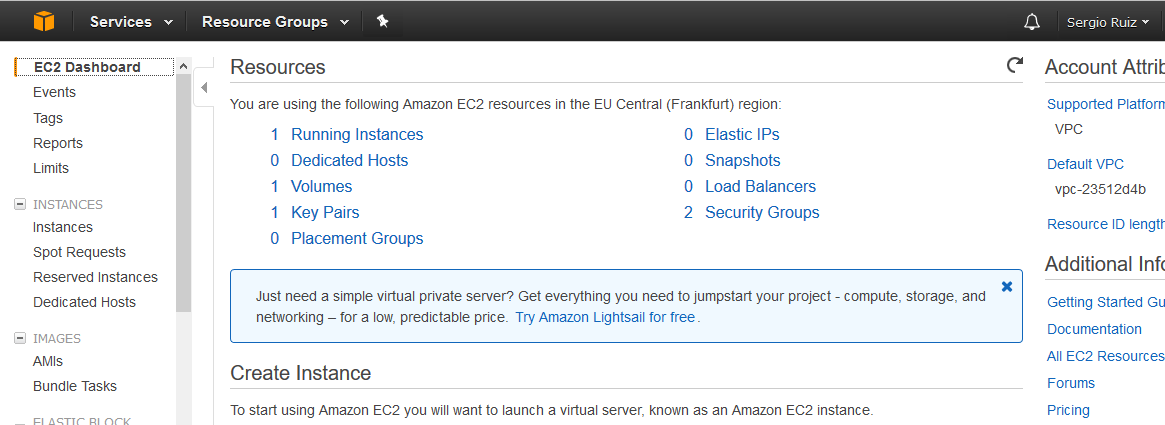


Figura 8.7 – AWS Instancia

Accedemos a Running Instances, y deberemos esperar hasta que el Status Check, muestre 2/2 checks.



Figura 8.8 – AWS Instancia Running

Una vez levantada la instancia, con el botón derecho encima de la instancia, seleccionamos Get System Log.

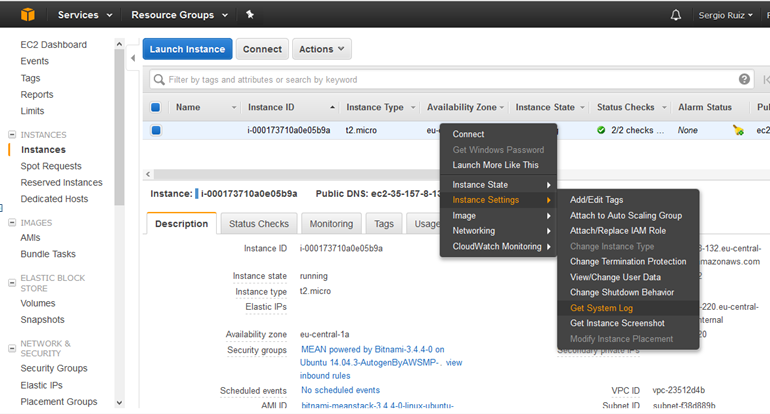


Figura 8.9 – AWS Instancia Get System Log

Y buscamos **Setting Bitnami applicaction password to** y guardamos la contraseña.

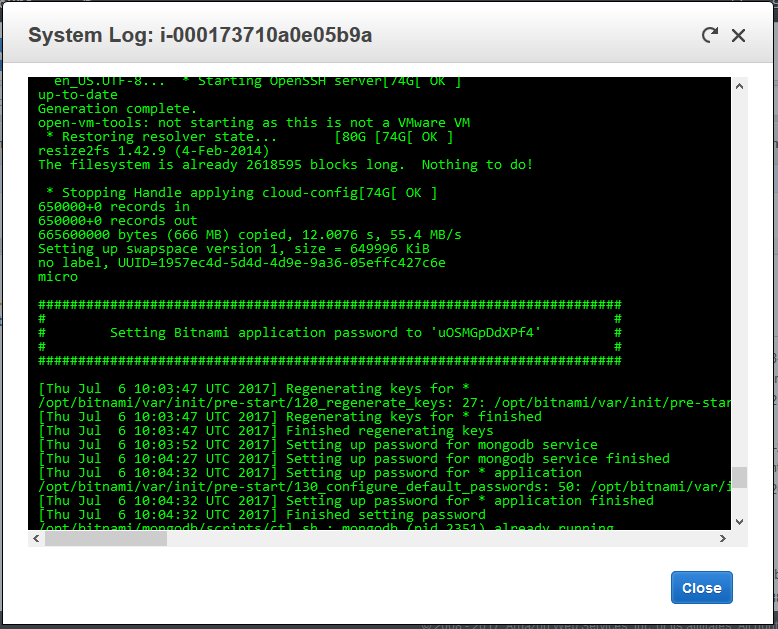


Figura 8.10 – AWS Log Bitnami

Ahora necesitamos subir y configurar nuestra aplicación en el servidor AWS. Por lo que nos descargamos Putty, una herramienta gratuita para conectarnos a través de SSH. A su vez nos descargamos también, Putty Key Generator para transformar el fichero PEM previamente descargado a un fichero compatible con Putty.

Para ello abrimos el fichero jUCAbox.pem con Putty Key Generator y seleccionamos Save private key.

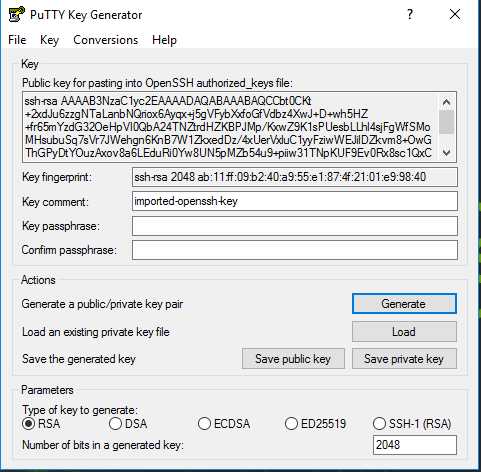


Figura 8.11 – Putty Key Generator

Se nos generará un fichero compatible con Putty. Con lo que abrimos Putty e introducimos los datos necesarios para la configuración. Obtenemos la dns de AWS.

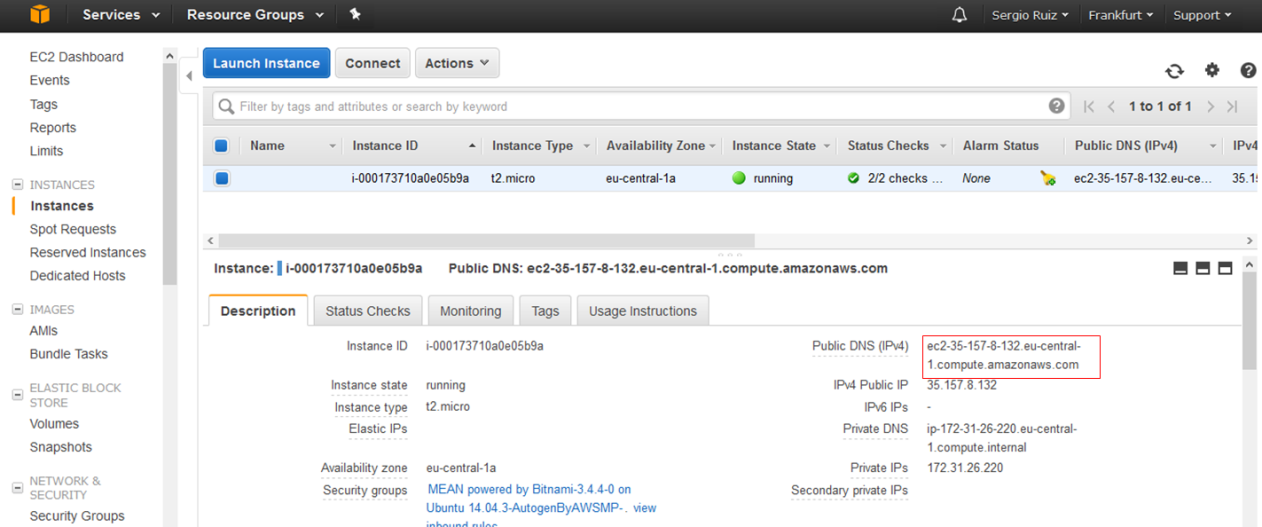


Figura 8.12 – AWS Dns

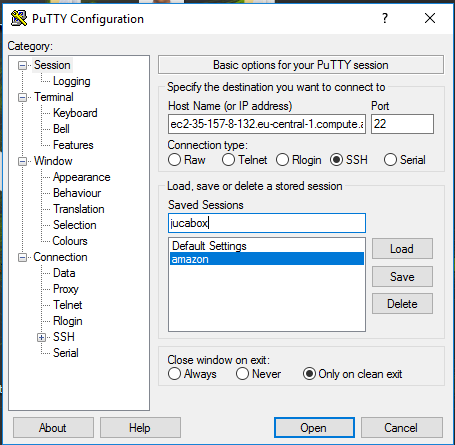


Figura 8.13 – Putty Session

Y añadimos en el apartado Connection 🡪 SSH 🡪Auth , el fichero generado por Putty Gen.

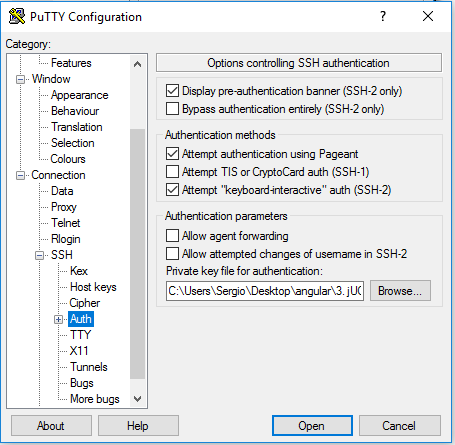


Figura 8.14 – Putty Session SSH Auth

Y nos logamos como bitnami.

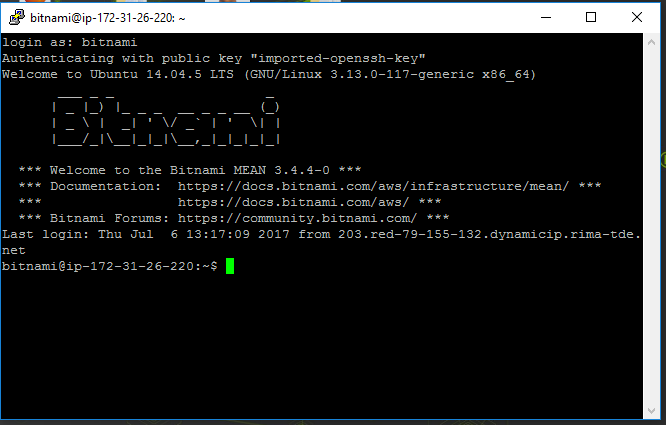


Figura 8.15 – Putty Inicio Sesion

Una vez dentro tendremos varias carpetas, entre ellas apps, así que accedemos a ella y clonamos nuestro repositorio GIT. En este caso los comandos serían:

cd

cd apps

sudo git clone https://github.com/kodorniz/jUCAbox\_PFC.git

cd jUCAbox\_PFC

Una vez terminado de clonar el repositorio

bitnami@ip-172-31-26-220:~/apps$ sudo git clone https://github.com/kodorniz/jUCAbox\_PFC.git

Cloning into 'jUCAbox\_PFC'...

remote: Counting objects: 1458, done.

remote: Compressing objects: 100% (374/374), done.

remote: Total 1458 (delta 250), reused 365 (delta 131), pack-reused 948

Receiving objects: 100% (1458/1458), 11.98 MiB | 5.84 MiB/s, done.

Resolving deltas: 100% (791/791), done.

Checking connectivity... done.

bitnami@ip-172-31-26-220:~/apps$

Una vez terminado de clonar el repositorio, realizamos la opción **npm install** para instalar las dependencias del proyecto.

Una vez finalizado, deberemos de configurar los puertos que hayamos puesto para la aplicación, en este caso para Angular el puerto es el 4200, para el API es el 3000 y para Mongodb es el 27017

Por lo que accedemos al Dashboard de AWS, en el apartado de NETWORK & SECURITY, accedemos a Secutiry Groups.

En el grupo creado por MEAN powered by bitnami, editamos los puertos habilitados.



Figura 8.16 – Editar puertos AWS

Una vez abierto los puertos, probamos que nuestra aplicación está corriendo correctamente. Para ello ejecutamos el comando npm start.

Y obtendríamos el siguiente resultado si todo ha ido ok:

\*\* NG Live Development Server is running \*\*

10% building modules 4/4 modules 0 active[HPM] Proxy created: /api -> port:3000

[HPM] Subscribed to http-proxy events: [ 'error', 'close' ]

11% building modules 14/19 modules 5 active ...\bootstrap\dist\css\bootstrap.min.csswebpack: wait until bundle finished: /

Hash: a89f75a53bd541960297

Time: 31565ms

chunk {0} polyfills.bundle.js, polyfills.bundle.map (polyfills) 232 kB {5} [initial] [rendered]

chunk {1} main.bundle.js, main.bundle.map (main) 405 kB {4} [initial] [rendered]

chunk {2} styles.bundle.js, styles.bundle.map (styles) 401 kB {5} [initial] [rendered]

chunk {3} scripts.bundle.js, scripts.bundle.map (scripts) 994 kB {5} [initial] [rendered]

chunk {4} vendor.bundle.js, vendor.bundle.map (vendor) 5.44 MB [initial] [rendered]

chunk {5} inline.bundle.js, inline.bundle.map (inline) 0 bytes [entry] [rendered]

webpack: Compiled successfully.

Ya tendríamos nuestra aplicación funcionando correctamente.

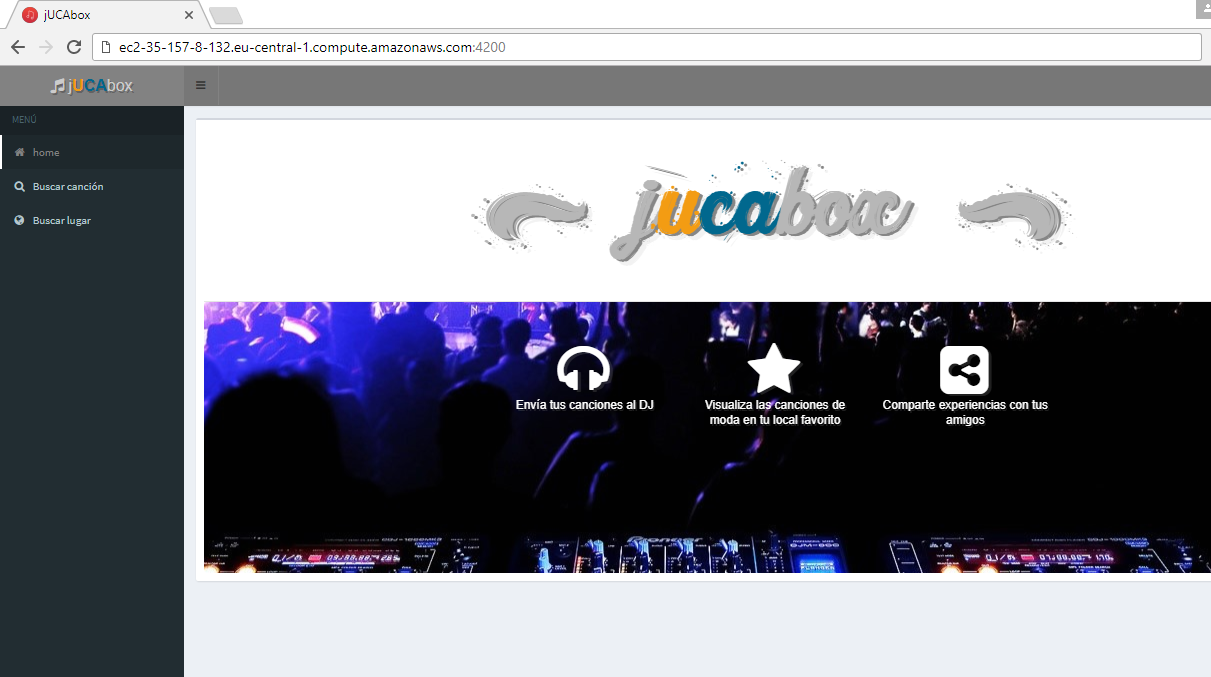


Figura 8.17 – jUCAbox en AWS

## **Pruebas de implantación**

Para comprobar el correcto funcionamiento simplemente es necesario acceder a la dirección creada, y verificar desde el Dashboard de AWS que todo funciona correctamente. Si existiera algún problema, AWS tiene reportes y controles de seguridad donde nos informan del estado de la aplicación en todo momento.

## **Procedimientos de operación y nivel de servicio**

Para garantizar y asegurar los datos del servidor es necesario realizar copias periódicas de seguridad, para ello AWS nos ofrece un amplio abanico para la configuración de estas copias. Aún así, existen herramientas para poder hacer copias de seguridad de la base de datos de manera manual, como es la herramienta mongodump, que podemos instalar en el servidor y ejecutar el siguiente comando:

mongodump -h localhost --directoryperdb -o alldump

Para realizar una copia de base de datos consistente, sería necesario para la instancia, lo que tendríamos que buscar un rango de tiempo adecuado para no intervenir en altos picos de uso de la herramienta.

# Capítulo 9

# Manual de usuario

Las instrucciones de uso del software se detallan a continuación. Este manual se dirige al usuario final del software objetivo de este proyecto.

## **Introducción**

Resumen de los principales objetivos, ámbito, características y alcance del software desarrollado.

## **Instalación**

Se detallarán los pasos necesarios para la obtención e instalación del software, así como los requisitos previos de hardware y software.

## **Uso del sistema**

Describir todos los aspectos necesarios para una utilización efectiva y eficiente de las caracterís- ticas del sistema por parte de los usuarios.

# Capítulo 10

# Conclusiones

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software desarrollado.

## **Objetivos alcanzados**

Este apartado debe resumir los objetivos generales y específicos alcanzados, relacionándolos con todo lo descrito en el capítulo de introducción.

## **Lecciones aprendidas**

A continuación, se detallan las buenas prácticas adquiridas, tanto tecnológicas como procedi- mentales, así como cualquier otro aspecto de interés.

Resumir cuantitativamente el tiempo y esfuerzo dedicados al proyecto a lo largo de su desarrollo que escribir un sencillo ’he trabajado mucho en este proyecto’.

## **Trabajo futuro**

En esta sección, se presentan las diversas áreas u oportunidades de mejora detectadas durante el desarrollo del proyecto y que podrán ser abarcadas en futuras versiones del software.

Los elementos aquí descritos deben estar en relación con lo relatado en el apartado de objetivos y alcance del proyecto descritos en la introducción.

# Bibliografía

[CCOO, 2010]CCOO (2010). Tablas salariales 2010 IV Convenio Colectivo. [http://www.uca.](http://www.uca.es/sindicato/ccoo/documentos/tabla-salarial-pas-laboral-2010.pdf) [es/sindicato/ccoo/documentos/tabla-salarial-pas-laboral-2010.pdf](http://www.uca.es/sindicato/ccoo/documentos/tabla-salarial-pas-laboral-2010.pdf).

[Hevner et al., 2004]Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105.

Información sobre Licencia

Incluir aquí la información relativa a la licencia seleccionada para la documentación y software del presente proyecto.