FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

DESARROLLO DE APLICACIÓN WEB CON INTEGRACIÓN DE SISTEMA DE CONTROL VERSIONES Y LIBRERÍAS DE EDICIÓN MUSICAL, PARA LA CREACIÓN DE PISTAS MUSICALES DE FORMA GRUPAL

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero en Sistemas de Computación e Informática

Profesor Guía

Juan José León

Autor

José Emmanuel Añasco Loor

Año

2016

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”

_____

Juan José León
Ing. Msc. Sistemas
C.I: 1707506760

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original de mi autoría que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes”

_____

José Emmanuel Añasco Loor

C.I: 1716599533

AGRADECIMIENTOS

Agradezco a cada una de las personas que hicieron posible este proyecto, incluyendo a aquellas que indirectamente fueron una fuente de inspiración para su culminación

RESUMEN

Este proyecto tiene como objetivo principal implementar la primera versión de una aplicación web destinada a la colaboración musical. La aplicación funcionará principalmente como una red social que nos permitirá compartir y copiar pistas, llamadas ‘loops’, creadas por medio de un editor de sonidos en línea.

Al ser una aplicación web, necesariamente se tendrá que crear y reproducir los loops en el navegador. Para lograrlo se usará la librería de Web Audio API, la cual es estándar en todos los navegadores web en la actualidad. Sobre Web Audio API se crearán otras librerías de audio, las cuales implementarán abstracciones para la creación de notas musicales y la manipulación del API en general.

El usuario podrá versionar ‘loops’ de otros artistas, realizando una copia de este ‘loop’ en su perfil. La aplicación usará el sistema de control de versiones GIT para almacenar la información referente a los cambios de cada ‘loop’.

Las librerías y la aplicación serán implementadas en su totalidad con el lenguaje de programación Ruby, las cuales interactuarán entre si gracias al marco de trabajo Volt. Este marco de trabajo permite la creación de aplicaciones ‘isomórficas’, capaces de ejecutar código en un solo lenguaje tanto en el back-end como en el front-end de nuestra aplicación. Por debajo, la aplicación compilará el código a JavaScript para así ejecutarlo en el navegador.

Con el objetivo de minimizar los errores en el desarrollo se crearán pruebas unitarias y de integración del proyecto. Los cambios en el código se los realizará siguiendo el proceso de integración continua. En la creación de pruebas se seguirá la práctica de desarrollo guiado por pruebas.

Heroku será el servidor usado para desplegar la aplicación. Las ventajas de Heroku son: su integración con el lenguaje de programación Ruby, la facilidad de uso y los plugins incorporados en la plataforma, entre los que se incluyen servicios para realizar pruebas de rendimiento, entre otros servicios.

ABSTRACT

The main purpose of this project is to implement the beta version of a web app for music collaboration. The app will work mainly as a social network, in which users will be able to share their audio tracks, called ‘loops’, created via the online audio editor.

Since the app is meant to be used online the loops must be created in the browser, this will be done through the Web Audio API which is the standard for manipulating sounds in any modern web browser. Other libraries will be created for wrapping the Web Audio API functionality. These libraries will enable the creation of notes and sequences of notes.

A user will be able to create a copy of a loop under his user profile. The application will save changes made to the loop with the GIT version control system.

The web application and the libraries will be developed with the Ruby programming language; these will communicate through the Volt framework. Volt can execute Ruby code both on the front-end and the back-end, enabling the creation of ‘isomorphic’ applications. Under the hood, the app will compile the code to JavaScript when it is executed in the front-end.

Unit and Integration tests will be created for minimizing the number of bugs in the development process. New features and fixes will be deployed trough continuous integration. For creating tests, The TDD approach will be followed.

The app will be deployed to the Heroku Platform. Some of the advantages of this service are the straightforward deployment process and the plugins available.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Antecedentes

La música, como cualquier otro campo, se ha beneficiado de excelentes contribuciones musicales tanto en el plano solista como en el plano grupal. Compositores como Beethoven o Mozart dejaron memorables sinfonías y sonatas, mientras que grupos como los Beatles dejaron grandes baladas. Dar vida a estas obras fue fruto del esfuerzo y desde luego de la colaboración directa e indirecta de varias personas. Sin la influencia de otros artistas un músico no tiene un punto de partida firme. El ascendiente de varias personas hacen más enriquecedora la creación y más elevado su resultado. La inspiración puede venir de diversas fuentes, y sus orígenes no son siempre las habituales. Sin la inspiración, la creación no fluye. Se dice que el 99 % del trabajo de un artista es buscarla constantemente y el 1 % de haberla encontrado.

En cualquier tipo de trabajo en grupo, al sumar esfuerzos la creatividad se multiplica. Pero no siempre podemos estar de acuerdo con toda la lluvia de ideas de un grupo heterogéneos de personas. Así, en el plano musical el fruto de todo el trabajo se logra en base al consenso. Se desechan unas propuestas y se mejoran otras con el fin de obtener el mejor producto final, en este caso una canción.

A medida que avanza la tecnología, y las formas en las que interactuamos como individuos, se multiplican las ideas. Hemos visto como de una simple plataforma web para compartir videos, se han creado nuevas formas de expresión en el campo audiovisual. Así por ejemplo se han generado aplicaciones para la edición de videos con otras personas.

En el campo musical este es un tema que se está tratando de explotar. Si bien existen aplicaciones web que buscan ser un centro de colaboración

musical, están atadas a aplicaciones de edición musical de terceros, con lo que el proceso se vuelve menos intuitivo para el usuario. De igual manera en aplicaciones con sistemas de edición propios, no se aplica obligatoriamente un sistema de versiones y su implementación podría facilitar la colaboración entre varios artistas.

Por los motivos mencionados se ha planteado este sistema, que servirá como punto de colaboración, creación y edición para artistas.

Alcance

La aplicación que se plantea es una red social implementada en una plataforma web, en la cual se incentivará la colaboración musical entre artistas. Los usuarios dentro del sistema podrán realizar una copia de las canciones de otros artistas, creando así su propia versión. Cada canción será almacenada dentro del servidor mediante el sistema de control de versiones GIT.

La parte principal de la aplicación será la edición de una canción en sí. Se podrá optar entre añadir pistas que tendrán contenido de batería, piano, etc. También se podrá cambiar la distribución de los mismos a lo largo de la canción de una manera intuitiva para el usuario.

Los sonidos para las pistas serán suministrados por la página. El usuario utilizará estas librerías y podrá editar las notas y tiempos para su pista.

Para el desarrollo se utilizará la metodología SCRUM. Es un campo todavía no muy explotado que permitirá generar nuevas ideas a medida que se desarrolle el tema, permitiendo a su vez una mejor respuesta a los cambios que requiera la aplicación.

Se hará uso también de la técnica de programación TDD o Test Driven Development (Desarrollo guiado por pruebas). También se hará uso de un

framework o marco de trabajo para implementar la aplicación de manera rápida.

Justificación

En lo que se refiere a música, actualmente existen muchas páginas para escoger, lo cual incentiva su difusión. En cuanto a la colaboración musical, podemos encontrar las siguientes aplicaciones web, cada una con sus pros y sus contras: Sistemas como Blend (https://blend.io/) y Wavestack (http://www.wavestack.com/). Las dos se enfocan en ser repositorios de música, pero no realizan un manejo de control de versiones ni permiten crear versiones propias de otros artistas. La aplicación web Wavestack no integra algún sistema de edición musical, como es el caso de Blend.

Existen otras páginas web como Splice (https://www.splice.com) que nos permiten colaborar con otros artistas e incluso integran un control de versiones, pero que necesitan acoplarse a un software de edición ya existente.

Esta situación puede llegar a ser confusa y no muy amigable para el usuario final, el cual requerirá primero configurar la aplicación en su equipo para empezar a usar los servicios de la página.

Objetivos

Objetivo General

Crear una aplicación web que permita integrar la edición de sonidos, el control de la copia o versiones de estos sonidos y la colaboración entre varios usuarios.

Objetivos Específicos

- Implementar y adaptar un sistema de edición en línea que permitirá editar la pista como tal.
- Adaptar un sistema de control de versiones y adaptarlo conjuntamente

con el sistema de edición musical.

- Crear pruebas unitarias y de integración que nos servirá como documentación del código creado.
- Crear una documentación más extensa de las historias de usuario por medio de un gestor de tareas que use metodologías ágiles.

Metodologías

Se utilizarán el método exploratorio y el método experimental. A lo largo del uso de los mismos estos se complementarán el uno con el otro (una vez que se haya explorado una posible solución se la implementará de la manera más óptima).

El método exploratorio nos permitirá encontrar las librerías necesarias tanto para implementar el sistema de control de versiones, como la interfaz de edición musical.

El método experimental nos ayudará a diseñar de manera más clara la interfaz para cada una de las características del sistema expuestas anteriormente.

1. Capítulo I. Metodologías de Desarrollo

1.1. Metodologías Ágiles

En el software se tiene un amplio número de metodologías de desarrollo. Las primeras fueron inspiradas en metodologías usadas en otras ramas de ingeniería y adaptadas para su uso en la construcción de sistemas informáticos. Estas promueven un desarrollo lineal, haciendo énfasis en la planificación exhaustiva para su posterior ejecución. Un ejemplo de estas metodologías es el desarrollo en cascada.

Uno de los motivos por los cuales no se recomienda usar metodologías de desarrollo lineales, como el desarrollo en cascada, es que son procesos muy rígidos, los cuales contemplan de antemano, que todos los proyectos de software sean similares y que no necesiten gestionar muchos cambios a lo largo de su existencia.

Son contados los tipos de proyectos en el mundo del software que tienen una base y aplicación parecidas, y al ser el desarrollo de sistemas una rama relativamente joven en comparación a otras ingenierías, no se puede confiar ciegamente en un proceso que considera la construcción de sistemas de software como un proceso probado y casi infalible. Las metodologías ágiles se crearon con el objetivo de solventar estas deficiencias y obtener un mejor tiempo de respuesta a posibles errores o cambios que se den a lo largo de un proyecto, con iteraciones cortas para obtener un mejor feedback por parte del cliente.

Manifiesto Ágil

Como una alternativa a métodos formales, un grupo de desarrolladores decidió reunirse para establecer una forma más flexible, adaptable y acorde a las nuevas necesidades del desarrollo de software. Entre estos se encontraba Kent Beck, creador de la metodología Extreme Programming. En esta reunión se

estableció lo que hoy se conoce como el manifiesto ágil (Agile Manifesto, n.d.). , el cual enumera los siguientes principios:

Valorar más a los individuos y su interacción que a los procesos y las herramientas

En la industria generalmente se tiende a enaltecer los procesos como la única manera de lograr un objetivo en conjunto, dejando de lado el factor humano. Se debe considerar a los procesos como una ayuda, más no como la única forma de llevar adelante un proyecto.

Valorar más el software que funciona que la documentación exhaustiva

La documentación siempre es importante, tanto como respaldo o como constancia del trabajo realizado pero no es lo primero que ve el cliente. Un proyecto en funcionamiento es prueba suficiente de lo realizado.

Valorar más la colaboración con el cliente que la negociación contractual

Las metodologías tradicionales tienden a sobreplanificar los proyectos de desarrollo como una manera de acordar el trabajo a ser realizado con el cliente. Esto generalmente termina en un producto erróneo y en estimaciones incorrectas del trabajo a realizarse. Al tener una colaboración constante con el cliente se obtiene un acuerdo instantáneo con el mismo.

Valorar más la respuesta al cambio que el seguimiento de un plan

Al igual que en una negociación contractual, regirse completamente a un plan inicial aumenta el riesgo de no obtener el producto deseado. Se debe considerar el tener un plan inicial, pero tomando en cuenta que este puede cambiar en medio del proceso de desarrollo.

1.1.1 Tipos

Scrum

Scrum forma parte de las metodologías ágiles que consideran un proceso iterativo e incremental de desarrollo. Un equipo de Scrum es auto organizado y altamente comunicativo. Las tareas se cumplen en sprints que por lo general duran entre dos y cuatros semanas, cada sprint contiene un backlog de estas tareas, las cuales son ordenadas de acuerdo a su prioridad. El equipo entero es el encargado de estimar el esfuerzo necesario para acabar cada una de las tareas.

Figura 1. *Proceso Scrum*

Tomado de (Scrum Alliance, n.d.)

Extreme Programming (XP)

Al igual que Scrum, XP considera también un proceso iterativo, pero este hace mayor énfasis en el proceso de desarrollo de software. Sus principales características son: creación pruebas unitarias de tú código, programación en parejas, refactorización constante del código y retroalimentación constante de los cambios por parte del cliente.

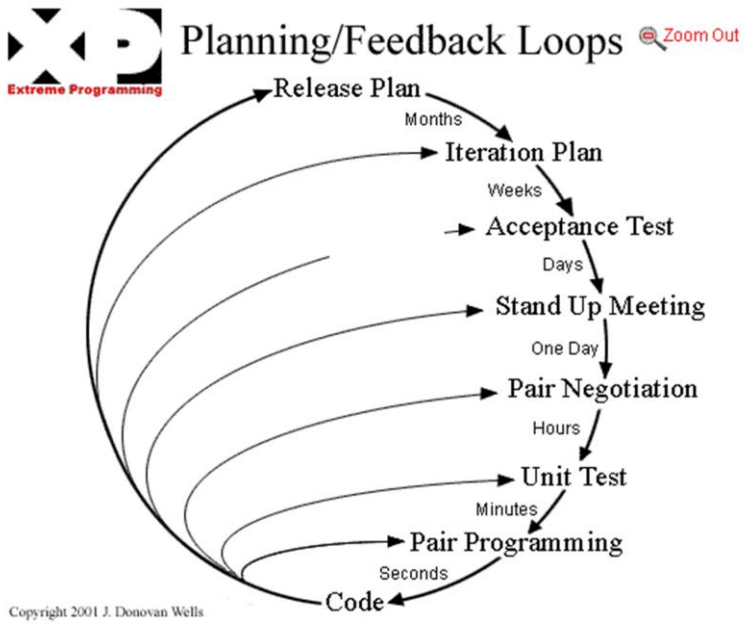


Figura 2. *Proceso XP*

Tomado de (Wells, n.d.)

1.2. Desarrollo guiado por pruebas

El desarrollo guiado por pruebas (o TDD) es una práctica de software que contempla la creación de pruebas automatizadas. Estas pruebas son creadas y mantenidas por los desarrolladores. Entre los beneficios del TDD están la mejora del código y una mayor tolerancia al cambio.

1.2.1. Red, Green, Refactor

TDD considera un proceso estricto, con el objetivo de analizar lo que se quiere implementar, para así tener una mejor idea de lo que se necesita desarrollar, y así mejorar nuestro código; a este proceso se lo denomina Red, Green, Refactor.

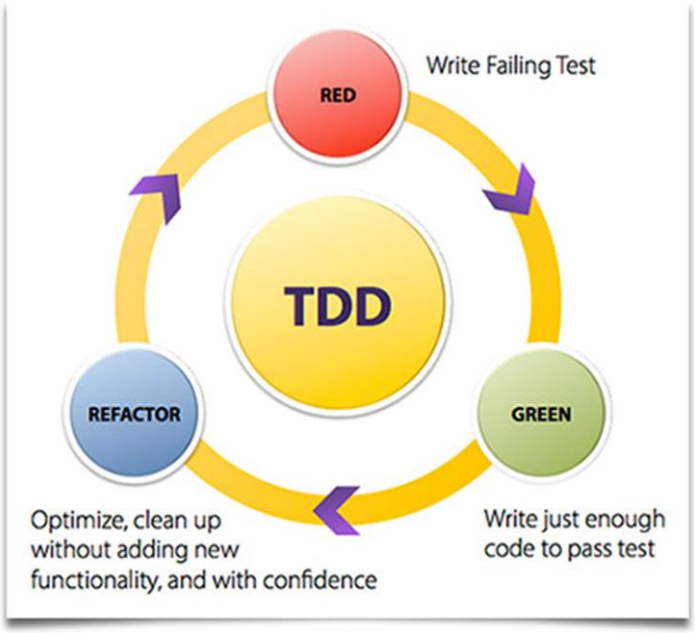


Figura 3. *Agile Test Driven Development*

Tomado de (Centric Consulting, n.d.)

Red (prueba fallida)

El Red se refiere a que primero debemos crear nuestra prueba antes de crear cualquier código. Al estar el test en color rojo significa que este no ha pasado.

Green (prueba superada)

El siguiente paso es hacer pasar nuestra prueba creando el código correspondiente; una vez que el test pase, este se mostrará con un color verde.

Refactor (mejorar código)

Una vez que hemos acabado de programar procedemos a mejorar nuestro código. Cada vez que realizamos un refactor debemos aseguramos de que nuestra prueba siga pasando. Si nuestro test vuelve a dar error, se debe repetir el ciclo.

1.2.2. Errores comunes en el proceso de crear pruebas

Pensar en el porcentaje de cobertura de nuestras pruebas

La cobertura de pruebas es el porcentaje de código que cubren nuestros test. Se sugiere un índice de cobertura del 100% de nuestro código. Sin embargo, las pruebas que se enfocan solo en la cobertura tienden a crear código poco refactorizable y generan pruebas muy frágiles y poco tolerantes a los cambios.

No tomar un tiempo equivalente para realizar refactor

Idealmente se debe tomar el mismo tiempo tanto para tanto hacer refactoring como para crear el test. Esto a la larga produce un código más limpio y mantenible.

1.2.3. Dobles de prueba

Las pruebas deben crearse en base solo a la funcionalidad que vayan a implementar. Cualquier dependencia debe mantenerse al mínimo, tanto por rendimiento del sistema como por diseño del código. Otro motivo para mantener la dependencia al mínimo en tests es que no podemos probar una dependencia de manera correcta, como por ejemplo llamadas a servicios web externos; estas llamadas pueden cambiar constantemente y quebrar nuestros tests. Con el objetivo de ocultar estas dependencias se crearon los mocks o dobles de prueba. Un mock es un objeto ficticio que reemplaza a un objeto real dentro de nuestro test.

Ventajas

Nuestros tests corren más rápido y no dependen uno de otro, lo cual permite probar cada parte de manera aislada y con mayor facilidad.

Desventajas

Crear muchos mocks puede llevar a la creación de pruebas no fiables, ya que asumimos una salida en nuestro mock que puede cambiar en cualquier momento en el objeto real.

1.2.4. Pruebas y su impacto en el desarrollo

Diseño del código

El realizar pruebas antes de escribir el código ayuda a entender mejor la implementación y su relación con otras partes del sistema, esto genera una mejora en el diseño. La forma en la que los tests indican que pueden haber problemas de diseño, es cuando el test se vuelve difícil de implementar, lo cual indica que el código tiene muchas responsabilidades . Parte de esas responsabilidades deben ser extraídas y testeadas de manera más profunda.

Documentación

Las pruebas no solo ayudan a tener un código más claro. Aparte de que tienen un impacto en el entendimiento del código, también sirven como documentación extra, indicando cuales son los parámetros de entrada y cuál es la salida que se obtendrá al correr un código específico. Ciertas librerías de pruebas permiten al desarrollador incluso escribir tests como si se estuviera escribiendo texto, dando lugar a pruebas descritas de manera más clara.

Depuración de errores

Uno de los objetivos de hacer pruebas es deshacernos del proceso de búsqueda de errores manual; si nos encontramos haciendo este proceso significa que no hemos realizado las suficientes pruebas a nuestro código y que este oculta una responsabilidad que debe ser extraída, por lo que se debe probar más a profundidad fuera del contexto de nuestro primer test.

1.2.5. Pruebas y proceso de Integración Continua

Las pruebas automatizadas son una parte esencial dentro del proceso de integración continua. Por lo general, para añadir nuevos cambios se sigue el siguiente proceso:

Escribir Pruebas

Antes de escribir código para una nueva funcionalidad, según el proceso de TDD, se debe escribir los tests automatizados.

Escribir Código

En este punto se crea el código y se verifica que todas las pruebas de la aplicación, incluyendo las creadas para esta funcionalidad, pasen localmente.

Correr pruebas en servidor de versiones

Una vez que se hayan verificado los cambios localmente se procede a crear un permiso para añadir los nuevos cambios al repositorio. El servidor de control de versiones corre las pruebas automatizadas cuando se realizan cambios en el código y estos son añadidos al servidor.

Correr pruebas en distintos ambientes

Cada vez que se añaden nuevos cambios al servidor de versiones se procede a desplegar los cambios en los distintos ambientes; esto se le puede hacer automáticamente, en el momento que todas las pruebas pasen, o de forma manual. Se aconseja hacer un despliegue manual en servidores de producción.

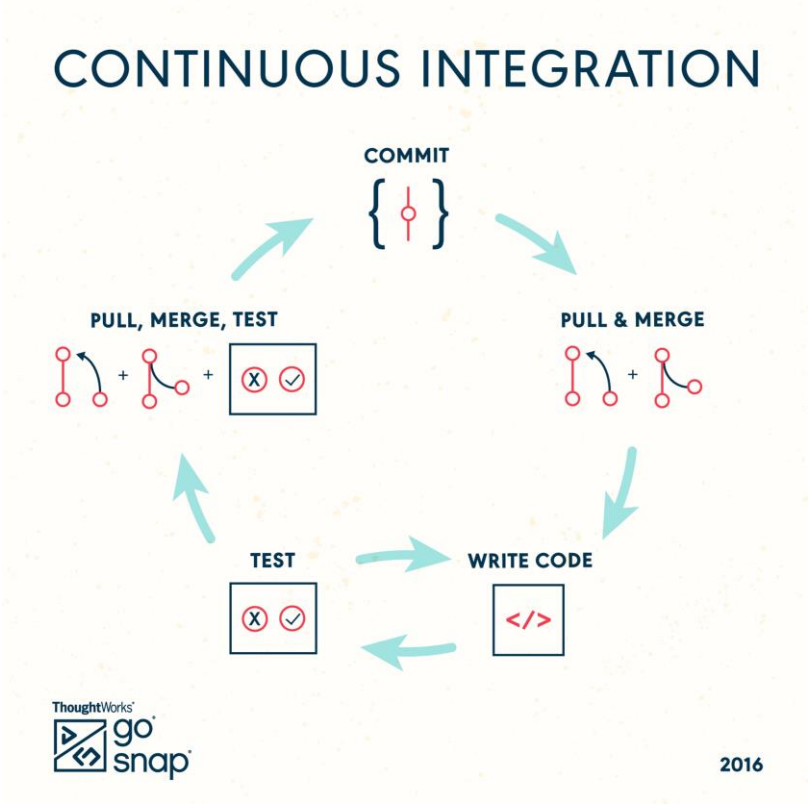


Figura 4. *Integración Continua*

Tomado de (Rice, n.d.)

1.3. Marcos de desarrollo

Un framework o marco de trabajo es una herramienta que permite crear aplicaciones con mayor rapidez sin tener que empezar a desarrollar desde cero. Los marcos de trabajo web incorporan conexiones a múltiples bases de datos, comunicación de datos a la web, manejo de requests por parte de los navegadores y los presenta de una forma sencilla para el programador. Esto se logra siguiendo patrones de diseño.

1.3.1. Modelo Vista Controlador

MVC es un patrón de diseño que separa su lógico por medio de tres objetos: modelo, vista y controlador; el modelo se encarga de la parte de

persistencia de datos, la vista de la parte gráfica y el controlador de lo referente a manejo de requerimientos activados por la vista.

Elementos MVC

Modelo: El modelo abstrae todo lo referente a conexión con bases de datos por medio de un objeto, este objeto representa una tabla en el sistema. Cuando la clase modelo se hereda esta concede múltiples funcionalidades como validaciones de datos, relaciones con otros objetos y una interfaz clara tanto para guardar como para obtener datos de la base.

Vista: En lo referente a web, la vista representa el html creado. Algunos frameworks incorporan templates (páginas con un diseño incluido) que permiten generar html usando código desde nuestro backend.

Controlador: Se encarga de manejar todos los requests provenientes de la vista. En este caso un navegador será el que lo active. A su vez, el controlador redirecciona los datos a las vistas correspondientes.

Proceso MVC

El controlador es el eje principal dentro de un esquema MVC. El ruteador es el encargado de recibir un URL y redirigirlo al controlador correspondiente. En caso de que se necesiten datos para desplegar la página, el controlador recurre al modelo para tomar los datos de la tabla que representa. Una vez que el controlador obtiene los datos los redirige a la vista y los despliega en el navegador.

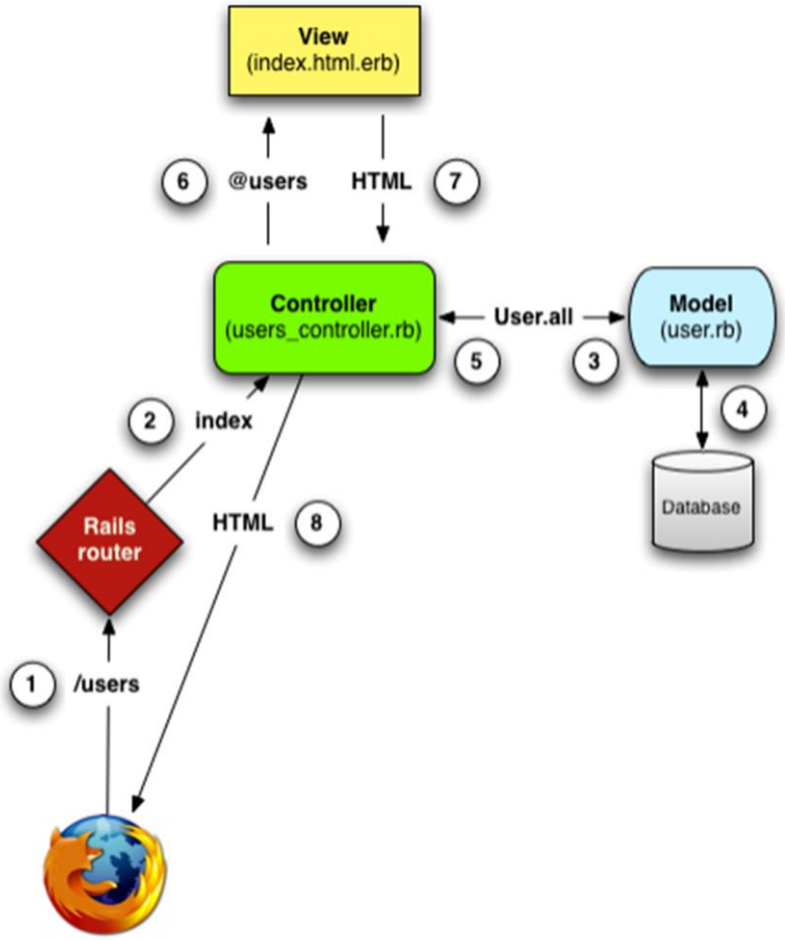


Figura 5. *Diagrama MVC*

Tomado de (Hartl, 2013)

Ventajas

Una de las principales bases de MVC y su mayor fortaleza es la separación correcta de los roles y su flujo dentro de una aplicación. El patrón MVC permite pensar en el flujo de los datos a través de estas capas, sin que el desarrollador tenga que preocuparse por lo que pasa por debajo. Este patrón de diseño lleva bastante tiempo siendo utilizado en el mercado, lo cual ha permitido la evolución del mismo y de los frameworks que adoptan este patrón a tal punto que ahora es muy fácil crear una aplicación desde cero.

Desventajas

MVC fue pensado para aplicaciones de escritorio con un solo lenguaje de programación en común a través de todas sus capas. En el caso de un ambiente web se hace uso lenguajes de programación distintos, sobre todo en la capa de vista; esto da a lugar a aplicaciones más complejas que requieren una mayor cantidad de programación por fuera del framework, para crear aplicaciones más dinámicas en el front-end. Por ejemplo, para desarrollar aplicaciones que no necesitan refresco en el navegador es necesario usar librerías externas, con su propio patrón de diseño, el cual a su vez tiene que comunicarse con en el framework MVC para la obtención de datos.

1.3.2. Modelo Vista Vista-Modelo

MVVM es un patrón generalmente usado en aplicaciones que requieren una constante actualización de datos. Los datos del modelo no pasan por un controlador, sino que son manejados por una capa intermedia llamada vista-modelo. La capa Vista-Modelo sirve de contexto para la publicación de datos en las vistas.

Elementos MVVM

Modelo: Al igual que en el patrón MVC, este se encarga de la obtención y validación de datos de manera transparente, independiente del tipo o fuente de datos.

Vista-Modelo: Maneja la actualización de datos del modelo por medio de eventos. En esta capa se asume que los datos se actualizan automáticamente, por lo cual no es necesario definir métodos explícitos para manejar la creación o eliminación de datos.

Vista: Se encarga de mostrar los datos y ejecutar los eventos definidos en la capa de Vista-Modelo. Los datos se actualizan automáticamente sin necesidad de hacer refrescos de página.

Proceso MVVM

En el momento en el que un usuario realiza una acción en la vista, esta ejecutará una acción en la capa vista-modelo; a su vez se encargará de realizar las consultas o cambios respectivos en el modelo. No existe necesidad de enviar los datos de nuevo a las vistas ya que estas se actualizan de manera automática.

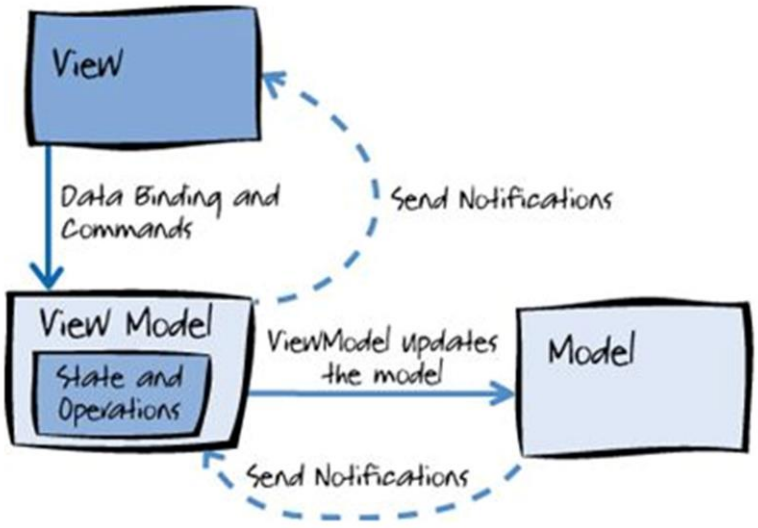


Figura 6. *Diagrama Patrón MVVM*

Tomado de (Silva, 2015)

Ventajas

Este tipo de marcos de trabajo ayudan a crear aplicaciones más ricas en comparación con un marco de trabajo de tipo MVC. Debido a su interactividad pueden ser transportadas fácilmente a otro tipo de interfaces como los dispositivos móviles. Gracias a que necesitan un refresco de datos constante, estos frameworks proveen abstracciones que permiten manejar datos en tiempo real de manera más fácil, sin necesidad de crear código adicional en el lado del cliente.

Desventajas

El patrón MVVM asume que todas las vistas deben estar constantemente actualizadas; a la larga esto puede generar problemas desempeño. Estos problemas generalmente son más notorios en el lado del cliente, pero también pueden pasar al lado del servidor. Por razones de seguridad, se puede requerir que los datos pasen primero por un proceso de validación directamente en el servidor, antes de que puedan ser mostrados en el cliente. La validación de datos en el servidor no es posible en estos tipos de frameworks ya que la mayor parte del procesamiento pasa primero en el cliente.

2. Capítulo II. Herramientas de Desarrollo

Como herramientas se escogieron aquellas que permiten un desarrollo rápido de aplicaciones web, particularmente del lado del cliente. La mayor parte del procesamiento de audio sucede en el navegador, por lo cual es necesario un entorno de desarrollo que facilite la creación de páginas interactivas en el front-end.

Tanto para el despliegue como para la creación de nuevas funcionalidades la aplicación contará con un servidor de integración continua, el cual verificará nuevos cambios almacenados tanto en el servidor de versiones como en el servidor de hosting.

2.1. Lenguajes de Programación

En la actualidad se cuenta con una variedad de lenguajes de programación para el desarrollo. En el ámbito web contamos con lenguajes de tipo estático y dinámico.

Entre los lenguajes de tipo estático tenemos a Java, #C, Haskell entre otros. Estos lenguajes proveen sus respectivos entornos de desarrollo. En el caso de C# y Java es necesario el uso de herramientas específicas para el desarrollo y/o despliegue, y en el caso de C# la mayor parte son software propietario.

Los lenguajes de tipo estático son una gran elección cuando se cuenta con grandes equipos de desarrollo y un proyecto establecido.

En los últimos años se ha tratado de potenciar el uso de lenguajes dinámicos. Estos permiten un desarrollo rápido en proyectos en los que se cuenta con un limitado número de recursos y tiempo.

2.1.1. Ruby

Según (Flanagan & Matsumoto, 2008) Ruby es un lenguaje de programación dinámico con un gramática expresiva, de fácil lectura, el cual cuenta con un API poderoso. Ruby fue inspirado por lenguajes como Lisp, Smalltalk y Perl pero con una gramática de fácil entendimiento para desarrolladores más familiarizados en lenguajes como C y Java. Ruby es un lenguaje orientado a objetos, pero también puede ser usado en un estilo funcional o procedural. Incluye características como meta programación, la cual permite la creación de lenguajes específicos de dominio, característica muy usada por frameworks de desarrollo como Ruby on Rails.

Este lenguaje, gracias a su facilidad de uso y popularidad dentro del desarrollo web, provee las librerías necesarias para dotar la aplicación con características como registro e inicio de sesión de usuarios, facilidades de despliegue en servidores gratuitos como Heroku, entre otros.

2.1.2. Opal

Opal es un compilador de código Ruby a Javascript el cual permite usar gran parte de la implementación de Ruby (librerías Core y Standard) dentro del navegador.

Gracias a Opal se pueden crear librerías o gemas de Ruby que corran dentro del navegador. Esto ayudará en la manipulación de audio en el browser, ya que se puede usar Ruby directamente para controlar el audio y compartir esta lógica entre el front-end y el back-end.

2.2. Volt Framework

Según (Volt Framework, n.d.) Volt es un marco de desarrollo en el cual el código Ruby corre tanto en el servidor como en el cliente (gracias al transpilador Opal). Los elementos HTML dentro de la página se actualizan al mismo tiempo que el usuario interactúa con la misma.

En lugar de sincronizar los datos entre el cliente y el servidor por medio de http (requests POST y GET) Volt usa una conexión permanente (WebSockets). Cuando los datos son cambiados dentro de un cliente específico estos se verán reflejados tanto en el servidor como en otros clientes.

Volt usa programación reactiva para actualizar automáticamente el contenido HTML, cuando una parte de estos datos cambia solo esa sección del HTML es actualizado. Esto facilitará la actualización y obtención de datos concernientes tanto a las características sociales de la aplicación, como al editor de audio, ya que se requiere almacenar los datos de los usuarios automáticamente sin necesidad de refrescos de página.

Volt usa el patrón de diseño MVVM pero también se puede cambiar a MVC si así se lo requiere, por conveniencia se usa la capa VistaModelo como controlador

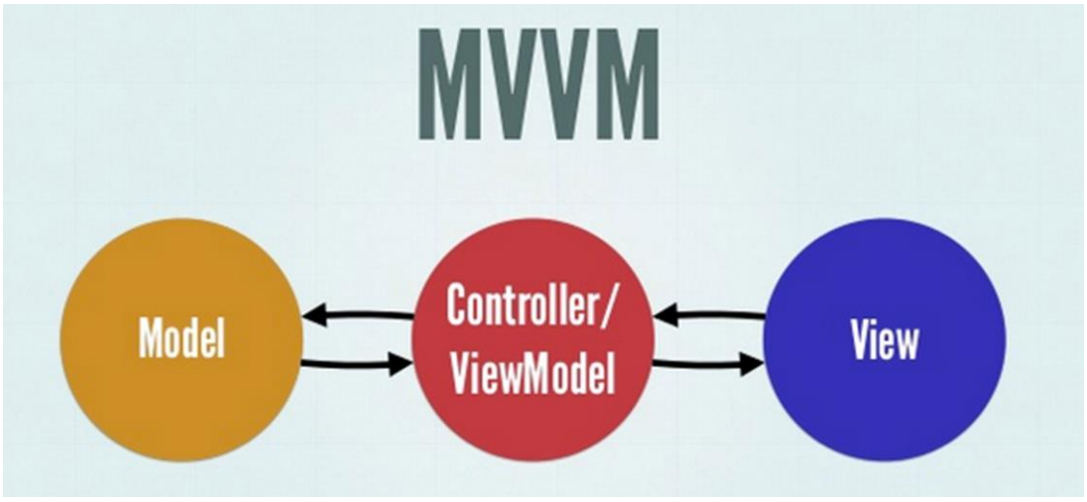


Figura 7. *Diagrama Volt MVVM*

Tomado de (Stout, 2014)

2.3. Mongo DB

Mongo DB es una base de datos tipo NOSQL orientada al almacenamiento de datos en una base tipo documento. Los datos son guardados como datos JSON, lo cual permite una mayor libertad al momento de almacenar información.

(Taylor, 2015) indica que una base NOSQL no usa el lenguaje SQL para realizar consultas, no sigue el modelo relacional, ni requiere de un esquema para relacionar diferentes tipos de datos, por lo que es muy útil en casos en los que se necesitan guardar datos adicionales dentro de un mismo tipo de record.

El usar bases de datos de tipo SQL restringe de antemano el esquema y los tipos de datos a usarse para una determinada tabla. Por medio de una base de tipo NOSQL como Mongo se pueden almacenar nuevos tipos de datos dinámicamente, en el instante en el que sea requerido. En el caso de una aplicación de tipo social esto resulta útil ya que se puede almacenar datos adicionales de un usuario, como datos estadísticos, sin tener que cambiar toda la estructura de la tabla.

2.4. Git

Git es un sistema de control de versiones de código abierto. Este fue desarrollado por Linus Torvalds con el objetivo de ser usado en las contribuciones hechas al kernel de Linux.

(Chacon & Straub, 2015, pp. 32-36) Git almacena información de forma distinta en comparación a sistemas como Subversion. Conceptualmente otros sistemas almacenan información como una lista de cambios hechos a un archivo; por el contrario, GIt guarda múltiples versiones de estos archivos, a manera de snapshots, o capturas del estado del archivo en un punto determinado en el que se realizaron estos cambios.

Las operaciones de registro de cambios son realizadas localmente, sin necesidad de un servidor central; esto lo convierte en un sistema de control de

versiones distribuido y descentralizado. Si un repositorio principal desaparece, este puede ser reemplazado fácilmente por cualquier otro servidor o usuario que posea el repositorio Git.

Gracias a que Git es de código abierto, existen en la actualidad múltiples servidores gratis para almacenar proyectos y librerías en la nube. Para la aplicación y librerías se usará Git por medio de la plataforma Github; esta puede enlazarse fácilmente con otras plataformas como Travis para integración continua, Heroku para despliegue de aplicaciones, entre otros.

2.5. Heroku

(Middleton & Schneeman, 2013) Heroku es un servicio de tipo 'PaaS' o plataforma como servicio. Permite el alojamiento de aplicaciones web sin necesidad de realizar configuraciones adicionales en el servidor. Heroku requiere cumplir con ciertas opiniones para correr el código. Aunque estas opiniones pueden parecer severas, al aplicarlas permiten tener una aplicación flexible, escalable y tolerante a fallos.

Heroku puede correr aplicaciones en código Python, PHP, Node.js entre otros, pero comenzó alojando aplicaciones en el lenguaje Ruby, por lo que es muy fácil correr aplicaciones creadas en frameworks como Ruby on Rails, Sinatra e incluso Volt.

Esta plataforma también tiene la ventaja de ser gratuita, en comparación a otras alternativas virtualizadas.

2.6. Servidor de Integración Continua

La principal función del servidor de integración continua es la de verificar nuevos cambios subidos a un proyecto; esto lo hace corriendo los tests automatizados del proyecto dentro de un servidor o máquina virtual. Este servidor debe funcionar independientemente del servidor de la aplicación.

El servidor de CI se enlazará con el servidor de control de versiones para ejecutar la suite de pruebas en cada nuevo cambio y a su vez, manejar

automáticamente el despliegue de nuevas versiones de la aplicación. Estos procedimientos se ejecutarán en todos los ambientes disponibles.

Este ciclo de desarrollo ayuda a mejorar la calidad del código en general, permitiendo a los desarrolladores crear sus pruebas para verificar su código y para tener una mejor respuesta a nuevos cambios.

2.6.1. Rspec

Rspec es un lenguaje de dominio específico para la creación de pruebas en el lenguaje de programación Ruby. Sigue el estilo de pruebas guiadas por el comportamiento, por lo que resultan más descriptivos en comparación a otras librerías de pruebas como minitest.

El uso de esta librería es muy extendido dentro de la comunidad y es usada por otras librerías para crear pruebas de integración, aceptación entre otros.

2.6.2. Travis

Travis es un servidor de integración continua el cual puede ser usado de manera gratuita en proyectos de código abierto. Este servidor se encarga de correr nuestros tests por medio de un archivo de configuración el cual contiene la versión del lenguaje, la versión del framework de pruebas, etc.

El uso de Travis es muy extendido dentro de la comunidad de código abierto. Este tiene integración con servidores repositorio como Github, por lo que beneficiara a las librerías que serán creadas para este proyecto.

2.7. Web Audio Api

(Smus, 2011) Indica que antes de existir HTML5 el audio en la web era generado por medio de flash o con plugins. A pesar de que con HTML5 no es necesario plugin alguno para generar audio (gracias al tag <audio>), este es

muy limitado en su uso.

Web Audio API es un API Javascript de alto nivel que permite procesar audio dentro de una aplicación web. El objetivo principal de este API es el de imitar la capacidad de generar audio en la web como si se lo hiciera en un computadora de escritorio.

Funcionamiento

(Mozilla Developer Network, n.d.) indica que la idea principal detrás de este API es la de producir y configurar sonidos por medio de nodos de audio. Los nodos se pueden enrutar y juntar para producir distintos resultados. Cualquier componente de Web Audio API se ejecuta en un contexto específico de audio llamado AudioContext.

Cada nodo tiene una entrada y una salida. El nodo fuente es aquel que genera el sonido, este puede ser un oscilador, una matriz de datos (generalmente un archivo de audio) o un dispositivo externo (por medio de WebRTC, por ejemplo).

2.8. Edición Musical

La edición musical conlleva el cambio y la manipulación de determinados sonidos, pudiendo ser estos pregrabados o creados por medios analógicos o digitales. Con el fin de suavizar el resultado final de una canción o melodía es necesario mezclar los sonidos de tal manera que no se noten los cambios abruptos en el sonido, pudiendo darse estos en la frecuencia o la amplitud del sonido. También se busca realzar ciertas partes de una melodía agregando efectos y combinarlos con cambios en la amplitud del sonido. Gracias a los medios digitales el proceso de edición se ha vuelto mucho más fácil y de mejor calidad.

2.8.1. Audio Digital

Antes de existir las computadoras las grabaciones eran realizadas de forma

analógica, es decir se almacenaban en medios físicos como cassettes o vinilos. Conseguir equipos de mezcla para medio analógicos era considerado costoso y muy difícil de manejar.

Ahora mediante las grabaciones digitales podemos almacenar nuestras grabaciones en un archivo digital y procesarlo en software de edición de sonido. De igual forma podemos generar sonidos digitales, los cuales pueden simular sonidos de instrumentos reales.

2.8.2. Sintetizadores

Los sintetizadores son la base para la generación de sonidos electrónicos, (Danzeria, n.d.) señala que en un principio funcionaban de manera analógica (manipulando directamente la corriente eléctrica), luego se empezó a crear sonidos por medio de la manipulación de ondas FM (dando origen a los sintetizadores digitales) y finalmente por medio de la manipulación de valores discretos (números enteros) de un sistema binario dio origen a la manipulación de sonidos por medio de software (sintetizadores virtuales).

(Danzeria, n.d.) Indica que en términos básicos los sintetizadores crean sonidos por medio de circuitos. Un sintetizador produce una onda de audio que puede ser manipulada para crear distintos sonidos. Las ondas son generadas por medio de voltaje o digitalmente por medio de un DCO (Oscilador controlado digitalmente). Las ondas son modificadas por medio de chips digitales, los cuales recrean muestras de distintos tipos de sonidos (con distintos tipos de ondas).

Para poder generar distintos tipos de sonidos el sintetizador modifica los siguientes elementos:

Oscilador

Es el componente principal para la generación de sonidos. Este produce una onda que dará como resultado un sonido (dependiendo de las propiedades de esta onda los sonidos creados pueden variar).

Manipulación de Onda

Distancia: Especifica el tiempo que va a durar el sonido generado

Frecuencia, longitud de Onda: Es el número de ciclos generado por la onda, este se traduce en una nota musical; por ejemplo los 440Hz corresponden a la nota musical LA.

Amplitud: Es el tamaño de la onda. Esta se traduce en el volumen del sonido generado.

Tipos de Onda: Las formas de onda transforman el sonido generado (cambiando su timbre). Existen los siguientes tipos de onda: Senoidal, triangular, cuadrada y diente de sierra.

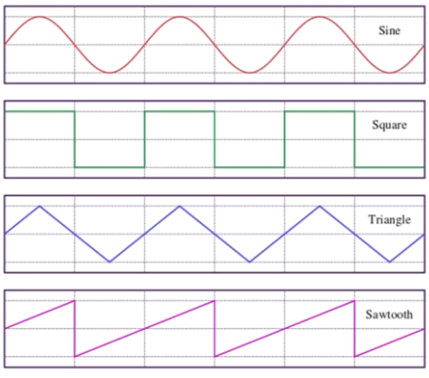


Figura 8. *Tipos de Onda Básicos*

Tomado de (omarsdiscodojo, n.d.)

2.8.3. Secuenciadores Musicales

El secuenciador es un forma de edición musical muy básica. Los secuenciadores pueden generar sonidos y ordenarlos en un punto en el tiempo para su reproducción. Entre los primeros secuenciadores musicales tenemos la caja musical, la cual cuenta con dientes que reproducen una determinada nota al momento de accionarlo. Dependiendo de la velocidad del rodillo, la pista grabada en este se reproduce más rápido o más lento. En la actualidad se cuenta con software secuenciadores que nos permiten grabar y programar pistas.

Secuenciador

Un sequencer o secuenciador cuenta con varias pistas; cada pista contiene notas musicales en las cuales se puede modificar el volumen, añadir efectos y cambiar su sonido modificando el tipo de onda. Posee también controles para su reproducción y un piano para identificar las notas. Se puede usar el teclado proporcionado en el software o podemos conectar un teclado físico por medio del protocolo MIDI.

Tracker

El tracker en un tipo de sequencer. Cada nota es representada por medio de códigos. Cada código contiene datos como el volumen, la nota, el efecto entre otros. Una canción entera consta de una grilla, donde las filas representan un momento en el tiempo y las columnas un canal específico. La edición en un tracker es más compleja y menos amigable que la de un sequencer normal. Los trackers fueron creadas de esta forma debido a la carencia en su tiempo de interfaces gráficas más ricas como las que se tiene en la actualidad.

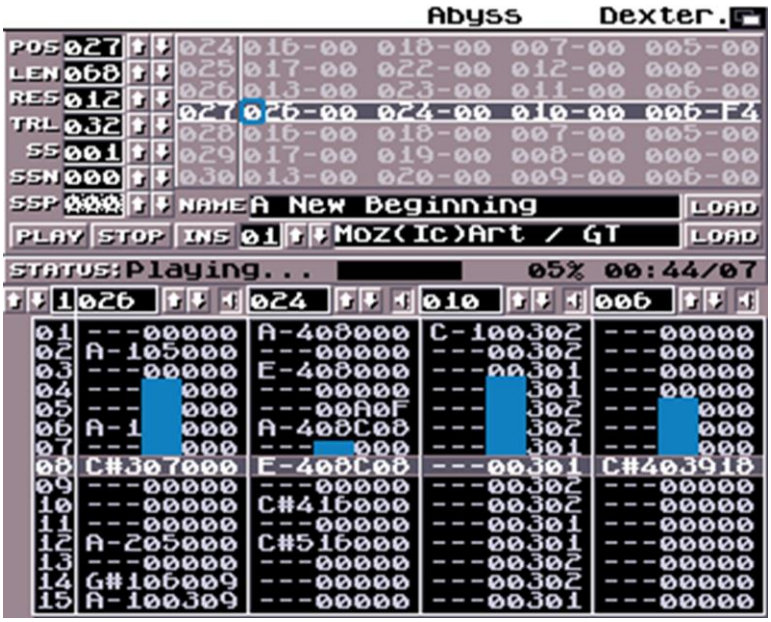


Figura 9. *Music Tracker*

Tomado de (Tracker Base, n.d.)

3. Capitulo III. Diseño

3.1. Historias de usuario

Como metodología de desarrollo se escogió Scrum. Esta metodología nos permite tener avances rápidos y modificaciones de las características de nuestro software en medio del proceso de desarrollo. El seguimiento de estas características se lo hace por medio de las historias de usuario. Cada historia contiene el detalle completo de la funcionalidad a ser implementada.

A continuación, por medio de los siguientes diagramas de caso de uso, se detallará de manera general la funcionalidad general de la aplicación, desde el punto de vista del usuario.

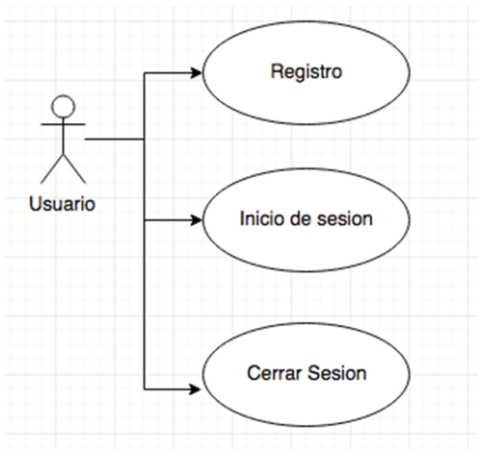


Figura 10. *Caso de Uso: Autenticación de Usuario*

Para crear las historias de usuario se debe tomar en cuenta todos los detalles de la funcionalidad y la estimación del esfuerzo necesario para completar la tarea (por medio de los puntos de la historia). Como nombre del proyecto se escogió Loopify, la aplicación contempla la creación de la aplicación web y una librería personalizada (Opal-Music) para la generación de notas musicales.

A continuación se detalla las historias iniciales para la creación del proyecto.

3.1.1. Aplicación Web (Loopify)

La aplicación podrá generar loops por medio de secuencias. Una secuencia representa un sección con notas. Estas notas pueden repetirse múltiples veces a lo largo del loop. Todas las secuencias ocuparán un mismo espacio de tiempo de tiempo; esto significa que al agregar más notas estas simplemente dividirán la secuencia por el número de notas requeridas, por lo que al agregar más notas tendremos notas de menor duración y al agregar menos notas tendremos notas con mayor duración. Otra ventaja de tener secuencias con la misma longitud es que se vuelva más fácil secuenciarlas en el loop.

La principal funcionalidad del sistema es el editor de loops. A continuación se muestra un diagrama con las actividades principales de esta funcionalidad.

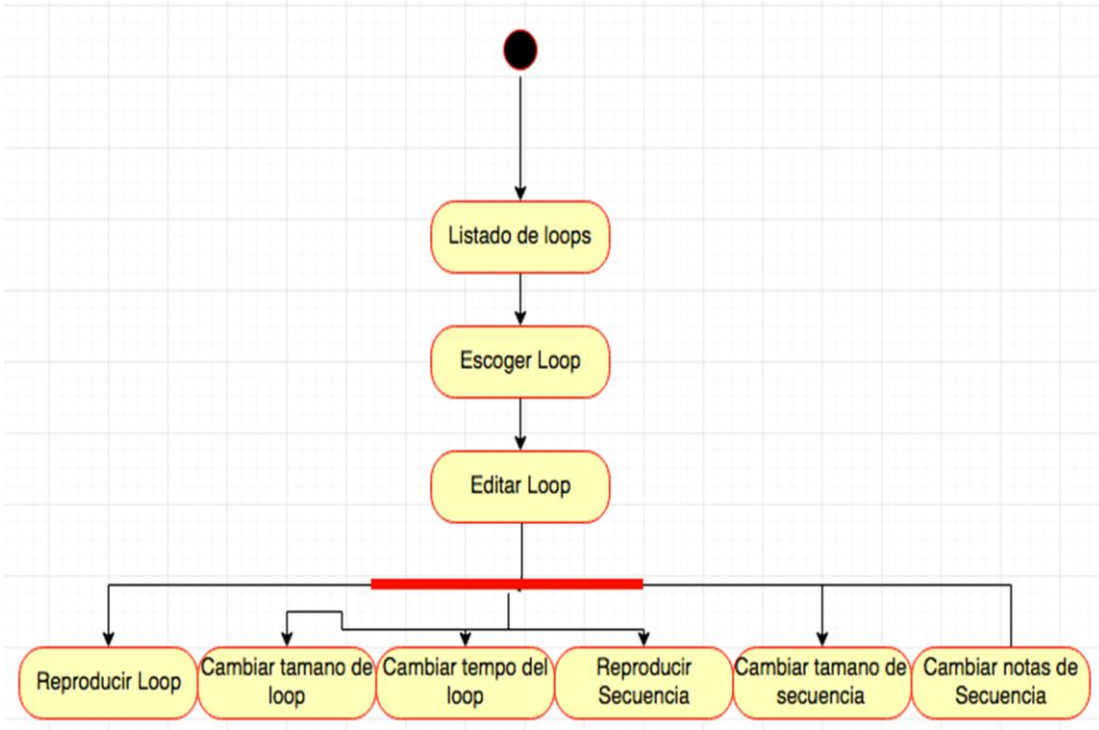


Figura 11. *Diagrama de Actividades: Loops*

Creación de múltiples secuencias

Figura 12 *Editor con Secuencias*

Para poder crear distintas secciones de un loop crearemos las secuencias, las cuales se podrán programar a lo largo del loop. Cada secuencia tendrá la posibilidad de tener un tipo de efecto/timbre, un volumen determinado y la posibilidad de escucharla por medio de un botón de reproducción. Los cambios en el loop y las secuencias serán almacenados en la base de datos y en el sistema de control de versiones GIT. En la tabla 1 se incluye a más detalle la implementación de esta historia.

Tabla 1. *Historia de Usuario: Como cliente puedo crear un loop con múltiples secuencias*

HISTORIA DE USUARIO			
NÚMERO:	10659321	TIPO:	Feature
PUNTOS ESTIMADOS:	2	TÓPICO:	Editor Loops
TÍTULO:	Como cliente, puedo crear un loop con múltiples secuencias		
DESCRIPCIÓN :			

Este editor permitirá la creación básica de loops, los cuales tendrán más de una secuencia
TAREAS :
El usuario puede modificar el título y volumen del loop Crear botón de play que reproducir la secuencia Crear control de volumen Se puede añadir efectos para cada loop (usar tipo de onda de web audio API (sine, square, etc)) Los cambios se almacenarán en la base de datos y en un archivo. Este archivo se actualizará cada vez que existan nuevos cambios y usará el sistema de control de versiones GIT para almacenarlo

Notas en una secuencia

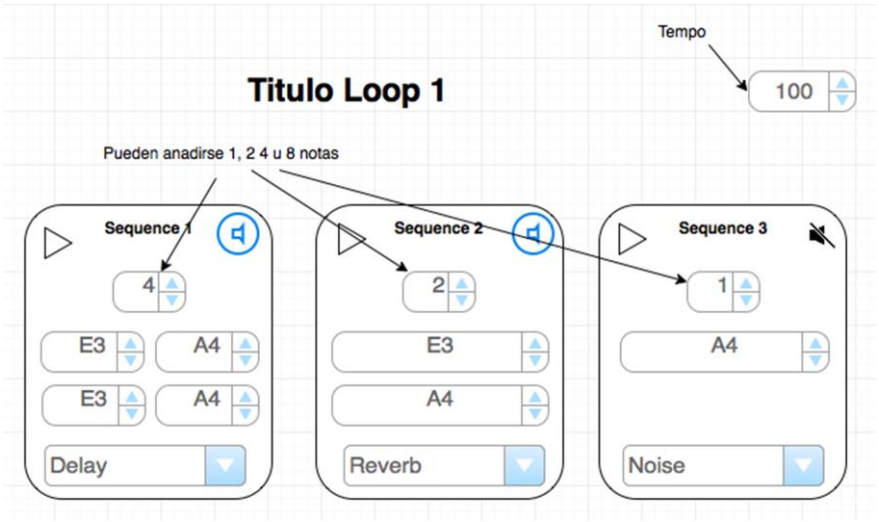


Figura 13. *Secuencias con notas*

Para las notas de la secuencia se podrán escoger 1, 2, 4 u 8 notas. (se escogieron pares para su división en partes iguales y para su uso interno en la librería de edición musical). Al escoger el número de notas

se crearan/disminuirán automáticamente el número de dropdowns con notas. En la tabla 2 se muestra un resumen de esta historia.

Tabla 2. *Historia de Usuario: Como cliente, puedo agregar notas a mi secuencia*

HISTORIA DE USUARIO			
NÚMERO:	10877534	TIPO:	Feature
PUNTOS ESTIMADOS:	2	TÓPICO:	Editor Loops
TÍTULO:	Como cliente, puedo agregar notas a mi secuencia		
DESCRIPCIÓN :			
Se podrá cambiar las notas de una secuencia específica			
TAREAS :			
Agregar notas por medio del atributo ’notes’ de ’sequence’ Agregar select con opción para agregar 1,2,4 u 8 notas. En la base de datos se creará el campo ’quantity’ para agregar el tiempo en cada nota (1 = whole note, 2 = half note, 4 = quarter note, 8 = eigth note) Al escoger el número de notas se debe desplegar el número de selects respectivo (las cuales tendrán como opciones las notas musicales existentes); por defecto empezará con una nota vacía o ’-’			

Cambio de notas en el momento de la reproducción

Con el fin de poder escuchar los cambios realizados a una secuencia, esta debe tener la posibilidad de cambiar internamente sin interrumpir la reproducción. La secuencia se reproducirá de manera continua, a menos que se pare su reproducción. Si se cambia una nota este cambio se verá reflejado en la siguiente reproducción de la secuencia, sin tener que reiniciar su reproducción. Como se detalla en la tabla 3 esto debe realizarse junto con la librería de edición creada.

Tabla 3. *Historia de Usuario: Habilidad para cambiar notas al momento de reproducirlas*

HISTORIA DE USUARIO			
NÚMERO:	11028088	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Editor Loops
TÍTULO:	Habilidad para cambiar notas al momento de reproducirlas		
DESCRIPCIÓN :			
Al momento de reproducir la secuencia, añadir la habilidad de cambiar las notas; estos cambios deben estar presentes, a su vez, en la siguiente reproducción del loop			
TAREAS :			
Anadir la habilidad de mutar las notas en el momento de reproducción Realizar estos cambios junto con la librería Opal-Music			

Programación de múltiples secuencias en el loop

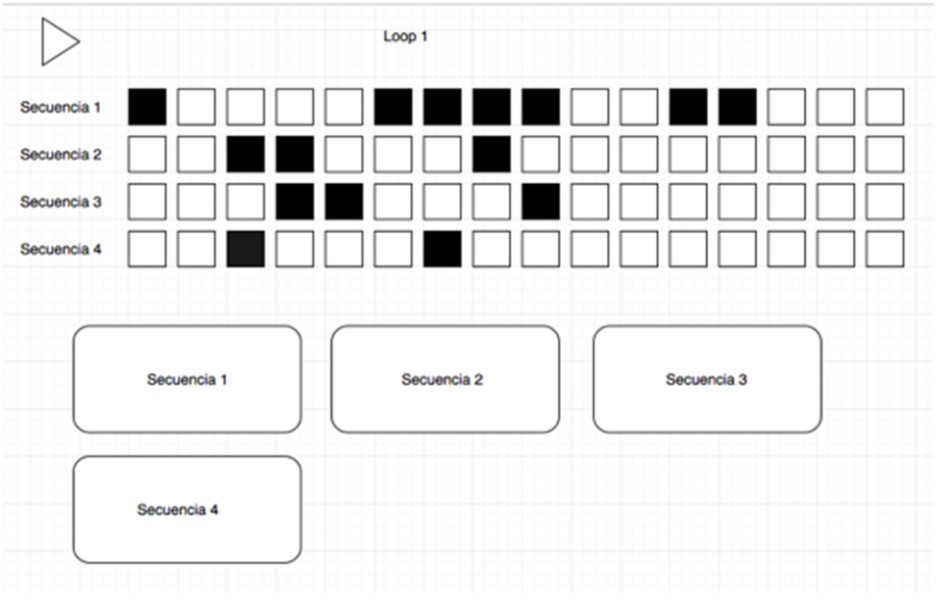


Figura 14. *Programación de secuencias*

Para programar las secuencias se creará una matriz grafica de checkboxes; cada fila representa a una secuencia y cada columna el instante en el que se las reproducirá (si está marcada se reproduce, sino simplemente se reproduce como una secuencia en silencio). En la tabla 4 se detalle lo necesario para crear esta historia.

Tabla 4. *Historia de Usuario: Habilidad para programar las secuencias por medio de una matriz gráfica*

HISTORIA DE USUARIO			
NÚMERO:	11105693	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Editor Loops
TÍTULO:	Habilidad para programar las secuencias por medio de una matriz gráfica		
DESCRIPCIÓN :			
Se usará una matriz gráfica para especificar el momento de reproducción de una secuencia. Cada fila representa una secuencia y las columnas son el punto de reproducción de la secuencia. Si una celda de la matriz se encuentra seleccionada esta será reproducida, caso contrario se lo toma como un espacio en blanco (no se reproduce)			
TAREAS :			
Añadir una matriz de selects para programar cada una de las secuencias. Añadir la posibilidad de aumentar o disminuir la longitud de todo el loop, lo cual aumentará o disminuirá el número de selects en la matriz gráfica de reproducción.			

Otra funcionalidad del sistema es la exploración de loops y usuarios. En el siguiente diagrama se muestra con mayor detalle esta funcionalidad.

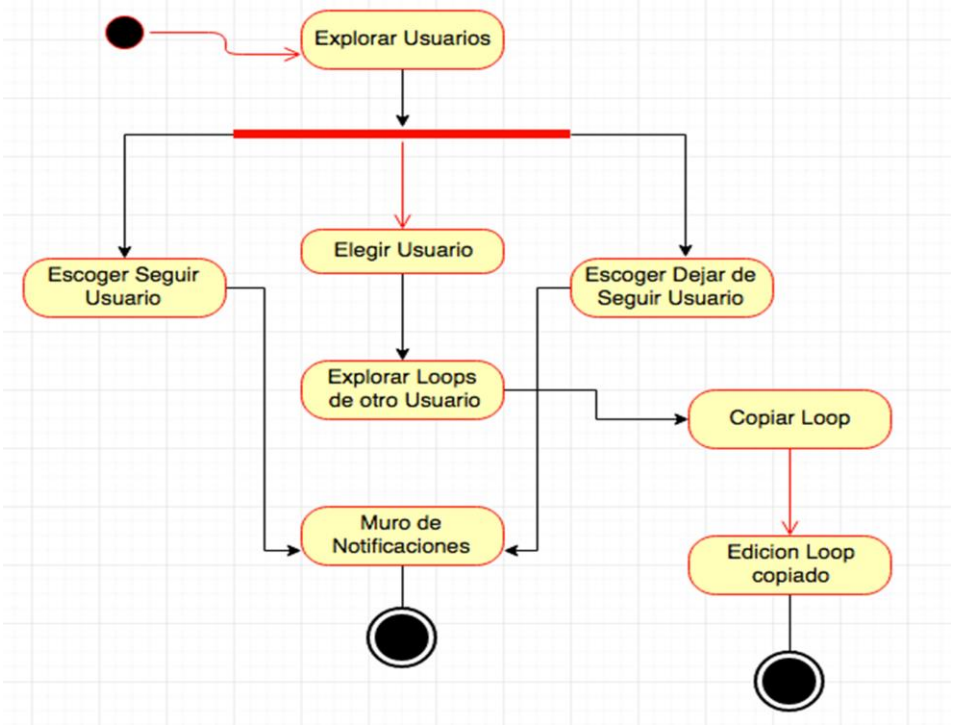


Figura 15. *Diagrama de Actividades: Explorar*

Usuarios con múltiples loops

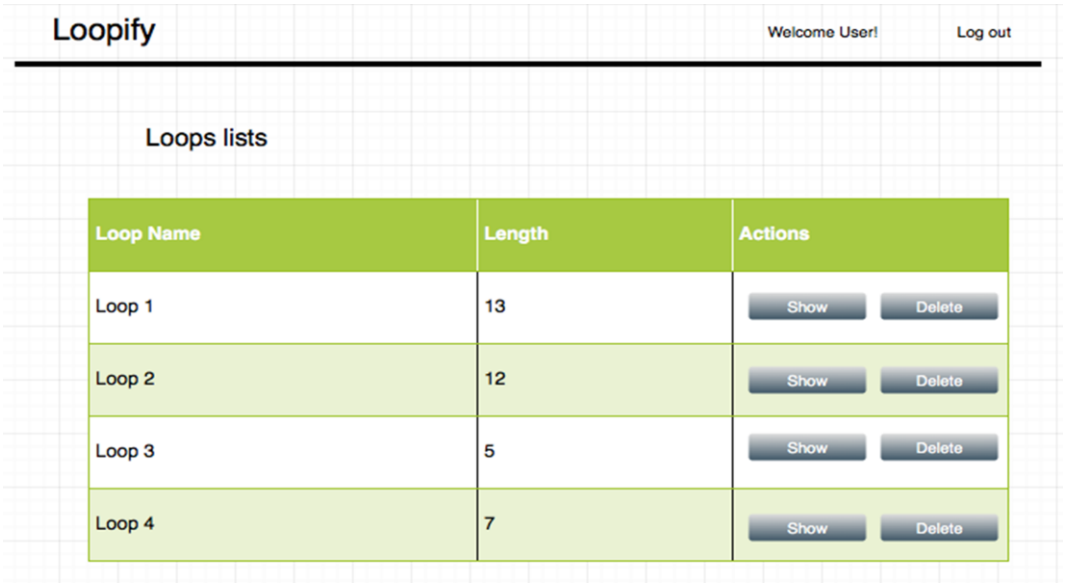


Figura 16. *Lista de Loops de usuario autenticado*

Una vez creada la posibilidad de tener un loop y programar nuestras notas por medio de secuencias se creará la funcionalidad que permita a un usuario tener múltiples loops. En la tabla 5 se detalla esta historia.

Tabla 5. *Historia de Usuario: Como usuario, puedo tener más de un solo loop*

HISTORIA DE USUARIO			
NÚMERO:	10925000	TIPO:	Feature
PUNTOS ESTIMADOS:	2	TÓPICO:	Usuario
TÍTULO:	Como usuario, puedo tener más de un solo		
DESCRIPCIÓN :			
El usuario podrá crear múltiples loops y visualizarlos en una lista			
TAREAS :			
Crear página de índice para listar loops. Añadir formulario para crear un nuevo loop. Añadir habilidad de redireccionar al nuevo loop una vez creado			

Usuario puede copiar un loop

Cada usuario podrá crear su propia versión de un loop específico; para esto cada usuario tendrá la posibilidad de navegar entre los listados de loops de otros usuarios, pero solo podrán modificarlos si hacen una copia propia. En la tabla 6 se resume esta historia.

Tabla 6. *Historia de Usuario: Como usuario puedo hacer una copia de un loop perteneciente a otro usuario*

HISTORIA DE USUARIO			
NÚMERO:	11935091	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Usuario
TÍTULO:	Como usuario puedo hacer una copia de un loop perteneciente a otro usuario		
DESCRIPCIÓN :			

Un usuario podrá hacer copia de un Loop perteneciente a otro usuario para realizar sus propios cambios
TAREAS :
Añadir habilidad de navegar en la lista de Loops de otros usuarios Agregar botón con texto ’Copiar a mi perfil’ Al hacer click se añadirá una nueva copia de ese Loop para el usuario que lo copió

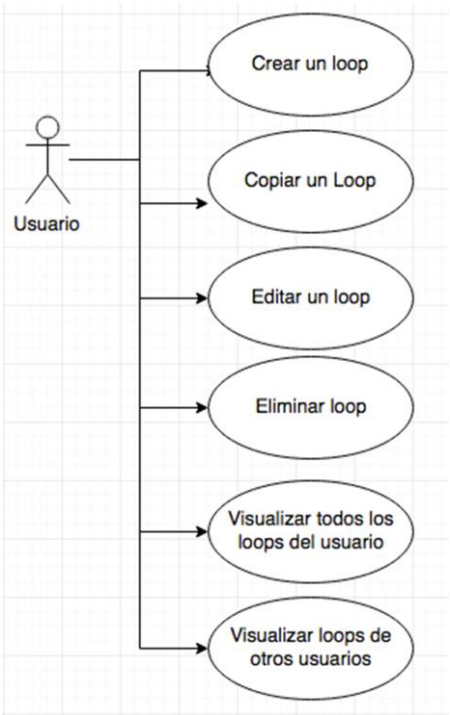


Figura 17. *Caso de Uso: Manejo de loops*

Agregar amigos

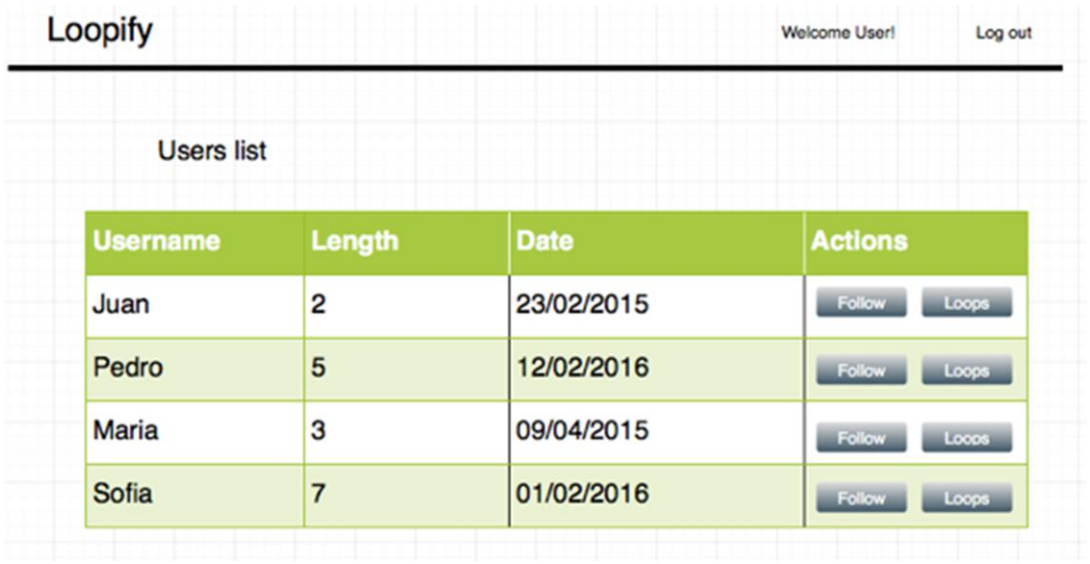


Figura 18. *Lista de usuarios en el sistema*

Para poder visualizar nuestros loops se creará la funcionalidad de agregar a otros usuarios como amigos, esto permitirá ver las actividades realizadas por los usuarios. En la tabla 7 se explica a mayor detalle la historia

Tabla 7. *Historia de Usuario: Como usuario, puedo agregar otros usuarios como amigos*

HISTORIA DE USUARIO			
NÚMERO:	12238111	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Usuario
TÍTULO:	Como usuario, puedo agregar otros usuarios como amigos		
DESCRIPCIÓN :			
El usuario podrá agregar amigos y enterarse de sus actividades			
TAREAS :			

Crear una lista de usuarios dentro de la aplicación Añadir botón ’Agregar como amigo’ alado de cada usuario Crear página de noticias con las actualizaciones de los amigos agregados

Muro de actualizaciones

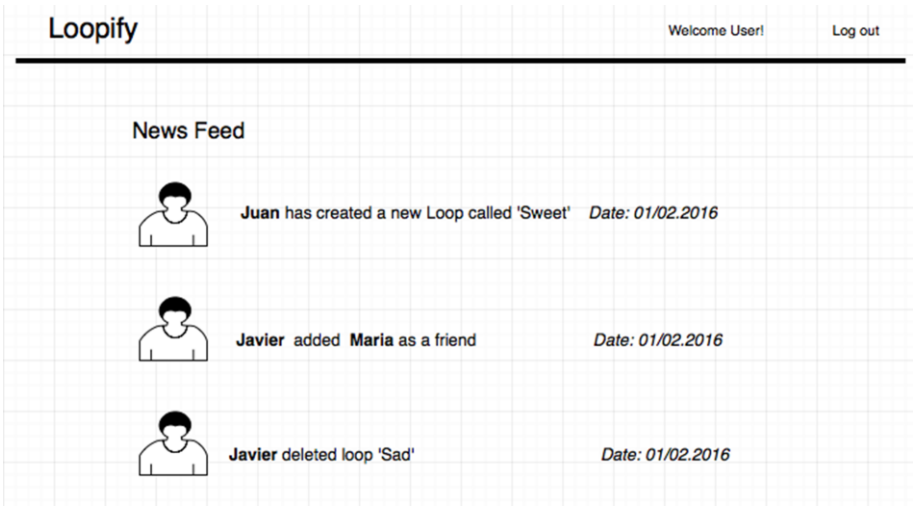


Figura 19. *Lista de noticias*

Las visualización de actualizaciones es una funcionalidad importante dentro de una ’red social’. Con el fin de ver estas actualizaciones se creará un muro de noticias donde se podrá ver a cada uno de nuestros amigos y sus respectivas actividades. En la tabla 8 se detalla los tipos de actualizaciones que tendrá la aplicación y en la figura 11 los casos de uso para las actualizaciones

Tabla 8. *Historia de Usuario: Como usuario puedo revisar mi muro de notificaciones*

HISTORIA DE USUARIO			
NÚMERO:	12245891	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Usuario
TÍTULO:	Como usuario puedo revisar mi muro de notificaciones		
DESCRIPCIÓN :			

El usuario recibirá actualización específicas de las actividades de sus
TAREAS :
Actualizar muro cuando amigos agregan otros amigos Actualizar muro cuando amigos crean otros loops Actualizar muro cuando amigos copian otros loops

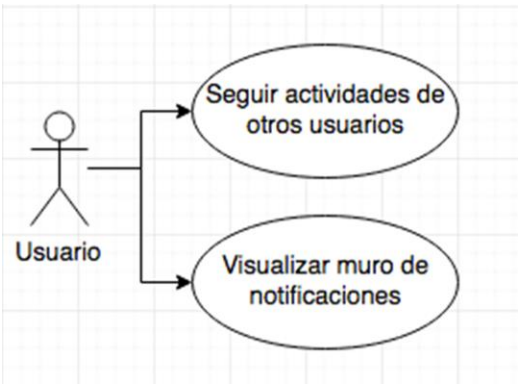


Figura 20. *Caso de Uso: Notificaciones de otros usuarios (amigos)*

3.1.2. Librería Edición Musical (Opal-Music)

El motor de la aplicación web será la librería Opal-Music, esta permitirá la creación y manipulación de notas y efectos musicales por medio de Web Audio Api. Web Audio Api es un librería creada por Google para la manipulación de sonidos en la web; funciona en cualquier navegador con HTML5. Esta librería solo funciona con el lenguaje de programación JavaScript. Para poder integrarlo junto con la aplicación se lo desarrollará con Opal. Opal es un lenguaje de programación que se encarga de compilar código Ruby a JavaScript.

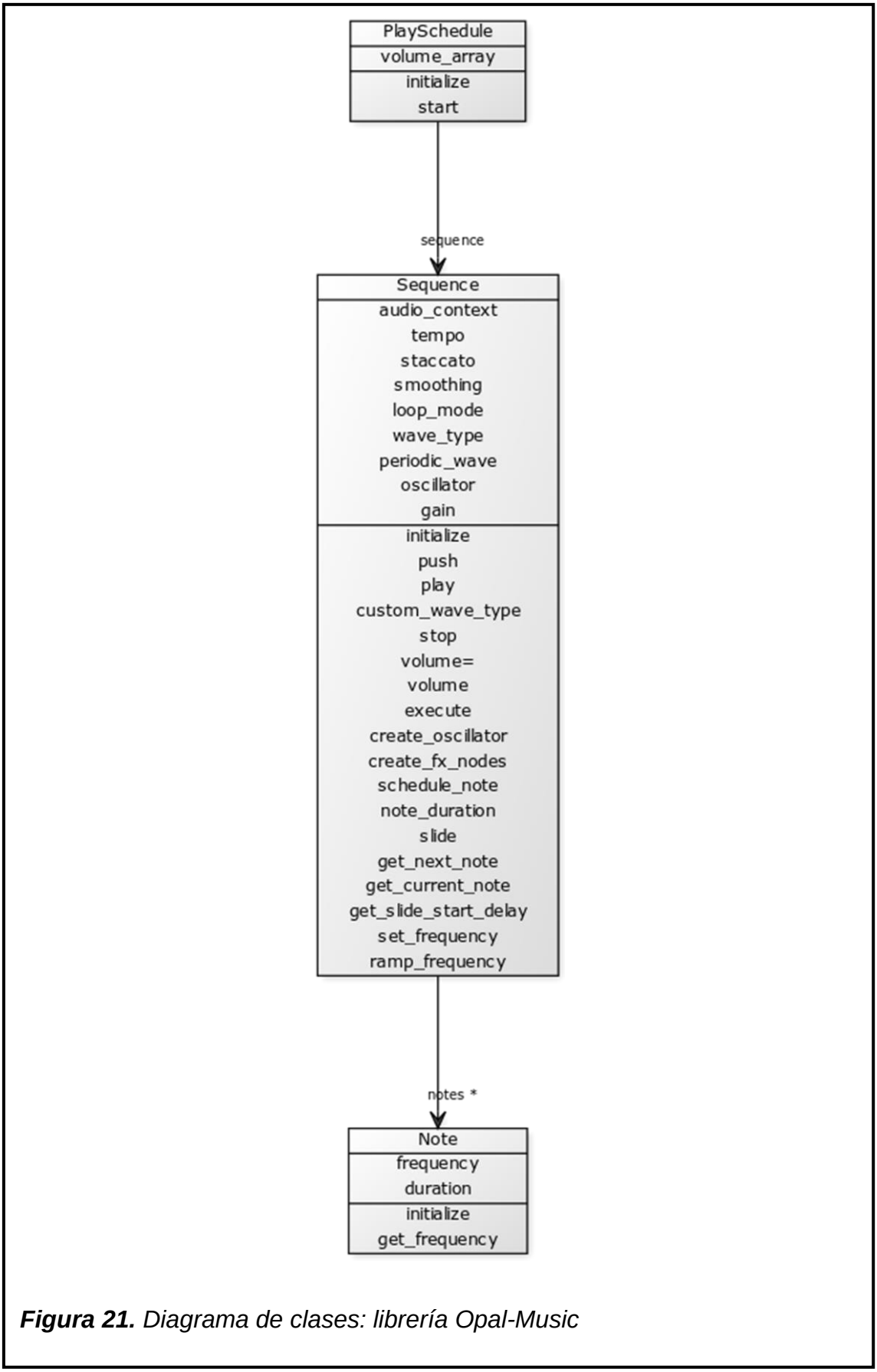


Figura 21. *Diagrama de clases: librería Opal-Music*

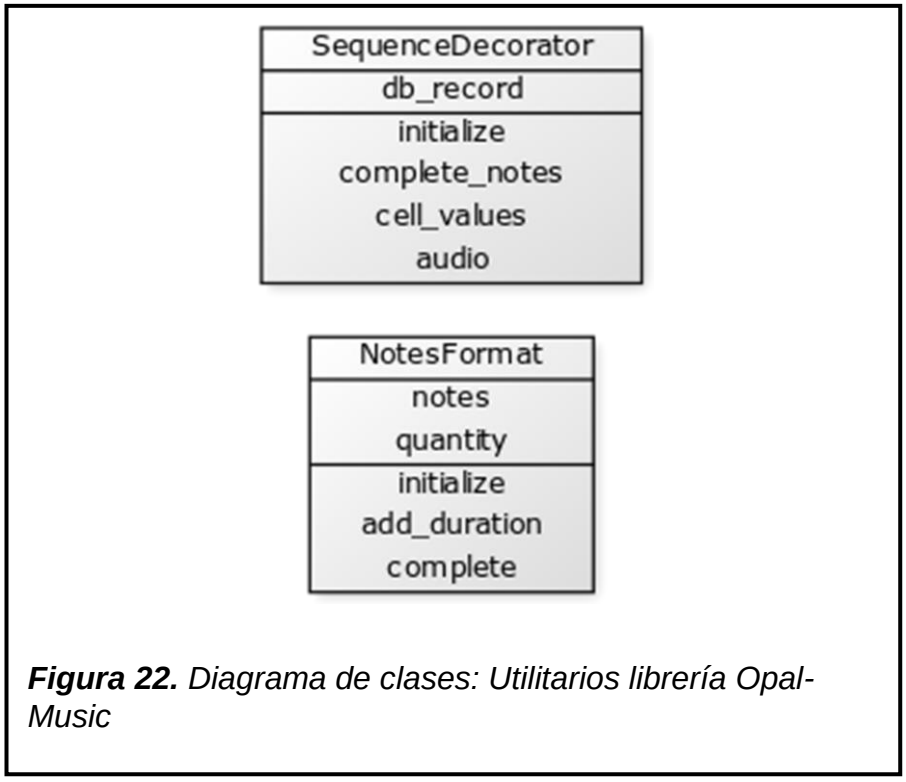


Figura 22. *Diagrama de clases: Utilitarios librería Opal-Music*

Notas musicales con Web Audio Api

Las notas se crearán por medio de la clase llamada 'Note', la cual se encargará de crear notas con un volumen y una frecuencia específica (el número de la frecuencia define la nota). En la tabla 9 se detalla esta historia

Tabla 9. *Historia de Usuario: Habilidad crear notas por medio de Web Audio API*

HISTORIA DE USUARIO			
NÚMERO:	328	TIPO:	Feature
PUNTOS ESTIMADOS:	3	TÓPICO:	Librería
TÍTULO:	Habilidad crear notas musicales por medio de la librería Web Audio API		
DESCRIPCIÓN :			
Esta librería podrá generar notas musicales por medio de osciladores configurados a una frecuencia específica (relacionado con historia: 108775342)			

TAREAS :
Creación de sonidos por medio del oscilador de Web Audio API
Especificar frecuencias para generar cada nota
Modificación de la duración de una nota
Modificación del volumen por medio de la amplitud de onda
Modificación efectos por medio del tipo de onda

Notas en secuencia

La clase ’Sequence’ podrá crear múltiples notas y secuenciarlas en un determinado periodo de tiempo. Se podrá configurar el volumen y efecto de cada secuencia. En la tabla 10 se detalla la forma en que se manejarán notas vacías y la forma en la que se secuenciarán las notas.

Tabla 10. *Historia de Usuario: Habilidad para reproducir notas en secuencia*

HISTORIA DE USUARIO			
NÚMERO:	426	TIPO:	Feature
PUNTOS ESTIMADOS:	2	TÓPICO:	Librería
TÍTULO:	Habilidad para reproducir notas en secuencia		
DESCRIPCIÓN :			
Esta librería permitirá la reproducción de varias notas en secuencia (relacionado con historia: 108775342)			
TAREAS :			
Programar una secuencia por medio de sus notas y duraciones Reproducir la siguiente nota al momento de terminarse la reproducción de la nota anterior			

Programación de secuencias

Con el fin de repetir y/o programar una secuencia a lo largo del tiempo se creará la clase ’Schedule’. Esta se encargará de guardar y reproducir una programación para una secuencia determinada. En la tabla 11 se detalla la historia creada para esta característica.

Tabla 11. *Historia de Usuario: Habilidad para programar la reproducción de varias secuencias*

HISTORIA DE USUARIO			
NÚMERO:	840	TIPO:	Feature
PUNTOS ESTIMADOS:	1	TÓPICO:	Librería
TÍTULO:	Habilidad para programar la reproducción de varias secuencias		
DESCRIPCIÓN :			
Esta librería permitirá la programación de una secuencia a través de un tiempo determinado (relacionado con historia 111056932)			
TAREAS :			
Crear un límite de duración En el límite provisto se repetirá la misma secuencia n veces o se omitirá su reproducción n veces en el límite de duración provisto			

3.1.3. Librería Decorador Web Audio Api (Opal-Audio)

Para integrar Web Audio con Opal-Music es necesario crear un decorador sobre el código Javascript del Api para poder usarlo con código Ruby, por medio de Opal. En la siguiente historia se detalla la creación de esta librería

Tabla 12. *Historia de Usuario: Habilidad para programar la reproducción de varias secuencias*

HISTORIA DE USUARIO			
NÚMERO:	843	TIPO:	Feature
PUNTOS ESTIMADOS:	4	TÓPICO:	Librería
TÍTULO:	Creación Librería Opal-Audio		
DESCRIPCIÓN :			
Esta librería permitirá usar Web Audio Api por medio de Opal. Debe crear un contexto de audio y generar los nodos de audio existentes en Web Audio. Los parámetros de cada nodo pueden generarse como AudioParam o como valores concretos. También generar los respectivos métodos para conectar y desconectar los nodos y otros métodos presentes en los distintos nodos			
TAREAS :			
Audio Context con generación de AudioNodes Invocación de métodos presentes en cada Nodo Creación de getters y setters de Nodos			

4. Capitulo IV. Desarrollo

4.1. Product Backlog

El product backlog tendrá todas las tareas destinadas tanto a la creación de las librerías como de la aplicación web. Para poder clasificar las tareas, estas se dividieron en tres historias de tipo Epic : Loopify-Editor, Loopify, Opal Music y Opal Audio**,**

A continuación se muestra el product backlog general del proyecto, con los epic, los tipos de historia y la estimación. Las historias de tipo Chore y Bug no contienen estimación ya que no aportaron nuevas funcionalidades al proyecto.

Tabla 13. *Product Backlog*

Historia	**Epic**	**Tipo**	**Estimación**
1. Crear clase Sequence	Opal-Music	Feature	3
2. Creación de Secuencias	Loopify Editor	Feature	1
3. Usar el mismo tempo en todas las secuencias del loop	Loopify Editor	Chore	
4. Actualizar versión de Volt en branch Local	Loopify	Chore	
5. Crear clase Note	Opal-Music	Feature	1
6. Como Usuario, puedo agregar Notas a mi Secuencia	Loopify Editor	Feature	2
7. Como Usuario, puedo crear un loop con múltiples secuencias	Loopify Editor	Feature	2
8. Habilitar reproducción infinita de secuencia (modo loop)	Opal-Music	Feature	1
9. Habilidad para programar una secuencia	Opal-Music	Feature	1

10. Programar las secuencias por medio de una matriz gráfica	Loopify Editor	Feature	3
11. Cambio de notas en el momento de la reproducción	Loopify Editor	Feature	3
12. Crear tests de unidad	Opal-Music	Chore	
13. Al agregar notas, se anula la secuencia	Opal-Music	Bug	
14. Valores por defecto en loops	Loopify Editor	Chore	
15. Como usuario, puedo tener uno o más loops	Loopify Paginas	Feature	1
16. Iniciar Creación Gema Opal Audio	Opal-Audio	Chore	
17. Eliminar Tick al enlazar secuencias	Loopify Editor	Bug	
18. Al cambiar el número de notas, deja de funcionar la reproducción del loop	Loopify Editor	Bug	
19. Crear modelos Volt en lugar de usar variables underscore	Loopify	Chore	
20. Configurar Travis CI para librerías y aplicación	Loopify	Chore	
21. Al hacer click en una secuencia, mostrar solo la secuencia editada	Loopify-Editor	Feature	1
22. Crear landing page	Loopify	Feature	1
23. Publicar Opal-Music Y Opal-Audio en Ruby Gems	Loopify	Chore	
24. Desplegar aplicación en Heroku	Loopify	Chore	
25. Configurar deploys	Loopify	Chore	

automáticos con Travis y Heroku			

4.2. Sprints

Se consideraron tres sprints para completar todo el proyecto, cada uno con una duración de un mes.

El primer Sprint ocurrió en el mes septiembre, consta de las historias enumeradas en la tabla 13: 1-8 . En estas historias consta el desarrollo inicial de loopify y el inicio de la creación de la librería Opal-Music

El segundo Sprint transcurrió en el mes de octubre, con las historias 9-17. Este sprint contiene tareas concernientes a la creación de Opal-Audio, la culminación de Opal-Music y la depuración del editor de Loopify.

El tercer sprint, culminado en el mes noviembre, contiene el resto de historias. En su mayoría contienen historias concernientes a otras páginas de loopify, depuración de errores y configuración de servidores, tanto de despliegue como de integración continua.

4.3. Librerías

Opal-Audio Y Opal-Music Fueron creadas como gemas de Ruby con el objetivo de reutilizarlas en otros proyectos.

Opal Music consta con la siguiente estructura de archivos:

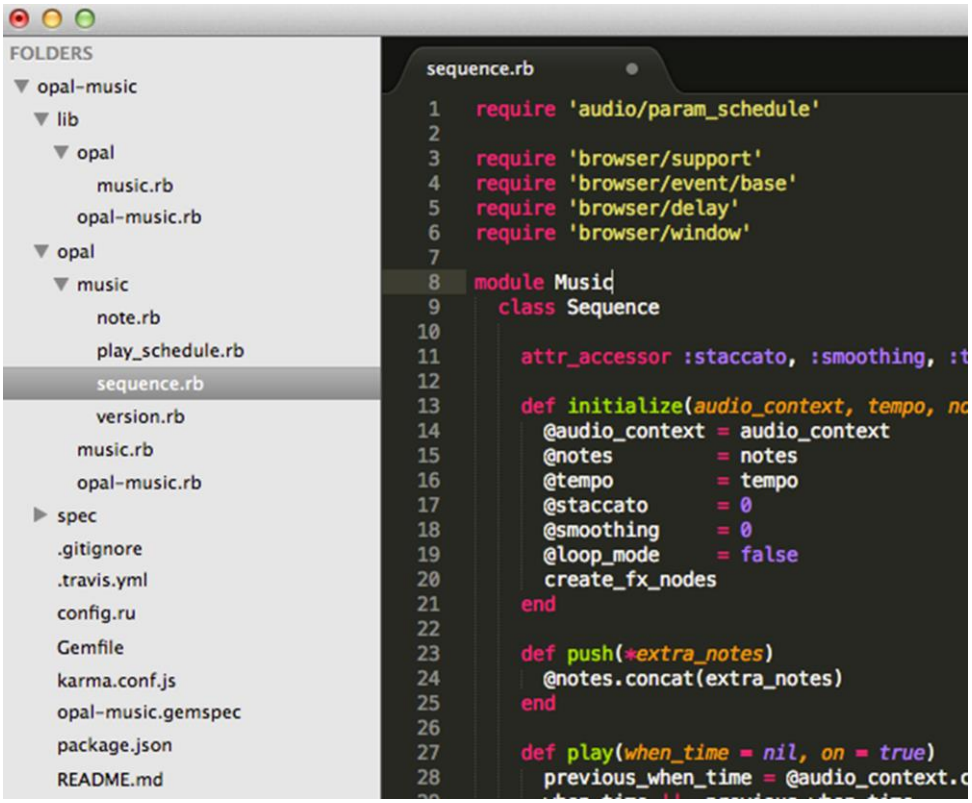


Figura 23. *Estructura de Archivos, Opal-Music*

Las librerías tienen una estructura similar a una gema de Ruby. La diferencia entre una gema de Opal y una gema normal es que la primera se carga en el front-end, por lo que los archivos no son requeridos de la misma forma (estos se cargan junto con el servidor de Opal). Opal realiza la carga de estos archivos por medio de Sprockets.

En el caso de nuestras librerías ubicamos todos los archivos Opal dentro de la carpeta Opal y las requerimos con Sprockets, dentro de los archivos de configuración ubicados en la carpeta lib.

Entre otros archivos tenemos los tests , archivos de configuración de la gema y el archivo de configuración del servidor de integración continua.

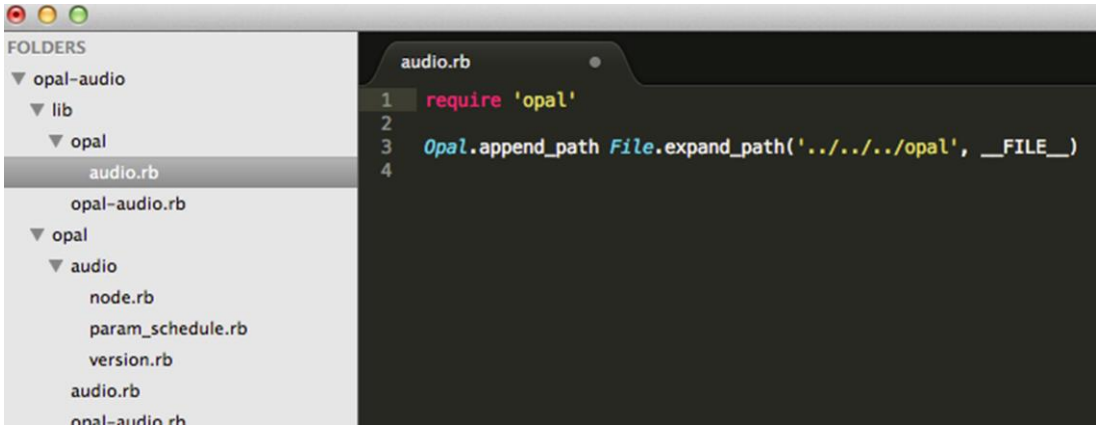


Figura 24. *Especificación carga de archivos Opal, Opal-Audio*

A pesar de que las librerías tienen pruebas automatizadas fue necesario realizar pruebas manuales, esto para comprobar que los sonidos eran generados por la librería. Las pruebas manuales fueron realizadas por medio de opal-irb, un REPL (Read eval print loop) que corre en el navegador.

Figura 25. *Prueba de Web Audio Api con Opal-irb*

En un principio muchas de las características de Opal-Audio formaban parte de Opal-Music. Se decidió separar ambas con el objetivo de tener más clara las funcionalidades a implementar en Opal-Music

Opal-Audio en gran parte es una librería que llama directamente a métodos Javascript, siendo estos parte de Web Audio Api. Opal realiza estos llamados por medio de métodos que permiten indicar que atributos nativos podemos llamar como código Ruby.

4.2. Aplicación

La aplicación consume directamente las funcionalidades dentro de Opal-Music para crear los controles dentro del editor de Loops.

La estructura de la aplicación está divida en Volt-apps, los cuales son subaplicaciones dentro de un proyecto Volt. Cada una de estas aplicaciones contienen sus respectivos modelos, vistas y controladores.

El proyecto contiene las aplicaciones Main y Music. Music contiene todo lo referente al editor de Audio y Main implementa el resto de funcionalidades de la aplicación como login, exploración de loops, copia de loops, etc. En la siguiente figura podemos ver la estructura de archivos del proyecto.

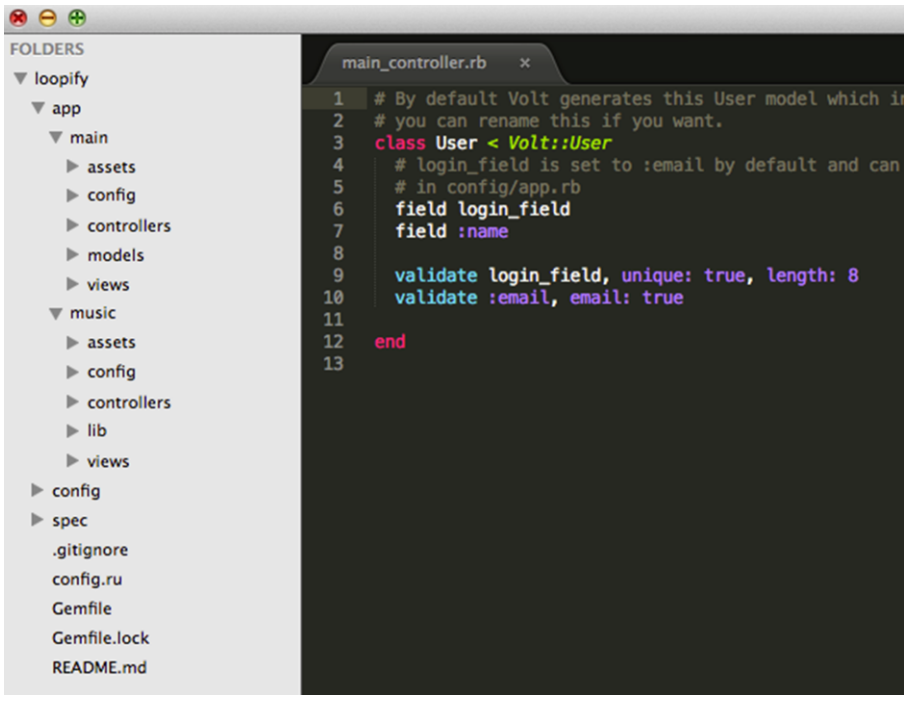


Figura 26. *Estructura de carpetas, loopify*

Opal-Music se usa dentro de Loopify por medio del controlador. El controlador en Volt se ejecuta directamente en el browser por lo que podemos hacer llamadas a Web Audio Api desde el mismo.

main_controller.rb | sequences_controller.rb | sequence_decorator.rb

```ruby
require 'opal-music'

require 'music/lib/sequence_decorator'

module Music
  class SequencesController < Volt::ModelController
    attr_reader :audio_sequence

    def controls

    end

    def play(audio_context, tempo)
      @audio_sequence = sequence_decorator.audio(audio_context, tempo.to_i)
      @audio_sequence.loop_mode = true
      @audio_sequence.play
    end

    def stop
      @audio_sequence.stop
    end

    def set_volume
      @audio_sequence.volume = _volume.to_i / 100 / 2
    end
```

Figura 27. *Controlador de Secuencias, loopify*

Otra ventaja de los controladores de Volt es que podemos enlazar los controles de audio con la persistencia de estos datos en la base. Gracias a la reactividad en las vistas los datos cambiados en Web Audio Api se ven también reflejados en el modelo.

5. Capítulo V. Integración Continua

5.1. Pruebas Unitarias

Las pruebas unitarias fueron creadas tanto para las librerías opal-audio como opal-music. Estas librerías corren enteramente en el navegador gracias al compilador Opal, por lo que las pruebas tienen que correr en Javascript. Localmente se las corrió en un servidor local, por medio de la librería Sprockets/Rack, la cual se encarga de cargar los archivos Javascript en un servidor web hecho en Ruby.

Para crear los tests se usó la librería opal-rspec, la cual usa la misma sintaxis de la librería Rspec original.

```
require 'opal-audio'

describe Audio::Node do
  #NOTE: let! doesn't work
  audio_context = Audio::Context.new

  describe 'reader methods' do
    let(:oscillator) { audio_context.oscillator }
    let(:frequency) { oscillator.frequency }

    it 'returns a raw value' do
      expect(frequency).to be_a Number
    end

    context 'for audioparam values' do
      it 'returns audio params for especial values' do
        expect(oscillator.frequency false).to be_a Audio::ParamSchedule
      end

      it 'returns the same raw value' do
        expect(oscillator.frequency(false).to_f).to eq frequency
      end
    end
  end

```

Figura 28. *Sintaxis Opal-RSpec, opal-audio*

En la librería opal-music se crearon pruebas para verificar la reproducción de notas, modo loop y reproducción de un conjunto de notas en modo secuencia. En la figura 29 podemos observar el conjunto de tests que forman parte de esta librería.

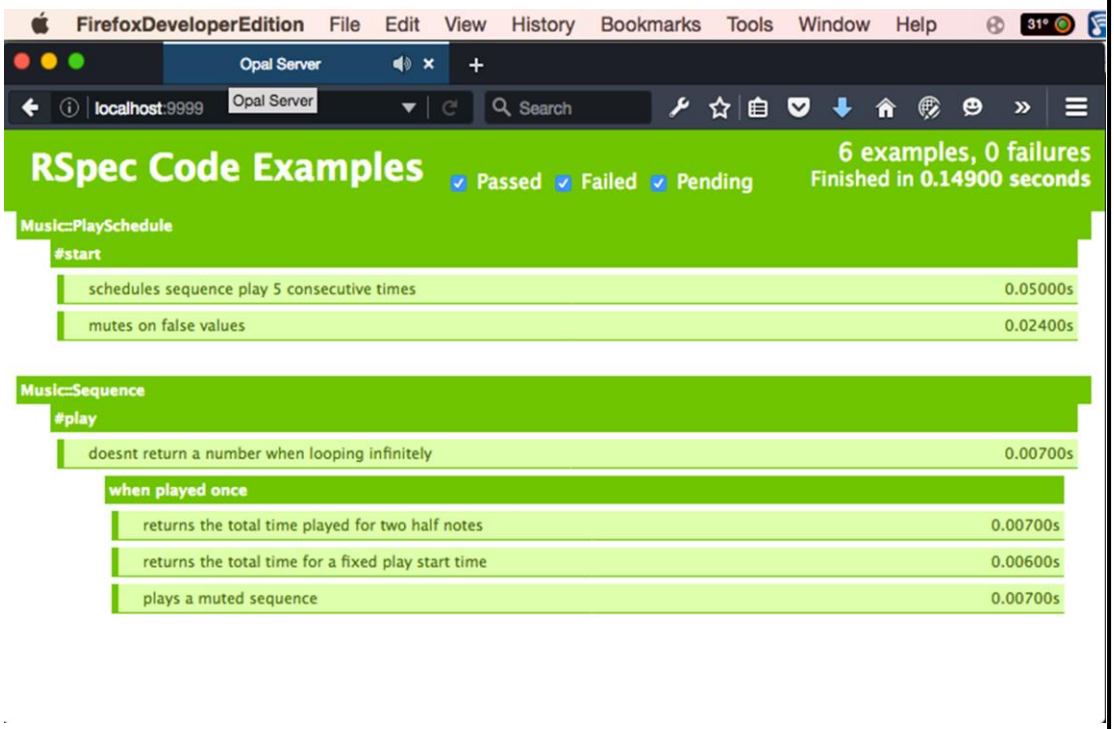


Figura 29. *Test suite librería opal-music*

La librería opal-audio es un decorador en opal sobre Web audio Api por lo cual no se añadieron nuevas funcionalidades, pero si contiene pruebas unitarias que verifican la creación de los respectivos métodos de este api.

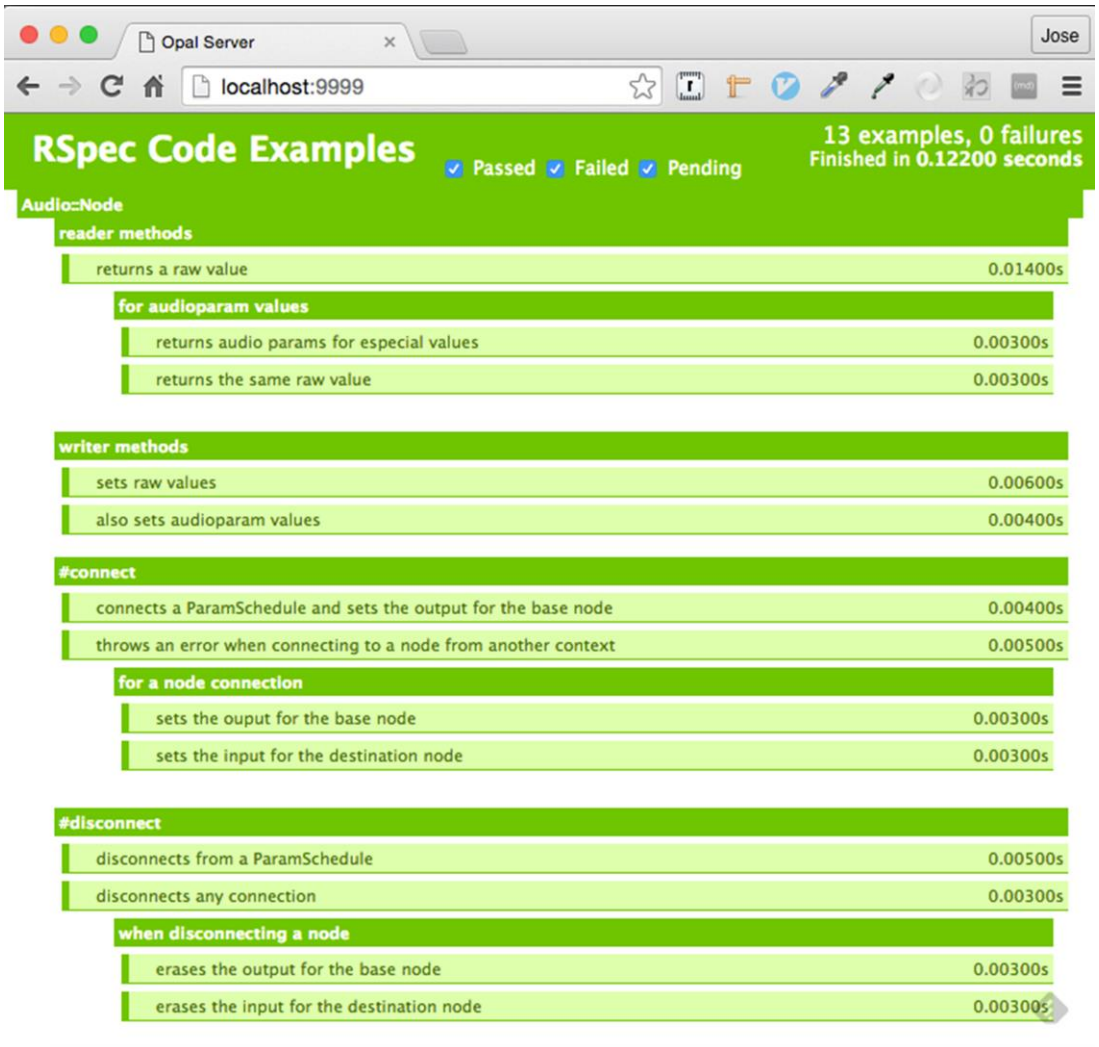
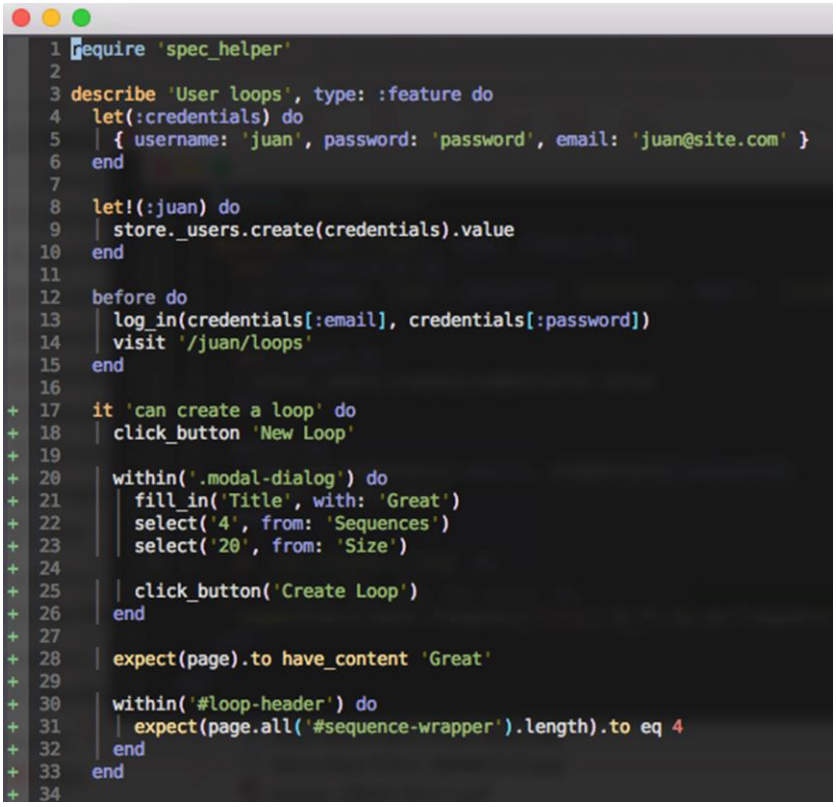


Figura 30. *Test suite librería opal-audio*

5.2. Pruebas de Integración

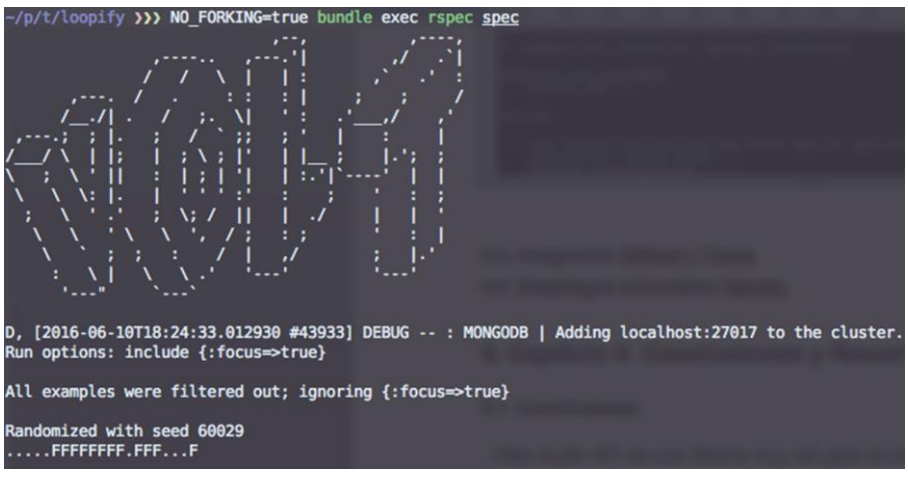
Para la aplicación web loopify se crearon, en su mayoría, pruebas de integración. Estas pruebas nos ayudan a verificar de manera general como funciona toda nuestra aplicación. Las pruebas realizan clicks en páginas, llenado formularios, etc. Esto lo realiza gracias a la librería Capybara, la cual se usa en conjunto con Rspec para crear pruebas que ejecutan pasos sobre el navegador.



```
require 'spec_helper'

describe 'User loops', type: :feature do
  let(:credentials) do
    { username: 'juan', password: 'password', email: 'juan@site.com' }
  end

  let!(:juan) do
    store._users.create(credentials).value
  end

  before do
    log_in(credentials[:email], credentials[:password])
    visit '/juan/loops'
  end

+ it 'can create a loop' do
+   click_button 'New Loop'
+
+   within('.modal-dialog') do
+     fill_in('Title', with: 'Great')
+     select('4', from: 'Sequences')
+     select('20', from: 'Size')
+
+     click_button('Create Loop')
+   end
+
+   expect(page).to have_content 'Great'
+
+   within('#loop-header') do
+     expect(page.all('#sequence-wrapper').length).to eq 4
+   end
+ end
+
```

Figura 31. *Prueba de Integración: Creación de loop*

Al contrario que con las librerías, las pruebas de la aplicación se pueden correr directamente en la terminal. Al correr las pruebas Volt arranca el servidor automáticamente para correrlas dentro de la consola.



```
~/p/t/loopify ››› NO_FORKING=true bundle exec rspec spec

D, [2016-06-10T18:24:33.012930 #43933] DEBUG -- : MONGODB | Adding localhost:27017 to the cluster.
Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}

Randomized with seed 60029
.....FFFFFFFF.FFF...F
```

Figura 32. *Arranque de pruebas en aplicación*

5.3. Integración Github y Travis

La aplicación y la librería tienen repositorios en el servidor Github. Gracias a que se cuenta con tests se puede subir nuevos cambios de manera segura a los repositorios. Para correr los tests se usará el servidor de integración continua Travis.

El servidor de CI solo puede visualizar las pruebas dentro de una máquina virtual, por lo que se necesita correr y obtener los resultados de los tests dentro de la terminal.

Para correr pruebas de Javascript y obtener los resultados se usó la librería Karma. Esta se encarga de ejecutar los resultados en el navegador y obtenerlos en la terminal para su análisis.

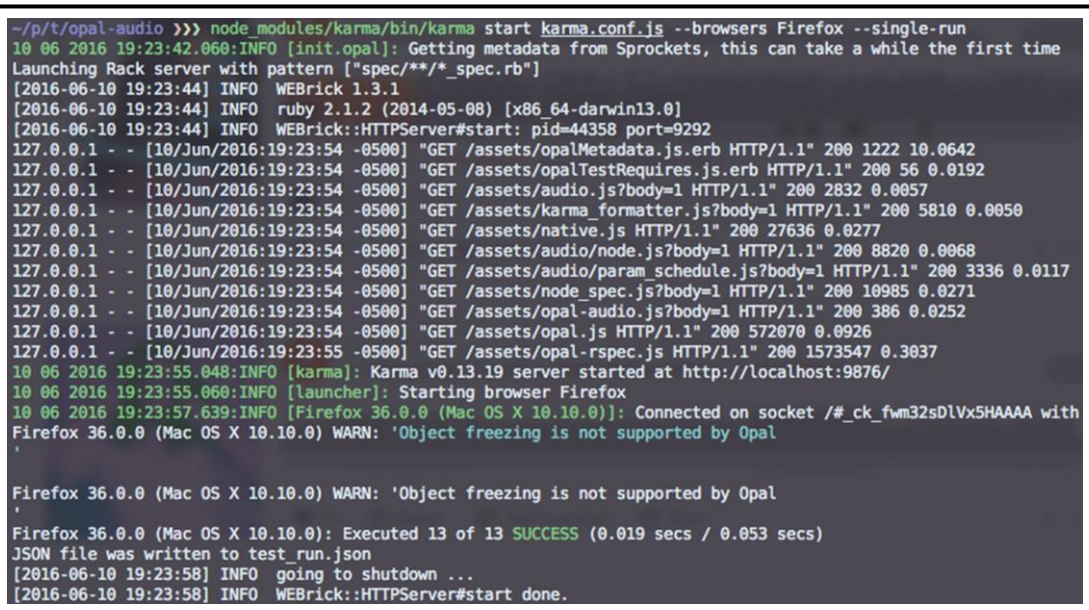


Figura 33. *Ejecución de pruebas con karma*

Ahora se puede correr las pruebas dentro de Travis automáticamente al realizar cambios, para esto se enlaza la cuenta del usuario dueño del repositorio dentro de Travis y se especifica el repositorio a ser usado. El repositorio debe tener un archivo de configuración '.travis' con los datos necesarios para poder correr el

repositorio. En el archivo de configuración podemos especificar la versión de Ruby, especificar comandos node.js para correr pruebas de Javascript, entre otras opciones.

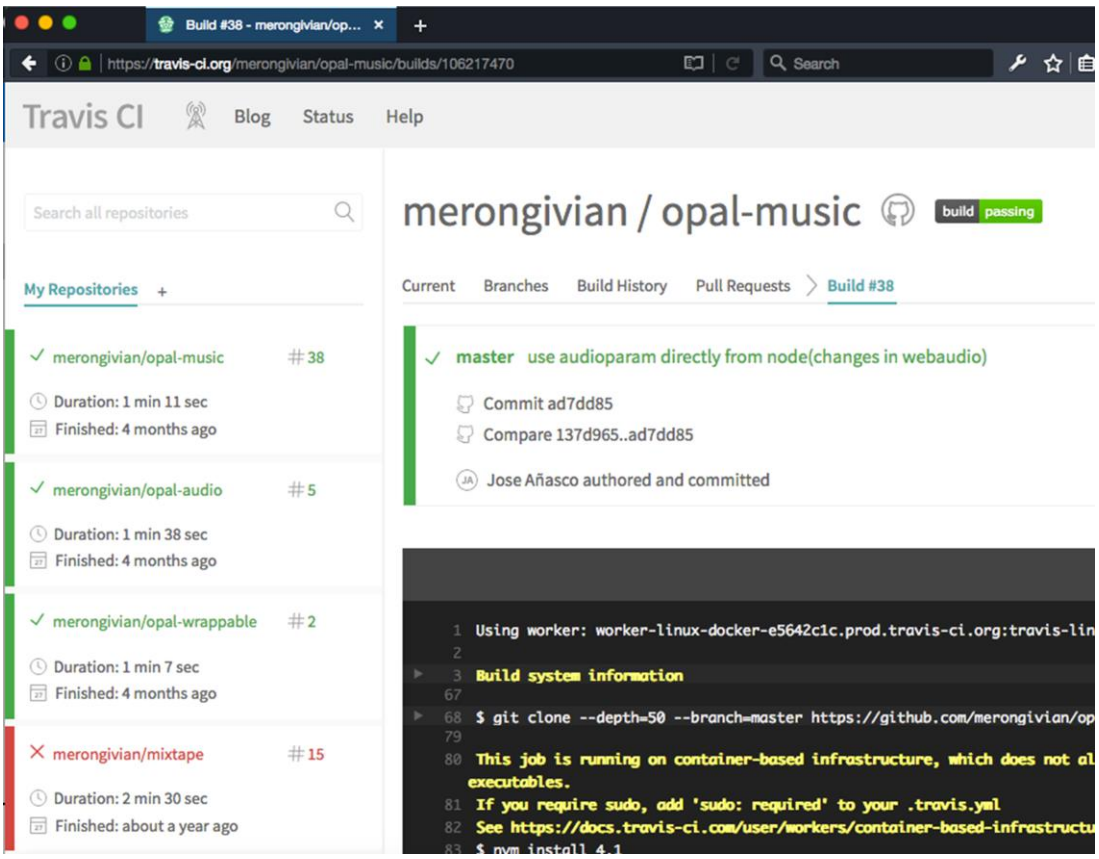


Figura 34. *Ejecución de pruebas en Travis CI*

5.4. Despliegue automático con Heroku

Una vez configurado travis se puede desplegar la aplicación automáticamente en cada build exitoso, esto se logra configurando el archivo '.travis' y añadiendo el key provisto por Heroku.

6. Capítulo VI. Pruebas de Rendimiento

El aplicativo será usado como una red social, por lo que tendrá que soportar una considerable carga de usuarios. Para empezar se escogieron las características mínimas para el servidor en Heroku y dependiendo del crecimiento de los usuarios se aumentará la cantidad de recursos necesarios. Esta decisión se tomará en base a métricas de desempeño. Las métricas nos mostrarán el número de cargas por segundo y su impacto en el rendimiento.

6.1. Loader.io

Las pruebas serán realizadas por medio de Loader.io. Este es un servicio en línea que genera cargas al servidor de la aplicación con el objetivo de medir la respuesta del sistema a condiciones adversas, como la cantidad excesiva de usuarios.

La instalación y uso de este servicio se lo realiza a través de Heroku. El servidor de la aplicación, al ser un servidor más restrictivo, no permite realizar pruebas directas con aplicaciones como JMeter; estas son bloqueadas inmediatamente por los servidores de Heroku al ser consideradas como un ataque.

Una de las ventajas de este servicio es el despliegue de resultados por medio de reportes, con gráficas que muestran los resultados a través del tiempo.

Para poder comenzar con las pruebas primero se debe proceder verificando que se tiene acceso al servidor de la aplicación a ser probada, esto se lo hace creando una nueva ruta en la aplicación, con un código proporcionado por Loader.io. Una vez comprobado el acceso se puede comenzar con la creación de los escenarios de pruebas, los cuales pueden ser ejecutados inmediatamente o calendarizados para una posterior ejecución.

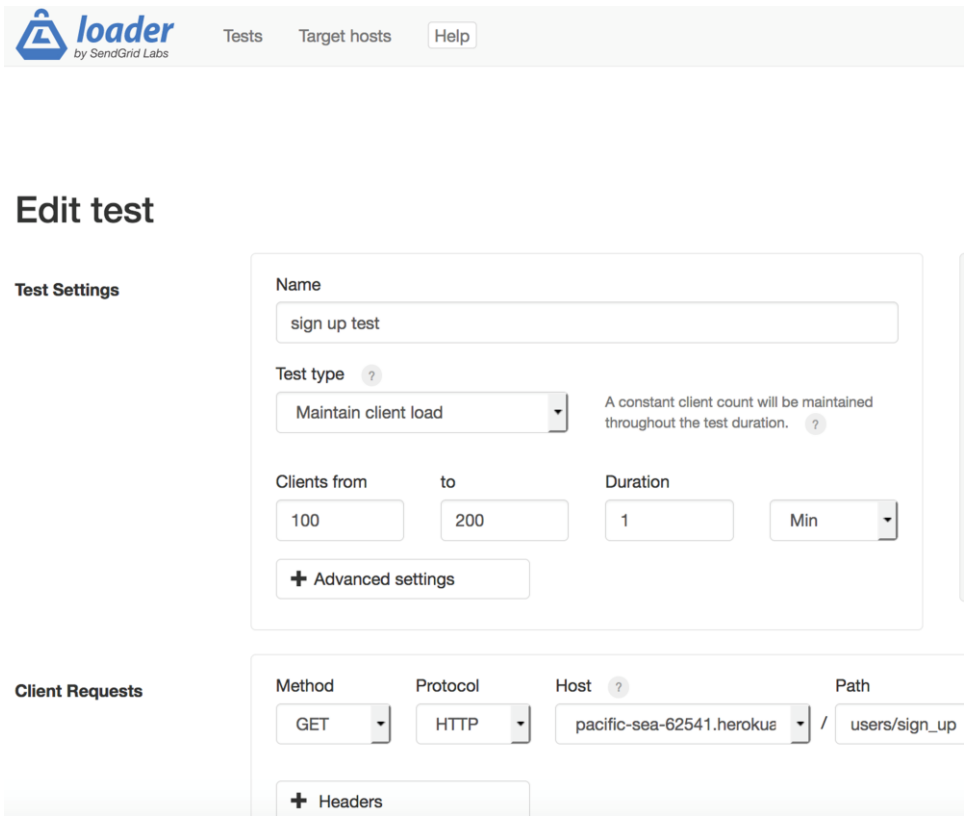


Figura 35. *Creación de pruebas de carga*

6.2. Pruebas de Carga

Existen varias formas de realizar las pruebas de carga: A lo largo de un lapso de tiempo se manda una carga constante de usuarios por segundo, un número total de usuarios distribuidos a lo largo de la duración de la prueba, o una carga constante de usuarios en un rango determinado.

Según las pruebas realizadas se pudo constatar que el máximo número de conexiones soportadas por un servidor standard de Heroku, con 512mb de ram es de 150 a 200 usuarios por segundo.

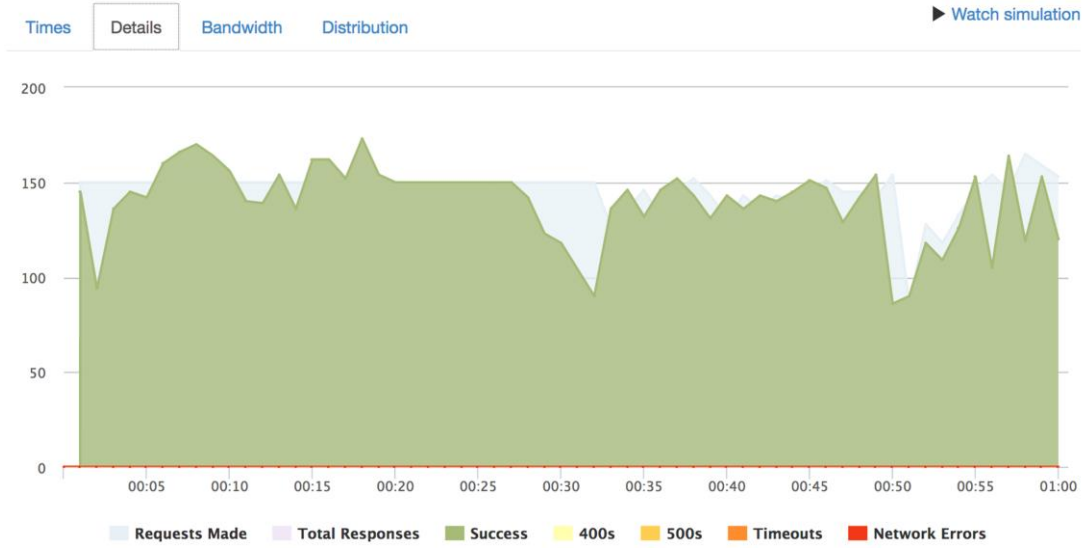


Figura 36. *Datos de carga en 150 usuarios/sec*

La máxima cantidad de usuarios soportados por minuto fue de 8000 usuarios.

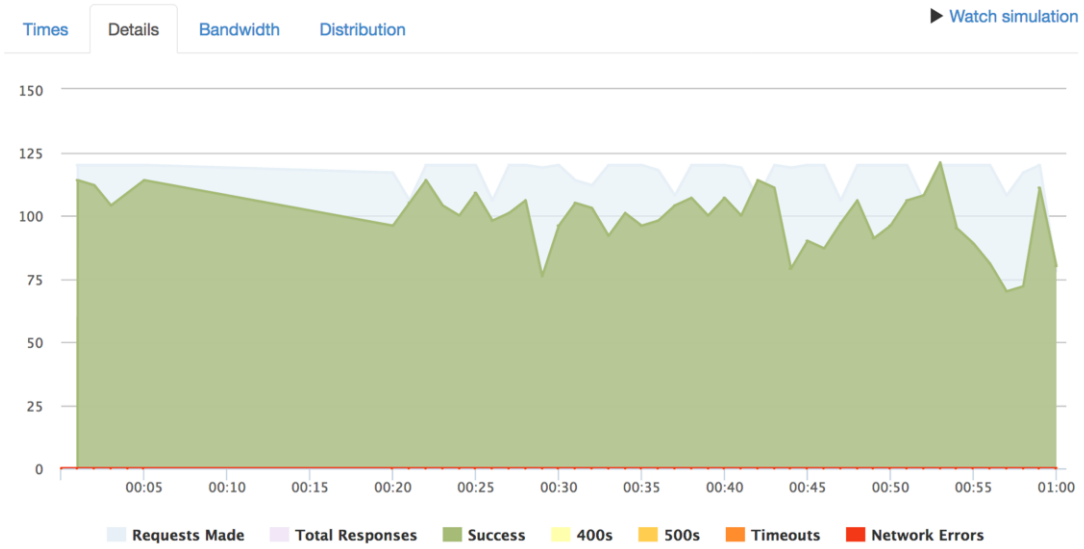


Figura 37. *Datos de carga en 8000 usuarios/min*

Pasado el umbral de los limites antes mencionados provoca una caída de conexión en los usuarios, dado que el servidor mantienen conexiones abiertas con otros clientes antes de aceptar nuevos usuarios.

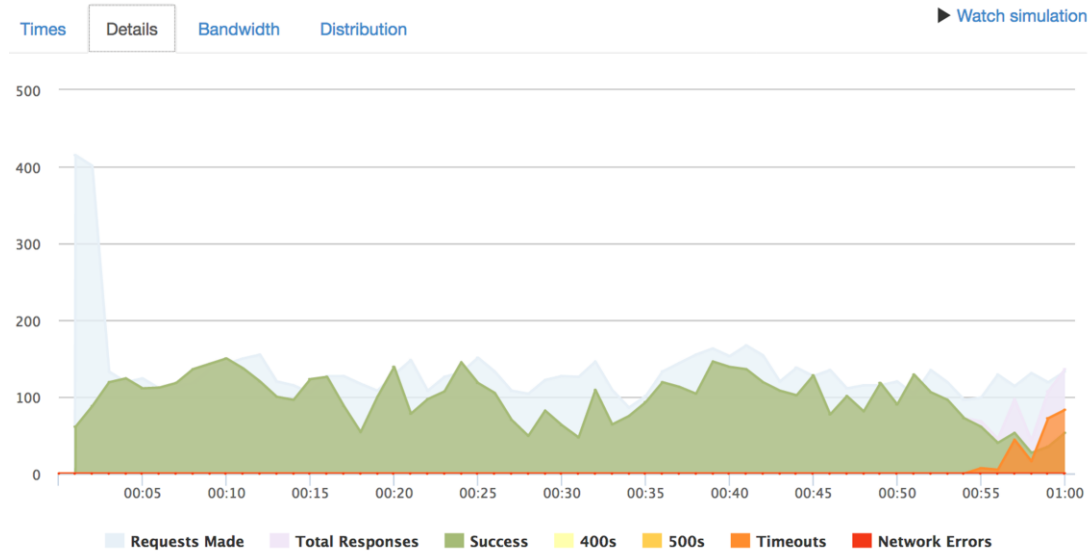


Figura 38. *Datos de carga en 400 usuarios/sec*

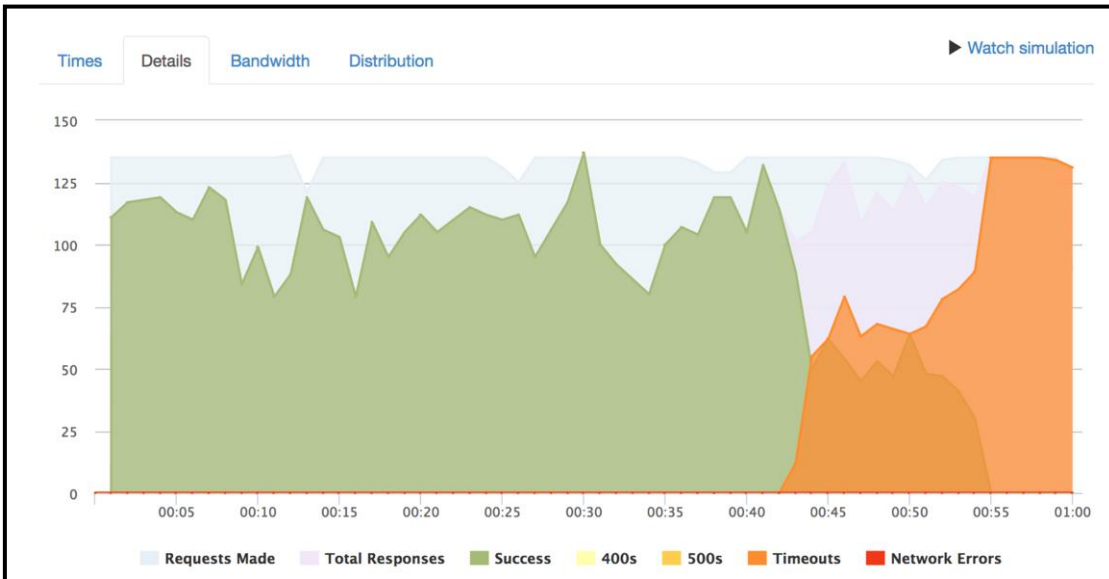


Figura 39. *Datos de carga en 9000 usuarios/min*

5. Conclusiones y Recomendaciones

5.1. Conclusiones

Web Audio API es una librería muy útil para la creación de aplicaciones web con contenido sonoro. Gracias a que forma parte de HTML5, Web Audio Api permite manipular sonidos en el navegador e incluso enlazarlos con otros API's.

El uso de una arquitectura MVVM fue beneficioso para la implementación de características de 'tiempo real'. La aplicación cuenta con funcionalidades que dependen en gran medida en la actualización inmediata de datos. Al usar MVVM la implementación de estas funcionalidades fue relativamente sencilla en comparación con otros frameworks (Angular, React) los cuales requerían el uso extenso de Javascript y la repetición de lógica usada en el back-end

El uso y creación de librerías de código abierto beneficia a la comunidad. Este proyecto en su totalidad fue creado con librerías de código abierto y se crearon librerías específicas para su uso en conjunto con la aplicación. Esto tiene, como objetivo principal, el obtener la retroalimentación de otros desarrolladores y la mejora del código.

5.2. Recomendaciones

Usar TDD o el desarrollo guiado por pruebas para mejorar el proceso de desarrollo en proyectos de Software. El desarrollo guiado por pruebas no solo ayuda a identificar posibles bugs, también mejora el proceso en general: desde la calidad del código hasta en el mantenimiento

Desplegar la aplicación en conjunto con integración continua para verificar que nuestra aplicación funcione correctamente. El crear tests automatizados de nuestro código nos permite asegurarnos que nuestra aplicación siga funcionando, incluso al realizar cambios o incluir nuevas funcionalidades.

Identificar un framework o marco de desarrollo acorde a las necesidades de nuestra aplicación. Las funcionalidades de la aplicación creada dependen, en gran parte, en la actualización de datos en tiempo real. El escoger Volt como framework permitió implementar estas funcionalidades de manera más fácil.

6. Referencias

Agile Manifesto. (s.f.). Recuperado el 2016, de http://agilemanifesto.org/

Centric Consulting. (s.f.). Recuperado el 22 de Julio de 2016, de http://centricconsulting.com/case-studies/agile-test-driven-development/

Chacon, S., & Straub, B. (2015). Pro Git. Mountain View, California, United States.

Danzeria. (s.f.). Recuperado el 23 de Junio de 2016, de http://danzeria.com/2016/03/06/lo-basico-que-necesitas-saber-sobre-los-sintetizadores/

Flanagan, D., & Matsumoto, Y. (2008). *The Ruby Programming Language* (1ra Edición ed.). (M. Loukides, Ed.) Sebastopol, California, Estados Unidos: O'Reilly.

Hartl, M. (2013). *Ruby on Rails Tutorial* (2nd edition ed.). Upper Saddle River, New Jersey: Addison-Wesley

Middleton, N., & Schneeman, R. (2013). *Heroku: Up and Running.* Sebastopol, California, United States.

Mozilla Developer Network. (s.f.). *Mozilla Developer Network*. Recuperado el 13 de Agosto de 2016, de https://developer.mozilla.org/es/docs/Web_Audio_API

omarsdiscodojo. (s.f.). Recuperado el 22 de Julio de 2016, de https://omarsdiscodojo.wordpress.com/2012/05/04/synthesis-basics-basic-wave-types-13/

Rice, D. (s.f.). Recuperado el 21 de Julio de 2016, de https://www.go.cd/2016/03/15/are-you-ready-for-continuous-delivery-part-2-feedback-loops/

Scrum Alliance. (s.f.). Recuperado el 20 de Julio de 2016, de https://www.scrumalliance.org/why-scrum

Silva, S. (26 de Abril de 2015). *C-Sharpcorner*. Recuperado el 24 de Julio de 2016, de http://www.c-sharpcorner.com/UploadFile/3789b7/xamarin-guide-9-use-mvvm-pattern/

Smus, B. (14 de Octubre de 2011). *HTML 5 Rocks*. Recuperado el 12 de Agosto de 2016, de http://www.html5rocks.com/en/tutorials/webaudio/intro/

Stout, R. (20 de Noviembre de 2014). *SlideShare*. Recuperado el 25 de Julio de 2016, de http://es.slideshare.net/ryanstout/isomorphic-app-development-with-ruby-and-volt-rubyconf2014

Taylor, J. (28 de Octubre de 2015). *CodeSchool*. Recuperado el 21 de Junio de 2016, de https://www.codeschool.com/blog/2015/10/28/conjuring-nosql-with-mongodb/

Tracker Base. (s.f.). Recuperado el 12 de Mayo de 2016, de http://trackerbase.blogspot.com/2014/01/ahx.html

Volt Framework. (s.f.). Recuperado el 22 de Junio de 2016, de http://voltframework.com/docs

Wells, D. (s.f.). *Extreme Programming*. Recuperado el 22 de Julio de 2016, de http://www.extremeprogramming.org/map/loops.html

ANEXOS

GUIA DE DESPLIEGUE LOOPIFY

Autor: José Añasco

2016

Requerimientos de Hardware y Software

Para poder desplegar la aplicación con el siguiente manual se requiere un servidor con las siguientes características

- Sistema operativo basado en Unix (OSX o Ubuntu) con memoria ram de 4(GB) y procesador de Intel Core 3
- Ruby versión 2.2 (mínimo)
- Base de Datos Mongo 3.0.11
- Git 2.8.2
- Volt versión de desarrollo (tomado de Github)
- Web Server Nginx (para usarlo en un ambiente de producción)

PASOS INSTALACIÓN GIT

1. **Ejecutar el siguiente comando para instalar la versión básica de git**
   sudo apt-get install git
2. **En el caso de que se requiere usar git como repositorio de despliegue, ejecutar el siguiente comando:**
   sudo apt-get install git-core

PASOS INSTALACIÓN LENGUAJE RUBY

1. **Usar el siguiente comando, el cual contiene utilitarios tanto para el funcionamiento de Ruby como para el despliegue en producción**
   sudo apt-get install build-essential bison openssl libreadline6 libreadline6-dev libffi-dev curl zlib1g zlib1g-dev libssl-dev libyaml-devlibxml2-dev libxslt-dev autoconf libc6-dev nodejs -y
2. **Instalar los siguientes paquetes: libpq y libffi**
   sudo apt-get install libpq-dev libffi-dev

3. **Usar el paquete imagemagick para el manejo de imágenes**

   sudo apt-get install imagemagick --fix-missing -y

4. **Descargar rbenv para el manejo de versiones de Ruby, mediante el siguiente comando del utilitario CURL**

   curl

   https://raw.githubusercontent.com/fesplugas/rbenv-

   installer/master/bin/rbenv-installer | bash

5. **Añadir el siguiente código dentro del archivo ~/.bashrc para la ejecución de rbenv**

   ```
   export RBENV_ROOT="${HOME}/.rbenv"
   if [ -d "${RBENV_ROOT}" ]; then
   export PATH="${RBENV_ROOT}/bin:${PATH}"
   eval "$(rbenv init -)"
   fi
   ```

6. **Instalar versión 2.2 de Ruby por medio de rbenv**

   rbenv install 2.2.0

7. **Configurar la versión global de Ruby**

   rbenv global 2.2.0

8. **Instalar bundler para el manejo de dependencias**

   sudo gem install bundler

PASOS INSTALACIÓN FRAMEWORK VOLT

1. **Instalar gema specific_install**

   gem install specific_install

2. **Instalar la versión de desarrollo de Volt (desde el repositorio) mediante el siguiente comando**

   gem specific_install –l

PASOS INSTALACIÓN BASE DE DATOS MONGO

1. **Para instalar la versión 3.0.11 de mongo correr el siguiente comando:**
   sudo apt-get install -y mongodb-org=3.0.11 mongodb-org-server=3.0.11 mongodb-org-shell=3.0.11 mongodb-org-mongos=3.0.11 mongodb-org-tools=3.0.11

2. **Ejecutar servicio Mongo**
   sudo service mongod start

PASOS DESCARGA CODIGO FUENTE

1. **Una ves instalado git descargar el código fuente público con el siguiente comando (dentro del path en el que se quiera guardar):**
   https://github.com/merongivian/loopify.git

PASOS EJECUCION AMBIENTE DE DESARROLLO

1. **Dentro del directorio en el que se descargo la aplicación ejecutar el siguiente comando para instalar las dependencias**
   bundle install
2. **Ejecutar la aplicación**
   Bundle exec volt server
3. **Abrir la aplicación en el siguiente url**
   Localhost:3000

PASOS EJECUCION AMBIENTE DE PRODUCCION

1. **Instalar Node JS**
   sudo apt-get install nodejs
2. **Instalar nginx**

sudo apt-get install nginx

3. **Abrir el siguiente archivo**

   /etc/nginx/nginx.conf

4. **Seguir las instrucciones del siguiente url**

   http://docs.voltframework.com/en/deployment/nginx.html

5. **Ejecutar servicio de Nginx**

   sudo service nginx start

6. **La aplicación es ahora visible en el puerto 80 del servidor de producción**

MANUAL DE USUARIO LOOPIFY

Autor: José Añasco

2016

INTRODUCCIÓN

Que es loopify

Loopify es una aplicación web que permite la generación de sonidos por medio de loops, a su vez permite compartir estos loops con otros usuarios, permitiéndole crear sus propias versiones. Contiene las siguientes características:

- Edición de loops

  Permite la creación de loops y la visualización/guardado de datos de manera automática. Cuenta también con un listado de loops

- Explorar Usuarios

  Se pueden navegar entre usuarios y sus loops. Mediante los botones de las acciones podemos seguir las actividades de otros usuarios, así como copiar loops pertenecientes a otros usuarios

- Muro de Noticias

  La aplicación generará automáticamente un listado de noticias de todos los usuarios a los cuales seguimos

FLUJO DE LA APLICACIÓN

Acceso Usuario

Para acreditarnos como usuarios de la aplicación debemos estar registrados, esto podemos realizarlo en la página de registro de nuevos usuarios

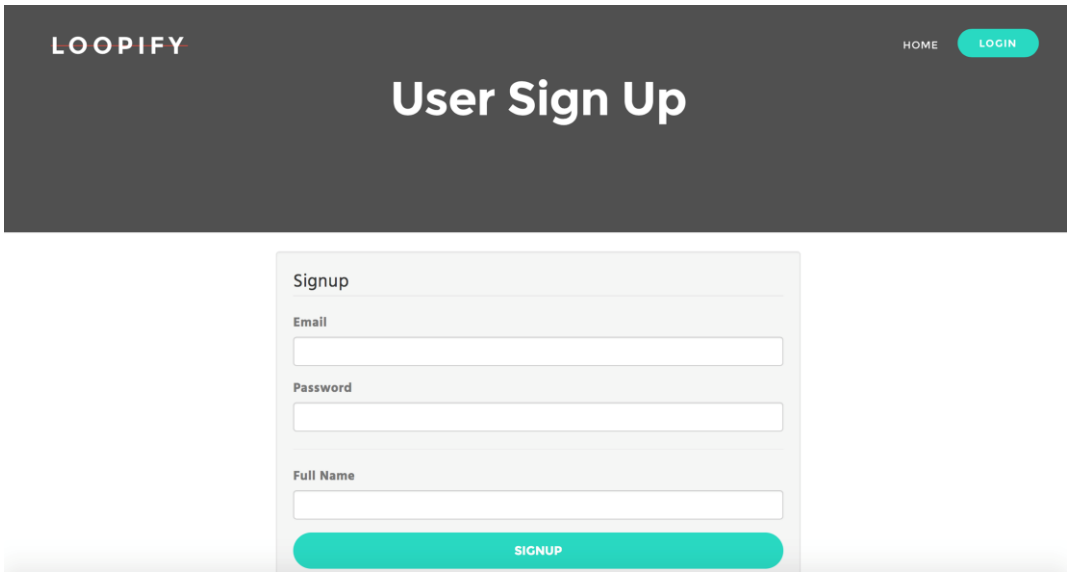


Figura 40. *Registro de Usuarios*

Una vez registrados podremos iniciar la sesión dentro de la página

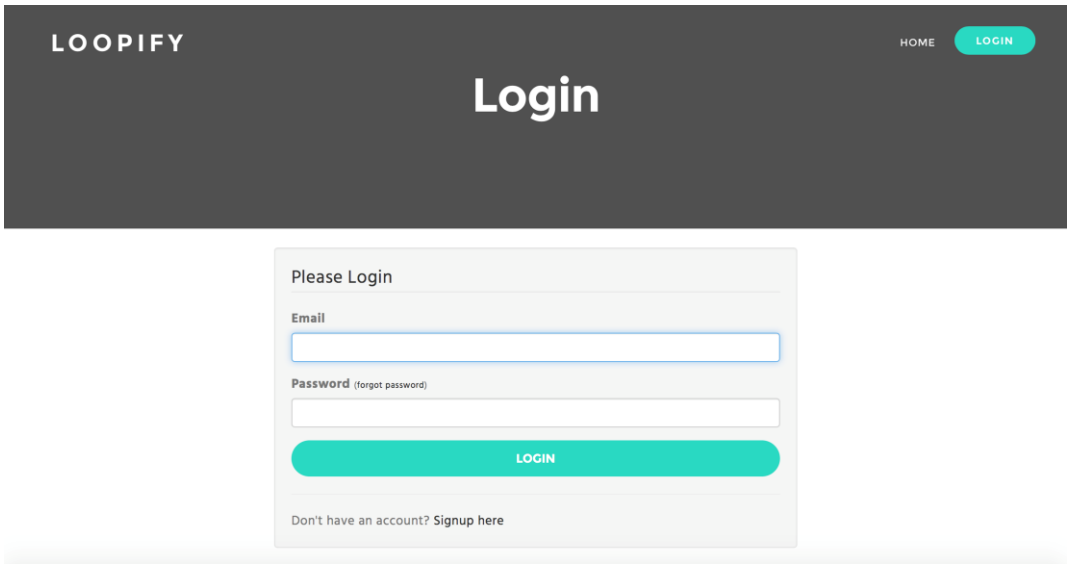


Figura 41. *Inicio de Sesión*

Loops

Esta es la sección más importante dentro de la aplicación. Desde aquí podemos editar nuestros loops y también podemos visualizar todos los loops que hemos creado

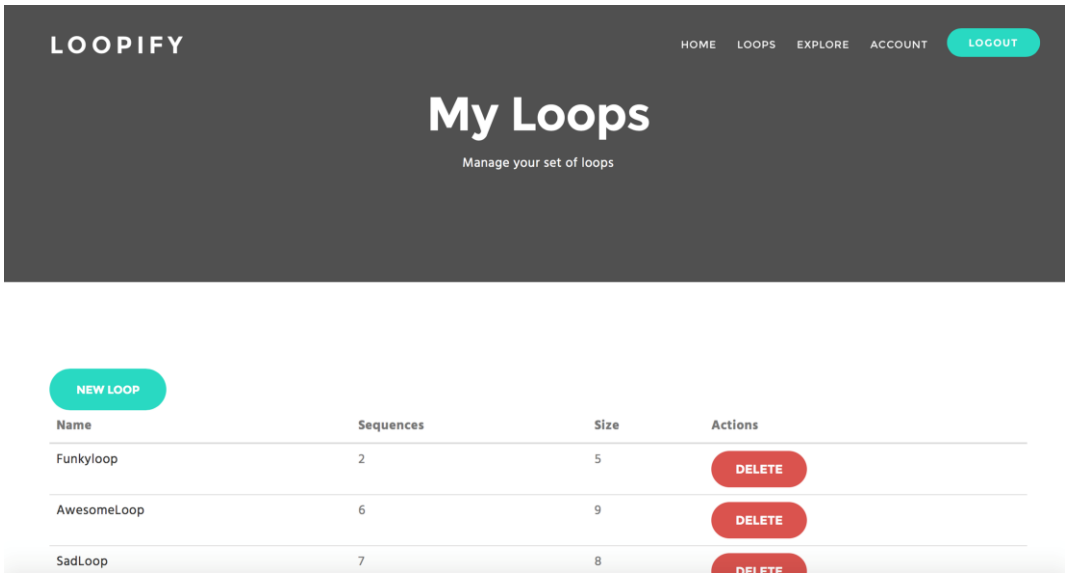


Figura 42. *Lista de Loops*

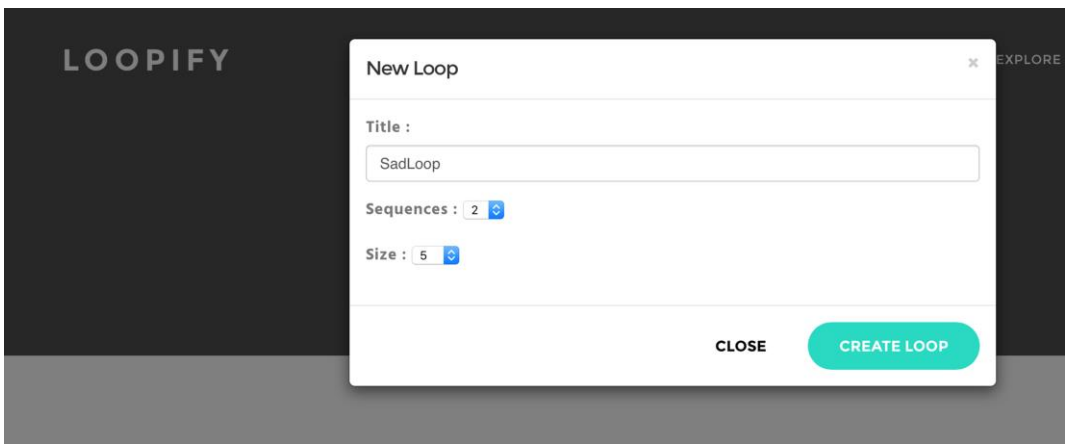


Figura 43. *Crear nuevo Loop*

Cada loop es creado con un tamaño, el cual determina la duración del loop, así como con datos como el tempo, volumen, la grilla de reproducción y las respectivas secuencias con sus notas

Tempo

Determina la velocidad de reproducción del loop

Length

Es el tamaño del loop. Este número es usado para especificar el número de casillas que tendrá nuestra grilla de reproducción

Grilla de Reproducción

Mediante esta grilla podemos programar la reproducción/repetición de una secuencia a lo largo del loop, si la casilla se encuentra activa esta será reproducida, caso contrario es omitida

Secuencia

Es la parte esencial de un loop, contiene las notas a ser reproducidas (en secuencia), así como el volumen, efectos y el número de notas

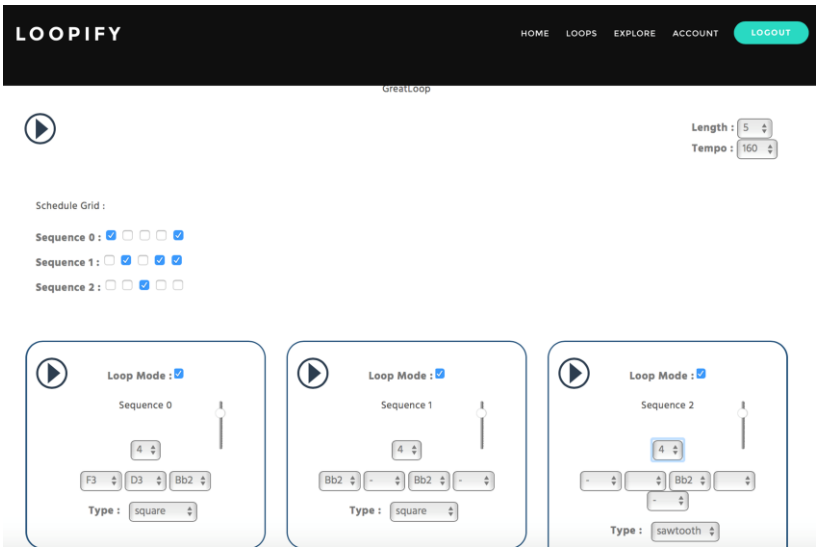


Figura 44. *Edición de Loops*

Muro de Noticias

Dependiendo de los usuarios a los que se sigue, veremos actualizaciones de los mismos dentro de esta sección. Esta se actualiza con datos de creación de nuevos loops, copia de loops o información de a quien siguen otros usuarios

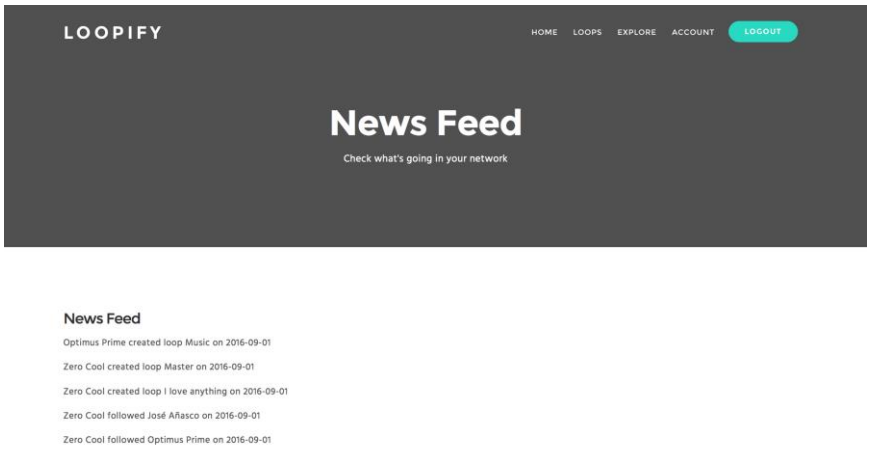


Figura 45. *Muro de noticias*

Explorar

Desde esta sección podemos navegar entre la lista de usuarios, visualizar sus loops y seguir las actualizaciones de usuarios específicos

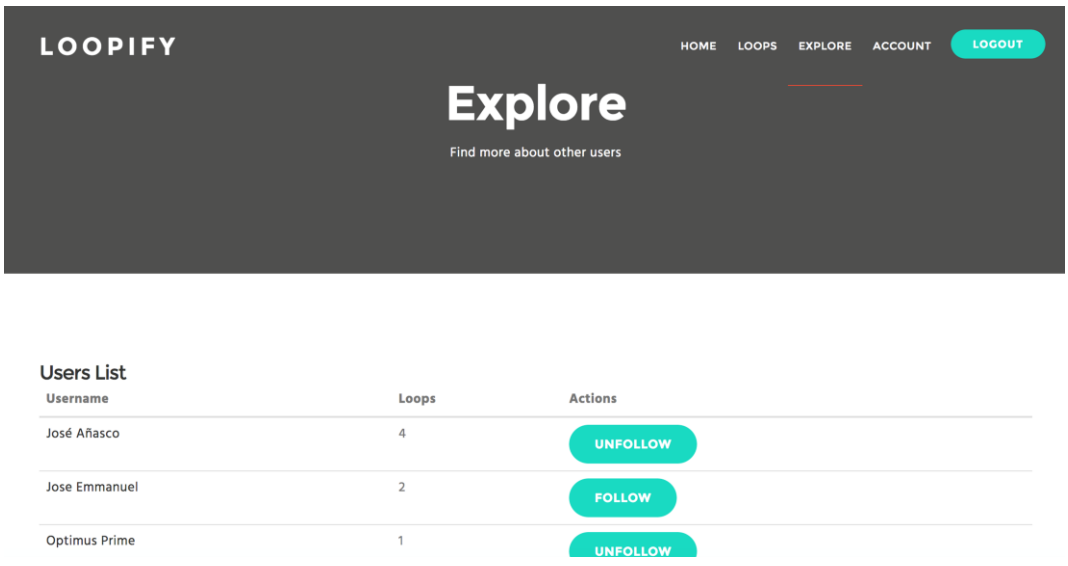


Figura 46. *Lista de usuarios*

Jose Emmanuel's loops

Name	Sequences	Size	Actions
SomeBeats	6	5	COPY LOOP
SomeBeats	3	12	COPY LOOP

Figura 47. *Lista de Loops de otros Usuarios*