Grado en Ingeniería en Tecnologías de la Telecomunicación

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2015-2016

Trabajo Fin de Grado

TITULO
TITULO

Autor: Jorge Ortega Morata

Tutor: Pedro de las Heras Quirós

Madrid 2016

Índice general

Índice de figuras

Capítulo 1

Introducción

Este Trabajo Fin de Grado se desarrolla en el ámbito del desarrollo y tecnologías web. En este capítulo introduciremos los conceptos y tecnologías básicas utilizadas y expondremos el paradigma actual en lo que a desarrollo web se refiere.

1.1. Desarrollo Web

En la actualidad el desarrollo web se encuentra muy arraigado en nuestra sociedad. Avanza a una velocidad increíble, tanto que ni nos percatamos de su impacto en todo lo que nos rodea. Acogemos cada nueva tecnología con los brazos abiertos y al poco transcurso de tiempo desde su primer uso nos es insuficiente. Esa insuficiencia impulsa su crecimiento e impide su decadencia.

Para conocer la historia del desarrollo web es necesario remontarse a los orígenes de *Internet*. En 1969 se crea la primera red de comunicación interconectada entre dos computadoras de dos universidades estadounidenses (UCLA y Stanford) llamada **ARPAnet**[1] (Advanced Research Projects Agency Network o Red de la

[1] https://es.wikipedia.org/wiki/ARPANET

Agencia para los Proyectos de Investigación Avanzada de los Estados Unidos) que funcionaba con un protocolo de intercambio de paquetes llamado **NCP**[2] (Network Control Program) y que más tarde se sustituyó por **TCP/IP**[3] por su robustez frente a colisiones. En 1986 comenzó la construcción de la primera infraestructura en forma de árbol de Internet llamada **NSFNET**[4] la cual se complementó con otras redes en EEUU. Después se crearon otras redes troncales en Europa y junto con las anteriores ya formaban el *backbone* o red troncal básica de Internet. Más tarde, en 1989 con la creación de la arquitectura de capas OSI en los computadores, comenzó a ser tendencia el utilizar diversos protocolos de comunicación a través de dicha red.

El desarrollo web se inició con la propia web o lo que conocemos como **World-WideWeb**[5] (WWW) que fue el primer cliente web creado por el *CERN*, cuyo equipo también creó el lenguaje *HTML*[6] (HyperText Markup Language). Lenguaje básico a la hora de estructurar la información en un sitio web.

El desarrollo web comprende todo el proceso de creación de un sitio web. La elección de las herramientas a utilizar, de diseño y desarrollo, qué metodología seguir, prototipado, propuesta final y lanzamiento. Es una labor compleja transformar una idea en algo físico y dotarla de utilidad para la sociedad. Es un reto que yo, como alumno, nunca había tenido la oportunidad de afrontar y me alegro de haber tenido la ocasión de realizar este proyecto.

La labor del desarrollador web en la actualidad es mucho más sencilla y accesible gracias a las tecnologías que surgen cada pocas semanas o días. Con el término

[2] `https://en.wikipedia.org/wiki/Network_Control_Program`
[3] `https://es.wikipedia.org/wiki/Modelo_TCP/IP`
[4] `https://es.wikipedia.org/wiki/NSFNet`
[5] `https://es.wikipedia.org/wiki/World_Wide_Web`
[6] `https://es.wikipedia.org/wiki/HTML`

open source en auge, millones de usuarios de Internet quieren aportar su granito de arena a esta labor y propician un crecimiento en herramientas web increíble y en ocasiones vertiginoso.

1.2. Tecnologías Web

Entran en este ámbito todas aquellas herramientas creadas específicamente para generar contenido web. Según su finalidad dentro del desarrollo web podemos diferenciarlas en Frameworks, librerías y lenguajes.

1.2.1. Frameworks

Según su definición un **framework**[7] es un *conjunto de conceptos y prácticas estandarizados* diseñado para afrontar un problema en particular, en este caso, el proporcionar una *infraestructura de software* a la hora de crear una aplicación web. Al usar un framework debemos seguir una serie de reglas establecidas según su diseño a la hora de organizar el código. Hoy en día estamos siendo testigos de la *"Batalla de los frameworks"* en lo que a desarrollo web se refiere y es que su crecimiento y potencial es increíble. Es muy importante conocer los puntos fuertes y débiles de cada uno de ellos a la hora de diseñar una aplicación ya que no todos incorporan los mismos conceptos y prácticas. Además también es imprescindible entender el *entorno de ejecución* de cada uno de ellos a saber: cliente (**Front-End**]), servidor (**Back-End**)[8] o ambos (**Full Stack**).

[7] `https://es.wikipedia.org/wiki/Framework`
[8] `https://es.wikipedia.org/wiki/Front-end_y_back-end`

En la actualidad se utilizan numerosos frameworks entre los que destacan: **AngularJS**[9], **EmberJS**[10], **Django**[11], **ReactJS**[12], **MeteorJS**[5], **BackboneJS**[13] y **ExpressJS**[14]. Todos utilizan *Javascript*[15] como lenguaje de desarrollo a excepción de Django que utiliza *Python*[16].

AngularJS

Es un framework front-end. Hace posible realizar peticiones REST y se pueden desarrollar proveedores que brindan servicios al cliente que se encuentran en el lado del servidor. La principal característica de Angular es que mediante su concepto de directiva podemos construir toda la aplicación de forma modular. Además cuenta con two data-binding que permiten el renderizado reactivo y dinámico de sus plantillas o módulos.

AngularJS es una creación de Google y es el framework más utilizado hoy en día, por lo que posee una gran comunidad, y uno de los más pesados en lo que respecta a tamaño.

EmberJS

Se trata de un framework front-end muy potente diseñado para crear aplicaciones grandes. Mediante la librería Handlebars [17] que incorpora podremos crear plantillas dinámicas gracias al data-binding que presenta. Además también posee un CLI (Interfaz de Línea de Comandos) que nos permitirá configurar todo

[9] `https://angularjs.org/`
[10] `http://emberjs.com/`
[11] `https://www.djangoproject.com/`
[12] https://facebook.github.io/react/
[13] `http://backbonejs.org/`
[14] `http://expressjs.com/es/`
[15] `https://es.wikipedia.org/wiki/JavaScript`
[16] `https://www.python.org/`
[17] `http://handlebarsjs.com/`

mediante comandos. Posee el módulo de routing más avanzado de todos los frameworks. Es una excelente herramienta para desarrollar un cliente con un buen nivel de potencial.

Django

Django, al igual que Ruby on Rails o MeteorJS no se debería de considerar framework sino plataforma de desarrollo web, ya que es full-stack y posee su propio CLI. El lenguaje de desarrollo es Python. Posee todas las herramientas necesarias para generar un servidor y un cliente. Es una herramienta muy completa. En el momento de su lanzamiento experimentó una gran acogida y aún conserva su fama tras sus numerosas actualizaciones.

ReactJS

Este es el framework más limitado en lo que a ámbito de ejecución se refiere (desarrollo de las vistas en el cliente y poco más). Pero es increíblemente flexible. La forma en la que se crean las plantillas en ReactJS (sin necesidad de escribir HTML) es increíble. Ha sido creado por Facebook y se utiliza cada vez más por su concepto de reactividad.

MeteorJS

MeteorJS [5], como decíamos de Django, se le debería tratar como una plataforma de desarrollo. Corre sobre NodeJS[18] y la principal ventaja que ofrece es que el servidor es completamente transparente al desarrollador. Solo debe preocuparse de crear el modelo de datos (más bien enunciarlo porque cuenta con MongoDB[10] como base de datos), las rutas, las publicaciones de datos reactivos y de las plantillas de la aplicación. Ha sido el pionero en lo que respecta al concepto de reactividad

[18] `https://nodejs.org/en/`

antes que ReactJS. Es una herramienta a tener en cuenta por su gran comunidad y sus paquetes específicos y por su capacidad para el prototipado rápido.

Exploraremos más a fondo las características de este framework más adelante ya que es el framework elegido para el desarrollo de nuestra aplicación que es de lo que trata este proyecto.

BackboneJS

Se trata de otro de los frameworks front-end más usados en la actualidad y no es de extrañar debido a su sencillez y capacidad. Es ideal para aplicaciones pequeñas y medianas.

ExpressJS

Este es un framework back-end y permite crear un servidor NodeJS en pocos minutos para sólo preocuparse del desarrollo front-end. Actualmente tiene cabida en cualquier proyecto por su fácil integración con cualquiera de los frameworks front-end.

Bootstrap [8]

Es más una librería que un framework pero dado que posee un paradigma especial y unas reglas claras a la hora de usarlo se le considera un framework front-end. Permite el dotar a tu aplicación de estilo rápidamente puesto que ya tiene definidos estilos por defecto, y crear módulos funcionales para organizar el contenido, además de animaciones. Es un framework destinado a la labor de maquetación y es el más utilizado hoy en día. Aunque también se utilizan otros como Foundation[19] o Pure[20] que también poseen buenas características para dicha labor.

[19] `http://foundation.zurb.com/`
[20] `http://purecss.io/`

1.2.2. Librerías

Las librerías son un conjunto de utilidades programadas en un lenguaje dado que proporcionan un servicio concreto al desarrollador. Al contrario que un programa no están pensadas para un uso automático, es decir, no tienen un inicio. Sólo son usadas por programas. En lo que se refiere a desarrollo web son utilizadas para aportar servicios al desarrollador de la aplicación. La librería líder en este ámbito es JQuery[11][12] y sus descendientes como JQueryUI[21]. Otra libería cada vez más utilizada es UnderscoreJS.

JQuery

Se trata de una librería que permite manipular el DOM[22] (Document Object Model) desde la lógica de la aplicación. No necesitamos hacer uso del objeto document para realizar búsquedas, añadir etiquetas y demás operaciones de manipulación. Mediante JQuery podremos establecer eventos, manipular dinámicamente el DOM añadiendo etiquetas y estilos y mucho más. Es una herramienta indispensable para el desarrollador web. Lo es tanto que incluso otras librerías y paquetes destinados a diferentes frameworks la usan.

UnderscoreJS [13]

Esta es una librería que permite manipular objetos, colecciones de objetos y arrays. Permite realizar operaciones de filtrado muy avanzadas sobre una colección de objetos y además realizar iteraciones de forma funcional mediante sus métodos

[21] `https://jqueryui.com/`
[22] `https://es.wikipedia.org/wiki/Document_Object_Model`

1.2.3. Lenguajes

Existen numerosos lenguajes en desarrollo web: Java (para servidores principalmente), Javascript, CoffeeScript, TypeScript, HTML5, Jade, CSS, Less, Sass, Python, Ruby, etc. Pero hay que tener en cuenta que lo que el navegador compila e interpreta son ficheros CSS, HTML y lenguajes de lógica de la aplicación como Javascript. Por lo que si desarrollamos con Jade, Less, Sass o CoffeeScript o TypeScript debemos de preprocesarlos hacia los tres principales lenguajes mencionados anteriormente.

HTML5

Se trata de la quinta versión del lenguaje HTML. Posee una variante de sintaxis básica conocida como HTML5 y otra XHTML conocida como XHTML5 y se sirve como sintaxis XML (eXtensible Markup Language) concebido por el World Wide Web Consortium (W3C) con el fin de almacenar datos de forma legible y que complementa en la mayoría de aplicaciones al documento HTML.

En esta quinta versión se han desarrollado nuevas etiquetas que ayudan a realizar un documento más semántico y legible, además de proporcionarnos herramientas de dibujo en 2D y 3D, etiquetas para introducir audio y video o de formato. Etiquetas como: <article>, <aside>, <audio>, <video>, <canvas>, <datalist>, <details>, <dialog>, <embed>, <figure>, <footer>, <header>, <mark>, <meter>, <nav>, <output>, <progress>, <ruby>, <rp>, <rt>, <section>, <source> y <time>.

También incorpora herramientas nuevas como un visor de fórmulas matemáticas (MathML), tecnología Drag & Drop (arrastrar y soltar objetos basado en eventos), ejecución en paralelo mediante WebWorkers, comunicación bidireccional

entre páginas mediante WebSockets, APIs para almacenamiento (Local & Global Storage), geolocalicación y para trabajar en local (Off-line).

La incorporación de estas nuevas herramientas reduce la necesidad del desarrollador a utilizar plugins externos.

CSS3

CSS es el lenguaje utilizado para crear la presentación y dar estilo al documento HTML o XML. En esta tercera versión se introducen nuevas funcionalidades como animaciones 3D, transiciones, estructuración mediante propiedades grid (rejilla), media querys para establecer cambios de estilo conforme el tamaño de la pantalla varía, etc.

Javascript [9]

Es el lenguaje de programación más usado hoy en día en el desarrollo web (sobretodo en el lado del cliente). Fue creado por Brendan Eich de Netscape como dialecto de ECMAScript. Su primer nombre fue Mocha, después LiveScript y finalmente Javascript. Es un lenguaje orientado a objetos, basado en prototipos, funcional (las funciones son objetos), posee un tipado muy débil y es dinámico.

Como se ha comentado, el marco de utilización de este lenguaje en el desarrollo web son los frameworks front-end. Aunque también se utilizan frameworks basados en NodeJS o el propio NodeJS para desarrollar mediante este lenguaje en el lado del servidor en numerosos proyectos.

Javascript sigue creciendo y actualizándose. En 2015 fue lanzado el estándar ECMAScript6, el cual dota a javascript de nuevas funcionalidades y módulos co-

mo un nuevo paradigma de orientación a objetos basado en clases, iteradores o promesas para programación asíncrona.

1.3. MeteorJS

Meteor es una *plataforma* que permite crear aplicaciones Web en *tiempo real* basada en **NodeJS**. Fue creado con el propósito del *prototipado rápido* y en este ámbito supera a la mayoría de *frameworks*. Ha sido el primero en introducir el principio de *reactividad* en el desarrollo web. Soporta **MongoDB** como tecnología de base de datos y posee un asombroso concepto de *subscripciones* a *publicaciones* procedentes de *colecciones* que hacen del *renderizado* del **DOM** un proceso muy rápido gracias a que mantiene en el cliente una mini base de datos llamada *Mini-Mongo*. El cliente es capaz de modificar dicha base de datos y observar los cambios directamente que más tarde se actualizarán en la base de datos del servidor.

1.3.1. Reactividad

EL gran potencial de Meteor es debido a este *principio*. Se basa en observar los cambios sobre una *fuente* en tiempo real y actuar en consecuencia, dotando a las aplicaciones de un *dinamismo* muy especial. Adiós a los *Listeners* y al *binding* sobre elementos **HTML**, no hacen falta si sabes aprovechar este principio y sus *entidades*. En meteor existen numerosas entidades que proveen *reactividad* y otras muchas que permiten crear nuevas *entidades reactivas*. Las **fuentes reactivas** que puedes controlar de manera simple en Meteor son las *variables de sesión* almacenadas en *Session*. Mediante la sentencia *Session.set(key[String],value)* ya tienes una fuente reactiva a tu disposición. Sólo necesitas algo que sepa escuchar sus cambios y actuar (*helpers* o *Tracker*).

El modulo **Tracker** posee un método *.autorun()* que permite ejecutar código cuando una fuente reactiva cambia. Además puede directamente asociarse a la lógica de cualquier plantilla dentro del método *.rendered()* del controlador. Esto permite dotar a la plantilla de dinamismo, por ejemplo realizar **subscripciones dinámicas** sobre una colección enlazado con los eventos de la plantilla. (Auto-completados basados en subscripciones, Botones para cargar más contenido, etc).

1.3.2. Sistema de plantillas reactivas

Meteor utiliza una *biblioteca* muy poderosa para crear *interfaces de usuario* que se actualizan en tiempo real llamada **Blaze**. Cumple el mismo propósito que **Angular**, **Backbone**, **Ember**, **React**, **Polymer** o **Knockout** en este ámbito pero es mucho más sencilla de utilizar, incluso *transparente* para el programador. Su labor no sería posible sin *Tracker*, un módulo de Meteor que permite gestionar *procesos reactivos* de manera limpia, y sin **Spacebars** (parecido a **Handlebars**), el lenguaje particular de Meteor para definir las plantillas y que aprovecha al máximo la funcionalidad de *Tracker*.

1.3.3. Comunicación con el servidor

La comunicación con el servidor se basa en el protocolo HTTP que Meteor integra de manera transparente al programador mediante su módulo methods al que se accede mediante Meteor.methods() y para la petición de recursos se utiliza algún paquete creador de rutas como IronRouter o FlowRouter. También posee el módulo http para realizar peticiones desde el cliente al servidor y a terceros. Todas las peticiones son asíncronas y como tales se les asocian callbacks (funciones que se ejecutarán una vez haya terminado la ejecución de la petición).

Cada vez que se define una plantilla mediante Spacebars se crea un objeto plantilla Template.name y que lo tendremos accesible a la hora de dotarla de funcionalidad mediante javascript. Es un tipo de controlador. Spacebars permite el paso de datos de la plantilla al controlador y viceversa, esto se denomina two data-binding y supone una poderosa herramienta a la hora de crear componentes aislados puesto que su configuración puede realizarse vía Spacebars. Este proceso lo realiza mediante la declaración de helpers o ayudantes de plantilla y se definen mediante la función helpers() del controlador. La gran ventaja de esto radica en que los helpers son funciones javascript asociadas a una fuente reactiva. Esto quiere decir que en el momento que esa fuente cambia los helpers se actualizan y se actualiza el contenido HTML asociado a ellos. Además de permitir crear componentes reusables, éstos son dinámicos (reactivos) en su instanciación y durante su uso.

Aparte de los helpers al controlador se le puede asociar un mapa de eventos relacionados con la plantilla mediante la función events() que toma como parámetro un objeto tipo:

Jerarquía de carpetas y orden de carga

Para aplicaciones pequeñas es posible escribir el código ejecutable por el cliente y por el servidor en una misma carpeta. Para ello Meteor cuenta con las funciones *.isClient()* y *.isServer()* para especificar qué código debe ejecutarse en cada entorno. Para aplicaciones más grandes la estructura es un poco peculiar y se debe generar teniendo en cuenta el orden de carga según la *jerarquía de carpetas*. Este orden de carga aunque sea muy estricto provee de una flexibilidad asombrosa y permite crear cualquier aplicación de forma *modular*. La jerarquía de carpetas sería la siguiente:

```
/app
    /lib --primero en cargar tanto en el servidor como en el cliente
        /collections
        /router.js
    /client --solo se carga en el browser
        /lib --primero en cargar
        /modules --plantillas
            /module_name
                /module_name.js
                /module_name.html
                /module_name.scss
        /components --componentes reutilizables
            /component_name
                /component_name.js
                /component_name.html
                /component_name.scss
        /styles --estilos reutilizables
        /main.js --ultimo en cargar
        /main.html --ultimo en cargar
        /index.scss --primera hoja de estilos en cargar
    /server --solo se carga en el servidor
        /publications.js --declaración de las publicaciones de la base de datos
        /services --servicios ofrecidos para el cliente (Meteor.methods)
            /service_name.js
    /public --fuentes e imágenes estáticas
```

Figura 1.1: jerarquía de carpetas

La carpeta */lib* de más alto nivel dentro de la jerarquía contiene todos los *ficheros comunes* a ambos entornos (cliente y servidor) y es la primera en cargar. */server* y */client* contienen todos los *ficheros ejecutables* por el servidor y por el cliente respectivamente. Dentro de cada una de las carpetas anteriores existe un orden de carga. Si poseen carpeta */lib* será la primera en cargar, después es el turno de los demás ficheros en *orden alfabético* y, por último, los ficheros *main.** sea cual sea su extensión. En la carpeta */public* se encuentra el *contenido estático* de nuestra aplicación (fuentes, imágenes, etc).

Como podemos ver según el orden y ámbito de ejecución Meteor ofrece un *entorno de ejecución simétrico* (se ejecuta tanto en el cliente como en el servidor) dentro de la carpeta */lib* de más alto nivel en la jerarquía. La principal ventaja de esto es que no tenemos porqué replicar código. Podemos crear *constructores* y demás funcionalidad necesaria en ambos entornos una sola vez y Meteor se encarga de saber que cargar en cada uno.

Además, al ser MeteorJS una plataforma de desarrollo, no es necesario un *gestor de tareas* como **GruntJS** o **GulpJS**. Estos se encargan de **automatizar tareas** tales como establecer la carga de ficheros sobre el *documento HTML*, *minificar* todos los ficheros, establecer la *configuración del servidor* o arrancar nuestra aplicación. Meteor posee un **CLI** (Interfaz de línea de comandos) que mediante el comando *meteor* ya se encarga de establecer las configuraciones iniciales, cargar todos los ficheros según el orden descrito e incluirlos en el documento y arrancar nuestro servidor. El *minificado* no es necesario en un primer momento, puesto que con el *deploying* (despliegue) se realizará. Existen una gran cantidad de paquetes y constructores que lo harán de forma automática.

1.4. MongoDB

MongoDB es una base de datos **no relacional** cuya arquitectura *se basa en documentos*. Al no ser *relacional* como **mySQL**, **Oracle** o **PostgreeSQL** carece de *claves primarias*. No hay que declarar un *modelo de datos* puesto que lo que se almacenan son objetos en formato **BSON** (muy parecido a **JSON**) en el que todos los *atributos* pueden ser utilizados como *clave* a la hora de realizar búsquedas. Cada vez que un documento es insertado se le asocia un *índice único*. Aunque no sea relacional ofrece la posibilidad de realizar un diseño de este tipo. Esto es enlazando objetos pertenecientes a *colecciones* (tablas) diferentes mediante su identificador o cualquier atributo válido. La gran ventaja de utilizar este tipo de base de datos es la **rapidez de acceso**. Como cualquier *objeto javascript* un documento puede *embeber* otros documentos (objetos) a los que se les puede establecer un índice y realizar **búsquedas indexadas**. Además *MongoDB* cuenta con módulos como *$agregation* que permite establecer reglas para cada colección que permiten realizar búsquedas más complejas como por ejemplo establecer para qué campo del

documento se realiza una búsqueda mediante *expresiones regulares*.

Debido a que es una base de datos no relacional podemos *embeber entidades* que dependan de otras en éstas. De no hacerlo así el proceso de *borrado de datos* hay que tomarlo con calma debido a que este tipo de base de datos no posee **joins** ni de herramientas que hagan que la **atomicidad** de los datos se mantenga como ocurre en las bases de datos relacionales. Aunque si queremos también realizar un diseño desde un enfoque más limpio deberíamos de separar todas las entidades.

1.4.1. MongoDB y MeteorJS

Al crear una aplicación mediante el CLI de MeteorJS directamente se crea una base de datos MongoDB. Aunque Meteor no trabaja con otro tipo de base de datos en un principio, se puede cambiar mediante la instalación de paquetes. En la actualidad existen paquetes de bases de datos relacionales para Meteor que poseen reactividad y es este principio en el que se basa Meteor.

Las tablas creadas en MongoDB en Meteor se convierten en colecciones, un wrapper para ofrecer funcionalidad desde la aplicación y dotarlas de reactividad (las convierte en fuentes reactivas). Este objeto en el que se engloba a la tabla o entidad ofrece los métodos .find(), findOne(), update(), remove(), insert(), allow() y deny() que son los más usados. Hay que tener en cuenta que las colecciones en Meteor se declaran en la carpeta /lib, cuyo contenido será ejecutado tanto en el entorno del cliente como en el del servidor. Esto quiere decir que cada entorno tendrá una instancia de cada colección y esto al igual que es ventajoso en cuanto a rapidez en el cliente (puede acceder a la base de datos directamente "miniMongo"), es peligroso por el mismo motivo. Para ello existen los métodos deny() y allow() que establecen qué operaciones sobre la colección están permitidas en el cliente y

cuáles no.

Lo más sensato es permitir insertar y denegar el permiso para realizar cualquier alteración sobre otros documentos ya presentes, al menos directamente. Para ello se usa el módulo methods de Meteor que permite configurar métodos a los que llamar desde el cliente (también se declaran en la carpeta /lib) y que se ejecutan en el servidor (donde no existe ningún tipo de restricción).

1.4.2. Publicaciones y Subscripciones

Puesto que las instancias de las colecciones se encuentran accesibles también en el cliente debe haber un control sobre el contenido de las mismas dentro de MiniMongo. Para ello se utilizan las publicaciones y las subscripciones. Las publicaciones se realizan en el lado del servidor y el cliente se subscribe a ellas. La moneda de cambio son los cursores. Un cursor es una fuente reactiva que engloba uno, varios o ningún documento procedente de una colección. El cliente al subscribirse a una publicación obtiene el cursor y este lo transforma en documentos que se almacenan dentro de MiniMongo donde tendrá accesibles los documentos. Lo bueno de esto es que como se ha dicho los cursores son fuentes reactivas, esto quiere decir que en el momento que se produzca algún cambio que altere el cursor al que se está subscrito, la publicación cambiará y la subscripción se actualizará. Para aprovecharse de este fenómeno el cliente necesita extraer un cursor procedente de MiniMongo mediante NombreColección.find() o NombreColección.findOne() y asociarlo a un helper dentro de la lógica de la plantilla.

1.5. HTML5 Media

Con la llegada y estandarización de HTML5 cada vez se trabaja más en herramientas que faciliten la comunicación entre usuarios y que, en definitiva, brinden servicios interactivos a los mismos en tiempo real. Una de esas herramientas es WebRTC (Web Real-Time Communication).

1.5.1. WebRTC

Se trata de una API creada para permitir realizar llamadas de voz, chat de video e intercambio de archivos P2P (Peer to Peer) sin la necesidad de plugins. Es Open Source (Código abierto). Fue creado primero por Google y la W3C se encarga ahora de su estandarización. Su desarrollo está en proceso, por lo que se comporta de manera inestable en su funcionalidad avanzada. Los navegadores que la soportan son Chrome, Firefox y Opera hoy en día. Esto se debe a que su módulo navigator facilita el acceso a los recursos media del ordenador (micrófono, webcam).

1.5.2. RTCRecorder

Se trata de una API basada en WebRTC que proporciona una serie de herramientas para grabar video y audio de manera sencilla y pudiendo exportar el archivo creado y almacenarlo tanto en servicios cloud como en local. El creador de esta API también ha desarrollado otros módulos basados en WebRTC capaces por ejemplo de grabar video y audio sobre un elemento canvas. Hablaremos de esta API más adelante puesto que ha sido integrada en el proyecto.

1.6. Tecnologías Cloud

Hoy en día el término Cloud está muy extendido. La mayoría de aplicaciones y sitios web utilizan tecnologías cloud para almacenar grandes cantidades de datos y liberar memoria propia de la aplicación o bien son utilizadas como base de datos. Ejemplos de tecnologías cloud son Google Drive, Dropbox, Youtube, Amazon S3, Soundcloud, mLab o DigitalOcean.

La mayoría de los anteriores son utilizados como servicios cloud extra o para almacenar contenidos de la aplicación (ficheros, imágenes, audios, videos). Amazon S3 o mLab son utilizados como base de datos de la aplicación y su ventaja radica en que, a la hora de realizar el despliegue, utilizamos servidores externos en los que almacenaremos la base de datos de nuestra aplicación.

1.6.1. API Soundcloud

Soundcloud es un sitio web que permite alojar archivos de audio al estilo de Youtube con vídeos. Posee un API disponible en diferentes lenguajes de programación como Ruby, javascript y PHP para realizar peticiones REST y por tanto un espacio para desarrolladores en el cual crear diferentes aplicaciones con las que comunicarse la API. Se trata de una solución factible a la hora de desarrollar un proyecto pequeño ya que tiene limitaciones en lo que respecta a espacio. Si nos encontramos ante un proyecto de gran envergadura necesitaremos explorar otras vías como GridFS aplicado a otro servicio cloud de base de datos como Amazon S3.

También proporciona herramientas de Streaming de gran utilidad a la hora de reproducir dichos archivos de audio en nuestra aplicación de forma remota.

Exploraremos a fondo esta API puesto que es una de las herramientas más importantes incluidas en este proyecto.

1.7. Herramientas para trabajo en equipo

El mundo del desarrollo web es altamente competitivo y como tal exige la obtención de resultados muy a corto plazo. Para conseguir este objetivo surgen las metodologías ágiles como SCRUM que, según su definición, no es más que un proceso en el que se aplican una sucesión de buenas prácticas para trabajar colaborativamente y en equipo. La finalidad es conseguir un equipo altamente productivo. La base de esta metodología es la realimentación o feedback, es decir, es un proceso circular compuesto por varias fases y realimentado en el cual el cliente toma conciencia de cada ciclo aportando sus críticas.

Para reforzar este proceso y facilitar la labor del desarrollador y de todo el equipo existen distintas herramientas como GitHub, PivotalTracker y Slack entre otras. Estas herramientas han sido utilizadas a lo largo de este proyecto.

1.7.1. Git y GitHub

Git es el sistema de control de versiones más usado en el mundo del desarrollo. Crea una copia del proyecto en un repositorio y a través de sus commits se almacenan versiones recuperables del mismo. Incorpora un sistema para generar ramas de versiones que posteriormente pueden volver a unirse para conformar el resultado final del proyecto o finalizar alguna fase. Esto lo hace verdaderamente potente a la hora de utilizarlo en grupo y por tanto es necesario compartir el repositorio entre los integrantes del equipo de desarrollo. Esto se hace mediante GitHub, una plataforma en la que almacenar proyectos públicos o privados que

integra el sistema de control de versiones mencionado anteriormente. Permite la copia de cualquier versión desde un usuario a otro (fork) y puede desarrollarse un seguimiento de todo el proyecto mediante el espacio para Wiki.

Además permite la sincronización de otros servicios afines al proyecto como PivotalTracker.

1.7.2. PivotalTracker

La primera fase de toda metodología ágil en un proyecto de desarrollo se basa en el análisis y la extracción de requisitos. Estos requisitos son llamados historias de usuario y son el elemento base de PivotalTracker.

PivotalTracker es una plataforma de organización de proyectos mediante tareas o items a completar. Cada historia de usuario normalmente es dividida en distintas tareas. Una vez creado un proyecto en esta plataforma y establecido los miembros comienza la asignación de tareas. Se establecen diferentes espacios o ambientes de trabajo que indican la prioridad de las tareas (Icebox, current, done, etc). Con la creación de cada tarea viene la estimación del tiempo de trabajo de la misma y a mayor valor de estimación más miembros la tendrán asignada. Una vez finalizada la tarea es necesaria la validación de la misma por el resto del equipo. De esta manera está asegurado el correcto desarrollo del proyecto.

1.7.3. Slack

Slack es una plataforma de comunicación destinada a grandes proyectos. Se organiza mediante canales y permite el intercambio de ficheros, información y mucho más a través de chat. Además ofrece la posibilidad de vincular cada canal a los servicios utilizados en el proyecto como GitHub y PivotalTracker por lo que

cada acción y avance quedará reflejado y notificado a los miembros del equipo. Se trata de una herramienta muy útil para el trabajo en remoto y como historial de proyecto.

Capítulo 2

Objetivos y metodología

2.1. Motivación

En la actualidad existen distintas aplicaciones y sitios web destinados a facilitar la labor del desarrollador. Sitios como PivotalTracker, Github, GitBucket, CodePen, Codecademy o StackOverflow. Estos sitios proporcionan herramientas para un correcto desarrollo de cualquier proyecto. En el caso de Github o GitBucket facilitan la creación de repositorios remotos con un sistema de control de versiones. PivotalTracker permite organizar las tareas de un proyecto y CodePen, Codecademy y StackOverflow son sitios web para el aprendizaje y compartir conocimientos.

Es precisamente el aprender y compartir lo que mueve el mundo del desarrollo. Cualquier desarrollador utiliza ideas propias y de otros desarrolladores para realizar cualquier proyecto. Pero muchas veces esas ideas no son muy intuitivas o fáciles de aprender a partir de un artículo o un fichero de código completado. Por esto muchas veces recurrimos a plataformas de difusión de video para poder aprender rápidamente.

De esta necesidad surge la motivación de crear una aplicación web que sirva de plataforma para todos los desarrolladores. Una plataforma donde puedan aprender rápidamente, intercambiar conocimientos de manera interactiva y que permita la comunicación entre sus usuarios. Puesto que la información debería ser lo más visual e interactiva posible el formato se basará en grabaciones de código y audio sobre un editor.

2.2. Requisitos

Con la motivación surge el proceso de pensar en las necesidades que tendrá el usuario al utilizar la aplicación. Estas necesidades se traducen en requisitos que debe cumplir la aplicación y que hay que tener presentes en todo momento del diseño y del desarrollo. Tanto nuestro concepto de la aplicación como las necesidades del usuario pueden verse alteradas durante el desarrollo del proyecto por lo tanto pueden surgir nuevos requisitos y verse alterados los ya establecidos.

A continuación se exponen todos los requisitos que debería cumplir nuestra aplicación:

1. Se deberá proporcionar una interfaz de inicio para que el usuario pueda explorar las características de la aplicación, aprender, y poder crear un usuario para acceder a la misma.

2. Los usuarios deben ser autenticados para poder acceder a la mayoría de la funcionalidad de la aplicación.

3. Un usuario podrá registrarse introduciendo un nombre de usuario, un correo electrónico y una contraseña o bien mediante los servicios integrados de Facebook, Github y Google+.

4. Es necesario proporcionar al usuario un proceso de validación o verificación de email, un proceso de cambio de contraseña y otro de recuperación de la misma.

5. Un usuario podrá recuperar su contraseña siempre que tenga asociado un correo electrónico y éste haya sido verificado y puede estar autenticado o no.

6. Un usuario podrá acceder a la aplicación introduciendo su nombre de usuario o email asociado y su contraseña.

7. Un usuario podrá crear canales, lecciones, grabaciones y conversaciones.

8. Un usuario podrá realizar peticiones de contacto a otros usuarios para añadirlos a su lista de contactos.

9. Un usuario podrá aceptar o rechazar peticiones.

10. Un usuario podrá reenviar una solicitud de contacto que haya sido rechazada.

11. Un usuario podrá crear conversaciones con otros usuarios que estén en su lista de contactos.

12. Cualquier contenido se podrá etiquetar para facilitar la recomendación y búsqueda del mismo, con la excepcion de las conversaciones.

13. Cualquier contenido se podrá votar y comentar con la excepción de las conversaciones.

14. Se podrán crear respuestas a los comentarios.

15. Todo contenido deberá mostar contadores de votos, subcontenidos y usuarios subscritos (cuando proceda).

16. Todo contenido podrá ser editado.

17. Todos los contenidos podrán ser borrados a excepción de las conversaciones.

18. No se podrá borrar ningun contenido con subcontenidos.

19. Para una mejor experiencia es necesario que el usuario conozca lo que sucede dentro de aplicación para ello debe existir un módulo que permita crear notificaciones personalizadas y de interés para el usuario.

20. El usuario deberá tener acceso rápido a las listas de contenido.

21. Todas las listas de contenido podrán ser filtradas por dos tipos de filtros: más recientes y más populares.

22. Para todo tipo de contenido existen dos roles con un tipo de acceso diferente a las funcionalidades propias del mismo.

23. La aplicación deberá ser óptima y precisa y proporcionar al usuario una interfaz cuidada e intuitiva.

24. Las grabaciones serán grabaciones de código y audio sobre editor.

25. Las grabaciones serán el elemento atómico en lo que respecta a contenido dentro de la aplicación.

26. El objeto de cada grabación será grabar documentos de código.

27. Cada grabación podrá tener 1 o más documentos.

28. Los documentos tendrán un titulo único dentro de cada grabación, un modo (lenguaje en el que están programados) y un tema (aspecto en el editor).

29. Un usuario podrá crear grabaciones.

30. Un usuario podrá crear respuestas sobre cualquier grabación a partir de cualquier instante de su reproducción.

31. Las grabaciones podrán ser aisladas o pertenecer a un canal o lección.

32. El usuario podrá navegar, visualizar y reproducir cualquier grabación.

33. El usuario podrá acceder al padre de una grabación (respuesta) desde la página de visualización de la misma.

34. El usuario podrá visualizar tanto las respuestas a una grabación cómo grabaciones relacionadas a la misma.

35. El usuario podrá acceder al canal o lección a la que pertenece cualquier grabación.

36. Las grabaciones no tienen porqué poseer un título único, pero sí deben tener uno. Opcionalmente tendrán una descripción y una lista de etiquetas.

37. Las grabaciones podrán ser editadas mediante una respuesta.

38. El usuario podrá cambiar el instante de la reproducción, pausarla o reanudarla.

39. Un usuario podrá crear canales.

40. Un usuario podrá subscribirse a cualquier canal que no haya sido creado por él mismo para recibir notificaciones relevantes.

41. El contenido de los canales estará formado por una lista de grabaciones.

42. La creación de contenido no está restringida para los canales. Todos los usuarios podrán generar contenido dentro de cualquier canal.

43. Cualquier usuario podrá votar y comentar canales y sus contenidos.

44. Cada canal podrá ser editado por el creador.

45. Un usuario podrá crear lecciones.

46. El contenido de las lecciones serán secciones formadas por una lista de grabaciones.

47. La creación de contenido estará restringida para las lecciones. Sólo el creador de la lección puede crear contenido directo a las mismas a saber: secciones y grabaciones.

48. Un usuario podrá subscribirse a cualquier lección que no haya sido creada por él mismo para poder acceder a su contenido y recibir las notificaciones relevantes.

49. Cualquier usuario subscrito a una lección podrá crear respuestas a todas las grabaciones que existan dentro de la misma.

50. El contenido de una sección se reproducirá mediante el concepto de lista de reproducción.

51. El reproductor deberá identificar si la grabación procede de una lección y si forma parte de una lista de reproducción. Si es el caso deberá proporcionar una interfaz para navegar por la lista de reproducción y establecer opciones tipo: reproducción y repetición automática.

52. Las opciones reproducción y repetición automática estarán asociadas al usuario no a la sección.

53. La reproducción automática provocará que se reproduzca la siguiente grabación de la lista al finalizar la actual.

54. La repetición automática provocará que la reproducción de la lista sea circular (ni principio ni fin).

55. Cualquier usuario que esté subscrito a una lección podrá votar a la misma y a sus contenidos (grabaciones).

56. Cualquier usuario que esté subscrito a una lección podrá comentar la misma y sus contenidos (grabaciones).

57. Cada lección podrá ser editada por el creador.

58. El orden de las secciones podrá ser cambiado por el creador.

59. El orden de las grabaciones que forman una sección podrá ser cambiado por el creador.

60. Las secciones podrán ser borradas por el creador en cualquier momento, siempre que no tengan contenido.

61. Para una mejor experiencia es necesario realizar al usuario recomendaciones de contenido basadas en sus gustos y su recorrido dentro de la aplicación.

62. Para una mejor experiencia es necesario mostrar al usuario las tendencias o el contenido más popular dentro de la aplicación.

63. Para una mejor experiencia es necesario aportar al usuario un mini tutorial compuesto por tareas básicas que le ayude a dar sus primeros pasos en la aplicación.

64. Es necesario que los usuarios puedan explorar una descripción de las características de la aplicación para ello se debe crear un espacio en el que el usuario las conozca.

65. Es necesario que los usuarios puedan aprender cómo usar la aplicación para ello se debe crear un espacio con tutoriales.

66. Es necesario que los contenidos puedan mostrarse en listas para su navegación. Para ello cada contenido debe poseer una miniatura asociada que muestre la información más relevante para cada tipo de contenido.

67. Es necesaria la presencia de un buscador de contenido dentro de la aplicación.

68. Las búsquedas en la aplicación serán dinámicas y con un sistema de etiquetas que se irán sugiriendo de manera automámica.

69. Es necesario que los usuarios puedan recibir notificaciones de nuevos mensajes de sus conversaciones abiertas cuando no estén visualizando dicha conversación.

70. Cada usuario deberá poder acceder a toda su información y contenido. Para ello se les proporcionará un enlace en todo momento a su perfil y un acceso a sus contenidos principales.

71. Cada usuario podrá visualizar la lista de sus contenidos, sus subscripciones, sus conversaciones, una lista con las últimas reproducciones y sus contactos.

72. Cada usuario podrá editar su perfil a saber: su avatar, el banner, su descripción y sus emails.

73. Los perfiles podrán ser vistos por todos los usuarios por lo que se establecen dos roles: propietario, visitante.

74. Un visitante ya sea contacto o no del propietario podrá visualizar todo su contenido a excepción de sus conversaciones.

75. Un visitante ya sea contacto o no del propietario podrá acceder al perfil asociado al servicio integrado en el caso de que el propietario se haya registrado mediante dicho servicio.

76. Un visitante que sea contacto del propietario podrá iniciar una conversación o redactar un email al email del propietario en el caso de que éste tenga configurado y verificado algún correo.

77. Un visitante que no sea contacto podrá enviar una solicitud de contacto al propietario.

78. Se debe crear un espacio para realizar solicitudes de contacto que lo componga un buscador (autocompletado de usuarios) y listas de peticiones recibidas y enviadas y su estado.

79. Una conversación debe incluir al menos 2 contactos en el momento de creación.

80. Cada conversación podrá ser editada por todos los miembros.

81. Las opciones de edición de cada conversación están restringidas según roles: creador o líder, invitados.

82. Sólo el líder de la conversación puede cambiar el asunto y expulsar a miembros de la misma.

83. El líder sólo podrá dejar la conversación si antes ha nombrado como nuevo líder a alguno de los miembros invitados.

84. Todos los invitados podrán dejar la conversación en el momento que deseen.

85. Todos los miembros podrán eliminar el historial de mensajes.

86. Todos los miembros podrán invitar a la conversación a usuarios que sean sus contactos.

87. Todos los miembros podrán cambiar el fondo de la conversación y dicha configuración estará asociada a dicho usuario.

88. Las conversaciones son privadas, es decir, ningún usuario puede acceder a ninguna conversación de la que no sea miembro aunque alguno de sus contactos sea miembro.

89. Los emails no son entidades dentro de la aplicación. No se guarda registro de ellos. Simplemente se proporciona una herramienta que permite su redacción y envío.

2.3. Metodología y plan de trabajo

En este Trabajo Fin de Grado hemos seguido una metodología de realimentación o feedback basada en la metodología ágil SCRUM. Se basa en que el producto final es el resultado de una serie de prototipos los cuales han sido implementados en cada iteración del proceso. Podemos ver las etapas de cada iteración en la figura 2.1 a saber: extracción de requisitos, aprendizaje, etapa de diseño, etapa de desarrollo, realización de pruebas y evaluación del producto.

El código fuente de este TFG está disponible en un repositorio público en GitHub y se ha ido documentando en la Wiki del mismo. La documentación incluye manuales de uso de los módulos principales desarrollados, diseño de los objetos en la base de datos y un historial descriptivo de las reuniones realizadas con el tutor. La comunicación con el tutor se ha realizado mediante la plataforma Slack y mediante PivotalTracker se han ido estableciendo y completando las diferentes tareas de cada fase del desarrollo del proyecto.

El plan de trabajo se ha desarrollado en las siguientes fases:

- **Fase1 - Consolidación, búsqueda y aprendizaje**: esta fase corresponde al proceso de extracción de requisitos iniciales y consolidación del concepto

de la aplicación, a la búsqueda de herramientas necesarias y al aprendizaje de dichas herramientas (MeteorJS, Javascript, HTML5, CSS3, SASS, WebRTC y RTCRecorder, APIs Soundcloud, AceEditor, IronRouter y demás paquetes).

- **Fase 2 - Prototipado:** esta fase ha sido la que más tiempo ha requerido y en la que se ha empleado la metodología iterativa descrita anteriormente. En cada iteración se ha enunciado y desarrollado un prototipo cuyo resultado ha servido de base al siguiente. Cada fase ha llevado consigo la realización de las siguientes tareas:

  1. Extracción de requisitos
  2. Extracción de entidades
  3. Diseño de la base de datos
  4. Diseño front-end e Interfaces de usuario
  5. Desarrollo de Interfaces
  6. Pruebas y evaluación

- **Fase 3 - Final:** una vez implementado el prototipo final se ha procedido a la fase final del proyecto que corresponde a la mejora global y su despliegue.

Figura 2.1: Esquema metodología de realimentación

Capítulo 3

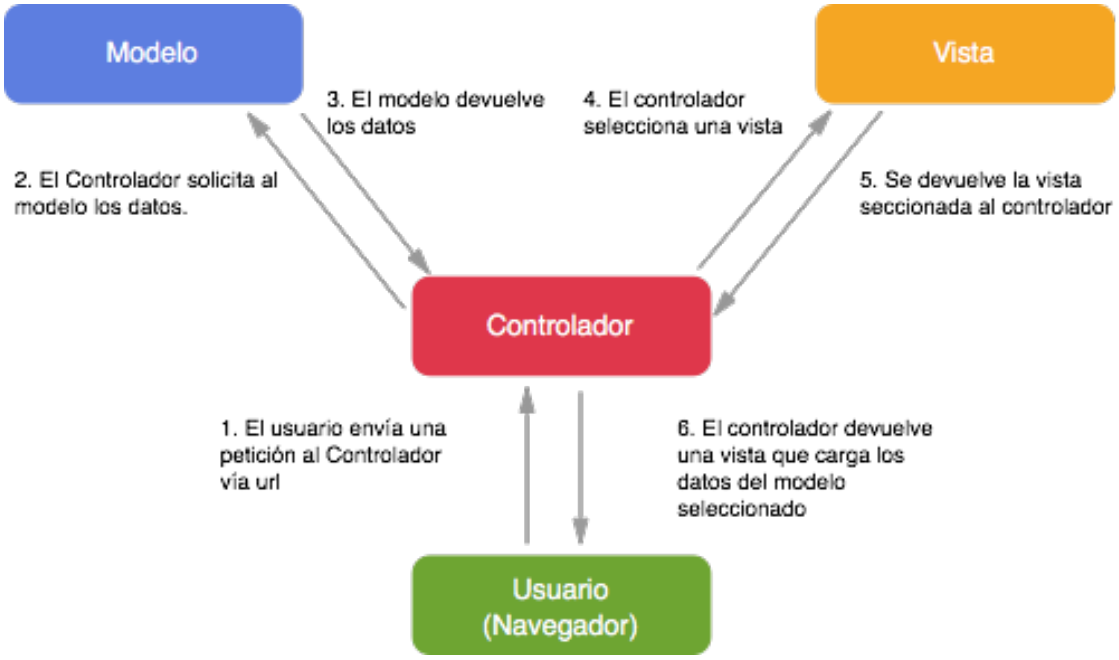
Diseño de la aplicación

El primer paso a la hora de desarrollar cualquier aplicación es el diseño de la misma. Este proceso engloba tanto el diseño visual como la elección de herramientas a utilizar según los requisitos de dicha aplicación. En este capítulo se expone todo lo relacionado con este proceso.

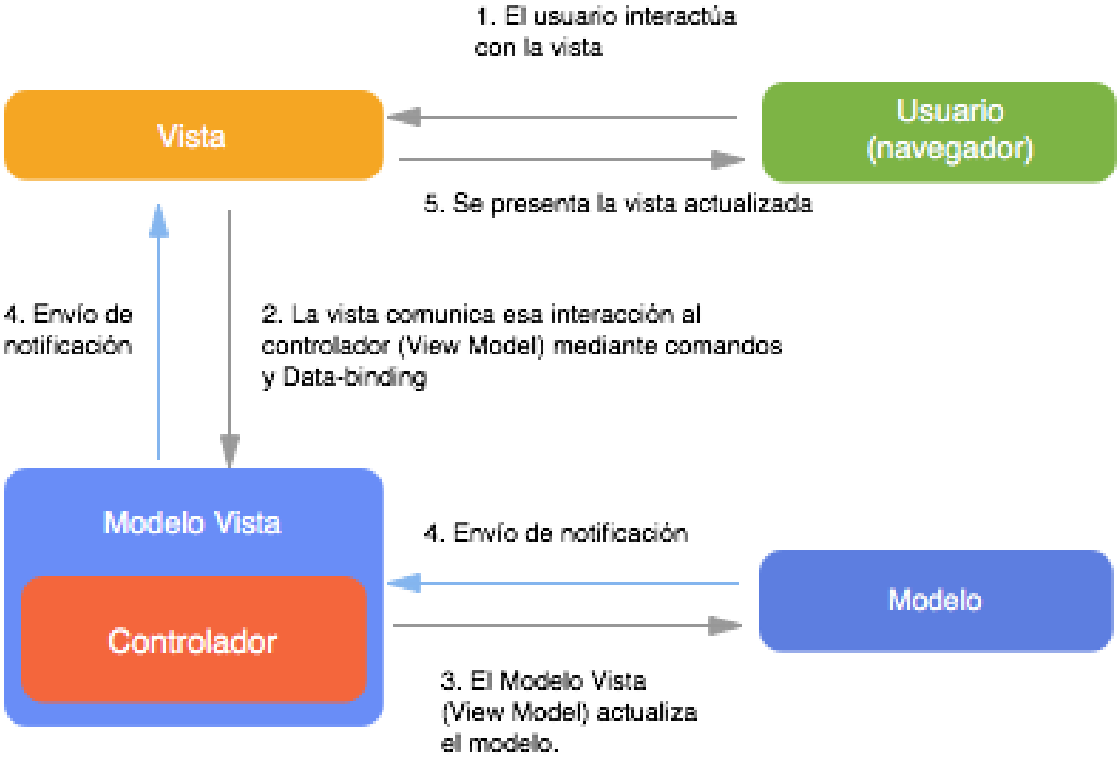
3.1. Tipo de arquitectura

El modelo de arquitectura más habitual es el **MVC** (Modelo Vista Controlador) (figura 3.1(a)). El modelo correspondería con la arquitectura de base de datos y el diseño de la misma, la vista son las interfaces o plantillas que se muestran al usuario y el controlador es el encargado de dotar a la aplicación de lógica y funcionalidad. Existen otros tipos de arquitectura tales como **MVVM** (figura 3.1(b)), donde el controlador del patrón MVC se sustituye por VM o *ViewModel* que establece que cada vista posee lógica y un sistema de *data-binding* entre plantillas.

La elección del patrón de arquitectura a usar es importante puesto que esa decisión nos limitará a la hora de usar determinadas herramientas.



(a) Patrón de arquitectura MVC



(b) Patrón de arquitectura MVVM

Figura 3.1: Patrones de arquitectura

Meteor posee una arquitectura peculiar. En la documentación se expone que posee una arquitectura MVC. Pero, dado que sus vistas poseen lógica y existe data-binding el patrón que según mis conocimientos se basa es MVVM. Por este motivo el patrón de arquitectura utilizado en esta aplicación será el MVVM (Model View ViewModel).

3.2. Búsqueda de herramientas de desarrollo

Partiendo de los requisitos establecidos que caracterizarán la aplicación podemos afirmar que necesitamos encontrar herramientas que nos permitan construir una aplicación en tiempo real, realizar grabaciones de audio, una interfaz con UX como parámetro fundamental y que trabaje de manera óptima.

Aplicación en tiempo real y prototipado rápido

Necesitamos algún framework o plataforma que trabaje con el concepto de reactividad o que lo simule. Elegimos MeteorJS por su flexibilidad y su asombroso concepto de reactividad. Al utilizar esta plataforma no tenemos que preocuparnos de qué frameworks usar para desarrollar el cliente por un lado y el servidor por otro o como gestionar los modelos de datos porque Meteor ya incluye toda la lógica necesario y según su paradigma o patrón de trabajo engloba ambos entornos. Además no hay que declarar el modelo de datos de forma tradicional o como en otras plataformas y frameworks ya que Meteor incluye desde el principio MongoDB como su base de datos.

Para el enrutamiento es precisa otra herramienta que nos proporcione la funcionalidad de especificar a qué recurso pertenece una plantilla y que datos asociamos a ella. Para ello existen varios paquetes para Meteor que realizan este proceso co-

mo IronRouter o FlowRouter. Ambos igual de válidos pero para este proyecto se ha optado por IronRouter, ya que dispone de mayor documentación.

Para el concepto de publicaciones y subscripciones de Meteor usaremos publishComposite, un paquete que permite realizar publicaciones compuestas (varias colecciones con relación de dependencia reactiva) y que siguen manteniendo el principio de reactividad y optimizando nuestro sistema de subscripciones. Sin este paquete realizar esta labor es más compleja.

Otro aspecto a tener en cuenta es la gestión de paquetes. En este ámbito Bower es excelente. Pero usando Meteor no lo necesitamos ya que posee su propio gestor de paquetes y mediante su CLI podemos añadir, actualizar y borrar paquetes de nuestro proyecto.

Grabaciones de audio

La grabación sobre editor con audio constituye el elemento base de esta aplicación. Existen numerosas formas de grabar audio vía web y algunas API de sitios como SoundCloud incorporan un grabador de audio directamente. Aunque esta hubiera sido la vía más rápida, la verdad es que no habría sido la más flexible, ya que el hacerlo de esta manera requería que el uploading se efectuara en SoundCloud. Por este motivo se ha utilizado la tecnología de WebRTC para esta tarea y construido un grabador modular que puede incorporarse facilmente a otros proyectos y que además si se desea usar el servicio de hosting de SoundCloud seguiría siendo factible.

Interfaz con UX como parámetro de diseño fundamental

Toda aplicación debe ser atractiva al usuario y no sólo en términos visuales sino en eficacia a la hora de gestionar acciones. Existen numerorsos frameworks y biblio-

tecas frontend para desarrollar dicha labor correctamente tales como Foundation, Bootstrap o Pure.css. En este proyecto nos hemos decantado por la biblioteca frontend Bootstrap, quizá la más explotada hasta la fecha. Su sistema de maquetación se ha convertido en estándar para el diseño web. También nos hemos visto con la necesidad de dotar de flexibilidad a las plantillas cuando el sistema de columnas de Bootstrap no mostraba los resultados deseados y para ello hemos utilizado la tecnología Flexbox.

Hoy en día trabar con CSS o CSS3 puro se podría considerar un error, ya que existen preprocesadores que permiten una mejor arquitectura para nuestros estilos como son Less y Sass. Ambos son muy parecidos, aunque Sass es mucho más legible. Por ello se han desarrollado las hojas de estilos mediante este preprocesador utilizando un paquete.

3.3. Diseño entidades y base de datos

Esta fase del diseño es primordial pues sin unas entidades bien definidas no sería posible la organización del contenido del sitio ni su correcta presentación. El diseño de entidades se basa en las necesidades del usuario al utilizar la aplicación y lo que queremos que dicha aplicación le aporte. La base de datos será la traducción del modelo entidad relación diseñado a partir de las características de cada entidad y su relación con las demás.

3.3.1. Entidades

Hemos dicho que las grabaciones en esta aplicación son el elemento atómico o base. Por lo que aquí tenemos nuestra primera entidad de la aplicación.

Ahora hay que pararse a pensar y a hacernos preguntas del tipo ¿Qué le gustaría hacer al usuario con esas grabaciones?¿Cómo estarían organizadas?¿Existen límites?¿Cuál es el fin de la grabación?. Esto nos lleva a crear un diseño con un enfoque didáctico y funcional, el cual corresponde con la finalidad de realizar cualquier grabación. Por esto esta aplicación lo que pretende es crear una Comunidad para desarrolladores y cualquiera que le interese el mundo del software. Cada usuario podrá compartir sus conocimientos y sus inquietudes, dudas en lo que a programación se refiere. Vista esta dualidad debemos establecer dos ámbitos en la aplicación. Uno correspondería con el aprendizaje y otro a la enseñanza. En definitiva se basa en el intercambio de conocimientos sobre software.

Por este motivo establecemos las entidades de Canal y de Lección. El elemento canal debe estar basado en un tema específico y cualquier usuario puede contribuir con sus conocimientos sobre el tema. Sería una especie de foro, pero con preguntas y respuestas propuestas mediante grabación de audio sobre editor. El elemento lección se diferencia del elemento canal en que sólo el autor de dicho elemento puede generar contenido de forma directa. Este contenido se generará y se organizará en torno a secciones, una nueva entidad. Las secciones tendrán un título y estará compuestas de una o más grabaciones. Además para una mejor experiencia de usuario estas grabaciones deberán componer una lista de reproducción. Esta lista tendrá las opciones de reproducción automática y de reproducción circular (al finalizar con la última comenzará con la primera de la lista).

Las lecciones deben ser participativas. No es muy didáctico que sólo el creador de la lección sea el que comparta sus conocimientos. Aunque sí que es verdad que los contenidos inmediatos de la lección sólo son creados por el autor, esos contenidos pueden ser respondidos o corregidos mediante una nueva grabación que no formará parte de la sección pero que sí quedará almacenada en la lista de respuestas de

dicha grabación inicial. Con esto surge la necesidad de que a cada grabación pueda grabarse una respuesta.

Otra entidad fundamental son los propios usuarios ya que las mayorias de las entidades que estamos describiendo mantienen una relación directa con ellos.

Ahora que ya tenemos nuestras cuatro entidades básicas: Grabaciones, Canales, Lecciones y Usuarios, podemos hacernos nuevas preguntas sobre funcionalidades de la aplicación y posibles acciones de los usuarios sobre dichas entidades, incluyendo a los mismos usuarios.

Para una mejor experiencia queremos realizar sugerencias de contenido a los usuarios por lo que necesitamos conocer sus pasos a través de la aplicación y una forma de lograrlo es fijándonos en sus gustos. Para ello cada entidad básica podrá ser votada y comentada. Surgen dos nuevas entidades: comentarios y votos. Con esto, las sugerencias se dividirán en dos secciones principales:

- Populares: los más votados
- Recomendados: se filtrarán según las etiquetas de los elementos votados por el usuario.

Puesto que pensamos en la aplicación como una comunidad de desarrolladores, debemos de existir alguna forma de comunicación entre los usuarios a parte de los comentarios y las grabaciones. Aquí surge la entidad de las conversaciones y nuevas preguntas: ¿conversaciones privadas o públicas?¿con qué usuarios puedo establecer una conversación?¿desde dónde? Para responder a estas preguntas necesitamos un punto de partida y ese es la página de perfil de cada usuario. Desde cualquier entidad un usuario podrá acceder a la página de perfil del autor y realizar una petición de contacto. Ahora han surgido dos nuevas entidades: relaciones

y peticiones de contacto (Relations y Requests). Una vez se establezca la relación de contacto entre dos usuarios, éstos podrán mantener una conversación. Con la creación de conversaciones surgen dos nuevas entidades:

- Mensajes (Messages): mensajes que se intercambian en cada conversación.
- Alertas (ConversationAlerts): notificará al usuario de cualquier cambio cuando no se encuentre en la página de la conversación.

Para más información sobre la entidad conversaciones y sus entidades agregadas y módulos ver capitulo ...

Los usuarios deben estar al tanto de lo que ocurre en la aplicación. Para esto se crea el módulo de notificaciones. A cada usuario se le notificará todo lo relacionado a aquello que se subscriba. Por esto se podrá subscribir a cada lección y cada canal y esto supone otra entidad distinta: subscripciones (usersEnrolled). Las notificaciones no se basan solamente en las subscripciones sino que también se le notificará al usuario todo lo relacionado con él. Este módulo se explicará con más detalle en el capítulo 4.

Finalmente, tras este diseño, contamos con las siguientes entidades:

- Grabaciones (Records)
- Documentos(Documents)
- Canales (Channels)
- Lecciones (Lessons)
- Etiquetas (Tags)
- Comentarios (Comments)

- Usuarios (Users)
- Relaciones (Relations)
- Peticiones de contacto (Requests)
- Conversaciones (Conversations)
- Mensajes (Messages)
- Subscripciones (UsersEnrolled)
- Alertas de conversación (ConversationAlerts)
- Notificaciones (Notifications)

3.3.2. Base de datos

Una vez realizado el diseño de entidades, necesitamos conocer las relaciones entre ambas para diseñar la base de datos. Para ello el primer paso es realizar un diagrama entidad-relación, que en nuestro caso es el siguiente: (diagrama entidad relación)

A través de este diagrama podemos extraer la información necesaria para diseñar los documentos de cada entidad que se almacenarán en la base de datos.

Por ejemplo el diseño del documento para una grabación contendrá la siguiente información:

- _id: será el identificador único para el documento en la base de datos.
- author: el identificador del documento usuario creador de dicha grabación.
- description: la descripción.

- RC: será la lista de eventos grabados sobre el editor indexados por una marca de tiempo (instante de grabación).
- createAt: fecha de creación.
- docs_count, votes_count, replies_count, comments_count: contadores para documentos, votos, respuestas y comentarios.
- channel_id: identificador del canal al que pertenece.
- lesson_id: identificador de la lección a la que pertenece.
- section_id: identificador de la sección a la que pertenece.
- order: orden dentro de una lista de reproducción.
- tags: etiquetas
- img: miniatura de la grabación.
- duration: duración de la grabación.
- isReply: indica si es una respuesta a una grabación.
- parent_id: identificador de la grabación a la que responde.
- track: objeto con los parámetros de identificación del audio almacenado en SoundCloud.

En el apéndice A se encuentran todos los diseños de los documentos para cada una de las entidades.

3.4. Organización del contenido e interfaces

Una vez establecidas las entidades que compondrán la aplicación, conocidas sus conexiones y traducidas éstas en documentos para la base de datos, es necesario diseñar interfaces que permitan organizar el contenido de la aplicación y la correcta comunicación y flujo de navegación entre dichas entidades. Esta organización del contenido la vamos a realizar a través de los recursos de nuestra aplicación y las conexiones a través del enrutamiento a dichos recursos. Por este motivo debemos diseñar correctamente la jerarquía de recursos. Para esto establecemos la siguiente categorización: recursos principales, de detalle, de creación, de configuración y especiales.

3.4.1. Recursos principales

/ o raíz

Corresponde a la página de inicio o página inicial de la aplicación. Este recurso será dual en su contenido. Esta dualidad dependerá del estado en el que se encuentre el usuario.

Si el usuario no ha iniciado sesión se mostraría la página de acceso a la aplicación donde se encontraría un formulario dinámico con la funcionalidad de inicio de sessión o de registro, además un enlace para poder recuperar la contraseña. También se mostraría un resumen de las funcionalidades de la aplicación y enlaces para los recursos de /tutorials y /features.

Si el usuario ha iniciado sesión o se ha registrado (una vez se ha creado el usuario se realiza el inicio de sesión) se visualizaría la página raíz de la aplicación. En este momento la mayoría de recursos compartirán el mismo diseño base que consta

de un sidebar y de un contenedor para el contenido asociado. La idea de utilizar un sidebar es proporcionar al usuario un acceso rápido a todos los contenidos incluyendo los creados por él (recursos principales y recursos de detalle propios del usuario) y a las notificaciones y alertas asociadas a la acción de otros usuarios y de él mismo. Además de la opción de poder cerrar sesión en cualquier momento. El contenido asociado a este recurso constará de un espacio para recomendaciones basadas en la navegación del usuario dentro de la aplicación, un módulo ofreciendo el acceso directo a los contenidos más populares y un espacio para realizar búsquedas rápidas y dinámicas. El motor de búsqueda se basará en auto-completados dinámicos asociados a etiquetas predefinidas que categorizarán la búsqueda.

/channels, /lessons y /records

Proporciona acceso a todos los canales de la aplicación mediante una lista y al recurso de creación /channels/submit que permite la creación de un nuevo canal. Se podrán visualizar los canales ordenados por número de votos (populares) o por fecha de creación inversa (más actuales). Además el motor de búsqueda se encuentra accesible en esta página, aunque restringida la búsqueda sólo para mostrar canales. La misma funcionalidad la encontramos para los recursos /lessons y /records aunque asociada al tipo de contenido correspondiente.

3.4.2. Recursos de detalle

Esta categoría abarca todos los recursos destinados a un contenido específico. En cada uno de estos recursos se realizarán restricciones en función del rol del usuario que los visite: creador o visitante.

/channels/:id

Este recurso corresponderá al canal que posea el identificador correspondiente al mismo. El contenido estará formado por una cabecera o header y por un sistema de tabs.

La cabecera constará de dos imágenes (una destinada para la miniatura del canal a la hora de mostrarlo como elemento de una lista y otra como elemento de personalización), una caja para mostrar información sobre el autor, la fecha de creación y los contadores, botones para votar el canal y aceptar/cancelar subscripción, un enlace al recurso de edición del canal, una descripción y la lista de etiquetas del canal.

El sistema de tabs estará formado por tres tabs: grabaciones, comentarios y usuarios.

- Grabaciones: miniaturas de las grabaciones asociadas al canal. Además un enlace al recurso de creación de grabaciones.
- Comentarios: comentarios del canal (se podrán comentar a su vez)
- Usuarios: lista de usuarios subscritos al canal. Cada elemento será un enlace al perfil de cada usuario.

/lessons/:id

Este recurso es similar al recurso destinado a un canal, por lo tanto tendrá la misma estructura. Una cabecera y un sistema de tabs. La cabecera mostrará el mismo contenido que el de un canal aunque visualmente será diferente. El sistema de tabs estará formado por tres tabs: secciones, comentarios y usuarios. En la tab

secciones se mostrará la lista de secciones que constituyen el contenido de la lección en sí.

Cada sección es un elemento editable y contendrá grabaciones. Por lo que cada elemento sección tendrá un título, un orden dentro de la lección, un enlace para reproducir el contenido, un enlace para crear nuevas grabaciones dentro de la sección, contadores y otro sistema de tabs formado por dos tabs: uno que mostrará la lista de grabaciones y otro para configuración (se podrán remover o cambiar de orden las grabaciones correspondientes a esa sección).

Al contrario que los canales, a las lecciones sólo el creador podrá añadirles contenido directamente y sólo los usuarios que se subscriban podrán explorar el mismo. Las subscripciones en los canales son para recibir notificaciones sobre nuevo contenido o como resultado de algún voto.

/records/:id

Este recurso se organizará de forma similar a los anteriores (cabecera y sistema de tabs), aunque diferirá en gran parte su contenido y su funcionalidad. La cabecera estará compuesta por un título, una descripción, un botón para realizar votos, un reproductor y una caja de autor que contendrá información sobre el autor, contadores para la grabación y la fecha de creación. Además una serie de enlaces:

- Enlace al canal/lección del que procede (si forma parte su contenido).
- Enlace a la grabación de la que es respuesta (si es una respuesta).
- Enlace al recurso creador de grabaciones para realizar una nueva grabación como respuesta. (sólo disponible cuando la reproducción ha terminado o ha sido pausada).

El sistema de tabs contendrá las siguientes tabs:

- Respuestas: se mostrará un timeline con las respuestas ordenadas según su instante de inicio correspondiente al instante de reproducción de la grabación.
- Comentarios: lista de comentarios para la grabación.
- Relacionados: lista de grabaciones relacionadas (contienen etiquetas comunes).

/profile/:id

Este recurso está destinado para mostrar los contenidos de cada usuario y funcionalidades destinadas tanto al propio usuario como al visitante como:

- usuario dueño: verificación de email, cambio de contraseña, acceso al perfil del servicio asociado (si lo hay), acceso al recurso de edición del perfil, acceso al tab conversaciones y al filtro peticiones del tab contactos, acceso al recurso de creación de conversaciones, canales, lecciones y grabaciones, acceso al recurso especial para enviar emails a los contactos con email y acceso al recurso especial para gestionar el borrado del contenido del propio usuario en la aplicación.
- usuario visitante: botón para enviar una solicitud de contacto (si es que no lo es ya)/ botón para enviar un email y crear una nueva conversación con el usuario (si es que ya se ha establecido la relación de contacto) y acceso a la lista de contactos, canales, lecciones y grabaciones.

Visualmente, en la cabecera, se mostrará un banner y una foto de perfil personalizable.

/conversations/:id

Este recurso corresponderá a una conversación en concreto. La interfaz mostrará un asunto para la conversación, una lista de miembros, una lista de opciones (salir, añadir nuevos miembros, borrar historial de mensajes y echar a un miembro que sólo tendrá acceso el líder de la conversación), la lista de mensajes (los del usuario actual a la derecha y los demás a la izquierda) y un espacio para escribir cada mensaje en el que se podrán incorporar enlaces y emoticonos.

3.4.3. Recursos de creación

Estos recursos están destinados a la creación de nuevos elementos dentro de la plataforma. Por lo tanto habrá uno por cada tipo de entidad elemental o que posean un recurso detalle asociado.

/channels/submit

/lessons/submit

/records/submit

/conversations/submit

3.4.4. Recursos de configuración (settings resources)

/channels/:id/edit

/lesson/:id/edit

/conversation/:id/edit

3.4.5. Recursos especiales (special resources)

/verifications

/verify-email/:token

/forgot-password

/recover-password/:token

/change-password

/send-email

/redirect

/not-found

3.5. Grabación, reproducción y respuesta sobre editor

Como se ha dicho las grabaciones constituyen el elemento base del contenido de la aplicación. Por este motivo es necesario diseñar correctamente el funcionamien-

to del grabador y del reproductor. El usuario podrá crear una nueva grabación asociada a un canal o a una sección dentro de una lección o bien una grabación totalmente independiente. Además podrá realizar respuestas a estas grabaciones mediante una nueva grabación comenzando el proceso en cualquier punto de su reproducción. Esto hace que sea posible la explotación del contenido de la aplicación.

El primer paso es distinguir entre audio y video. El audio se grabará desde el micrófono del usuario y el vídeo mostrará el proceso de edición sobre un editor de código. Puesto que en la actualidad no existen herramientas a la hora de realizar una grabación sobre un elemento HTML a no ser que posea la etiqueta de canvas de HTML5, la grabación consistirá en realizar una captura de los eventos sobre el editor e indexarlos mediante el instante en el que ocurren durante la misma.

El separar audio y vídeo en el proceso de grabación supone un problema a la hora de realizar la reproducción puesto que es necesario realizar la sincronización entre los dos elementos. Pero, con un buen enfoque, se puede lograr la sincronización completa.

Al realizar la búsqueda de herramientas elegimos una tecnología para grabar el audio (RTCRecorder) y un editor de código online que proporcionaba una API muy completa y sencilla (Ace Editor). Las características realmente útiles en este contexto de estas dos tecnologías son:

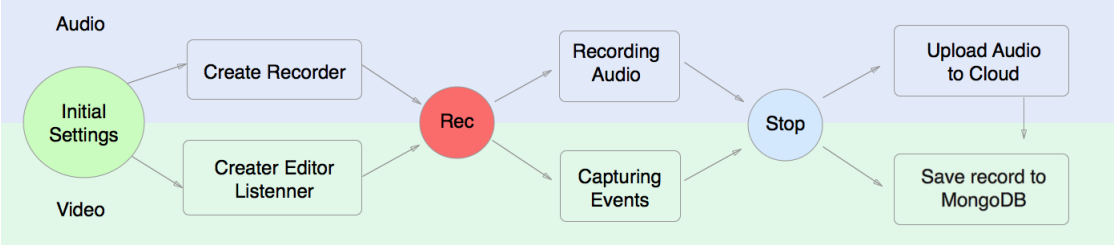
- RTC Recorder: utiliza HTML5 Media para captar los servicios necesarios del usuario (acceder al micrófono) y para generar un grabador que mediante métodos nos permitirá comenzar la grabación, pausarla, finalizarla y conseguir el audio resultante en un objeto tipo Blob que podremos subir a nuestro servicio de almacenamiento en la nube.

- Ace Editor: el API nos proporciona métodos para generar un editor a partir de un identificador HTML y poder capturar sus eventos y almacenarlos en una lista indexados, como se ha dicho, por su instante durante la grabación. Además podremos elegir el lenguaje de programación y el tema que queramos para nuestras grabaciones.

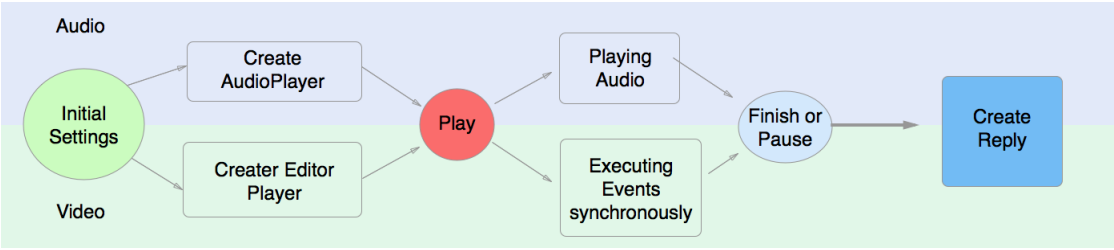
Para hacer más completas cada grabación, se ha trabajado con la idea de grabación de documentos, es decir, cuando queramos realizar una grabación, deberemos crear al menos un documento o archivo con un título único e identificativo, escogeremos el lenguaje de programación y el tema en el editor. De esta forma una grabación estará compuesta por uno o más documentos. Por lo que hay que manejar los eventos creación y cambio de documento durante el proceso de grabación para que se muestre el cambio durante su reproducción. Ni que decir tiene que este enfoque afectará a la hora de realizar una respuesta.

Las respuestas al igual que las grabaciones normales se realizarán en el recurso creador asociado /records/submit. Por este motivo será necesario dotar de lógica a la plantilla o interfaz capaz de diferenciar si se trata de una respuesta o de una grabación normal, y además se debe mantener el estado aunque el usuario refresque la página o retorne. Para ello se utilizarán queryStrings en el enrutamiento y la plantilla cargará unos datos u otros dependiendo de su existencia.

El progreso de grabación y reproducción quedan reflejados al igual que el flujo de grabaciones y respuestas en la figura 3.2.



(a) Proceso de grabación



(b) Proceso de reproducción

Figura 3.2: Procesos de grabación, reproducción y esquema de flujo.

3.6. Cloud Storage

Necesitamos un servicio de almacenamiento de audio en la nube. Para ello hemos elegido el servicio de SoundCloud. SoundCloud nos proporciona una API REST muy sencilla de utilizar mediante su sdk. Esta nos permitirá subir archivos tipo blob que acabamos de grabar, crear un stream para una pista de audio concreta y borrar contenidos de manera limpia. Para utilizar su servicio es necesario la autenticación del usuario y una aplicación que crearemos en el espacio para desarrolladores de soundcloud. A esta aplicación se le atribuirán unos credenciales que permitirán el acceso al API REST de SoundCloud. La configuración de dicha aplicación y el servicio dentro de nuestra plataforma se explicará con más detalle en el capítulo 4.

Capítulo 4

Desarrollo de la aplicación

El desarrollo de cualquier aplicación viene después de su diseño. No necesariamente es el paso final, puesto que esta etapa sirve de feedback en el proceso general y muchas veces hay que retomar el diseño para establecer nuevos requisitos o cambiar los actuales.

4.1. Aprendizaje

Una vez diseñada la aplicación y establecidas las herramientas a utilizar, el siguiente paso es profundizar en cada una de ellas. El aprendizaje de nuevas herramientas es lento pero el afianzamiento de los conocimientos necesarios de manera correcta nos evitará muchos problemas en el futuro. Por esto hay que esforzarse en comprenderlos y la primera fase del Desarrollo de la aplicación es el aprendizaje.

4.1.1. CSS3 y preprocesadores

Aquí explicaré Qué he aprendido de css3 y de los preprocesadores less y saas.

4.1.2. API SoundCloud

Aquí explicaré el servicio de almacenamiento de SoundCloud.

4.2. Componiendo el escenario

Pequeña introducción

4.2.1. Entorno de desarrollo

WebStorm. Foto de la interfaz.

4.2.2. Paquetes

Lista de paquetes necesarios y sus funcionalidades

4.2.3. Collecciones

Explicar cómo se han desarrollado las entidades en colecciones y la estructura de los objetos guardados. Cómo se relacionan.

4.2.4. Enrutamiento

Explicar Iron Router y su potencial.

4.2.5. Mixins iniciales

Foto de los mixines iniciales y que son necesarios.

4.3. Layout Principal

Estructura general de la vista de la aplicación.

4.4. Registro de usuarios

Pequeña introduccion hablar de los servicios de meteor automáticos para sign.

4.4.1. Sign In y Sign Up

Funcionalidad y pantallas

4.4.2. Verificación de Email

Funcionalidad y pantallas

4.4.3. Cambio y recuperación de contraseña

Funcionalidad y pantallas

4.4.4. Servicios agregados

Funcionalidad y pantallas.

4.5. Sidebar

Explicar que contenido se muestra en sidebar y que es el enlace a todos los recursos de la aplicación desde cualquier pantalla.

4.6. Formularios

Pequeña introducción enumerar los formularios creados.

4.6.1. Diseño de plantilla dinámica

Explicar cómo se crea una plantilla dinámica para los formularios.

4.6.2. Creadores de entidades

Enumerarlos y pantallas.

4.6.3. Conversaciones

Hablar del componente inputMembers.

4.6.4. Grabaciones

Hablar del grabador. (extenso).

4.7. Entidades

Hay que dar forma a las entidades visualmente.

4.7.1. Canales

Hablar de los canales.

4.7.2. Lecciones

Hablar de las lecciones.

4.7.3. Secciones

Hablar de las secciones.

4.7.4. Grabaciones

Hablar de la reproducción de las grabaciones. (extenso)

4.7.5. Listas de reproducción

Asociadas a una sección.

4.7.6. Conversaciones

Hablar de las conversaciones y de los roles.

4.8. Perfil de usuario

4.8.1. Rol dueño

4.8.2. Rol visitante

4.8.3. Contactos

4.9. Tutoriales

4.10. Landing

4.11. BrowseCrossing

4.12. Full Responsive

4.13. Despliegue

Capítulo 5

Experimentación

5.1. Motivación

5.2. Planteamiento y objetivos

5.3. Proceso y realización

5.4. Resultados

Capítulo 6

Conclusión

Bibliografía

[1] Página oficial WebRTC: https://webrtc.org/

[2] Página para WebRTC experiments (recordRTC): https://www.webrtc-experiment.com/RecordRTC/

[3] Página oficial SoundCloud: https://soundcloud.com

[4] Página para desarrolladores SoundCloud: https://developers.soundcloud.com/

[5] Página oficial MeteorJS: https://www.meteor.com/

[6] Guía de MeteorJS: http://guide.meteor.com/

[7] Documentación de MeteorJS: http://docs.meteor.com/#/full/

[8] Página oficial Bootstrap: http://getbootstrap.com/

[9] W3C javascript tutorial: http://www.w3schools.com/js/

[10] Documentación de MongoDB: https://docs.mongodb.com/manual/

[11] Página oficial JQuery: https://jquery.com/

[12] Documentación de JQuery: http://api.jquery.com/

[13] Documentación de UnderscoreJS: http://underscorejs.org/

6.1. Principales Sitios Web

6.2. Libros

6.3. Artículos

6.4. Tutoriales

6.5. Paquetes

6.6. Repositorios

Apéndices

Apéndice A

Diseño de documentos para Mongo

```
var record = {
    _id: //idMongo,
    author: //idUser,
    title: //no único,
    description: //(opcional),
    RC: [{},...], //Funciones de reproducción sobre el
       editor
    createAt: //fechaCreación,
    docs_count: //contador de documentos,
    votes_count: //contador de votos,
    replies_count: //contador de respuestas,
    comments_count: //contador de comentarios,
    channel_id: //canal al que pertenece,
    lesson_id: //lección a la que pertenece,
    section_id: //sección a la que pertenece dentro de una
        lección,
```

```
15      order: ,//orden dentro de la lista de reproducción.
16      tags: [{},...], //etiquetas,
17      ready: //(boolean) para conocer la disponibilidad del
            record.
18      img: //imagen miniatura,
19      duration: //duración en milisegundos de la grabación.
20      isReply: //(boolean) indica si se trata de una
            respuesta a otro record.
21      parent_id: //idMongo del record al que responde.
22      track: //{_id: 'id del track en SoundCloud', link: '
            link SoundCloud'}
23  };
```

A.1: Diseño de documento para una grabación

```
1   var doc = {
2       _id: //idMongo,
3       record: //record al que pertenecen,
4       doc: {
5            title: //titulo del documento (único para el
                record),
6            theme: //tema del editor tras último cambio,
7            mode: //tema del editor tras el último cambio,
8            value: //último estado del contenido del editor
9       },
10      start: //(boolean) (True)? comienzo grabación : se ha
            creado durante la grabación
11              //o es el estado final de otro inicial.
12  };
```

A.2: Diseño de documento para los documentos de cada grabación

```
var channel = {
    _id: //idMongo,
    author: //id_user,
    title: //único,
    banner: //url img banner,
    img: //url img miniatura,
    description: //(opcional),
    tags: //etiquetas [{name: //nombre etiqueta}],
    createAt: //fechaCreación,
    votes_count: //contador para los votos,
    records_count: //contador para los records,
    comments_count: //contador para los comentarios,
    users_count: //contador para los usuarios que se
       subscriban
};
```

A.3: Diseño de documento para un canal

```
var lesson = {
    _id: //idMongo,
    author: //id_user,
    title: //único,
    img: //url img miniatura,
    description: //(opcional),
    tags: //etiquetas [{name: //nombre etiqueta}],
    createAt: //fechaCreación,
    votes_count: //contador para los votos,
    sections_count: //contador para las secciones,
    comments_count: //contador para los comentarios,
    users_count: //contador para los usuarios que se
```

```
        apunten
};
```

A.4: Diseño de documento para una lección

```
var section = {
    _id: //idMongo,
    title: //único,
    createAt: //fechaCreación,
    records_count: //contador para los records,
    lesson_id: //id de la lección a la que pertenece.
    order: //orden de la sección.
};
```

A.5: Diseño de documento para una sección

```
var userEnrolled = {
    _id: //idMongo,
    contextId: //id del canal o de la lección a la que se
       han subscrito,
    user_id: //id del usuario
};
```

A.6: Diseño de documento para cada subscripción de un usuario

```
var tag = {
    _id: //idMongo,
    name: //nombre para la etiqueta
};
```

A.7: Diseño de documento para una etiqueta

```
var conversation = {
```

```
2     _id: //idMongo,
3     subject: //asunto de la conversación,
4     author: //líder de la conversación.
5     last_modified: //fecha de ultima modificación (cada
          vez que se inserta un mensaje),
6     members: //[{},...], array de miembros,
7     members_count: //contador para los miembros,
8     messages_count: //contador para los mensajes
9 };
```

A.8: Diseño de documento para una conversación

```
1 var message = {
2     _id: //idMongo,
3     author: //id del creador,
4     createdAt: //fecha de creación,
5     message: //contenido del mensaje
6     conversation_id: //id de la conversación a la que
          pertenece.
7 }
```

A.9: Diseño de documento para los mensajes

```
1 var conversationAlert = {
2     _id: //idMongo,
3     user_id: //usuario al que se le muestra la alerta,
4     conversation_id: //id de la conversación de la que
          procede,
5     alertsAllow: //flag (boolean) si es true se muestran
          las alertas y si es false no.
6     alerts_count: //contador de alertas
```

```
};
```

A.10: Diseño de documento para una alerta de conversación

```
var user = {
    _id: //idMongo,
    username: //nombre de usuario (único),
    avatar: //avatar del usuario,
    banner: //banner de la pagina de usuario,
    description: //descripción del usuario,
    status: //estado de conexión,
    emails: //[{address: //emailAddress,verified: //
       Boolean,},...],
    ... //otros campos establecidos por meteor-accounts.
};
```

A.11: Diseño de documento para cada usuario

```
var request = {
    _id: //idMongo,
    requested: //{id: id_user, delete: boolean},
    applicant: //{id: id_user, delete: boolean},
    status: //(string) 'pending', 'accepted', 'refused'
};
```

A.12: Diseño de documento para las peticiones de contacto

```
var relation = {
    _id: //idMongo,
    createdAt: //fecha creación,
    users: //[id_user_requested, id_user_applicant],
```

```
};
```

A.13: Diseño de documento para establecer la relación de contacto

```
var comment = {
    _id: //idMongo,
    author: //id del creador,
    createdAt: //fecha de creación,
    isReply: //(Boolean) indica si es una respuesta o no,
    replies_count: //contador de respuestas,
    message: //contenido del mensaje
};
```

A.14: Diseño de documento para los comentarios

```
var notification = {
    _id: //idMongo,
    to: //id_user destinatario,
    from: //id_user origen,
    parentContextTitle: //es el título de la lección,
       record o channel en el que se ha producido,
    urlParameters: //son los parámetros para construir el
       enlace al clickar sobre la notificación,
    type: //(string) 'channel', 'record', 'comment', etc,
    message: //(string) de contenido HTML
};
```

A.15: Diseño de documento para una notificación