UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

BOOTSTRAPPING DATABASES EN EQUIPOS MÓVILES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

CARLOS ANDRÉS GAJARDO MAUREIRA

PROFESOR GUÍA:
JÉRÉMY BARBAY

MIEMBROS DE LA COMISIÓN:
BENJAMÍN BUSTOS CÁRDENAS
JAVIER BUSTOS JIMÉNEZ

SANTIAGO DE CHILE
AGOSTO DE 2012

Resumen

Un *Boostrapping Database* es un motor de base de datos colectivo que intenta dar solución al problema que enfrentan las comunidades en línea con respecto al control de la calidad del contenido y la colaboración de sus miembros; *Repositorium* es una implementación de *Boostrapping Database* que permite a cualquier usuario iniciar una comunidad para compartir documentos de texto, actualmente es utilizado en contextos académicos y se busca abrir la plataforma a nuevas aplicaciones que aprovechen la estrategia de participación propuesta; en particular se busca facilitar el desarrollo de aplicaciones móviles en torno a *Repositorium*.

Por consiguiente, el objetivo de esta memoria es dotar a *Repositorium* de una interfaz que permita a cualquier desarrollador implementar una aplicación móvil utilizando los datos y el modelo de participación disponibles. Para lograr éste objetivo se diseñó, implementó y documentó una API web que establece un canal de comunicación claro y consistente entre las aplicaciones móviles y *Repositorium.*

Para mostrar la eficacia de la API web construida se desarrolló una aplicación móvil que, utilizando la API, logró dar respuesta a los desafíos que presentaba una comunidad en línea y de paso mostró un ejemplo de uso en un contexto no académico. Además la herramienta fue presentada a los alumnos del Taller de Proyectos de Programación Android, quienes se mostraron abiertos a trabajar con ella.

Como resultado hoy *Repositorium* provee las herramientas para que desarrolladores de aplicaciones móviles puedan hacer uso de los datos, el modelo de participación y la estrategia de control de calidad disponibles.

Dedicatoria

Dedico este trabajo a María, Katherine y Solange; las mujeres más importantes en mi vida.

Agradecimientos

Quiero agradecer en primer lugar a mi familia quienes supieron soportar mi mal humor cuando el stress alcanzaba su máximo nivel y quienes constantemente me recuerdan el valor del trabajo y el esfuerzo.

Al profesor Jérémy, por la oportunidad de trabajar con él, por creer en mi y por el apoyo que me entregó durante el desarrollo de este trabajo.

A mis amigos DCC quienes han sido, y probablemente sigan siendo, un pilar de apoyo fundamental en los momentos en que nada parece tener sentido y con quienes disfruto mucho los momentos de distensión.

A mis amigos de primer año quienes siempre me instaron a seguir adelante cada vez que quise abandonar.

Índice

1 Introducción 1
1.1 Conceptos básico 2
1.1.1 Crowdsourcing 2
1.1.2 Bootstrapping Database 3
1.1.3 Repositorium 5
1.2 Motivación 6
1.3 Objetivos 7

2 Antecedentes 8
2.1 Estado actual de Repositorium 8
2.2 Conceptos básico sobre API Web 11

3 Análisis 13
3.1 Cliente Modelo 14
3.2 Requerimientos 15

4 Diseño e implementación de una API para Repositorium 18
4.1 Buenas prácticas en el diseño de API 18
4.2 Decisiones Arquitectónicas 20
4.2.1 Elementos y su representación 23
4.2.2 Recursos y rutas 26
4.3 Implementación 29
4.4 Documentación 35
4.5 Análisis 38

5 Diseño e implementación de una aplicación cliente 43
5.1 Contexto 44
5.2 Diseño e implementación 46

6 Conclusiones 52
6.1 Logros 52
6.2 Trabajo futuro 53

7 Bibliografía 54
Anexo A. Modelo de datos en Repositorium 56
Anexo B. Descripción de los casos de uso para el cliente modelo 57
Anexo C. Descripción extensa de los Recursos disponibles 62
Anexo D. Tabla de relación entre Casos de Usos y Recursos 69

Índice de figura

Figura 1: Diagrama de arquitectura MVC …… 10

Figura 2: Arquitectura simple de una API web …… 12

Figura 3: Casos de uso aplicación modelo. Requisitos críticos y deseables 16

Figura 4: Casos de uso aplicación modelo. Requisitos opcionales ……...17

Figura 5: Arquitectura propuesta para la API de Repositorium 22

Figura 6: Ejemplo test JMeter, petición HTTP 30

Figura 7: Ejemplo test JMeter, Aserción..... 31

Figura 8: Diagrama de clases, capa Enrutador ……..33

Figura 9: Diagrama de clases, capa Controlador 34

Figura 10: Diagrama de clases, capa Modelo 35

Figura 11: Ejemplo documentación API Facebook 36

Figura 12: Ejemplo documentación API Twitter 37

Figura 13 : Capturas de pantalla de la documentación 38

Figura 14 : Gráfico tiempo promedio total 39

Figura 15 : Tiempo promedio por petición..... 42

Figura 16 : Promedio tiempo total II 43

Figura 17: Captura inicio Fondue 45

Figura 18: Flujo de navegación con BSDB……47

Figura 19: Inicio aplicación móvil Fondue 49

Figura 20 : Desafío en aplicación Fondue..... 50

Figura 21 : Calibrar puntajes Fondue 50

Figura 22 : Puntaje Fondue 51

Índice de tablas

Tabla 1 : Listado de recursos para Repositories accesibles mediante GET 28

Tabla 2 : Listado de recursos para Repositories 28

Tabla 3 : Listado de recursos para Users..... 29

Tabla 4 : Tiempos promedio Test de esfuerzo 40

1 Introducción

Este trabajo de título se enmarca en las áreas de computación móvil y trabajo colaborativo en línea. Tiene por objetivo construir una base de trabajo común entre la aplicación web *Repositorium* y las aplicaciones para dispositivos móviles.

La computación móvil puede ser definida como el uso, mientras se está en movimiento, del poder computacional a través de dispositivos especialmente diseñados, como *laptops*, *smartphones* o *tablets*. El estado actual de las redes de comunicación móvil y del desarrollo de aplicaciones permite a los usuarios interactuar con servidores remotos para recuperar datos o completar tareas mientras se está en marcha, haciendo de la computación móvil una experiencia mucho más completa.

Repositorium es un motor de *Bootstrapping Database* que permite a cualquier usuario crear una comunidad para compartir documentos y controlar la calidad de estos. Los usuarios pueden unirse libremente a una comunidad, sin embargo, para obtener acceso a los documentos del repositorio deben aportar nuevos documentos o evaluar la calidad de los documentos existentes.

El conjunto de todos los documentos en una comunidad conforman un repositorio, dentro del cual se definen criterios de calidad parar los documentos. Cada comunidad es libre de elegir sus propias reglas, propósitos y tipos de archivo a compartir, de manera que los repositorios varían tanto en tema como en contenido.

El modelo de participación de *Repositorium* requiere que los usuarios realicen algunas tareas de evaluación y validación para obtener acceso a los datos almacenados en el repositorio; estas tareas pueden ser fácilmente realizadas a través de un *smartphone* o una *tablet* con conexión a Internet, permitiendo a los usuarios aprovechar cualquier ventana de tiempo para interactuar con el repositorio.

Como los temas y lo contenidos varían de comunidad en comunidad, una aplicación móvil de propósito general resulta ser un trabajo costoso en términos de esfuerzo y tiempo, por lo que ofrecer a los desarrolladores una interfaz para crear aplicaciones de acuerdo a las necesidades de cada comunidad parece una alternativa más razonable, y de paso permitiría nuevos usos a la plataforma *Repositorium*.

1.1 Conceptos básico

A continuación se define y describe el concepto de *Crowdsourcing* que permitirá comprender mejor la naturaleza de *Bootstrapping Databases* y por consiguiente facilitará el entendimiento de *Repositorium*.

1.1.1 Crowdsourcing

Es común que en grupos de trabajo las tareas sean asignadas a los usuarios de acuerdo a su rol dentro del equipo. Este modelo de trabajo, si bien permite la especialización de las personas, desestima el potencial que existe en aquellos con un rol menos influyente.

El trabajo por roles ha influenciado la manera en que las personas colaboran en Internet; en comunidades virtuales por ejemplo, las tareas de control y moderación de contenido son asignadas a un pequeño grupo de usuarios, a quienes se les asigna el rol de “moderadores”.

La Web 2.0 vino a descartar algunos de estos paradigmas. Comunidades como *Youtube*[1] confían en el criterio de todos sus participantes para moderar los contenidos y los comentarios, eliminando la noción de roles y entregándole la responsabilidad a la comunidad completa.

[1] Youtube: sitio web el en cual los usuarios pueden compartir video. Se puede acceder en http://www.youtube.com

Emulando estas estructuras de participación nuevos modelos de trabajo han sido propuestos. *Crowdsourcing* es uno de ellos. En este modelo las tareas son presentadas a un número indeterminado de personas, confiando en el conocimiento y las habilidades del grupo para hallar la mejor solución. *Amazon Mechanical Turk*[2] o *ImageBrief* [3] son ejemplos de *Crowdsourcing* en Internet.

Un ejemplo interesante de *Crowdsourcing* es el servicio *ReCaptcha*[4]*,* en él los usuarios deben responder pequeños desafíos que no toman más de un par de segundos [**1**] y que permiten completar, colaborativamente, una tarea que podría tomar mucho tiempo como es la transcripción de un texto. Para separar a los usuarios maliciosos de los "buenos usuarios" *ReCaptcha* presenta dos palabras, para una conoce la respuesta correcta, para la otra no. En caso de que el usuario no transcriba correctamente la palabra conocida, su respuesta no se considerará válida y no se le concederá acceso al recurso que solicita.

1.1.2 Bootstrapping Database

Un *Boostrapping Database*, BSDB de ahora en adelante, es un motor de base de datos colectivo que, usando los principio de *Crowdsourcing*, intenta incentivar la participación de los miembros de una comunidad al tiempo que se controla la calidad del contenido.

---

[2] Amazon Mechanical Turk (MTurk) es un mercado *Crowdsourcing* en Internet donde se publica y se postula a trabajos relacionados con inteligencia humana. Más información en https://www.mturk.com

[3] ImageBrief es una comunidad online que permite a los usuarios delegar la tarea de buscar una imagen en particular al resto de la comunidad, pagando una "recompensa" a quien halle la imagen adecuada. Para más información visitar el sitio web http://www.imagebrief.com

[4] ReCaptcha es un servicio de detección de *bots* el cual solicita a los usuarios reconocer un texto para intentar filtrar usuarios no humanos y ayudar al mismo tiempo a la digitalización de libros. Más información en http://www.google.com/recaptcha

El concepto de BSDB, creado por el profesor del Departamento de Ciencias de la Computación de la Universidad de Chile, Jérémy Barbay, plantea un modelo de participación en el cual los usuarios deben realizar un pago, en tiempo y esfuerzo, para conseguir acceso a los nuevos contenidos [2]. Tal pago se traducen en aportar nuevo material o realizar pequeñas tareas de control de calidad. Esto marca la principal diferencia con otro ejemplos de *Crowdsourcing*.

En los esfuerzos previos a BSDB los sistemas de *Crowdsourcing* recompensaban el esfuerzo del usuario con dinero (*Mechanical Turk*, *ImageBrief*) o con la prestación de un servicio de identificación (*ReCaptcha*). En BSDB la recompensa del usuario es el resultado del trabajo de toda la comunidad; un usuario recibirá un documento de buena calidad en la medida de que cada miembro haya aportado documentos y/o supervisado la calidad de los aportes del resto.

La tarea se supervisar los aportes se logra a través de un *Protocolo de Control de Calidad* en donde se definen criterios de calidad y se construyen pequeños desafíos; además se permite calibrar el sistema marcando ciertos documentos como validados, positiva o negativamente, para un criterio determinado y de esta manera intentar separar a los usuarios que responden el desafío de manera coherente, de aquellos que no.

El *Protocolo de Control de Calidad* de BSDB funciona entonces de manera similar a como lo hace *ReCaptcha*, presentando desafíos formados por preguntas de control (para los documentos validados) y preguntas libres (para los documentos no validados); si el usuario responde incorrectamente a las preguntas de control sus respuestas en las preguntas libres no se considerarán válidas y en la siguiente oportunidad que deba responder un desafío, éste aumentará en número de preguntas.

Duolingo [3] podría considerarse un sistema de BSDB. Esta plataforma, desarrollada por el grupo de investigación de Luis von Ahn, permite a los usuarios aprender un nuevo idioma de manera colectiva y al mismo tiempo traducir contenido web de un idioma a otro. Puede ser accedida en http://duolingo.com/

En *Duolingo* durante la primera etapa los usuarios aprenden las nociones básicas de cada idioma para luego comenzar a traducir pequeñas frases, estas traducciones son validadas por el resto de los usuarios que están aprendiendo el mismo idioma. Algunas frases corresponden a trozos de sitios web reales, por lo que una vez alcanzado cierto nivel de validación la traducción puede ser utilizada en el sitio original.

1.1.3 Repositorium

Como se mencionó anteriormente, *Repositorium* es un motor de BSDB de código abierto, construido para administrar comunidades donde intercambiar documentos de texto.

El modelo de participación subyacente en *Repositorium* intenta fomentar la contribución de los usuarios a la comunidad. Para lograr este objetivo *Repositorium* mantiene un registro de puntos en cada criterio para cada miembro de una comunidad. Los miembros pueden conseguir puntos ya sea aportando material o realizando tareas de control de calidad y pueden utilizar sus puntos para descargar nuevo contenido desde el repositorio. Cuántos puntos costará descargar un documento o cuántos puntos se asignarán al usuario cuando aporte con nuevo material son parámetros definidos al momento de crear una criterio.

Cada comunidad debe definir algunos criterios de calidad para los documentos (al menos un criterio). Los usuarios entonces deberán responder preguntas en relación a este criterio, por ejemplo “¿contiene este documento texto en idioma inglés?”. Al mismo tiempo los usuarios pueden seleccionar ciertos criterios al momento de realizar una búsqueda. En el largo plazo estos criterios de calidad modelarán el tipo de contenido en el repositorio, dado que cada comunidad puede elegir promover distintos criterios.

Para evitar que usuarios malintencionados consigan puntos aportando material que no es útil, el sistema no entrega puntaje a un documento agregado hasta que sea validado para algún criterio. Podría entenderse entonces que aportar material resulta ser una inversión a largo plazo, no así responder desafíos, lo cual sería una inversión a corto plazo.

Existe una instancia de la aplicación *Repositorium* que puede ser accedida en http://www.repositorium.cl y que ha sido desarrollada bajo el criterio de código abierto, el cuál se encuentra en http://github.com/jyby/repositorium

1.2 Motivación

El acceso a Internet desde los dispositivos móviles se ha convertido en un canal fundamental al momento de compartir, implícita o explícitamente, información generada por el usuarios o sus entornos [**4**]. Existen varios ejemplos [**5**, **6**] de aplicaciones que implementan el modelo de *Crowdsourcing* en dispositivos móviles, y se ha mostrado que éstas facilitan la realización de las tareas.

La mayoría de las tareas de control de calidad que se desarrollan en *Repositorium* requieren que el usuario revise un documento agregado por otro miembro de la comunidad (proceso de *revisión por pares*[5]). Este tipo de tareas pueden sacar provecho de las facilidades que ofrece la computación móvil, permitiendo a los usuarios realizar revisiones en cualquier lugar y espacio de tiempo disponible.

Proporcionar a los desarrolladores de aplicaciones móviles una interfaz que les permita interactuar con *Repositorium* facilitaría la participación y colaboración dentro de cada comunidad. Al mismo tiempo abriría un espacio para el desarrollo de nuevas aplicaciones que utilicen los conceptos de BSDB, como por ejemplo juegos serios [6] [**5**].

[5] Revisión por pares (*Peer Review*): método usado para validar trabajos escritos con el fin de medir ciertos criterios, como calidad, originalidad, rigor científico, etc.

[6] Juegos serios (*serious games*): de acuerdo a la definición de Mike Zyda, juego realizado con un propósito distinto a la sola diversión, como por ejemplo, entrenamiento corporativo o educación.

En este sentido abrir *Repositorium* al desarrollo de aplicaciones móviles debe considerar los siguientes aspectos:

- El tráfico de Internet en equipos móviles es más caro y limitado, lo cuál induce ciertas restricciones en la cantidad de datos a transferir.
- Las diferencias entre equipos móviles, en términos de tamaño de pantalla, disponibilidad de sensores y capacidad de computo limita el número de acciones que un usuario puede realizar.

Repositorium podría aprovechar las ventajas que ofrecen las aplicaciones móviles, en término de ubicuidad y comodidad, sin embargo la versión actual del sistema carece de las herramientas necesarias para permitir que terceras partes desarrollen tales aplicaciones. Será necesario también revisar el conjunto de acciones que se harán disponibles y ajustarlas a las capacidades de un equipo móvil.

1.3 Objetivos

El principal objetivo de este trabajo de memoria es proporcionar una interfaz para *Repositorium* que permita a cualquier desarrollador implementar una aplicación móvil utilizando los datos y el modelo de participación disponibles.

Objetivos específicos

Los objetivos específicos necesarios para alcanzar el objetivo principal son:

- Diseñar e implementar una API[7] Web para *Repositorium* que permita el desarrollo de aplicaciones móviles. Tal API debe proporcionar al menos las siguientes funcionalidades:
    - Ingresar y salir de *Repositorium*

[7] API (Application Programming Interface): es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- Buscar y descargar documentos desde un repositorio.
- Realizar tareas de control de calidad.
- Agregar documentos a un repositorio.

Opcionalmente debe permitir:

- Administrar una cuenta de usuario
- Buscar comunidades y unirse a una comunidad

Diseñar e implementar una aplicación para equipos móviles con sistema operativo Android[8] que utilice la API para acceder a los datos de *Repositorium*. Usar esta aplicación para mostrar la efectividad de la API.

2 Antecedentes

Previo al trabajo realizado en esta memoria fue necesario conocer el estado actual de *Repositorium*, analizando su arquitectura, modelo de datos y detalles de su implementación. Así también fue indispensable realizar un pequeño estudio acerca del diseño y desarrollo de APIs Web.

A continuación se presentan los antecedentes básicos obtenidos en la etapa de investigación.

2.1 Estado actual de Repositorium

Repositorium es un proyecto de código abierto [6] desarrollado por alumnos del Departamento de Ciencias de la Computación de la Universidad de Chile.

[8] Android: es un sistema operativo para equipos móviles diseñado por Google y basado en el *kernel* de Linux.

La versión actual fue implementado siguiendo el patrón de arquitectura de software Modelo-Vista-Controlador (MVC); esta arquitectura permite separar las tareas en tres grupos, aquellas relacionadas con el manejo y la persistencia de datos (Modelo), aquellas encargadas de desplegar los datos (Vista) y las que dicen relación con el problema de negocio y que se encargan de comunicar a las partes (Controlador).

En la Figura 1 se observa la interacción entre los componentes de esta arquitectura, el proceso se inicia con una petición web desde un navegador a un servidor web, este entrega la petición al controlador quien se encarga de solicitar los datos al modelo y de invocar a la vista correspondiente para presentar los resultados. El navegador recibe como respuesta la conjunción de datos y vista solicitados.

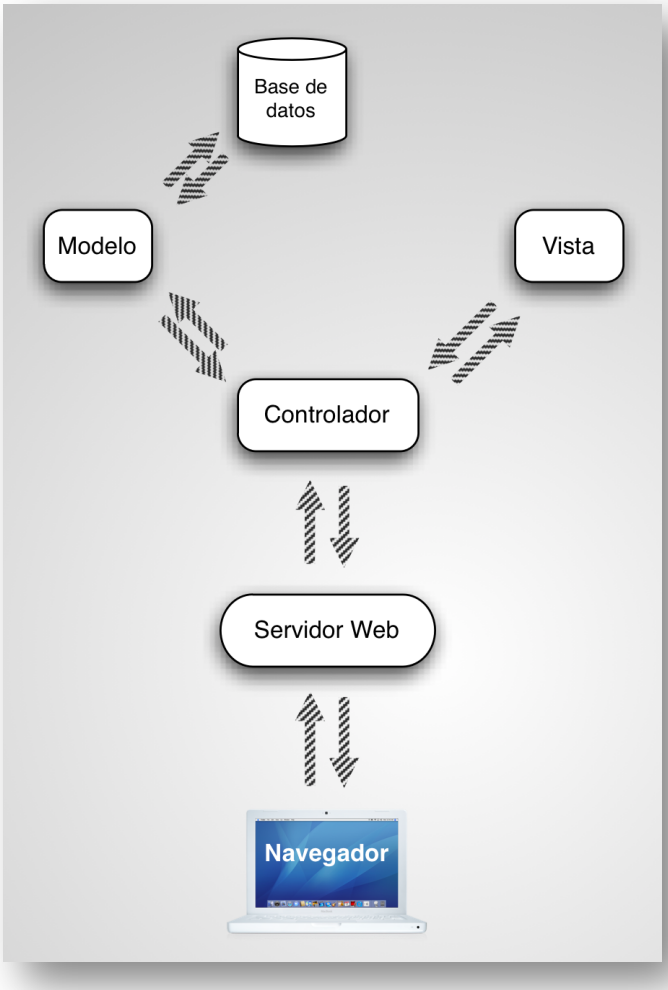


Figura 1: Diagrama de arquitectura de software MVC

En el caso particular de *Repositorium* la capa Modelo utiliza el lenguaje de programación PHP para representar los datos como objetos y el lenguaje de consulta SQL para interactuar con la base de datos; la capa Vista utiliza PHP para leer las datos que le entrega la capa Controlador, el lenguaje de marcas HTML y el lenguaje de programación JavaScript para desplegar los datos en el navegador; la capa Controlador está desarrollada en PHP en su totalidad.

El modelo de datos existente en *Repositorium* está compuesto de tres grupos de entidades: Repositorios, Comunidad y Documentos.

El grupo de Repositorios está compuesto por las entidades que se encargan de almacenar la información de los repositorios, así como los criterios y la relación con los usuarios.

El grupo de Comunidad se compone de las entidades que registran la información de los usuarios y sus relaciones con los repositorios. Un usuario puede unirse a un repositorio o seguirlo (agregarlo a su *watchlist*); los usuarios participan como miembros de unas comunidad y pueden ser promovidos a administradores.

Las entidades agrupadas en Documentos almacenan el contenido de los documentos, sus etiquetas (*tags*) y sus archivos adjuntos. Un documento pertenece a un repositorio y puedo o no contener un archivo adjunto.

En el Anexo AAnexo A se puede encontrar un diagrama del modelo de datos que soporta a *Repositorium*.

2.2 Conceptos básico sobre API Web

Una API general es un conjunto de funciones o métodos que un componente de software ofrece como canal de interacción para con otros componentes u otras aplicaciones. Se habla de una interfaz pues se presenta como una capa de abstracción en donde sólo es conocida la firma de las funciones o métodos y su implementación es irrelevante.

Una API Web, también conocida como *web service*, o servicio web, corresponde a la especificación de una API general, en este caso la comunicación entre las partes se produce mediante transacciones HTTP en donde la estructura de las llamadas y las respuesta es conocida. La Figura 2 muestra una arquitectura simplificada de una API Web.

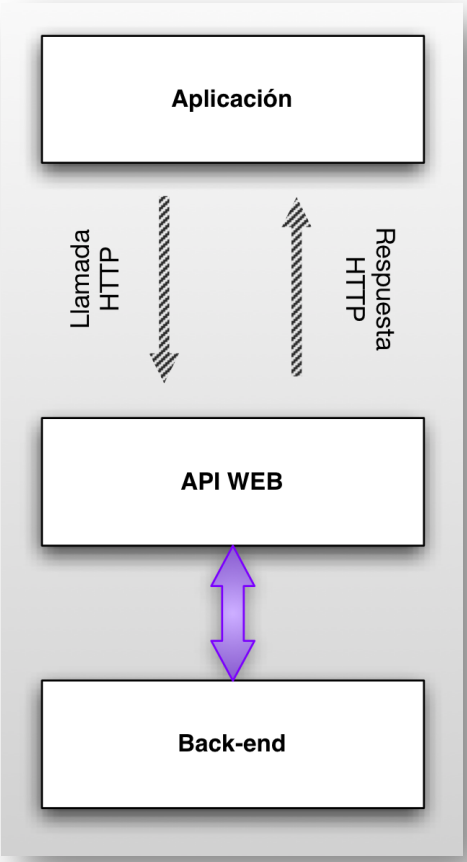


Figura 2: Arquitectura simple de una API web

Una llamada HTTP está compuesta por un encabezado donde se indica el recurso solicitado, el método mediante el cual se solicita, por ejemplo GET o POST, y el protocolo a utilizar, normalmente HTTP/1.1 y un cuerpo dónde, de ser necesario, se adjuntan parámetros y valores. Una respuesta HTTP incluye en el encabezado un código de respuesta, por ejemplo el código 200 en caso de que el requerimiento haya sido satisfecho correctamente, y en el cuerpo el recurso solicitado, normalmente en formato XML o JSON.

En el siguiente ejemplo una aplicación cliente solicitará los datos de un usuario llamado Alberto, para ello realizará la siguiente llamada HTTP:

```
HEADER: GET /usuarios/alberto  HTTP/1.1
```

La API responderá entonces con el siguiente mensaje, indicando que la petición fue ejecutada con éxito y entregando en el cuerpo del mensaje el contenido solicitado:

```
HEADER: HTTP/1.1 200 OK

BODY: <usuario><nombre>Alberto</nombre></usuario>
```

El formato de respuesta y la representación de los elementos deben ser conocidos a priori, por lo que toda API Web debe estar acompañada de una documentación adecuada y clara.

Hoy en día las API web han permitido el desarrollo de diversas aplicaciones que aprovechan el contenido y las herramientas ofrecidas por plataformas que implementan algún tipo de API. Es así como se pueden encontrar videojuegos que aprovechan la red social de los usuarios en Facebook[9], o aplicaciones de geolocalización que utilizan Twitter[10] como espacio de difusión.

3 Análisis

Para comprender las necesidades que podrían tener los usuario de *Repositorium* con respecto a la API, se modeló un cliente tipo que considerara los escenarios más comunes de uso.

Es importante mencionar que, por la naturaleza de la herramienta a desarrollar, es inviable capturar todas las necesidades que podría tener un desarrollador, especialmente si se tiene en consideración que los nuevos escenario en dónde se utilizará Repositorium son diversos en tema y en propósito.

[9] Facebook: sitio de web de redes sociales, cuenta con una API a disposición de desarrolladores y puede ser accedida en https://developers.facebook.com/docs/reference/api/

[10] Twitter: servicio de microblogging, cuenta con una API a disposición de desarrolladores y puede ser accedida en https://dev.twitter.com/docs/api

En las siguientes secciones se enlistan las necesidades de un cliente modelo y se recuperan a partir de ellas los requisitos funcionales y de calidad.

3.1 Cliente Modelo

Una API para Repositorium debe ser capaz de ofrecer las interfaces necesarias para el desarrollo de aplicaciones móviles cuidando de mantener un consumo de ancho de banda mínimo.

Una aplicación móvil genérica requerirá ser capaz de al menos:

1. Autentificar a un usuario.
2. Agregar a un usuario como miembro de un repositorio.
3. Crear un nuevo usuario o actualizar los datos de un usuario existente.
4. Realizar búsquedas y descargar documentos dentro de un repositorio.
5. Consultar las estadísticas de un repositorio.
6. Obtener y responder desafíos.
7. Agregar documentos en un repositorio.

Aplicaciones más específicas podrían requerir interfaces particulares, como por ejemplo:

- Consultar los repositorios existentes.
- Actualizar un documento.
- Consultar los usuarios que participan en un repositorio.
- Consultar la lista de *tags* disponibles en un repositorio.
- Eliminar documentos de un repositorio.
- Eliminar usuarios de un repositorio.

Un cliente modelo requerirá que la API tenga un bajo consumo de ancho de banda y que sus estructuras de datos sea simples y fáciles de manejar. En algunos casos particulares el cliente modelo podría necesitar actuar como un caché de los datos existentes en *Repositorium*.

En términos de seguridad se requiere que la API sea capaz de detectar y manejar ataques de usuarios malintencionados, ya sean ataques de fuerza bruta para tratar de adivinar una clave o recolectando paquetes que viajen por la red.

3.2 Requerimientos

Del análisis del cliente modelo se deprenden los siguientes requerimientos presentados en diagramas de casos de uso.

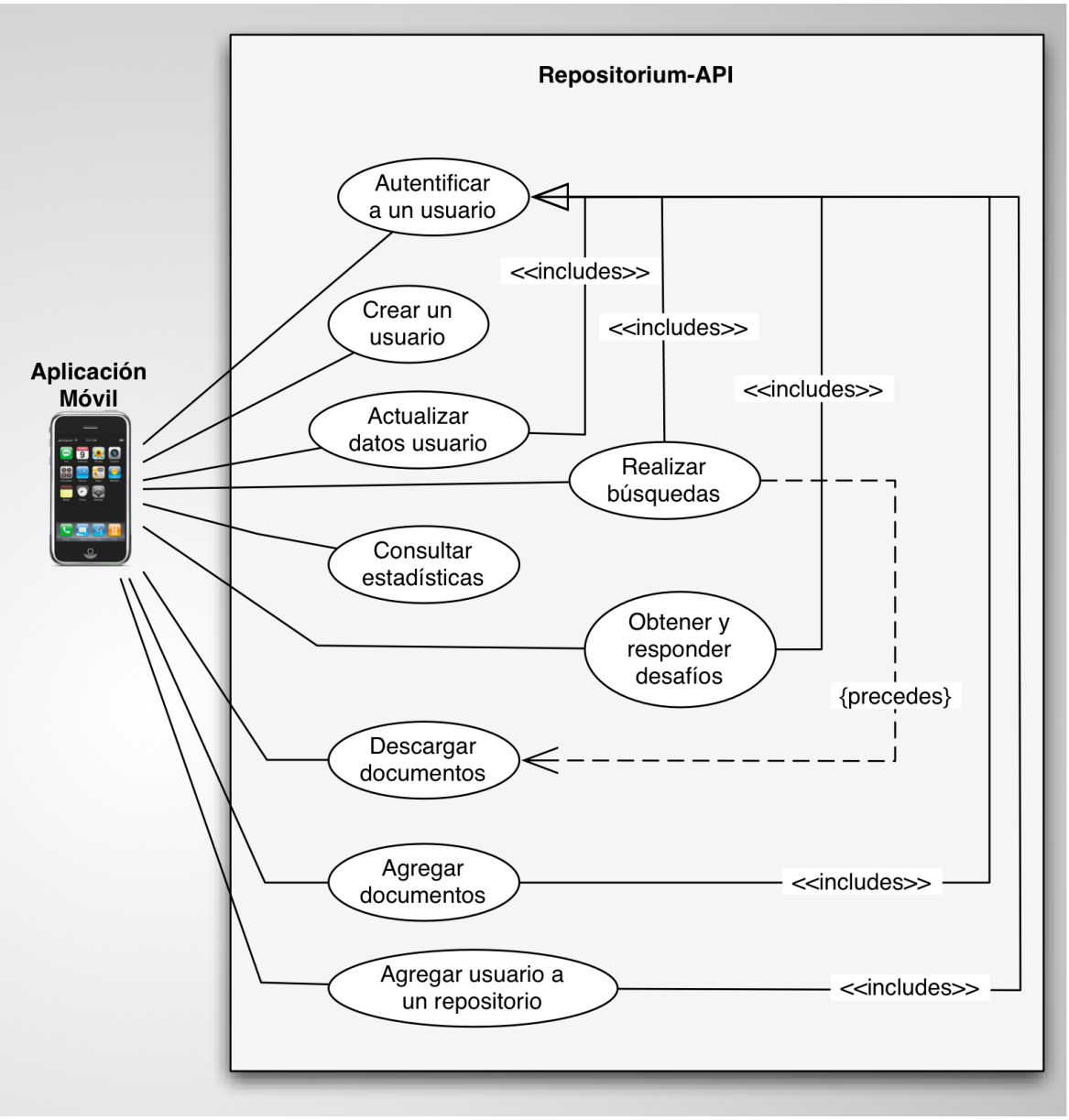


Figura 3: Casos de uso aplicación modelo. Requisitos críticos y deseables

Se observa en Figura 3 que los casos de uso relacionados con la obtención de documentos requieren que el usuario se autentifique. Implícitamente está el hecho de que el cliente debe haber seleccionado un repositorio en particular en el cuál agregar o consultar los documentos.

En la Figura 4 se presentan los casos de uso asociados a los requisitos deseables. Se puede observar que las tareas administrativas, como eliminar usuarios o documentos, requieren que el usuario posea rol de administrador para el repositorio en cuestión.

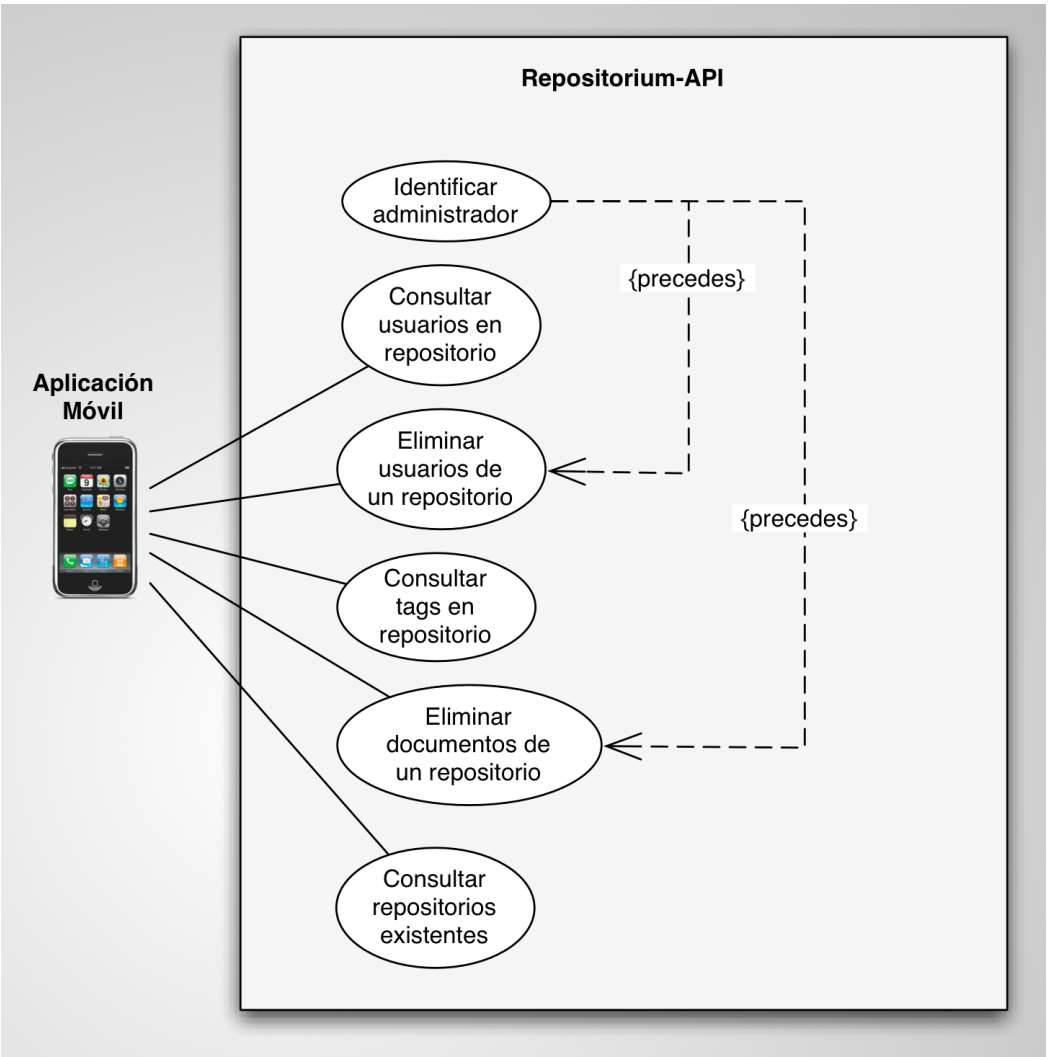


Figura 4: Casos de uso aplicación modelo. Requisitos opcionales

En el Anexo B se puede encontrar una descripción en extenso de los casos de uso indicados en las figuras anteriores.

4 Diseño e implementación de una API para Repositorium

Para diseñar y construir de manera adecuada una API para *Repositorium* se procedió definiendo un cliente modelo que permitiera descubrir las necesidades y restricciones que surgen en el desarrollo de aplicaciones móviles; en esta sección se presentan las decisiones arquitectónicas y los detalles de la implementación que permitirán satisfacer los requerimientos indicados en la sección anterior.

4.1 Buenas prácticas en el diseño de API

Antes de diseñar una API para *Repositorium* fue necesario revisar algunos antecedentes bibliográficos, leer las opiniones de expertos y observar otras APIs similares.

Según expertos el mayor desafío en el desarrollo de una API se encuentra en la etapa de diseño [7]; una API bien diseñada debe ser fácil de usar y debe facilitar el aprendizaje por parte de los desarrolladores, de manera tal que la documentación sea necesaria sólo en un comienzo y que en etapas siguientes su uso sea natural e intuitivo.

De acuerdo con los ingenieros de *Apigee*[11] una buena aproximación es identificar los elementos que se desean disponibilizar y aprovechar los métodos ofrecidos en las llamadas HTTP (GET, POST, PUT, DELETE) para simular las funciones básicas de almacenamiento [8], Crear, Obtener, Actualizar y Borrar, conocidas en inglés como CRUD.

Para mantener URLs cortas y fáciles de comprender se propone utilizar conjuntos y elementos. Considérese el siguiente ejemplo, se busca disponibilizar a través de una API las frutas y verduras que se encuentran en un supermercado, para ello se definen dos tipos de URL, conjuntos:

[11] Apigee es una empresa norteamericana especializada en el desarrollo de API, entre sus clientes se encuentras compañías como Netflix o AT&T. Para más información visitar su sitio web http://apigee.com

```
/frutas

/verduras
```

Y elementos:

```
/frutas/manzana

/frutas/naranja
```

Las acciones que se realicen sobre cada conjunto o elemento estarán definidas por el método HTTP con el cual se invoquen, de manera que, si por ejemplo, se realiza una llamada GET a `/frutas` se obtenga el listado de todas las frutas disponibles, o si se realiza una llamada POST a `/frutas` se agregue una nueva fruta al conjunto.

Esta aproximación podría resultar un tanto simple cuando se trata de resolver problemas más complejos, sin embargo se propone mantener siempre conjuntos y elementos en la URL y el resto de la complejidad manejarla como parámetros de las llamadas HTTP. Así por ejemplo, si se requiere un nivel más detallado de agrupación, se pueden ofrecer subconjuntos, en el ejemplo de las frutas se tendría:

```
GET /frutas/tropicales/piña
```

Y el resto de la complejidad, como por ejemplo, consultar frutas tropicales de una marca específica, o indicar los valores para agregar una nueva fruta, se resuelve en los parámetros de la llamada:

```
GET /frutas/tropicales

PARAMETROS marca=bandelsur

POST /frutas

PARAMETROS nombre=melón & marca=agrosuper
```

Por su naturaleza las APIs tienden a actualizarse y extenderse constantemente, sin embargo se debe cuidar de mantener las versiones anteriores disponibles pues algunas aplicaciones podrían están haciendo uso de ellas. Una recomendación de los ingenieros de *Apigee* es indicar la versión de la API lo antes posible en la URL, un buen ejemplo sería:

```
/api/v3.1/frutas/tropicales/banana
```

Una componente importante de toda API es su documentación, ésta debe ser clara, precisa e idealmente estar acompañada de ejemplo de uso.

Estos antecedentes se han tenido en consideración para el diseño de la API de *Repositorium*, en las siguientes secciones se describirán los elementos y los recursos que la componen.

4.2 Decisiones Arquitectónicas

Durante la etapa de diseño una de las tareas más críticas correspondió a la definición de la arquitectura que dará soporte y delineará el desarrollo de la API. A continuación se presentan las principales decisiones arquitectónicas que se tomaron previo a la implementación de la API Web.

Cabe mencionar en primer lugar la arquitectura física, que corresponde a una variable sobre la cuál no se tuvo control y que estuvo determinada por el servidor en el cuál se encuentra alojado el proyecto *Repositorium*. A junio de 2012 las características de este servidor eran las siguientes:

- 4 Procesadores Intel Xeon de 3.06GHz con 512 KB en caché de memoria nivel 2.
- 2 Gigabytes de memoria RAM.
- 28 Gigabytes en el disco duro donde se almacena el proyecto, con una tasa de uso del 38%.

Las características del servidor se ajustan a las requeridas para el desarrollo de una API Web.

En un nivel superior la arquitectura de software que se definió para este proyecto corresponde a cliente-servidor, donde los clientes serán dispositivos móviles que realizarán peticiones a un servidor web. La comunicación entre clientes y servidor se realizará mediantes llamadas y respuestas HTTP utilizando Internet como el canal de transmisión.

Para poder disponibilizar los contenidos existentes en *Repositorium* fue necesario definir un formato de representación que fuese a la vez expresivo y ligero; entendiendo por expresivo el que sea capaz de definir con claridad los elementos y sus propiedades, y por ligero, que consuma el menor ancho de banda posible.

Los formatos de representación más usados en el desarrollo de aplicaciones móviles son XML y JSON. El primero tiene como ventaja ser el estándar de intercambio de información entre plataformas, el ser extensible y fácil de leer e interpretar tanto por un humano como por una máquina. El segundo tiene como principal ventaja el tratarse de una notación comprimida y de menor tamaño que XML.

Teniendo en consideración que el tráfico de datos en los equipos móviles es limitado y costoso es que se decidió utilizar JSON como el formato de representación para los elementos de la API Web a construir.

En lo que respecta a la arquitectura de la API Web se decidió utilizar el patrón de diseño Modelo-Vista-Controlador como base y se realizaron algunos ajustes acordes al contexto; se agregó el componente “Enrutador” encargado de definir las URL válidas y de direccionar las peticiones al controlador correspondiente; la función de la capa Vista se reduce a declarar el formato de la respuesta como JSON y por lo tanto se incluye como parte del Enrutador. La Figura 5 muestra la arquitectura de software propuesta.

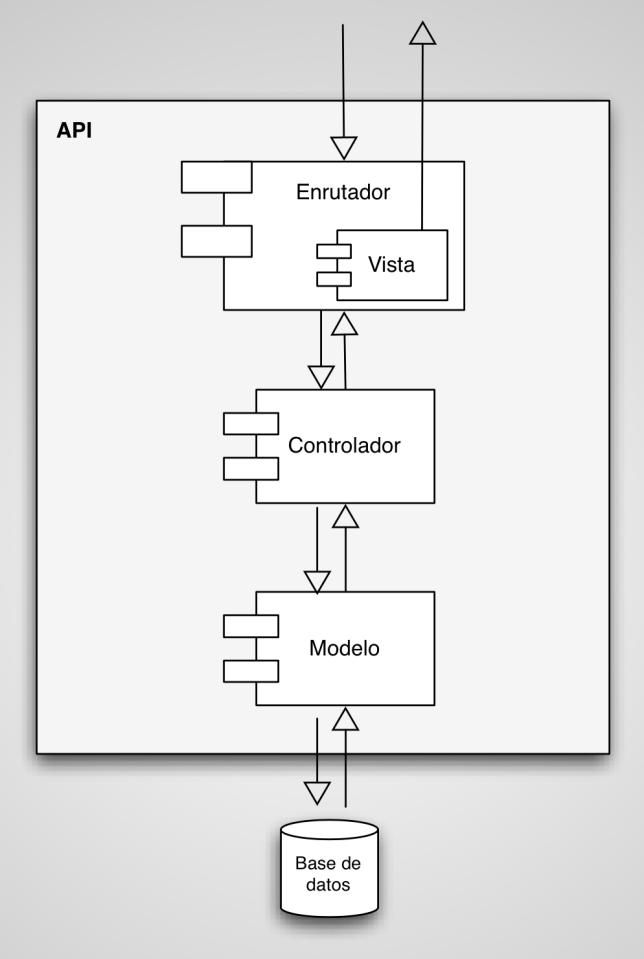


Figura 5: Arquitectura propuesta para la API de Repositorium

Por último, y previo a la implementación del proyecto, se decidió utilizar el lenguaje de programación PHP por tratarse de un lenguaje ampliamente documentado y soportado por la mayoría de los servicios de *hosting* y en particular por el servidor en dónde se ejecutará a API Web a desarrollar.

Las decisiones arquitectónicas aquí presentadas dieron paso a la definición de los elementos a disponibilizar y de los métodos a través de los cuales se accederá a ellos.

4.2.1 Elementos y su representación

Para satisfacer los requerimientos identificados en el capítulo anterior se procedió por definir los elementos a disponibilizar y su representación. Uno de los criterios considerados fue ofrecer siempre la información mínima necesaria de manera que la cantidad de datos transmitidos entre las aplicaciones cliente y el servidor fuera la menor posible.

Para mantener la consistencia con el sistema actual (*Repositorium*) el idioma en que se presentan los elementos es el inglés.

El elemento básico en las comunidades de *Repositorium* es justamente el repositorio (*repository*), para su representación se decidió considerar su identificador, su nombre, descripción, la identificación del autor y un indicador del estado del repositorio (activo o inactivo). El siguiente ejemplo muestra la representación de un repositorio en formato JSON:

```
{
id: 1,
name:Fondue,
description: This repository contains stimations of
nutritional facts for food,
author: USER,
active: 1
}
```

El identificador (*id*) permitirá acceder a las opciones disponibles dentro de repositorio, como la búsqueda de documentos; el nombre (*name*) y la descripción (*description*) permitirá a los usuarios conocer el tipo de contenido existente dentro del repositorio para, por ejemplo, decidir si unirse o no a éste repositorio; el autor (*author*) permitirá identificar al creador del repositorio en caso de que sea necesario establecer contacto con él; el indicador activo (*active*) permite conocer si el repositorio se encuentra activo o ha sido abandonado.

En el ejemplo anterior el autor corresponde a un elemento de tipo usuario (*user*); un usuario estará representado por su dirección de correo electrónico, su nombre y apellido, la fecha en que se inscribió en *Repositorium* y un estado activo o inactivo. Se decidió no incluir el identificar interno del usuario (id) para evitar que el acceso a los datos de toda la comunidad se pueda realizar iterando sobre este parámetro. El siguiente ejemplo muestra la representación de un usuario:

```
{
email:cgajardo@litmon.com,
name:carlos,
lastname:gajardo,
created:2011-09-25 20:26:51,
active:1
}
```

La dirección de correo electrónico (*email*) permitirá identificar a un usuario y será parte de los datos requeridos para autentificarlo en el sistema, esto para mantener la interfaz actual del la plataforma web; su nombre (*name*) y apellido (*lastname*) permitirán reconocerlo como miembro de la comunidad; la fecha en que se inscribió (*created*) en *Repositorium* permitirá reconocer la antigüedad del usuario; el identificador (active) activo o inactivo permitirá reconocer si el usuario se encuentra activo o ha sido eliminado del sistema.

Otro de los componentes relevantes de *Repositorium* es el documento (*document*); un documento estará representado por su título, el contenido o cuerpo, una identificación del autor, la fecha de creación, las etiquetas que lo identifican y, de existir, los archivos adjuntos. También en este caso se decidió no incluir el identificador interno del documento (id), de manera de que no se pueda acceso a todos los documentos del sistema iterando sobre este parámetro. A continuación se presenta un documento en formato JSON:

```
{
title:apple pie,
content:411 calories,
author:USER,
created: 2012-05-19 12:55:23,
files:FILES,
tags:deserve
}
```

El título (*title*), el contenido (*content*), los archivos (*files*) y las etiquetas (*tags*) son los elementos de interés para los usuarios de *Repositorium*; los archivos variarán en tipo dependiendo del propósito de cada repositorio; el autor (*author*) indicará quién creó este documento, la fecha de creación (*created*) permitirá identificar la antigüedad del documento en cuestión.

Los usuarios de *Repositorium* deben responder desafíos para obtener puntos y de esta manera acceder a la descarga de documentos. Un desafío (*challenge*) estará compuesto de un identificador, una pregunta, un documento y dos posibles respuestas o alternativas. A continuación un ejemplo de desafío:

```
{
id:23,
question: does this document describe a real meal?,
document:DOCUMENT[],
answera:yes,
answerb:no
}
```

El identificador (*id*) permitirá referir el desafío presentado; la pregunta (*question*) corresponderá a aquello que el usuario deberá responder con respecto al documento (*document*) que se le entregará y tendrá como alternativas las respuestas a (*answera*) y b (*answerb*).

Por último, un elemento no inherente a *Repositorium*, pero necesario en la comunicación con las aplicaciones clientes serán los errores. Un error estará representado por un código de estado HTTP y un mensaje, como se muestra en el siguiente ejemplo:

```
{
status: 401 Unauthorized,
message: This action can only be performed by admin users
}
```

El código de estado (*status*) indicará la naturaleza del error, este código también será parte del encabezado de la respuesta HTTP; el mensaje (*message*) entregará mayor información con respecto a la razón del error.

Con los elementos recién presentados y los métodos que se presentan más adelante, se pretende satisfacer los requerimiento identificados en la sección 3. Los elementos se han creado teniendo en consideración que el consumo de ancho de banda en equipos móviles es costoso y por lo tanto se buscó entregar la información mínima necesaria para una operación adecuada.

4.2.2 Recursos y rutas

Definidos ya los elementos y su representación el siguiente paso fue definir los recursos que se ofrecerán para que las aplicaciones cliente accedan a los datos almacenados en *Repositorium*; se entenderá por recurso una función ofrecida por la API Web que se invoca realizando una llamada HTTP a una URL definida, mediante un método (POST, GET, PUT o DELETE) establecido. En esta etapa se tuvieron en consideración los casos de uso presentados en la sección 3.2.

En concordancia con las buenas prácticas indicadas en la sección 4.1 se procedió a definir el siguiente esquema de recursos para *Repositorium*.

El conjunto base en la mayoría de los casos será aquel compuesto por los repositorios (*repositories*), desde ahí será posible realizar la búsqueda, descarga y agregación de los documentos. En las siguientes figuras se describen los recursos a disponibilizar, agrupados por conjunto y por método de invocación.

Para el conjunto Repositories se definieron los siguientes recursos que serán accesibles mediante el método GET:

Recurso	Descripción
GET /repositories	Devuelve un listado de elementos de tipo Repositoy con los datos de los repositorios existentes.
GET /repositories/:id	Devuelve un elemento de tipo Repository con los datos del repositorio :id
GET /repositories/:id/stats	Devuelve las estadísticas para el repositorio :id
GET /repositories/:id/tags	Devuelve una lista de tags para el repositorio :id
GET /repositories/:id/users	Devuelve un listado de elementos de tipo User con los datos de los usuarios que participan en el repositorio :id
***GET /repositories/:id/search:query**	Realiza una búsqueda de documentos en el repositorio :id y devuelve un elemento de tipo Challenge o Document dependiendo del puntaje del usuario.
***GET /repositories/:id/documents**	Retorna un elemento de tipo Document perteneciente al repositorio :id siempre que el usuario sea un administrador y provea el identificador del documento.
***GET /repositories/:id/documents/random**	Retorna un elemento de tipo Document con los datos de un documento aleatorio del repositorio :id

Recurso	Descripción
***GET /repositories/:id/challenges**	Retorna un elemento de tipo Challange para un criterio aleatorio dentro del repositorio :id

Tabla 1 : Listado de recursos para Repositories accesibles mediante GET

Los recursos marcados con * indican que el usuario deben haber proporcionado previamente los datos necesarios para su autentificación. Todos los recursos que se acceden mediante GET podrían entenderse como recursos para la lectura de los datos existentes.

En la siguiente tabla se listan los recursos accesibles mediante los métodos POST y PUT para el conjunto Repositories.

Recurso	Descripción
***POST /repositories/:id/join**	Permite al usuario autentificado unirse a un repositorio.
***POST /repositories/:id/disjoin**	Permite al usuario autentificado desunirse a un repositorio.
***POST /repositories/:id/documents**	Permite agregar un documento al repositorio :id
***POST /repositories/:id/challenge**	Permite al usuario enviar las respuestas de un desafío.
***PUT /repositories/:id/documents**	Permite a un usuario con rol de administrador actualizar los datos de un documento.
***DELETE /repositories/:id/documents**	Permite a un usuario con rol de administrador eliminar un documento.

Tabla 2 : Listado de recursos para Repositories

El segundo conjunto será aquel compuesto por los usuarios (users). A partir de este conjunto será posible agregar, autentificar y actualizar los datos de los usuarios. La siguiente lista muestra los recursos accesibles mediante los métodos GET, POST y PUT.

Recurso	Descripción
GET /users/:email	Entrega un elemento de tipo User con los datos del usuario :email
GET /users/:email/repositories	Entrega la lista de repositorios a los que el usuarios :email pertenece.
POST /users	Permite agregar un nuevo usuario al sistema.
POST /users/login	Permite autentificar a un usuario mediante su correo y su password.
***PUT /users/:email**	Permite actualizar los datos del usuario :email siempre que coincida con el usuario en sesión.

Tabla 3 : Listado de recursos para Users

Una descripción más exhaustiva de los recursos aquí presentados se puede encontrar en el Anexo C. En el Anexo D se presenta una tabla que relaciona los recursos disponibles y los casos de uso identificados en la sección 3.2.

Ya definidos los elementos y los recursos a disponibilizar la siguiente tarea correspondió a la implementación de estos.

4.3 Implementación

Ya definidos los elementos y su representación y establecidos los recursos disponibles se procedió con la implementación de estos.

Durante la implementación se trabajó de acuerdo a la metodología *Test Driven Development* o TDD, es decir, se inició el trabajo definiendo una batería de *tests* en base a los casos de uso determinados en la sección 3.2, luego en la medida de que se desarrollaba nuevo código se ejecutaban los test y se realizaban los ajustes necesarios, se iteraba tantas veces como fuese necesario hasta que el código cumpliera con todos los *tests*.

Los test definidos tuvieron por objetivo asegurar la correctitud del código implementado evaluando si las precondiciones y las respuestas eran las esperadas.

Para construir la batería de test se utilizó la herramienta *Jmeter* desarrollada por el grupo *Apache*. Esta herramienta permite simular llamadas HTTP y definir aserciones en las respuestas entregadas por el servidor. En la Figura 6 se muestra un ejemplo de una petición HTTP la cuál tiene por objetivo verificar que la búsqueda de documento funcione correctamente; en la Figura 7 se muestra la aserción correspondiente, para este caso se espera que el servidor responda con un mensaje indicando que el usuario debe estar autentificado.

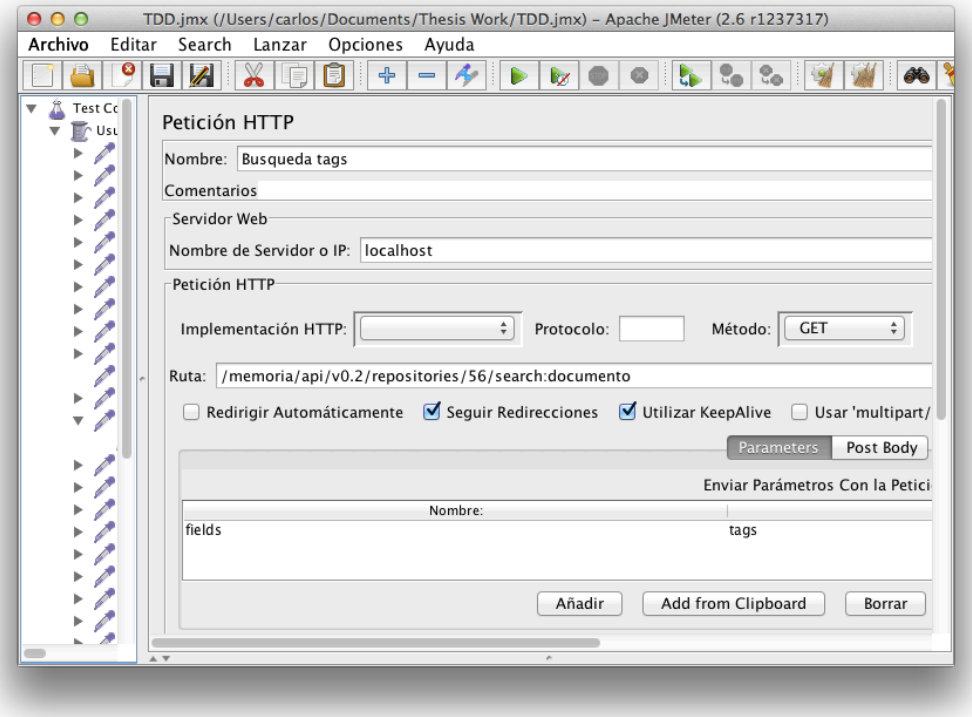


Figura 6: Ejemplo test JMeter, petición HTTP

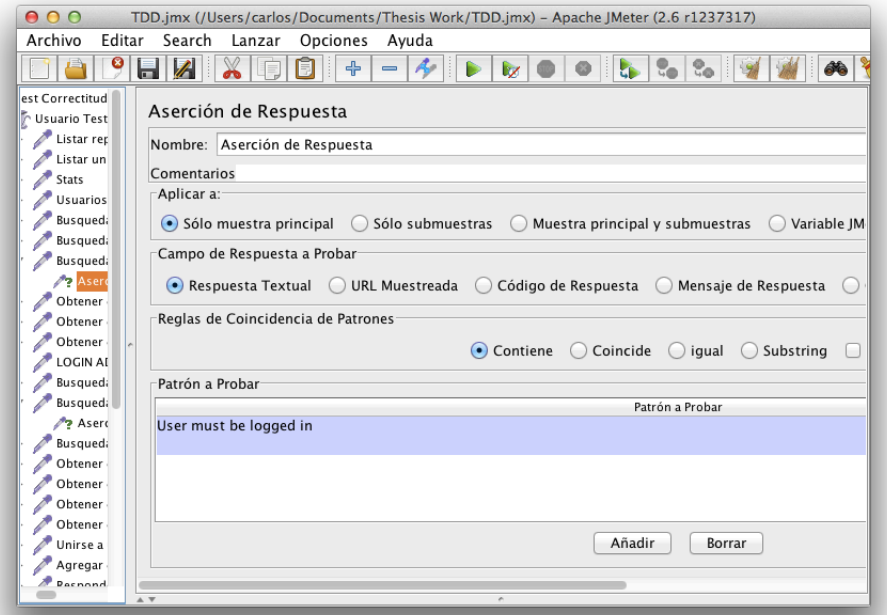


Figura 7: Ejemplo test JMeter, Aserción

Los test fueron definidos en base a los casos de uso presentados en la sección 3 y tienen por objetivo principal el asegurar que los requerimientos detectados se cumplan a cabalidad.

Definidos los test se procedió con la implementación propiamente tal, la arquitectura propuesta está descrita en la Figura 5 y se compone de tres capas, una capa denominada Enrutador que incluye la subcapa denominada Vista y las capas Controlador y Modelo. A continuación se entregan detalles de la implementación de cada una de estas capas.

Para la capa Enrutador se utilizó el *framework*[12] de código abierto *Epiphany* [9], el cuál facilitó la definición de las rutas para acceder a los recursos y entregó el entorno necesario para definir las vistas, que como se mencionó anteriormente, corresponden a la representación de los elementos en formato JSON.

[12] Framework: conjunto de librerías o clases reutilizables que funcionan como estructura de soporte para el desarrollo de software.

Se decidió utilizar *Epiphany* por tratarse de un *framework* de código abierto implementado en PHP, lo cuál se condice con la condición detectada en la sección 4.2 sobre los recursos disponibles en el servidor de *Repositorium*.

La Figura 8 muestra cómo la clase EpiApim definida por el *framework,* utiliza la clase EpiRoute para establecer las rutas de acceso; esta clase recupera las definiciones válidas desde un archivo de configuración.

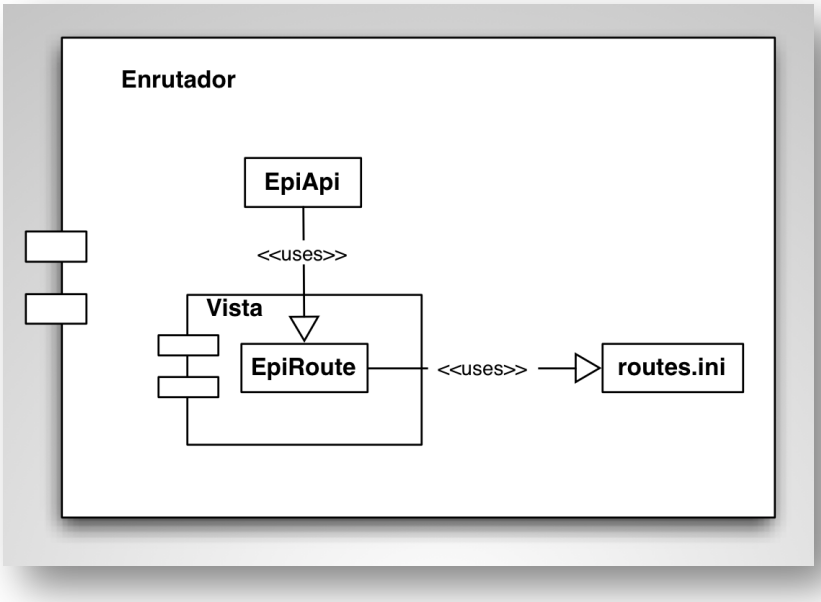


Figura 8: Diagrama de clases, capa Enrutador

En el archivo de configuración se declaran las rutas y métodos HTTP válidos y se asocian a una clase y una función de la capa Controlador. Las rutas pueden contener expresiones regulares; los valores que tomen estas expresiones regulares serán entregados cómo parámetros a la función, por ejemplo, la siguiente configuración indica que si la ruta es llamada mediante el método GET, se debe ejecutar la función *show* de la clase Repositories.

```
method = GET
path = "/repositories/(\d+)"
class = Repositories
function = show
```

La clase Repositories del ejemplo anterior pertenece la capa denominada Controlador; esta capa está compuesta además por las clases Challenges, Documents y Users las cuales implementan la lógica detrás de cada llamada HTTP y permiten entregar las respuestas esperadas por las aplicaciones clientes, en la Figura 9 se detallan los métodos disponibles en cada una de estas clases.

Las clases de la capa Controlador acceden a la base de datos mediante clases abstractas que se ofrecen en la capa Modelo, la invocación de las clases abstractas se realiza a través de una factoría, siguiendo el patrón de diseño *Factory*. Es también en la capa Modelo dónde se definieron los objetos para transmitir los datos, conocidos como DTOs (Data Transfer Objects).

Para consultar un registro de la base de datos las clases de la capa Controlador deben invocar el DAO (Data Access Object) correspondiente a través de la clase DAOFactory. Cada DAO devolverá el DTO correspondiente, salvo en los casos de inserción o actualización de registros, en donde devolverá un indicador de éxito o error.

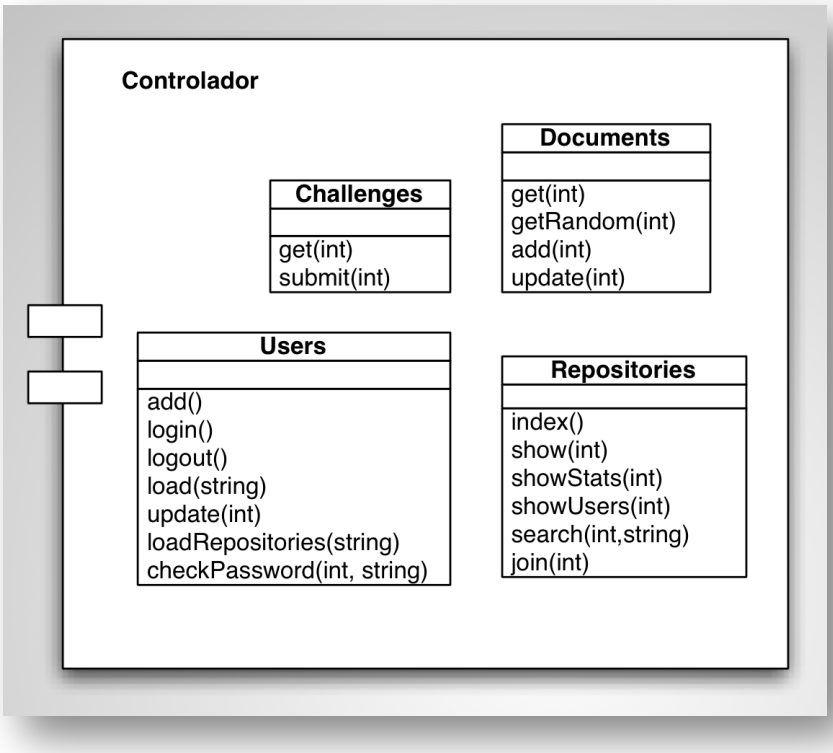


Figura 9: Diagrama de clases, capa Controlador

En la Figura 10 se observa un diagrama de la capa Modelo; este diseño permite independizar el modelo de datos y facilita la actualización de la API para ajustarse a futuras versiones de *Repositorium*.

Para construir la capa Modelo se buscó inspiración en la herramienta PHPDAO [**10**], la cuál además permite encapsular la conexión con la base de datos siguiendo el patrón de diseño *Singleton*.

La implementación de este trabajo se realizó con la política de código abierto, el cual se encuentra alojado en Github y puede ser accedido en la siguiente URL https://github.com/cgajardo/repositorium-api

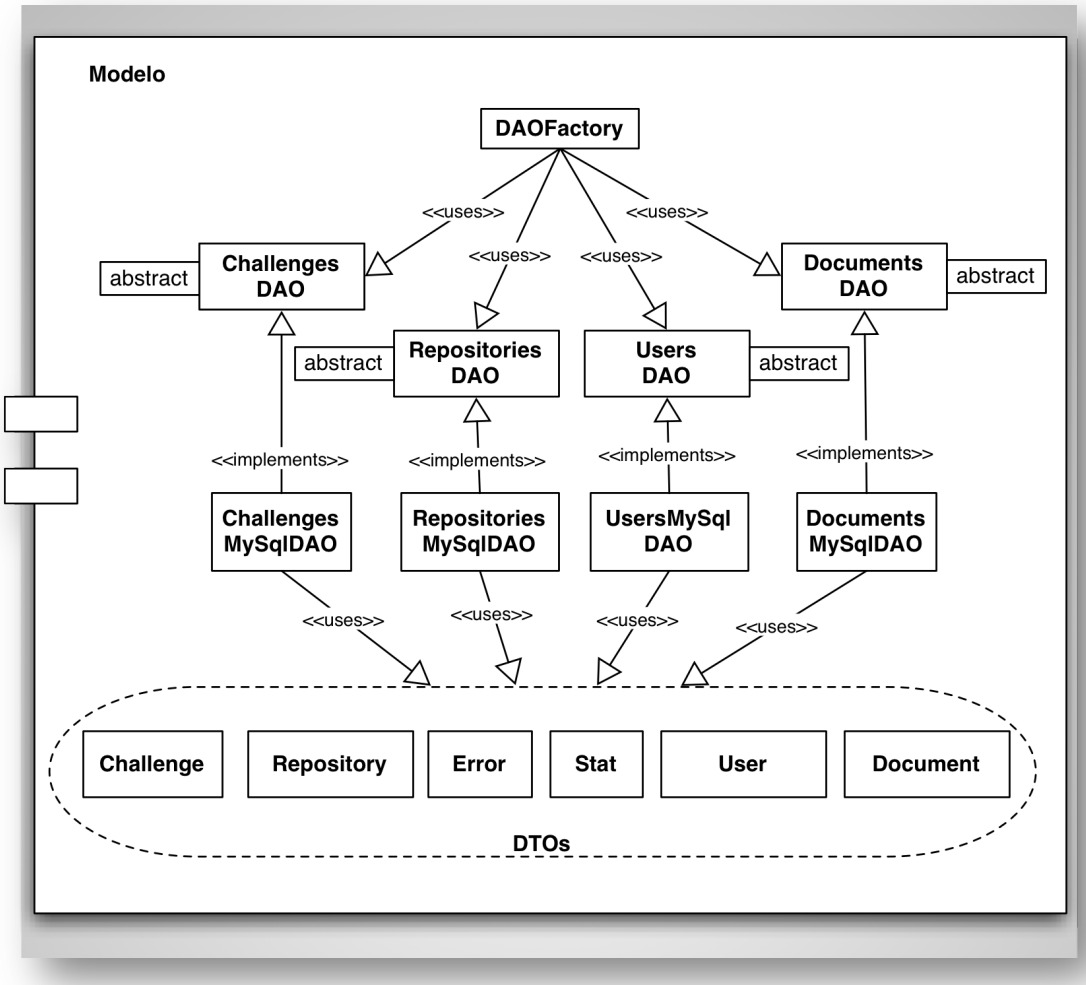


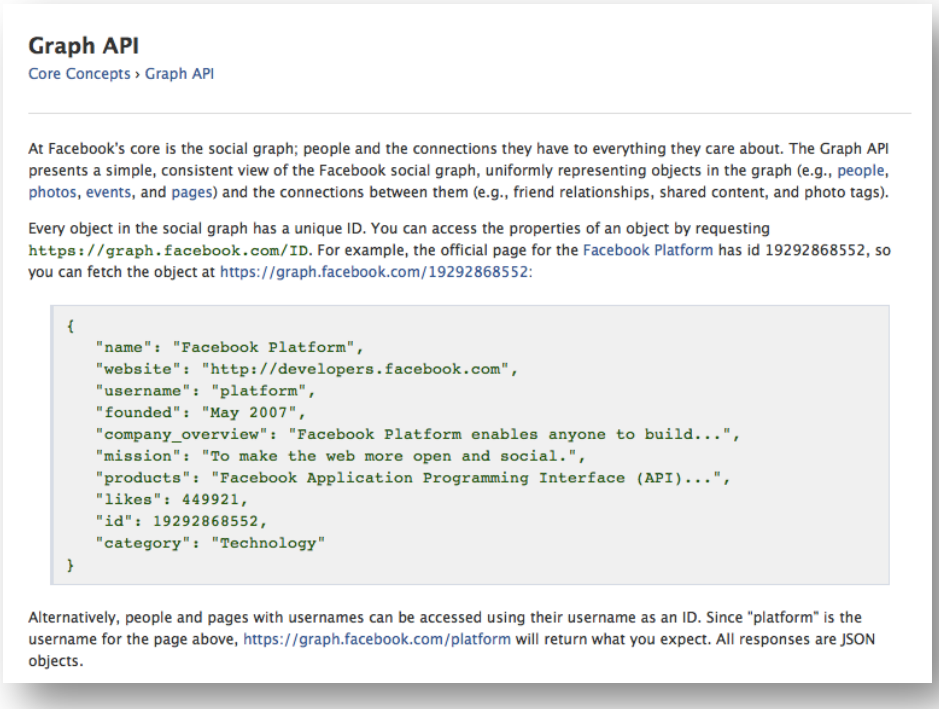
Figura 10: Diagrama de clases, capa Modelo

Completada la tarea de implementación el siguiente paso correspondió a la construcción de un documentación clara y consistente para poner a disposición de los futuros clientes de la API.

4.4 Documentación

Parte importante de una API corresponde a su documentación, el ingeniero de software en Google y autor de *Effective Java*, Joshua Bloch señala que “No importa que tan buena sea una API, esta no será usada si no cuenta con una buena documentación” [11] .

Es por ello que para buscar la mejora forma de documentar una API se estudiaron dos alternativas, la documentación de las APIs de Facebook y Twitter. En el primer caso se presenta una documentación basada en ejemplos, la cuál, si bien clara, requiere que los usuarios tengan un conocimiento previo acerca del funcionamiento de la API, presentando una mayor curva de aprendizaje.



Graph API

Core Concepts › Graph API

At Facebook's core is the social graph; people and the connections they have to everything they care about. The Graph API presents a simple, consistent view of the Facebook social graph, uniformly representing objects in the graph (e.g., people, photos, events, and pages) and the connections between them (e.g., friend relationships, shared content, and photo tags).

Every object in the social graph has a unique ID. You can access the properties of an object by requesting `https://graph.facebook.com/ID`. For example, the official page for the Facebook Platform has id 19292868552, so you can fetch the object at https://graph.facebook.com/19292868552:

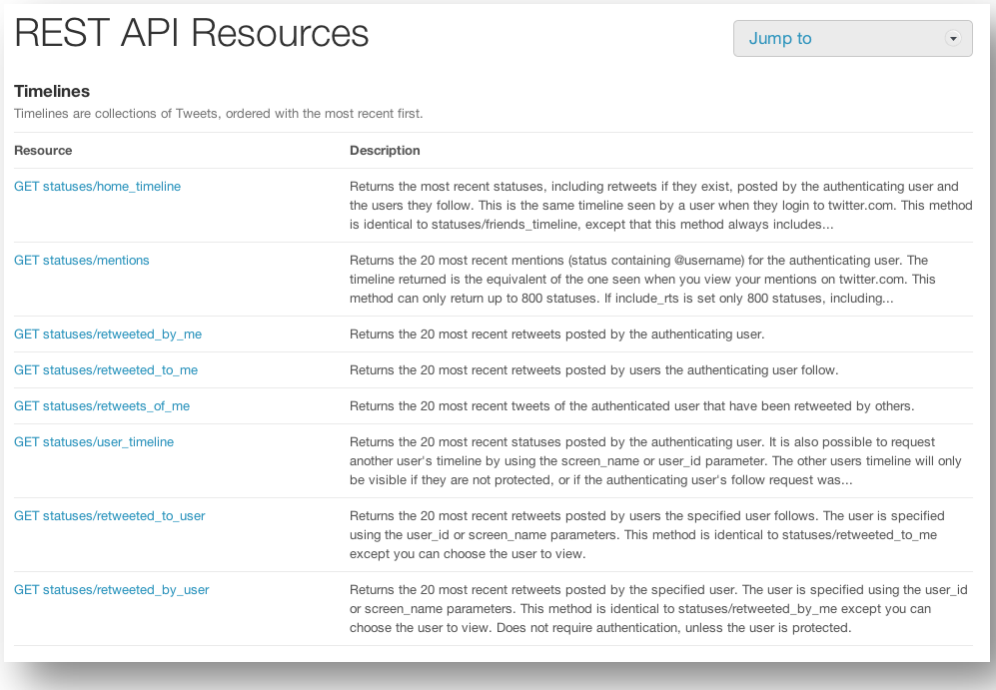

```
{
   "name": "Facebook Platform",
   "website": "http://developers.facebook.com",
   "username": "platform",
   "founded": "May 2007",
   "company_overview": "Facebook Platform enables anyone to build...",
   "mission": "To make the web more open and social.",
   "products": "Facebook Application Programming Interface (API)...",
   "likes": 449921,
   "id": 19292868552,
   "category": "Technology"
}
```


Alternatively, people and pages with usernames can be accessed using their username as an ID. Since "platform" is the username for the page above, https://graph.facebook.com/platform will return what you expect. All responses are JSON objects.

Figura 11: Ejemplo documentación API Facebook

Twitter en cambio presenta una documentación basada en la descripción de los recursos disponibles y en una segunda capa presenta ejemplo de uso de cada recurso. Este modelo permite a los usuarios tener una primera aproximación simple y ordenada, y permite, de ser necesario, explorar los ejemplo más complejos. Este modelo además presenta la ventaja de que la documentación de los métodos se puede utilizar como material de referencia durante el desarrollo.

Para poder realizar una mejor comparación se utilizaron ambas documentaciones para construir un pequeño ejemplo y se decidió trabajar con el modelo que propone Twitter, principalmente porque resulta simple de entender, fácil de consultar y no requiere que el desarrollador lea extensos ejemplo para comprender el uso de los recursos.



REST API Resources

Jump to

Timelines

Timelines are collections of Tweets, ordered with the most recent first.

Resource	Description
GET statuses/home_timeline	Returns the most recent statuses, including retweets if they exist, posted by the authenticating user and the users they follow. This is the same timeline seen by a user when they login to twitter.com. This method is identical to statuses/friends_timeline, except that this method always includes...
GET statuses/mentions	Returns the 20 most recent mentions (status containing @username) for the authenticating user. The timeline returned is the equivalent of the one seen when you view your mentions on twitter.com. This method can only return up to 800 statuses. If include_rts is set only 800 statuses, including...
GET statuses/retweeted_by_me	Returns the 20 most recent retweets posted by the authenticating user.
GET statuses/retweeted_to_me	Returns the 20 most recent retweets posted by users the authenticating user follow.
GET statuses/retweets_of_me	Returns the 20 most recent tweets of the authenticated user that have been retweeted by others.
GET statuses/user_timeline	Returns the 20 most recent statuses posted by the authenticating user. It is also possible to request another user's timeline by using the screen_name or user_id parameter. The other users timeline will only be visible if they are not protected, or if the authenticating user's follow request was...
GET statuses/retweeted_to_user	Returns the 20 most recent retweets posted by users the specified user follows. The user is specified using the user_id or screen_name parameters. This method is identical to statuses/retweeted_to_me except you can choose the user to view.
GET statuses/retweeted_by_user	Returns the 20 most recent retweets posted by the specified user. The user is specified using the user_id or screen_name parameters. This method is identical to statuses/retweeted_by_me except you can choose the user to view. Does not require authentication, unless the user is protected.

Figura 12: Ejemplo documentación API Twitter

La documentación propuesta se compone de 3 partes, la primera describe los elementos y sus propiedades, la segunda presenta un listado completo de los recursos disponible y sus método de invocación y la tercera muestra ejemplos de uso de cada recurso, detallando los parámetros a utilizar y su significado.

En la sección *Elements* se describieron todos los elementos y sus propiedades, además se entregó un ejemplo ilustrativo. En la sección *Resources* se listaron todos los recursos disponibles en la API indicando el método y la url mediante la cuál deben ser solicitados, estos recursos están vinculados a la tercera vista, *Examples*, que muestra ejemplos de uso de cada recurso.

Los ejemplos de uso son presentados como fragmentos en lenguaje de programación Java para facilitar el trabajo de aquellos que desarrollen aplicaciones en la plataforma Android.

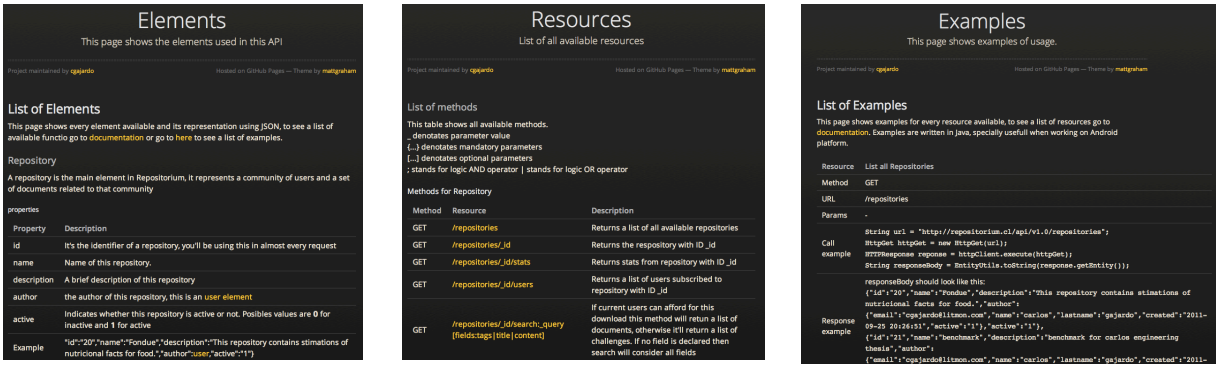


Figura 13 : Capturas de pantalla de la documentación

La documentación fue construida en HTML y se encuentra disponible en la siguiente URL http://cgajardo.github.com/repositorium-api/

4.5 Análisis

Una vez concluida la implementación de la API y su correspondiente documentación, se realizó un test de esfuerzo para medir el rendimiento en términos del tiempo de respuesta. Para ello se simularon peticiones de clientes y se sometió el sistema a distintos niveles de carga.

Para realizar las mediciones se ejecutaron 4 peticiones distintas, autentificar a un usuario, solicitar un desafío, solicitar un documento aleatorio y agregar un documento. Para comparar resultados se realizaron 5 cargas distintas, con 10, 50, 100, 500 y 1000 clientes realizando las 4 peticiones en 1 segundo. Los resultados obtenidos se muestran a continuación.

En la Figura 14 se puede apreciar el tiempo promedio, en milisegundos, que tomó completar las 4 peticiones para las 5 cargas, la curva muestra una tendencia logarítmica en relación al número de clientes por segundo.

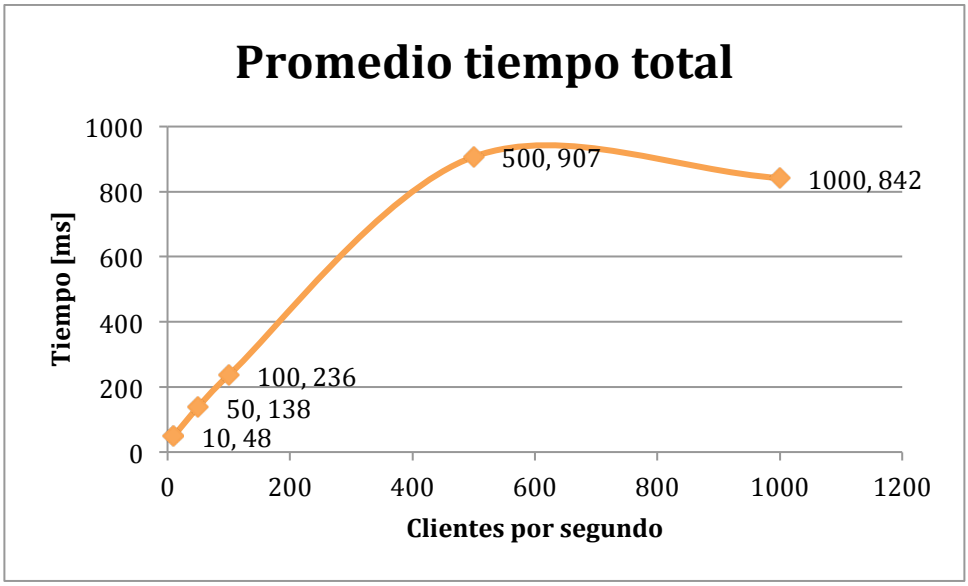


Figura 14 : Gráfico tiempo promedio total

Para determinar que tan cercana es la curva a una de orden logarítmico se realizó la siguiente regresión. Considérense los datos:

Clientes	Tiempo de Respuesta promedio [ms]
10	48
50	138
100	236
500	907
1000	842

Tabla 4: Tiempos promedio Test de esfuerzo

Supongamos que la curva se describe por la ecuación:

$$Y = aX^b$$

Donde a y b son coeficientes a determinar. Aplicando Ln en ambas partes de la ecuación se obtiene:

$$Ln(Y) = Ln(a) + b * Ln(X)$$

Para resolver esta ecuación línea haremos uso de los siguientes datos:

X	Y	$Ln(X)$	$Ln(Y)$	$Ln(X)^2$	$Ln(Y)^2$	$Ln(X) * Ln(Y)$
10	48	1,00	1,68	1,00	2,83	1,68
50	138	1,70	2,14	2,89	4,58	3,64
100	236	2,00	2,37	4,00	5,63	4,75
500	907	2,70	2,96	7,28	8,75	7,98
1000	842	3,00	2,93	9,00	8,56	8,78

Se puede obtener una estimación de b mediante la siguiente ecuación:

$$b = \frac{\sum(Ln(X) * Ln(y)) - \frac{\sum Ln(X) * \sum Ln(y)}{n}}{\sum Ln(X)^2 - \frac{(\sum Ln(X))\text{^}2}{n}}$$

En la cual, reemplazando los valores se obtiene que $b = 0{,}67$

Para estimar el grado de ajuste de los datos al modelo, se utilizará el siguiente coeficiente:

$$r^2 = \frac{b * (\sum \left(Ln(x)^{Ln(Y)}\right) - \frac{\sum Ln(X) * \sum Ln(Y)}{n}}{\sum Ln(Y)^2 - \frac{(\sum Ln(Y))\text{^}2}{n}}$$

Y reemplazando los valores se obtiene que $r^2 = 0{,}98$ lo cuál indica que los datos obtenidos se ajustan a una curva logarítmica y es posible realizar predicciones.

El menor tiempo que se obtuvo al realizar una carga con 1000 clientes debe tener en consideración que no todas las llamadas fueron atendidas por el servidor, al menos el 10% de las llamadas fueron rechazadas. Esto depende de varios factores, entre ellos la configuración del servidor web, en el cuál se puede definir el máximo número de clientes a atender; así como la carga actual del equipo, que en este caso aloja varias aplicaciones y proyectos académicos.

Para comprender mejor cómo aporta cada petición al promedio total se presentan en la Figura 15 los tiempo promedio de cada petición. Se observa desde ya que la petición más costosa en tiempo corresponde a solicitar un documento aleatorio. Esto se explica debido a que en gestor de bases de datos MySQL no se encuentra optimizado para realizar una selección aleatoria, más aún, el tiempo aumenta a mayor sea el conjunto sobre el cuál se realiza la selección.

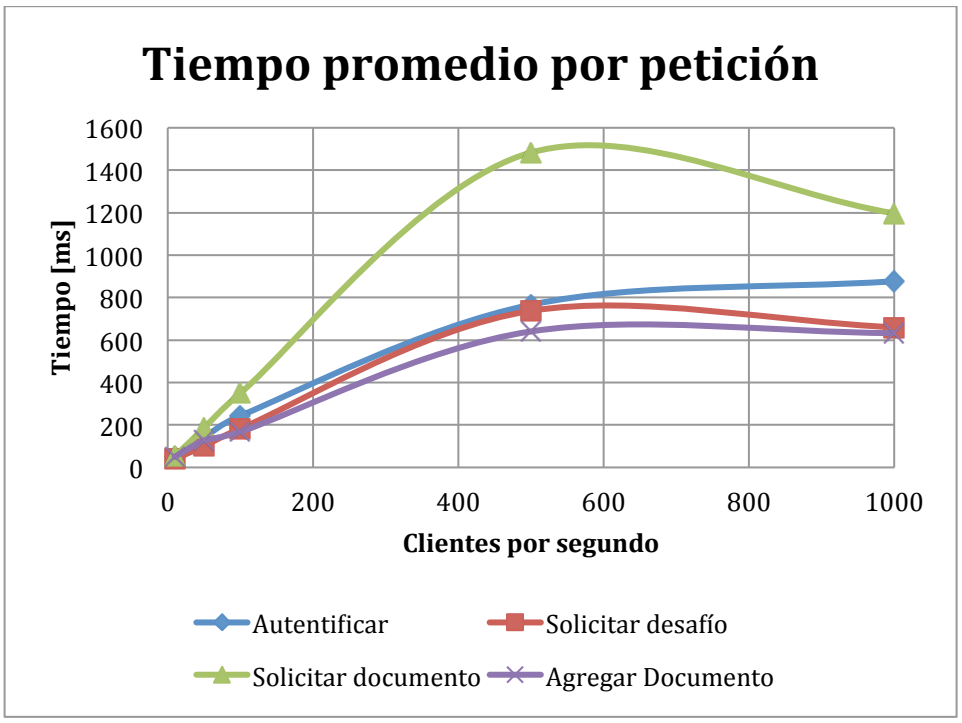


Figura 15 : Tiempo promedio por petición

El resto de las peticiones tienen un comportamiento similar, sin embargo la petición de agregar documentos se verá fuertemente afectada por el número de criterios que existan en el repositorio, pues por cada criterio debe agregarse un registro en la base de datos para permitir la validación del documento.

La petición para solicitar un documento aleatorio no forma parte de los requerimientos básicos identificados para el cliente modelo y más aún es una facultad que sólo disponen los usuarios con rol de administrador. Si no consideramos los tiempos de esta petición, el promedio de tiempo total cambiaría según se muestra en la Figura 16.

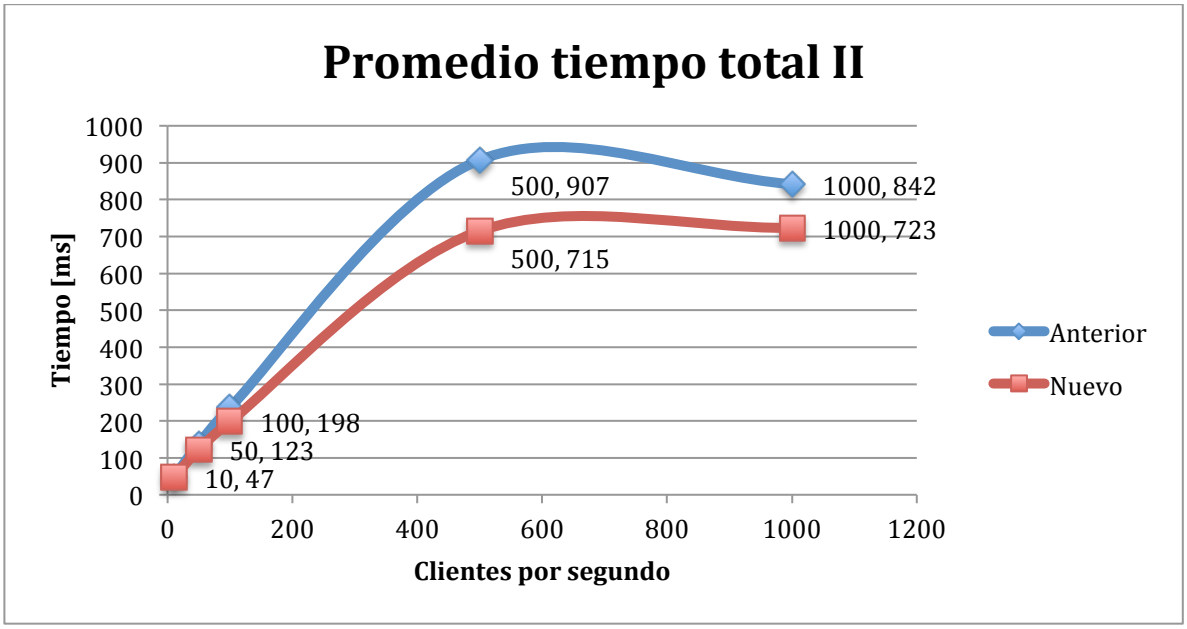


Figura 16 : Promedio tiempo total II

Tendiendo en consideración los resultados obtenidos es posible enunciar que la API es capaz de responder satisfactoriamente a 500 clientes por segundo realizando 4 peticiones cada uno, para un total de 2000 llamadas, superando este límite se tiene que, si bien los tiempos no aumentan considerablemente, si lo hace la tasa de llamadas no respondidas.

5 Diseño e implementación de una aplicación cliente

Una vez diseñada, implementada y documentada la API para *Repositorium*, una última etapa correspondió a la verificación de su eficacia como herramienta dentro del desarrollo de aplicaciones móviles en contextos distintos al académico.

En las siguientes secciones se mostrará como la API ayudó a resolver un problema de control de calidad que enfrentaba la aplicación *Fondue*.

5.1 Contexto

Fondue es una aplicación web que tiene por objetivo que sus usuarios compartan lo que están comiendo o cocinando al tiempo que les entrega *feedback* acerca de la calidad de su alimentación.

Existen sistemas similares que permiten llevar un registro de los alimentos consumidos y su información nutricional, sin embargo requieren que los usuarios indiquen con precisión las porciones consumidas y generalmente sólo permiten seleccionar alimentos desde una lista finita.

Fondue busca que los usuarios ingresen en texto simple aquello que están comiendo o cocinando y les permite compartir esta información con sus amigos en redes sociales como Facebook o Twitter. Los usuarios obtienen un primer *feedback* de su alimentación a través de un sistema de medallas que los premia, positiva o negativamente, de acuerdo a sus conductas alimenticias. El segundo feedback lo obtienen a través de un sistema de puntajes basado en la información nutricional de lo que consumen.

La tarea de estimar datos nutricionales precisos sobre un plato, como la cantidad de calorías o proteínas, resulta inviable al considerar la variabilidad en las recetas y en las porciones; es por eso que en *Fondue* se utilizan 4 indicadores que corresponden a estimaciones gruesas de la información nutricional, estos indicadores son Energía, Fuerza, Vitalidad y Glotonería.

Cada uno de los indicadores hace referencia a un valor nutricional (Energía a las calorías, Fuerza a las proteínas, Vitalidad a vitaminas y minerales y Glotonería a azúcares y grasas) y puede tomar un valor entre 0 y 10.

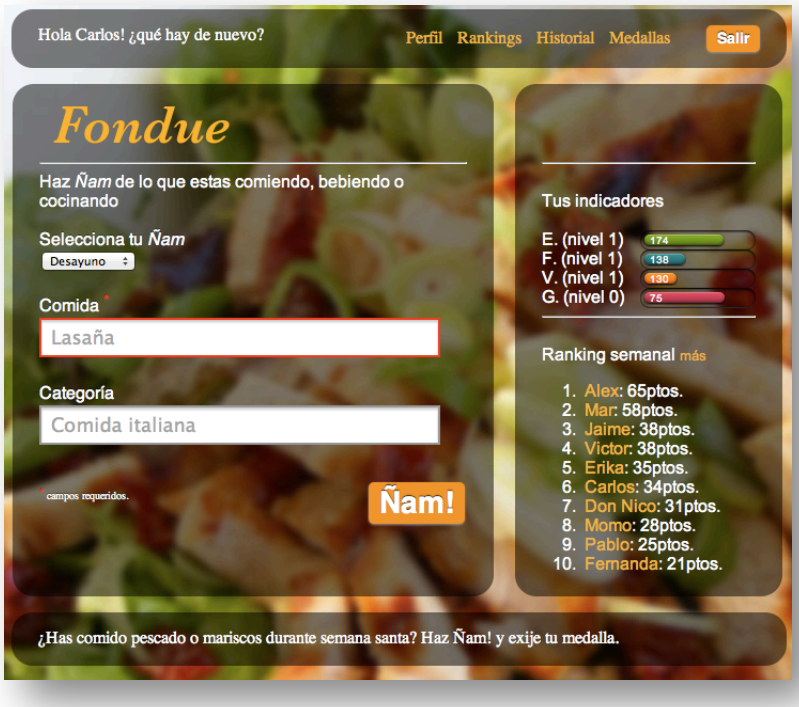


Figura 17: Captura inicio Fondue

Actualmente los valores de un plato se calculan de acuerdo a un diccionario construido manualmente, en donde se definen los grupos alimenticios y a cada grupo se le asigna un valor en cada indicador, luego los alimentos que ingresan al sistema se asocian manualmente a uno de estos grupos y heredan aquel puntaje.

Por ejemplo, para el plato “Arroz con pollo asado” el sistema reconocer las palabras “arroz” y “pollo” como parte del diccionario. Arroz pertenece al grupo de los cereales y hereda el puntaje de este grupo; pollo pertenece al grupo de las carnes y hereda el puntaje de este grupo. Así el puntaje del plato completo corresponde al promedio del puntaje de arroz y pollo.

Este modelo presenta dos problemas importantes, el primero es que depende en su totalidad del administrador para asociar los nuevos alimentos a un grupo alimenticio, y el segundo es que los valores asignados a cada grupo fueron elegidos arbitrariamente por el administrador del sitio, basado en su estimación personal sin ser un experto en el tema.

En este punto es dónde un BSDB podría ayudar, primero a mejorar las estimaciones de los indicadores aprovechando el conocimiento que la comunidad de *Fondue* puede tener respecto a la información nutricional de los alimentos y segundo, controlar que la información ingresada por los usuarios corresponda efectivamente a alimentos o platos reales.

5.2 Diseño e implementación

Para aprovechar las ventajas de BSDB fue necesario modificar el flujo de navegación de *Fondue*. En el sistema previo un usuario ingresaba un plato, el sistema calculaba el puntaje y lo mostraba.

La Figura 18 muestra el nuevo flujo de navegación que permite agregar las funcionalidades de BSDB a *Fondue*. Se aprecia que ahora el puntaje se obtiene desde un documento almacenado en *Repositorium*, la API responderá con un desafío si el usuario no cuenta con el puntaje necesario para descargar el documento, si por el contrario la API responde con un documento el sistema solicitará un segundo documento para calibrar el puntaje de éste.

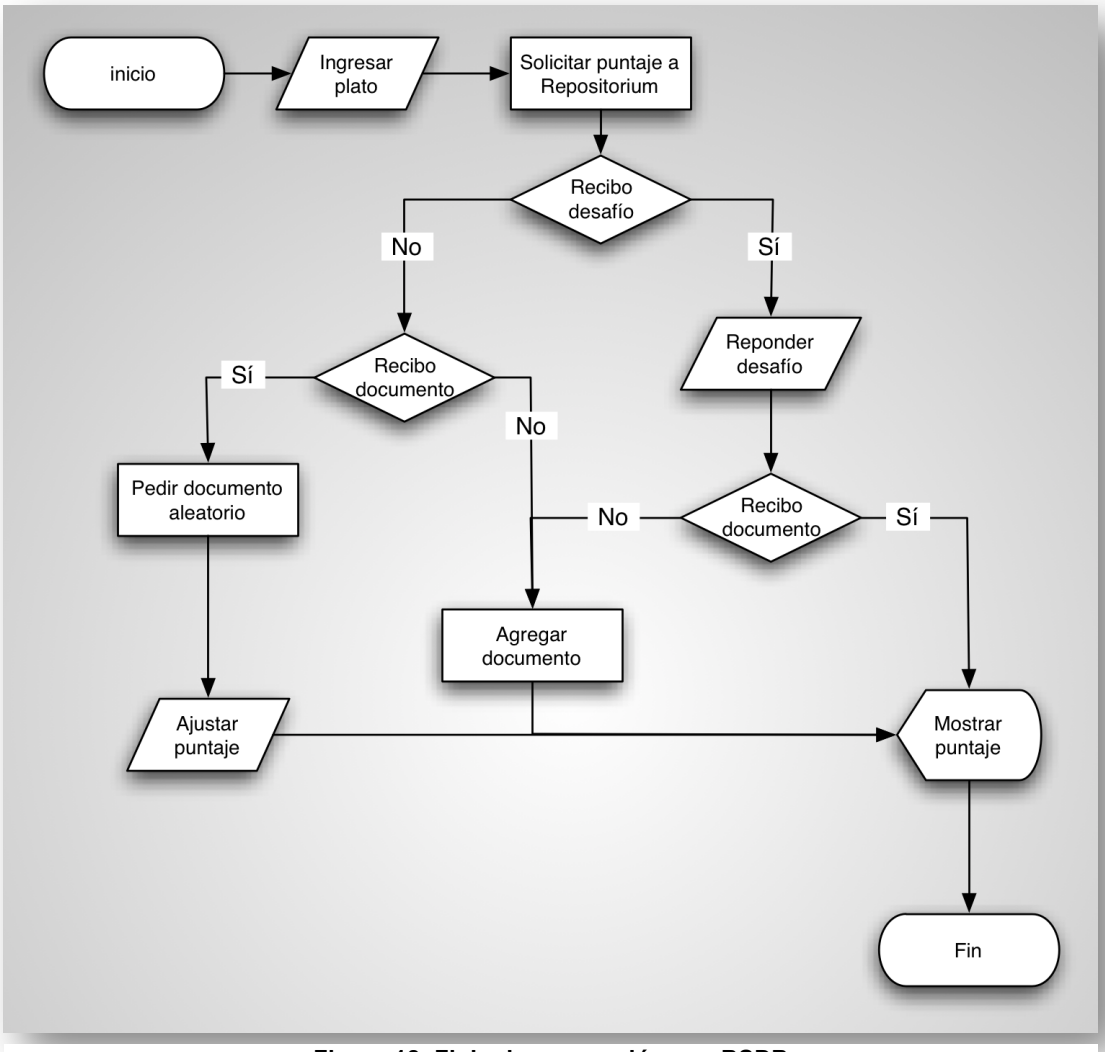


Figura 18: Flujo de navegación con BSDB

Como ya se ha mencionado *Repositorium* permite almacenar documentos de texto y adjuntar archivos. En el caso de *Fondue* se requiere almacenar varios datos (un puntaje para cada indicador), por lo tanto fue necesario adoptar una convención que permitiera almacenar en texto la información deseada; para ellos se utilizó el formato JSON por su simplicidad y menor tamaño.

Por otro lado, uno de los objetivos que se persiguen es poder realizar una estimación colaborativa del puntaje para cada indicador, por lo tanto, además de almacenar el puntaje, se almacena la cantidad de colaboraciones, de manera que el puntaje final sea el promedio de los puntajes asignados por los miembros de la comunidad. A continuación se muestra un ejemplo de cómo se almacenaron los datos de *Fondue* en *Repositorium*:

```
Title: Pizza
Content:{"strength":6,      "energy":8,      "vitality":2,
"gluttony":12, "ns":2, "ne":2, "nv":2, "ng":2}
```

Los campos *ns*, *ne*, *nv* y *ng* se utilizan para almacenar la cantidad de aportes que han hecho los usuarios para el indicador de fuerza, energía, vitalidad y glotonería respectivamente. En el ejemplo, dos miembros de la comunidad han emitido su aporte y el puntaje en cada indicador será el promedio de todos los aportes, así para una Pizza, la cantidad de energía corresponderá a $8 \div 2$, es decir 4 puntos.

Con el nuevo flujo y la representación en texto de los datos definidos, se procedió a la implementación de la aplicación móvil. Para acelerar el desarrollo se utilizó la herramienta *Cordova*, del grupo *Apache*, la cuál permite desarrollar aplicaciones para el sistema operativo Android utilizando herramientas web, como HTML y JavaScript.

Se decidió utilizar *Cordova* pues ofrecer facilidades para comunicarse con componentes del hardware, como la cámara o componentes lógicas como la galería de imágenes, y si bien no hace mayor diferencia con respecto al uso de la API, sí facilita el desarrollo de la aplicación.

Toda la lógica de comunicación con la API de *Repositorium* se implementó utilizando JavaScript en conjunto con la librería JQuery, mediante la técnica conocida como Ajax[13]. Cada solicitud de recursos está contenida en su propia función, lo cuál permite aislar el propósito de cada llamada al servidor de *Repositorium*.

Por tratarse de una aplicación pequeña y sin grandes complejidades, como muestra el diagrama de la Figura 18, la lógica se implementó usando el estilo de programación imperativo, es decir, cada función escrita se encarga de cambiar el estado del programa; la ejecución de una secuencia de funciones permite obtener el resultado deseado.

La aplicación móvil desarrollada mantiene la interfaz simple de la aplicación web *Fondue*, y permite a los usuarios ingresar en texto aquello que están comiendo, como muestra la Figura 19

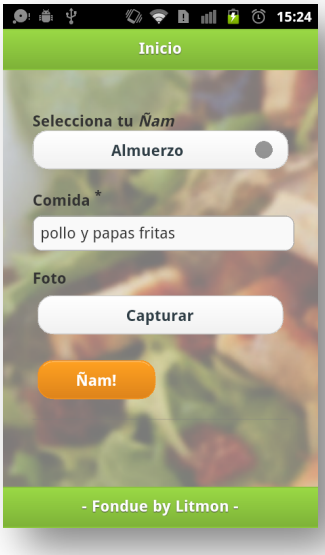


Figura 19: Inicio aplicación móvil Fondue

[13] Ajax: es una técnica de desarrollo web para crear aplicaciones interactivas, las cuales se ejecutan en el cliente (navegador) manteniendo una comunicación asíncrona con el servidor

La aplicación móvil resultante permite conectar el sistema de *Fondue*, con la aplicación web *Repositorium* a través de la API desarrollada durante esta memoria. Desde hoy los puntajes de cada comida se almacenarán y recuperarán desde Repositorium y se aplicarán las reglas de BSDB, es decir, para conocer el puntaje de una comida o plato en particular, el usuarios deberá contar un los puntos suficientes en *Repositorium*; de no ser así, deberá responder con un desafío como el que se muestra en la Figura 20.

Figura 20 : Desafío en aplicación Fondue

Del mismo modo, si un usuario cuenta con los puntos suficientes para conocer el puntaje de un plato, se le solicitará que ayude a calibrar los indicadores de otro plato, y se le presentará la interfaz que se muestra en la Figura 21.

En el caso de que el usuario busque conocer el puntaje de un plato que no existe aún en *Repositorium*, se agregará éste como un nuevo documento y se le asignará un puntaje aleatorio entre 0 y 5 a cada indicador.

Figura 21 : Calibrar puntajes Fondue

En todos estos casos el usuario podrá finalmente conocer su puntaje como muestra la Figura 22.

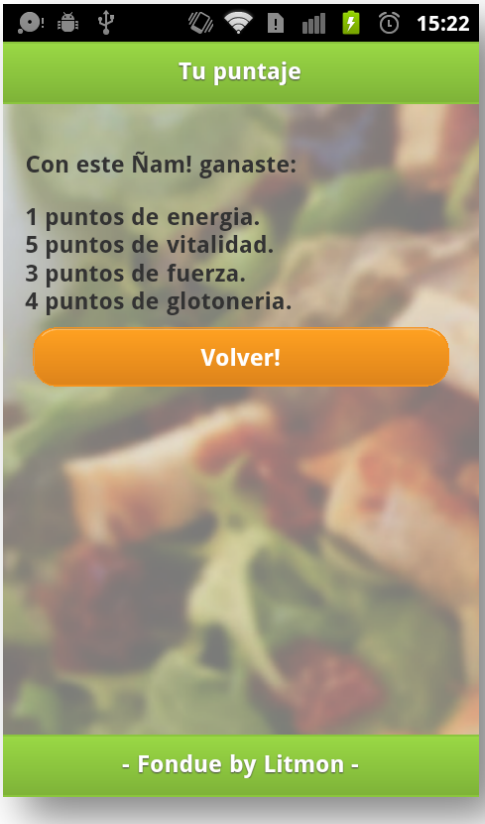


Figura 22 : Puntaje Fondue

Se tiene finalmente una aplicación móvil que logra establecer comunicación con el sistema web de *Repositorium* a través de la API desarrollada en este trabajo de título, lo cual permite mostrar que la API cumple con su objetivo principal de establecer un canal de comunicación entre aplicaciones móviles y el sistema web de Repositorium y que además abre un espacio para la existencia de aplicaciones no académicas.

6 Conclusiones

El objetivo principal, proporcionar una interfaz para *Repositorium* que permita a desarrolladores implementar aplicaciones móviles utilizando los datos y el modelo de participación disponibles, fue cumplido de manera exitosa mediante el desarrollo de una API web que abre un canal de comunicación simple y consistente con *Repositorium*.

Para cumplir con los objetivos específicos se construyó un cliente modelo que permitió identificar los requerimientos y documentarlos mediante casos de uso. Esto facilitó la organización del trabajo durante la etapa de desarrollo.

Para comprobar la correctitud de la API web se realizaron pruebas durante el desarrollo y posteriormente se analizó el rendimiento y la capacidad de la misma. Para comprobar la efectividad se desarrolló una aplicación móvil, basada en la aplicación web *Fondue*, que permitió mostrar las funcionalidades de la API y cómo ésta puede ayudar a resolver problemas de participación y control de calidad al estar basada en el concepto de BSDB, al igual que *Repositorium*.

6.1 Logros

Se logró diseñar y construir una API web funcional, orientada al uso en aplicaciones para dispositivos móviles. Se propuso un diseño tal que facilita el versionamiento de la API, permitiendo extenderla o actualizar en la medida de que cambié el modelo de datos o se actualice la lógica de la aplicación *Repositorium*. El diseño de capas simplifica la tarea de mantener el código y mejorar el sistema.

Se logró resolver un problema no académico mediante el uso de la API. Hasta antes de este proyecto *Repositorium* tenía un contexto de aplicación principalmente académico. Hoy es posible aprovechar las ventajas de BSDB, y los datos de *Repositorium* en contextos no académicos, por ejemplo, en una comunidad como *Fondue*.

6.2 Trabajo futuro

En paralelo al trabajo realizado en esta Memoria, el sistema *Repositorium* se encontraba en etapa de mejoras. Estas mejoras incluyeron nuevas opciones de clasificación de los documentos, un sistema de puntos basado en las áreas de conocimiento de los usuario y un nuevo modelo de datos.

Si bien este nuevo sistema no se encuentra en producción, se espera en el futuro que reemplace al sistema actual, en cuyo caso será necesario actualizar el modelo de datos de la API. Sin embargo el diseño de capas propuesto facilita la actualización de sus componentes, en este caso, de la capa Modelo.

Uno de los requisitos deseables identificados a través del cliente modelo, y que tiene relación con la seguridad y la capacidad de detectar y manejar ataques de usuarios malintencionados, no se abordó en este trabajo por razones de tiempo. Sin embargó se investigaron opciones para mejorar la seguridad del sistema; una de ellas es el uso de HTTPS[14] para la comunicación entre los dispositivos y el servidor que aloja a la API. El uso de HTTPS requeriría que toda la comunicación se encripte y se desencripte tanto en los dispositivos móviles como en el servidor, lo cuál podría afectar el rendimiento.

Una mejora interesante que podría realizarse en el futuro tiene relación con el manejo de las sesiones de usuario. Actualmente la sesión es almacenada a nivel del servidor, utilizando memoria caché, lo cuál está bien en la medida de que el número de clientes conectados sea bajo, sin embargo cuando el número aumente y las sesiones comiencen a ocupar un gran espacio de memoria será necesario cambiar de estrategia. Una opción es utilizar una base de datos para administrar las sesiones; otra opción sería manejar las sesiones en el cliente, utilizando el estilo REST.

[14] HTTPS: protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos.

7. Bibliografía

[1] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, y Manuel Blum, "reCAPTCHA: Human-Based Character Recognition via Web Security Measures," Science, vol. 321, no. 1465, pp. 1465-1468, Julio 2008.

[2]Jérémy Barbay, Edgar Chavez, y Marcelo Mylarz, Collective Production of Pedagogical Material, Noviembre 2011, Aplication for Funding to LACCIR.

[3] Luis von Ahn, "Three human computation projects," en Proceding of the 42nd ACM technical symposium on Computer Science Education, Dallas, 2011, pp. 691-692.

[4] Florian Alt, Alireza Sahami Shirazi, Albrecht Schmidt, Urs Kramer, y Zahid Nawaz, "Location-based crowdsourcing: extending crowdsourcing to the real world," en Proceeding NordiCHI '10, Reykjavík, 2010, p. Islandia.

[5]Nathan Eagle, "txteagle: Mobile Crowdsourcing," Lecture Notes in Computer Science, vol. 5623, pp. 447-456, agosto 2009.

[6] Yan Tingxin, Matt Marzilli, Ryan Holmes, Deepak Ganesa, y Mark Corner, "Demo Abstract: mCrowd - A Platform for Mobile Crowdsourcing," en Proceeding of the 7th ACM Conference on Embedded Networked Sensor Systems, 2009.

[7] Jorge Romo, "Tripdroid: Georrefenciación Colaborativa en Dispositivos Móviles," Departamento de Ciencias de la Computación, Universidad de Chile, Santiago, Reporte Trabajo Dirigido 2011.

[8] Jérémy Barbay. Github. [Online]. Visitado el 21 de marzo de 2012. HYPERLINK "https://github.com/jyby/repositorium" https://github.com/jyby/Repositorium

[9] Michi Henning, "API Design Matters," Communications of the ACM, vol. 52, no. 5, pp. 46-56, 2009.

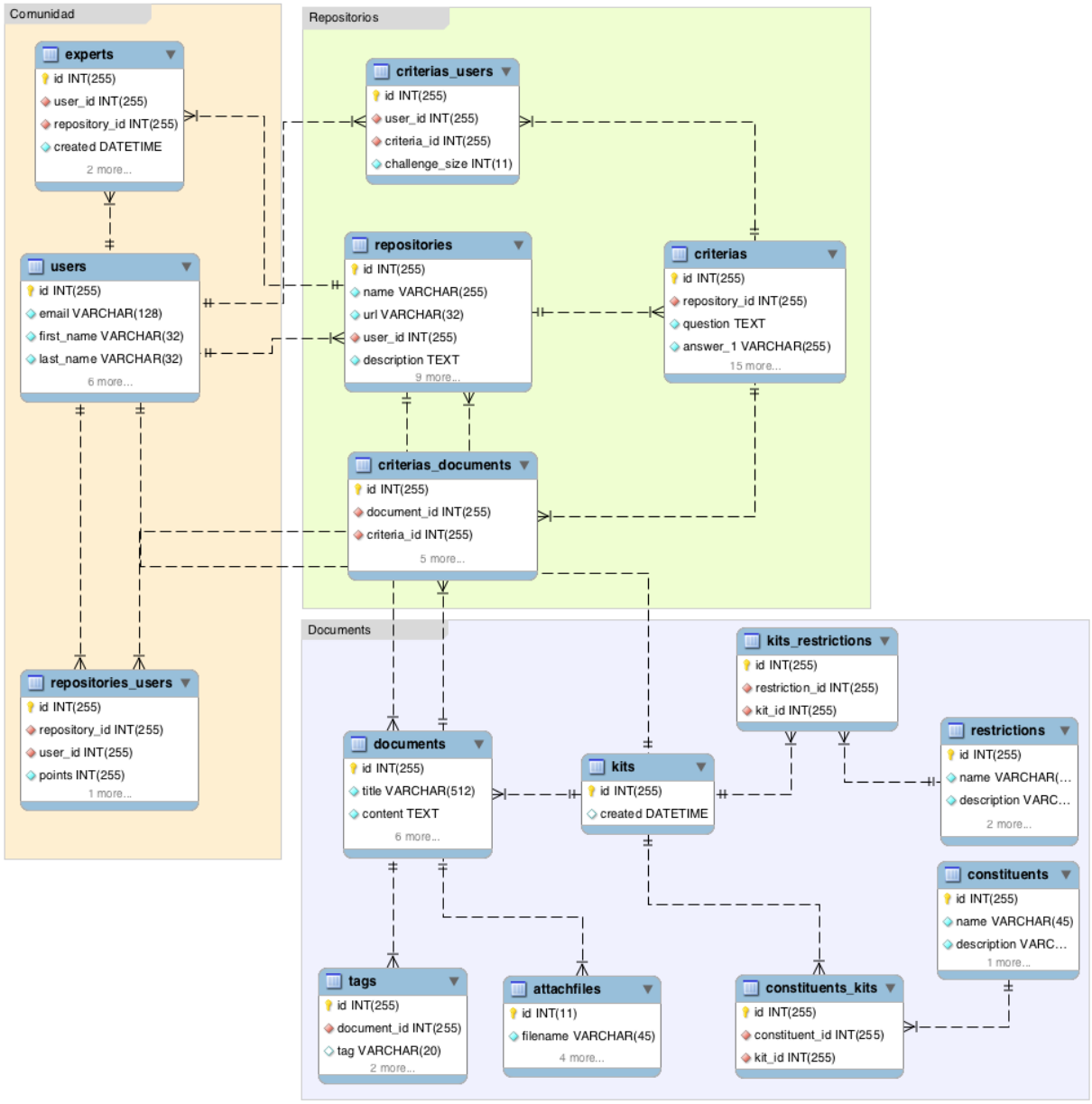
[10]Brian Mulloy. Teach a Dog to REST. [Online]. Visitado el 2 de abril de 2012. HYPERLINK "http://vimeo.com/17785736" http://vimeo.com/17785736

[11] Jaisen Mathai and Kevin Hornschemeier. Epiphany framework. [Online]. Visitado el 15 de Marzo de 2012. HYPERLINK "https://github.com/jmathai/epiphany" https://github.com/jmathai/epiphany

[12] Tomasz Jazwinski. DAO generator for PHP and Mysql. [Online]. Visitado el 5 de Febrero de 2012. HYPERLINK "http://phpdao.com/" http://phpdao.com/

[13] Joshua Bloch, "How to design a good API and why it matters," en OOPSLA '06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Oregon, 2006, pp. 506-507.

Anexo A. Modelo de datos en Repositorium



Anexo B. Descripción de los casos de uso para el cliente modelo

Caso de Uso	01. Autentificar a un usuario
Tipo	Crítico
Resumen	La API debe ser capaz de verificar un par usuario/clave. En caso de ser correcto debe entregar al usuario las autorizaciones necesarias, en caso contrario debe responder con un mensaje de error.
Precondición	Ninguna

Caso de Uso	02. Crear un nuevo usuario
Tipo	Deseable
Resumen	La API debe ser capaz de crear nuevos usuarios validando los datos ingresados. En caso de que los datos no sean válidos debe responder con un mensaje de error.
Precondición	Ninguno

Caso de Uso	03. Actualizar los datos de un usuario existente
Tipo	Deseable
Resumen	La API debe ser capaz de actualizar la información de un usuarios, incluyendo la clave. En caso de que los datos modificados no sean válidos debe enviar un mensaje de error.
Precondición	El usuario debe autentificarse previamente (CU 01)

Caso de Uso	04. Consultar los repositorios existentes
Tipo	Opcional
Resumen	La API debe ser capaz de consultar los repositorios activos. [Observación: este casos de uso quedará obsoleto en la siguiente versión de Repositorium]
Precondición	Ninguna

Caso de Uso	05. Realizar búsquedas dentro de un repositorio
Tipo	Crítico
Resumen	La API debe permitir realizar búsquedas en un repositorio. Si se encuentran resultados y el usuario tiene los puntos suficientes para descargar se debe responder con un conjuntos de documentos, en caso contrario la respuesta debe ser un conjunto de desafíos. Si la búsqueda no produce resultados debe responder con el mensaje correspondiente.
Precondición	El usuario debe autentificarse (CU 01) y ser miembro del repositorio a consultar (CU 14)

Caso de Uso	06. Obtener y responder desafíos
Tipo	Crítico
Resumen	La API debe permitir a los usuarios realizar desafíos (como resultado de una búsqueda o como solicitud directa) y de responder tales desafíos.
Precondición	El usuario debe autentificarse previamente (CU 01) y ser miembro del repositorio a consultar (CU 14)

Caso de Uso	07. Consultar las estadísticas de un repositorio
Tipo	Deseable
Resumen	La API debe permitir a las aplicaciones consultar estadísticas de un repositorio, tales como número de documentos y número de usuarios.
Precondición	Ninguna

Caso de Uso	08 Descargar documentos en un repositorio
Tipo	Crítico
Resumen	La API debe permitir a un usuarios, con el puntaje suficiente, descargar un documento del repositorio. A su vez debe descontar los puntos que costó tal descarga.
Precondición	El usuario debe autentificarse (CU 01), pertenecer al repositorio (CU 14) y realizar una búsqueda (CU 05) previamente

Caso de Uso	09 Agregar documentos en un repositorio
Tipo	Crítico
Resumen	La API debe permitir a los usuarios agregar documentos a un repositorio, incluyendo archivos.
Precondición	El usuario debe autentificarse (CU 01) y pertenecer al repositorio (CU 14)

Caso de Uso	10 Consultar usuarios en un repositorio
Tipo	Opcional
Resumen	La API debe permitir consultar un listado de usuarios miembros de un repositorio.
Precondición	Ninguna

Caso de Uso	11 Eliminar usuarios de un repositorio
Tipo	Opcional
Resumen	La API debe permitir, a los usuarios con rol de administrador, eliminar a otros usuarios de un repositorio. Se debe solicitar una confirmación de ésta acción indicando el nivel de riesgo que ésta presenta.
Precondición	El usuario debe autentificarse (CU 01) y contar con el rol de administrador

Caso de Uso	12 Consultar *tags* en un repositorio
Tipo	Opcional
Resumen	La API debe permitir obtener un listado de los *tags* asociados a los documentos de un repositorio.
Precondición	Ninguna

Caso de Uso	13 Eliminar documentos de un repositorio
Tipo	Opcional
Resumen	La API debe permitir, a los usuarios con rol de administrados, eliminar documentos de un repositorio. Se debe solicitar una confirmación de ésta acción indicando el nivel de riesgo que ésta presenta.
Precondición	El usuario debe autentificarse (CU 01) y contar con el rol de administrador

Caso de Uso	14 Agregar usuario a un repositorio
Tipo	Opcional
Resumen	La API debe permitir a un usuario suscribirse a un repositorio.
Precondición	El usuario debe autentificarse (CU 01)

Caso de Uso	15 Actualizar documentos de un repositorio
Tipo	Opcional
Resumen	La API debe permitir, a los usuarios con rol de administrados, actualizar el contenido de un documento.
Precondición	El usuario debe autentificarse (CU 01) y contar con el rol de administrador

Anexo C. Descripción extensa de los Recursos disponibles

Recurso	01 Lista de repositorios
Método	GET
Ruta	/repositories
Parámetros	-
Descripción	Entrega una lista de todos los repositorios disponibles en el sistema.

Recurso	02 Detalles de un repositorio
Método	GET
Ruta	/repositories/(\d+)
Parámetros	-
Descripción	Muestra el detalle del repositorio cuyo ID corresponda al número indicado en la URL.

Recurso	03 Estadísticas de un repositorio
Método	GET
Ruta	/repositories/(\d+)/stats
Parámetros	-
Descripción	Muestra algunas estadísticas, como numero de documentos o número de usuarios existentes en un repositorio.

Recurso	04 Tags de un repositorio
Método	GET
Ruta	/repositories/(\d+)/tags
Parámetros	-
Descripción	Muestra una lista de los tags existentes en un repositorio.

Recurso	05 Usuarios de un repositorio
Método	GET
Ruta	/repositories/(\d+)/users
Parámetros	-
Descripción	Muestra una lista de los usuarios que participan en un determinado repositorio.

Recurso	06 Buscar documentos
Método	GET
Ruta	/repositories/(\d+)/search:([^/]+)
Parámetros	fields=tags|title|content
Descripción	Permite realizar una búsqueda simple en un repositorio. El texto que acompaña a search en la URL corresponde al término que se busca, en el cuerpo de la petición se puede incluir opcionalmente el parámetro *fields* indicando si se desea buscar en los tags, el título o el contenido. Accesible sólo a usuarios autentificados

Recurso	07 Recuperar un documento
Método	GET
Ruta	"/repositories/(\d+)/documents
Parámetros	Id
Descripción	Permite recuperar un documento especificando su ID en el cuerpo de la petición. Este recurso sólo se permite a usuarios administradores previamente autentificados.

Recurso	08 Recuperar un documento aleatorio
Método	GET
Ruta	/repositories/(\d+)/documents/random
Parámetros	-
Descripción	Permite recuperar un documento aleatorio dentro un repositorio. Es necesario que el usuario esté autentificado.

Recurso	09 Obtener un desafío
Método	GET
Ruta	/repositories/(\d+)/challenges
Parámetros	-
Descripción	Entrega un desafío para un criterio aleatorio dentro del repositorio correspondiente. Es necesario que el usuario se haya autentificado previamente.

Recurso	10 Unirse a un repositorio
Método	POST
Ruta	/repositories/(\d+)/join
Parámetros	-
Descripción	Permite a un usuario unirse a un repositorio. Es prerrequisito que el usuario esté previamente autentificados.

Recurso	11 Desunirse a un repositorio
Método	POST
Ruta	/repositories/(\d+)/disjoin
Parámetros	-
Descripción	Permite a un usuario desunirse a un repositorio. Es prerrequisito que el usuario esté previamente autentificados.

Recurso	12 Agregar un documento
Método	POST
Ruta	/repositories/(\d+)/documents
Parámetros	Title, content, tags, files
Descripción	Permite agregar un nuevo documento al repositorio. Es necesario proveer al menos un titulo y un contenido en el cuerpo de la petición HTTP. El usuario debe estar autentificado para realizar esta petición.

Recurso	13 Eliminar un documento
Método	DELETE
Ruta	/repositories/(\d+)/documents
Parámetros	id
Descripción	Permite eliminar un documento de un repositorio. Es necesario proveer el id del documento en el cuerpo de la petición HTTP. Esta acción sólo está permitirá para usuarios con rol de administrador.

Recurso	14 Responder un desafío
Método	POST
Ruta	/repositories/(\d+)/challenge
Parámetros	Id, document_id
Descripción	Permite responder a un desafío. El usuario debe entregar el id del desafío y por cada documento debe agregar un par id:respuesta, donde id es el identificador del documento y respuesta es el valor a o b. El usuario debe estar previamente autentificado para realizar esta petición.

Recurso	15 Actualizar un documento
Método	PUT
Ruta	/repositories/(\d+)/documents
Parámetros	Id, title, content

Descripción	Permite a un usuario con rol de administrador actualizar el título o contenido de un documento. Debe entregar en el cuerpo de la petición el id del documento y el nuevo título ó contenido. El usuario debe estar autentificado para realizar esta acción.

Recurso 16 Agregar un nuevo usuario

Método	POST
Ruta	/users
Parámetros	Name, lastname, email, password
Descripción	Permite agregar a un usuario al sistema. Se deben proveer el nombre, apellido, email y un password.

Recurso 17 Autentificar a un usuario

Método	POST
Ruta	/users/login
Parámetros	Email, password
Descripción	Permite validar un par email-password para autentificar a un usuario en el sistema.

Recurso 18 Obtener los datos de un usuario

Método	GET
Ruta	/users/([^/]+)
Parámetros	-

Descripción	Permite a un usuario administrados obtener los datos básico de un usuario.

Recurso	19 Listar repositorios de un usuario
Método	GET
Ruta	/users/([^/]+)/repositories
Parámetros	-
Descripción	Permite obtener una lista de los repositorios en los que un usuario participa

Recurso	20 actualizar los datos de un usuario
Método	PUT
Ruta	/users/([^/]+)
Parámetros	Name, lastname, email, oldpassword, newpassword
Descripción	Permite a un usuario actualizar sus datos o su password. El usuario debe estar previamente autentificado y sólo podrá modificar sus propios datos.

Anexo D. Tabla de relación entre Casos de Usos y Recursos

El siguiente gráfico presenta una relación entre los casos de uso identificados en la Sección 3 y los recursos implementados, de manera de observar que recurso satisface que caso de uso.