Ingeniería de Software II

Sistema "Twitteando para Ahorrar"

Grupo 6

1º Cuatrimestre 2013

LU	Nombre	email
667/06	Daniel Foguelman	dfoguelman@dc.uba.ar
767/03	Hernán Modrow	hmodrow@gmail.com
511/00	Leonardo Tilli	leotilli@gmail.com

1. Parte II

En este informe presentamos el análisis realizado al respecto de los distintos atributos de calidad identificados, exponiendo varios escenarios para cada uno de éstos. También mostramos la arquitectura definida para la aplicación, utilizando distintos diagramas (de conectores y componentes, y de alocación), y complementamos estos últimos mediante una descripción detallada de la interacción entre los distintos componentes y artefactos. También explicamos las decisiones que nos llevaron a definir la arquitectura, y cómo ésta cubre los distintos escenarios de calidad.

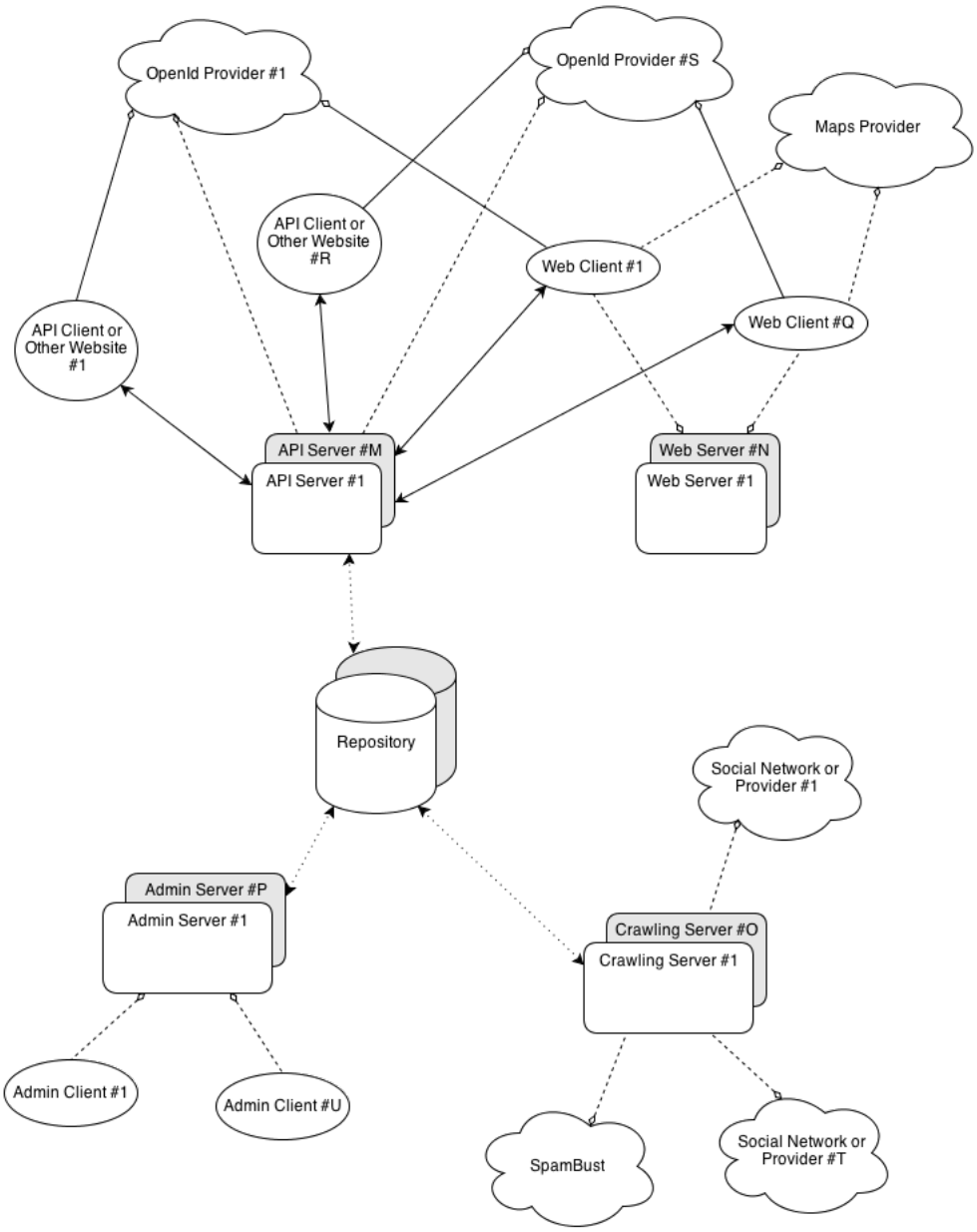
Por último, compartimos nuestra apreciación al respecto de las similitudes y diferencias entre la modalidad de trabajo para el primer trabajo práctico y éste, comparando metodologías y alcances.

Atributos de Calidad y Escenarios

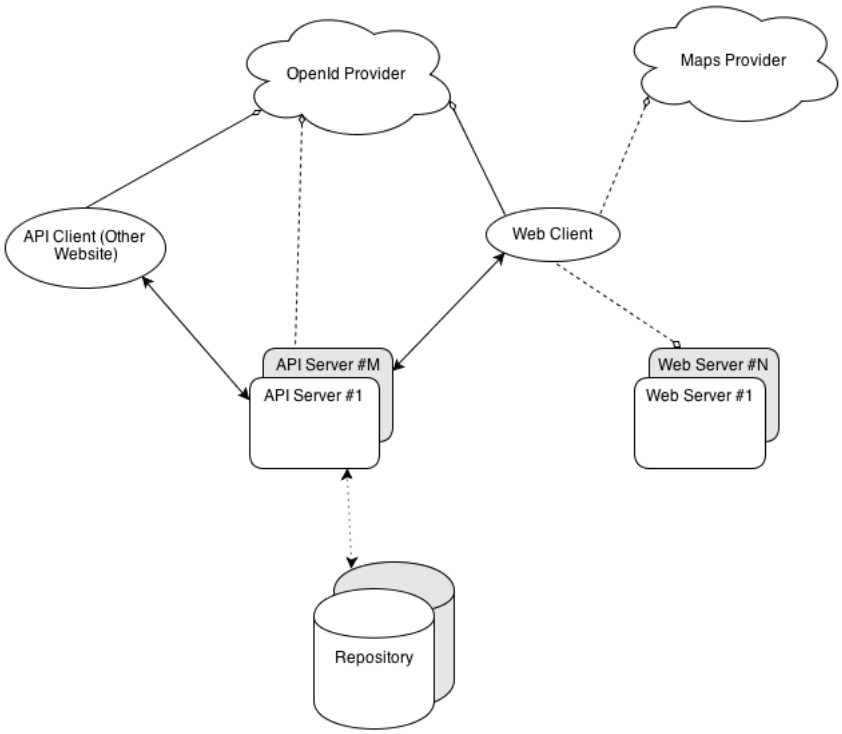
Explicación de la Arquitectura

Diagrama de Componentes y Conectores

Arquitectura Global



Consulta Web o API



En el diagrama de la consulta web se identifican dos tipos de clientes, que son los clientes del sitio Web y los que consumen únicamente la API web. Tenemos también dos tipos de componenetes servidor propios, que son los servidores del sitio Web y el servidores de la API web. Además, el diagrama nos muestra el repositorio de datos de configuración y consulta. Por último, tenemos distintos tipos de servicios externos que son consumidos por los componentes anteriores, como son los proveedores de OpenId y el proveedor de mapas.

Los clientes Web consumen el contenido estático directamente de los servidores del sitio Web, como puede ser el contenido HTML, JS, CSS e imágenes; se puede ver que el conector es de tipo cliente/servidor, lo que representa la sesión que arma el navegador con el servidor, y el bloqueo del thread de UI principal entre sucesivos requests (GET principalmente). En principio, este tipo de interacción es lo más simple posible, sin nigún tipo de autenticación; la mayor parte del contenido es cacheable y compone la UI de la aplicación (no tiene datos sensibles).

Tanto los clientes Web como los clientes de la API (como pueden ser otros sitios web) interactúan con el componente servidor API.

La autenticación es resuelta como parte de esta interacción, de acuerdo a la especificación de OpenId (`http://openid.net/specs/openid-authentication-2_0.html`):

- el cliente informa al componente servidor API el proveedor de OpenId que va a utilizar
- el componente API establece una sesión con el proveedor elgido (representamos esta sesión con un conector de tipo cliente/servidor) que podrá ser reutilizada para subsiguientes validaciones o para obtener datos extra del mismo cliente
- el cliente es redireccionado con el proveedor elegido para realizar la autenticación (representamos esta interacción como un envío de un mensaje a través de una conexión sincrónica)

- el componente API reutiliza la sesión establecida con el proveedor para verificar la información de autenticación y para obtener datos adicionales

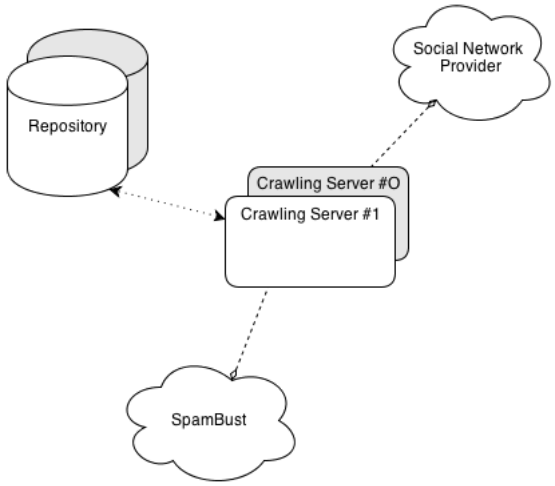
Más allá de manejar la autenticación, el componente servidor API responde a todos los pedidos de búsqueda o datos de configuración, de manera indistinta entre clientes Web o de API (usarían el mismo puerto); el tipo de conexión en este caso es asincrónica en ambos sentidos, para representar llamados tipo AJAX.

La interacción con el proveedor de mapas asumimos que se puede resolver directamente desde el cliente, mediante algún tipo de plugin descargado estáticamente desde el sitio Web.

El acceso al repositorio de datos y configuración lo hace únicamente el componente servidor API, para poder satisfacer los distintos requests, que agruparían todo el contenido dinámico de la aplicación.

Por último vale destacar que la cardinalidad del componente servidor API puede ser distinta a la del componente servidor Web, y dado que el primero atiende a consultas dinámicas, con workflows más complejos (como el de autenticación), y con mayor utilización de recursos, podría ser necesario escalarlo en mayor medida que al segundo, que sólo responde a requests de contenido estático. Además sólo es necesario implementar el proceso de autenticación en el componente servidor API, sustentado por el hecho de que sólo éste tiene acceso al repositorio, simplificando la arquitectura.

Crawling de Redes Sociales y Sitios Web



En el diagrama de crawling se identifica el componente de Crawling, el servicio de SpamBust, y el proveedor de red social (o sitio web de proveedor). Nuevamente, el diagrama nos muestra el repositorio de datos de configuración y consulta.

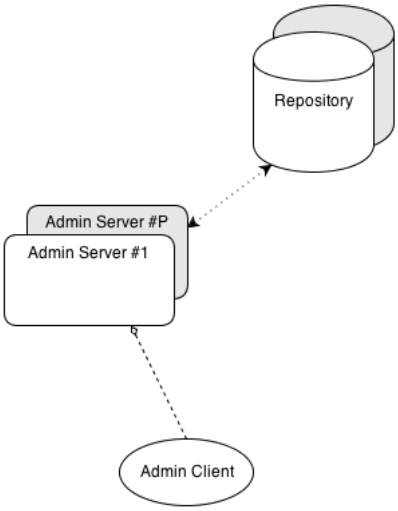
Los componentes de Crawling utilizan las distintas APIs provistas por las redes sociales para hacer las búsquedas de productos y ofertas; representamos esta interacción mediante un conector de tipo cliente/servidor, para generalizar el comportamiento de las distintas APIs. Podemos tener distintos componentes de Crawling, trabajando de manera independiente, cada uno realizando búsquedas distintas sobre la misma red social, o a distintas redes sociales.

El servicio de SpamBust es utilizado por los componentes de Crawling para filtrar ofertas de dudosa integridad, o que provienen de usuarios con baja reputación. Esta interacción también es representada mediante un conector tipo cliente/servidor, para generalizar las posibles implementaciones.

Por último, el repositorio es utilizado para consultar configuración de proveedores, usuarios, y redes sociales,

y también para persistir todos los datos de ofertas obtenidos en el proceso de crawling.

Administración



En el diagrama se identifican los componentes servidor y cliente de Administración. Además, el diagrama nos muestra el repositorio de datos de configuración.

El componente servidor de Administración provee tanto la UI como el contenido dinámico de la aplicación de adminsitración, que se utiliza para el ABM de proveedores, publicidad, y otras variables que puedan afectar el comportamiento de la aplicación. Además, maneja un esquema de autenticación y autorización propio, independiente de la aplicación principal, ya que la administración es realizada por usuarios internos. El acceso del componente cliente a la aplicación de adminitración se representa mediante un conector de tipo cliente/servidor.

Por último, el repositorio es utilizado para persistir toda la información de configuración, para ser utilizada por el resto de los sistemas.

Diagrama de Alocación

Arquitectura y Calidad

La arquitectura descripta en las secciones anteriores fue definida para cubrir los distintos escenarios detallados al comienzo del informe, y así cumplir con los atributos de calidad identificados.

Desde el punto de vista de usabilidad, el mayor aporte de la arquitectura es la separación total de la interfaz de usuario. Esto permite, en gran medida, independizar características de la UI que inciden directamente en la experiencia de usuario, como puede ser el maquetado.

Esta mayor independencia en el desarrollo de la UI nos ofrece varias ventajas:

- proveer distintas vistas que se adapten mejor a las preferencias del usuario, atacando temas como el uso eficiente, la personalización, y la sensación de confort
- ofrecer contenido estático de ayuda para toda la UI, accesible rápidamente y cacheable, cubriendo necesidades como el aprendizaje del sistema
- incluir validaciones simples y complejas del lado de cliente, minimizando la posibilidad de errores

Otro aspecto que aporta a la adaptabilidad y personalización, es el soporte de la arquitectura para identificar usuarios y permitirles persistir configuración propia.

El siguiente atributo de calidad más relevante, es el de rendimiento; en este caso la arquitectura lo ataca desde varios puntos. La arquitectura presenta varios componentes con responsabilidades diversas, y con bajo acoplamiento entre sí, facilitando partir las estrategias de performance entre todos estos componentes.

Desde el punto de vista de la experiencia de usuario, la arquitectura separa todos los request de contenido estático, de los requests de contenido dinámico, permitiendo paralelizar con distinta cardinalidad cada uno de esto puntos, destinando mayor cantidad de recursos a los componentes que más lo requieren.

En el caso de los componentes servidor API, se impulsa un aplicación stateless, en donde cada request tiene toda la información necesaria para ser atendido (header de autenticación y parámetros), lo que permite repartirlos de acuerdo a una estrategia de cola predefinida (round robin, recursos disponibles, etc.), aumentando el throughput (y disminuyendo el delay al no saturar recursos). Al alocar componentes relacionados en el mismo recurso físico, como una partición del repositorio y un componente servidor API, se minimiza el tráfico de red para requests que se puedan responder con esa partición, y así también el delay.

Desde el punto de vista de los componentes de crawling, se paraleliza el trabajo en distintos recursos físicos, aumentando el throughput.

El atributo de integrabilidad/extensibilidad se cubre desde varios puntos:

- el bajo acoplamiento entre componentes, permite su fácil modificación, en particular si el cambio no requiere cambiar la interfaz, y aún en estos casos se puede recurrir al uso de patrones conocido como el de adapter, bridge, etc
- la separación de la UI, da mayor flexibilidad a la hora de soportar nuevos dispositivos y plataformas
- los componentes de Administración dan soporte a la capacidad del sistema de modificarse en tiempo de ejecución.
- el acceso al servicio de SpamBust como un componente separado facilita el reemplazo de éste por una implementación propia, mientras que se respete la interfaz
- el escalamiento horizontal de recursos críticos, como el repositorio, permite adaptar su capacidad en tiempo de ejecución
- el acceso a las redes sociales de manera agnóstica de la API (utilizando patrones como Adapter, etc), agiliza la incorporación de nuevas redes sociales y sus APIs

Conclusión