API REST para Opinion Zoom

Metodología y detalles de implementación de API REST para Opinion Zoom.
Por Vicente Oyanedel Muñoz [vicente.oyanedel@ing.uchile.cl].

Contenidos

Introducción

En este informe se describe la metodología y el proceso llevado a cabo para implementar una API REST para el proyecto OpinionZoom (En adelante, OZ) en WIC.

Una API REST corresponde a un protocolo cliente/servidor sin estado que provee una biblioteca de recursos direccionables por un identificador unico (URI), publicados a través de HTTP y utilizando sus métodos (GET, POST y DELETE, en esta implementación).

Además, ciertas 'buenas practicas' han surgido en torno a esta arquitectura que proveen mejor usabilidad y robustez de la interfaz, las cuales fueron implementadas dentro del desarrollo de la API para OZ.

Las características de 'buenas practicas' que se implementaron son:

1. Interfaz HTTP: Para que los usuarios humanos puedan navegar por los recursos más facilmente que por JSON.
2. HATEOAS: Las respuestas incluyen vínculos a recursos relacionados para disminuir dependencia de la documentación y mejorar usabilidad.
3. Throttling: Limitación de consultas por unidad de tiempo para mejorar robustez.

Metodología

La metodología descrita corresponde a los pasos seguidos para llevar a cabo el desarrollo de la API, retro-alimentado por los aprendizajes adquiridos durante la implementación.

Esta metodología describe la implementación de una API REST para proyectos en producción que poseen un modelo de datos previamente existente.

La macro-estructura de la metodología se desprende de la Ingeniería de Software.

Dentro de el proceso genérico hay dos principales actores:

1. El desarrollador: Quien diseñará e implementará el producto de software.
2. La contra-parte: Quien sabe lo que se requiere que el producto de software provea.

Esta metodología asume una alta disponibilidad de la contra-parte, con el fin de recibir constante retro-alimentación y solución de dudas acerca del producto a desarrollar.

1. Concepción de la solución.

El objetivo de esta fase es llegar a un acuerdo con la contra-parte de que se hará, y de que forma la propuesta tecnológica va a satisfacer los requisitos. Esta fase concluye en cuanto se hace una propuesta técnica, y la contra-parte acepta la propuesta (luego de verificar que la propuesta técnica cumpla con sus requerimientos).

Dentro de esta fase es imperativo que el desarrollador logre capturar los requisitos de su contra-parte, lo cual considera intercambios de información, por medio de documentos y reuniones.

La propuesta del desarrollador debe ser, por ejemplo, pero no limitado a una lista de recursos, identificados unicamente y descritos, para mejor comprensión de la contra-parte, quien deberá verificar que dicha propuesta satisface sus necesidades. Quien, finalmente, deberá notificar al desarrollador en caso de ser aceptado, o retro-alimentar al desarrollador en caso contrario.

Esta fase concluye con la mutua aceptación de la propuesta técnica.

Otros temas relevantes a saldar en esta fase son:

- Definir los permisos: Un usuario (y súper-usuario) a que datos puede acceder/escribir y cuales no.
- Definir nombres de campos: Sintaxis CamelCase o underscore_case.
- Definir idioma de recursos y sus campos: Hay que ser consistente.
- Definir sintaxis de rangos de fechas: Al especificar un rango, ambas son inclusivas; inclusiva-exclusiva; etc.
- Definir tipo de respuesta: Solo JSON? O además XML? HTML también?
- Definir estrangulamiento ('Throttling'): Limite de cantidad de consultas para los usuarios, por hora, día, etc.

La urgencia de definirlos en esta fase depende de la disponibilidad de la contra-parte; si la disponibilidad es alta, puede consultarse por estas interrogantes en la medida que sean necesarias durante la implementación. Si es baja, mejor definirlos durante la concepción.

2. Diseño de la solución.

Antes de comenzar con la implementación, se debe diseñar el producto que se construirá, en base a la propuesta.

Estas bases clarificarán la factibilidad técnica de los requerimientos. Y en caso de surgir dificultades que impliquen la modificación de la propuesta técnica, ésta se cambiará y se ratificarán estos cambios con la contra-parte.

Esta fase concluye con un entregable a la contra-parte que describa más técnicamente los recursos.

2.1. Definición de recursos Se deben definir, para cada recurso individual:

- Identificador único de recurso (URI).
- Métodos HTTP (GET, POST, etc) que soporta para cumplir con la funcionalidad.
- Entradas y el medio de transmisión de estas (URL, JSON, Query, etc).
- Descripción del (recurso, metodo).

Cabe destacar que las definiciones pueden variar ampliamente entre un recurso y otro; de acuerdo a la funcionalidad pedida.

Y el medio de transmisión de las entradas pueden estar combinados dentro de un mismo recurso, por ejemplo, `clients/{client_id}/?keyword_id={keyword_id}` recibe entradas por URL y por Query.

Cliente (Web Auth)		
	api/clients/<cid>/keywords	Duplas de Idkeyword y trackeable del cliente asociado al usuario
	api/clients/<cid>/fb_pages	Duplas de idpagina y nombre de las páginas de Facebook contratadas
	api/clients/<cid>/user_info	Nombre de Cliente, nombre de usuario, correo, fecha de incorporación, fecha de último login
	api/client/<cid>/twitter_alerts	Duplas de nombre y fecha de término
	api/client/<cid>/fb_alerts	Duplas de nombre y fecha de término
	api/client/<cid>/contract_info	Plan Contratado, Servicios, Número de Keywords, Fecha de contratación, Próxima facturación, Monto de facturac
	api/client/<cid>/keywords	(trackeable; por post) Inserta una palabra en la base de datos
	api/client/<cid>/pages	(Nombre de pagina; por post) Inserta una página en la base de datos
Keywords		
	api/keywords/<kid>/polarities(?start=2012-01-01&end=2012-01-31)	Duplas de fecha y polaridad promediada diariamente
	api/keywords/<kid>/frequencies(?start=2012-01-01&end=2012-01-31)	Duplas de fecha y cantidad diaria de tweets
	api/keywords/<kid>/impacts(?start=2012-01-01&end=2012-01-31)	Duplas de fecha y impacto promediado diariamente
	api/keywords/<kid>/influences(?start=2012-01-01&end=2012-01-31)	Duplas de fecha e influencia promediada diariamente
	api/keywords/<kid>/influentials(?start=2012-01-01&end=2012-01-31)	Lista de los 12 usuarios más influyentes. Los elementos son duplas con id y screen_name
	api/keywords/<kid>/influentials/<uid>	Tres últimos tweets del usuario que mencionen la keyword
	api/keywords/<kid>/tusers/<uid> (redundante, nombre mas descriptivo)	Tres últimos tweets del usuario que mencionen la keyword
	api/keywords/<kid>/word_cloud(?start=2012-01-01&end=2012-01-31)	Duplas de palabra y peso relacionados a los tweets de la keyword
	api/keywords/<kid>/word_cloud/<palabra>(?start=2012-01-01&end=2012-01-31)	Últimos 5 tweets relacionados con la keyword que contengan esa palabra, que hayan sido emitidos en el período
	api/keywords/<kid>/average_age(?start=2012-01-01&end=2012-01-31)	Edad promedio de los usuarios que han hablado de la keyword en el período
	api/keywords/<kid>/gender_ratio(?start=2012-01-01&end=2012-01-31)	Porcentajes de hombres y mujeres hablando de la keyword en el período
	api/keywords/<kid>/complaints(?start=2012-01-01&end=2012-01-31)	Tuplas de fecha, cantidad de reclamos y cantidad de no reclamos
	api/keywords/<kid>/twitter_alerts	Tuplas de alertas, con nombre, frecuencia, polaridad, impacto, influencia y número de reclamos

Figure 1: Ejemplo de una definición de recursos. En blanco los recrusos con GET y azul con POST.

2.2. Inspección del modelo de datos Con el fin de validar la factibilidad de la propuesta, se debe inspeccionar el modelo de datos, antes de comenzar a implementar.

La fase de inspección se puede apoyar fuertemente de:

1. Documentación de la aplicación.
2. Consultas existentes en el código legado.
3. Reuniones con otros desarrolladores.

Es necesario que el desarrollador inspeccione el modelo de datos, por si mismo, para verificar que los recursos requeridos puedan entregar ('output') y aceptar ('input') la información de la propuesta técnica. Cualquier infactibilidad debe ser notificada a la contra-parte y ver como solucionar el problema para poder cumplir con el requisito.

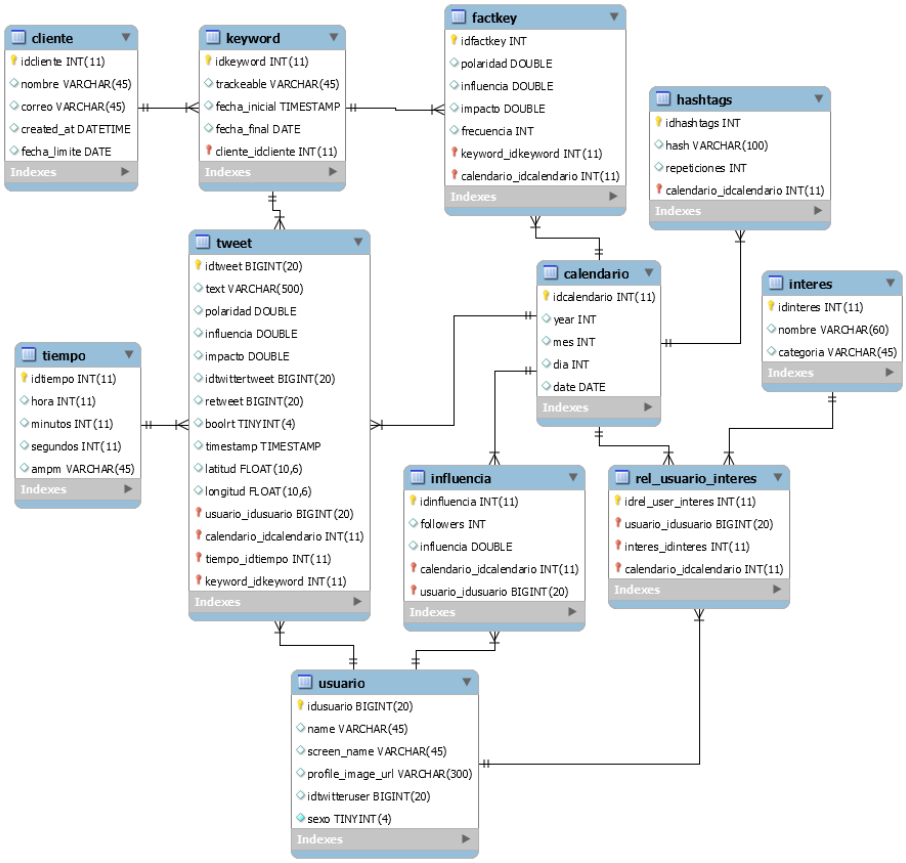

Figure 2: Ejemplo de buena documentación que puede entregarse para inspeccionar el modelo de datos

3. Implementación.

Como resultado de la fase anterior se tiene:

1. Listado de recursos, que incluye sus URI, métodos HTTP, entradas y descripción de su salida.
2. Análisis del modelo de datos, y por consiguiente fe en que el modelo satisface la propuesta.

Notar que la fase previa no se tiene clara la salida exacta que tendrán los métodos, dado que el modelo de datos existente puede imponer restricciones (funcionales o de rendimiento) improvistas.

Esta fase presenta un esquema iterativo/incremental, en el cual se comienza implementando la funcionalidad básica de cada recurso, y luego se va añadiendo más funcionalidades (seguridad, confiabilidad, nombres de campos, estandarización, navegabilidad, etc). Luego de cada iteración de esta fase, se debe hacer una fase de pruebas para comprobar el correcto funcionamiento del código.

La metodología de programación propuesta para cada iteración es de 'Code & Fix' para cada recurso.

Esta fase presenta un esquema iterativo, compuesto de los siguientes hitos.

3.1. Prototipos Dar funcionalidad básica a los recursos propuestos. Es decir, dado el set de entradas y la URI, obtener una respuesta en el formato pedido (JSON, XML, HTML, etc).

Esta fase incluye la codificación de las URL en base a las URI, y de los procedimientos para resolver las peticiones a los recursos.

Concluye con el funcionamiento mínimo viable de la API, de acuerdo a la funcionalidad pedida en la propuesta.

3.2. Seguridad Codificar los mecanismos que aseguren la lectura, escritura, y deleción de datos por los usuarios autorizados; de acuerdo a lo requerido.

Además definir las formas de autenticación. La más recomendable es por Token dado que mantiene el principio state-less; aun que se suele utilizar autenticación por sesión.

3.3. Nombres de campos Asegurar la consistencia de los nombres de campos y recursos, que tengan el mismo idioma y convención de nombres.

3.4. Definir HATEOAS En base a las relaciones entre los recursos, añadir a las respuestas enlaces (hipervínculos) a otros recursos.

```
{
    "client_id": 1,
    "user_id": 1,
    "name": "juaco",
    "email": null,
    "last_login": "2018-01-08T14:05:03Z",
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8000/rest-api/clients/1/"
        },
        {
            "rel": "contract_info",
            "href": "http://localhost:8000/rest-api/clients/1/contract_info"
        },
        {
            "rel": "keywords",
            "href": "http://localhost:8000/rest-api/clients/1/keywords/"
        },
        {
            "rel": "facebook_pages",
            "href": "http://localhost:8000/rest-api/clients/1/fb_pages/"
        },
        {
            "rel": "twitter_alert_definitions",
            "href": "http://localhost:8000/rest-api/clients/1/twitter_alerts/"
        },
        {
            "rel": "twitter_alert_records",
            "href": "http://localhost:8000/rest-api/clients/1/twitter_alert_records/"
        },
        {
            "rel": "facebook_alert_definitions",
            "href": "http://localhost:8000/rest-api/clients/1/fb_alerts/"
        },
        {
            "rel": "facebook_alert_records",
            "href": "http://localhost:8000/rest-api/clients/1/fb_alert_records/"
        }
    ]
}
```


Figure 3: Respuesta JSON navegable por su campo de link muestra HATEOAS

3.5. Definir Documentación En este punto, la entrada y salida de todos los métodos sobre los recursos debe estar completamente definida. Por lo que la definición de la documentación es directa de la implementación.

Se propone utilizar la especificación OpenAPI para documentar, dado que provee un lenguaje simple, es extensible a cambios y permite transformarse directamente a HTML.

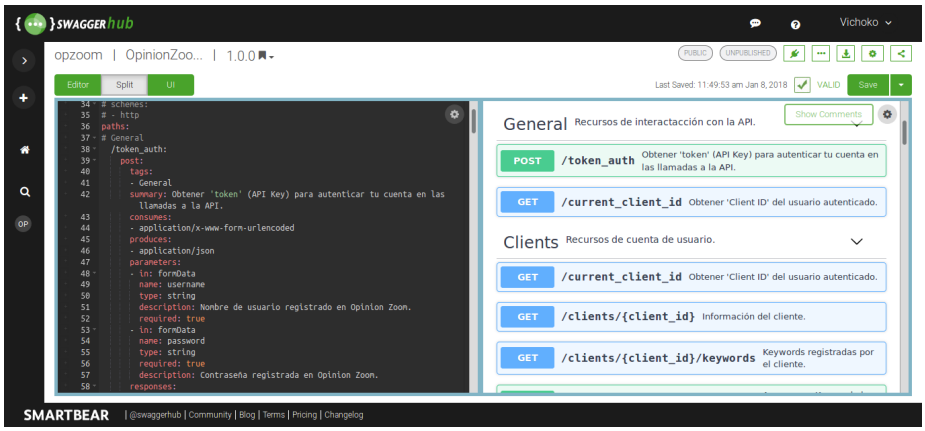

Figure 4: Herramienta para escribir y visualizar documentación en OpenAPI

El archivo YAML con la especificación, se puede exportar a HTML utilizando diversas herramientas en internet. La herramienta esteticamente más parecida a SwaggerHub es Swapy.

4. Implantación Luego de tener el producto listo, el producto debe ser puesto en marcha. Capacitando a los usuarios clave para que puedan mantener la API.

Implementación para OpinionZoom

Requisitos

- Desarrollador: Vicente Oyanedel
- Contra-parte: Andrés Córdova

Para la implementación de la API REST para OZ, la fase de concepción fue bastante directa dado que Andrés entregó un documento que muestra las métodos que debe proveer la API. Luego, el mapeo a recursos es bastante directo, produciendo una propuesta de API compuesta de una lista de recursos con sus URI y descripción que se desprendió del documento entregado.

Luego, se validó y modificó la propuesta en conjunto con Andrés, hasta que se llegó un acuerdo y comenzó la implementación.

Decisiones de diseño

Se utilizó Django Rest Framework, dado que es el framework más actualizado y recomendado para implementar este tipo de API en proyectos Django. Se intentó hacer uso de la mayor parte de las facilidades que provee, sin embargo, dado que ciertos recursos proveen información que se deriva de procesar datos de varias tablas; algunos recursos se debieron implementar mediante consultas SQL puras.

Para la seguridad, se implementaron métodos para asegurar que el dato al que se quiere acceder está relacionado con el usuario que consulta. Dado que la relación dato -> usuario es fuertemente dependiente del modelo de datos, no se pudo utilizar herramientas del framework para esta característica.

Se optó por utilizar la arquitectura REST con HATEOAS, dado que lo hace más usable y se recomienda como una buena práctica. El framework utilizado no provee herramientas para incorporar esta característica, por lo que se implementó manualmente apoyado por características de Django para adquirir las URL de recursos relacionados.

Se utiliza el idioma ingles y la convención de nombres underscore_case.

Para los rangos de fecha, se optó por especificarlas con fecha_inicio y fecha_final, ambas inclusivas; por requisito.

Para la documentación se opta por el uso de la especificación OpenAPI para definir API, dado que se visualiza estéticamente con SwaggerHub.

Implementación

La implementación se llevó a cabo durante 20 días hábiles.

Bitácora Para la **primera semana**, se levantó la API REST de lectura.

Para la **segunda semana**, se terminó el producto mínimo viable y se tradujo todo a ingles. Se comenzó con el sistema de autentificación con Token, la entrada de información mediante POST, la eliminación con DELETE, configuración de permisos y HATEOAS.

Para la **tercera semana**, se recibe nuevo modelo de datos. Se termina de implementar HATEOAS (navegación). Se comienza la documentación y se crean nuevos recursos que aparecieron. Se termina de hacer las fechas ambas inclusivas.

Para la **cuarta semana**, se termina de refactorizar la navegación, los modelos, las vistas antiguas y la API REST desarrollada las semanas anteriores. Se termina la documentación. Se conecta la API REST al JavaScript de la pagina actual. Y se ajustan los últimos detalles.

Principales dificultades

1. Modelo de datos cambió: Adecuarse al nuevo modelo de datos, refactorizar avances hasta el momento, refactorizar vistas antiguas.
2. Requisitos nuevos: Aparecieron nuevos recursos durante la marcha, para asegurar el completo funcionamiento de la API.

Resultados

La API REST fue implementada completamente con todas las características acordadas. Ésta fue documentada completa y rigurosamente en OpenAPI.

Referencias

El código puede encontrarse en la aplicación rest_api, en el repositorio: https://bitbucket.org/WICTeam/oz-django/src/?at=develop

Herramienta para escribir la documentación: https://app.swaggerhub.com/apis/opzoom/OpinionZoom/1.0.0#/

Herramienta para exportar YAML OpenAPI a HTML: https://gist.github.com/Vichoko/3c02fe94b5ee3d34c9d0d0

Django REST Framework: http://www.django-rest-framework.org/