Google Cloud EndPoints (Parte I)

1. Google cloud platform: Google Cloud EndPoints
- 1. 1. ¿Qué es Google Cloud Platform?

2. Google Cloud EndPoints.
- 2.1 Precios.
- 2.2. Instalación.
- 2.3. Crear la aplicación backend.
- 2.4 Componer la URL.
- 2.5. Testear el API.
  - 2.5.1 Probando el API por consola.
- 2.6 Usando anotaciones.

3. Mobile backend starter
- 3.1. Desplegar el backend.
- 3.2. Activar las notificaciones PUSH.
- 3.3. Librería cliente para Android.
- 3.4 Activando los mensajes Push
- 3.5 Estructura del cliente Android.

Google cloud platform: Google Cloud EndPoints.

Las aplicaciones móviles, en general, necesitan mecanismos para persistir o almacenar y gestionar la información que manejan. Si nos centramos en Android, disponemos de varios mecanismos; SharedPreferences, utilizar la base de datos relacional SQLite o guardarlo en ficheros json etc. Sin embargo, todas estas opciones sólo nos permite almacenar la información de forma local.

¿Qué pasa si necesitamos compartir esa información?. Pensemos en una red social, vamos a necesitar recibir información de la últimas actividades de publicación o mandar al servidor nuestras nuevas publicaciones para que estén disponibles para el resto de los miembros que puedan acceder a las mismas. En este caso, el almacenamiento local ya no es válido. Sólo se podría usar como mecanismo de persistencia si hay problemas con la red y no podemos enviar la información al servidor.

Para poder solventar este problema vamos a necesitar una aplicación a nivel de servidor (backend) que contenga la base de datos o el sistema de persistencia. También necesitaremos un protocolo o una técnica para transmitir los datos. Tenemos que pensar que la solución que tengamos a nivel transmisión debería ser útil no sólo para un tipo de cliente, sino soportar la gran mayoría o los más representativos. Si pensamos en el mundo móvil y web, deberíamos de poder tener una solución que sea operativa al menos tanto en android como en iPhone y trasladable mediante javascript a cualquier web.

Como servidor podemos usar entre otros muchos Google App Engine (GAE). GAE es una solución (PaaS: plataforma como servicio) desarrollada por Google que nos permite usar sus servidores para poder publicar nuestras aplicaciones web. Podemos ejecutar aplicaciones desarrolladas en Phyton, Java, Go, PHP y cualquier lenguaje que utilice la máquina virtual de java.

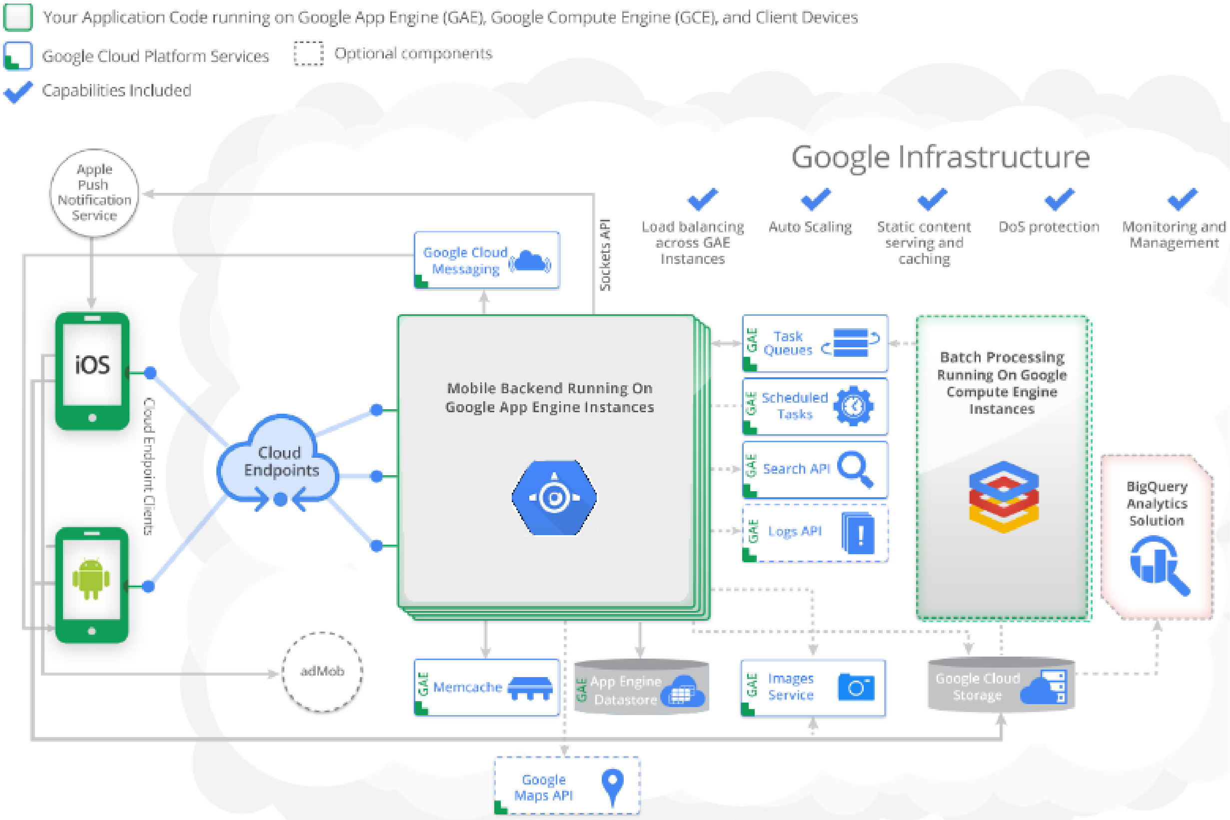
Para la comunicación entre el cliente y el servidor, el estilo arquitectónico más extendido hoy en día es REST (Representational State Transfer). REST aprovecha toda la potencia y simplicidad del protocolo HTTP. Mediante HTTP podemos intercambiar recursos entre el cliente y el servidor. Para ello, a través de los métodos HTTP podemos definir nuestra API web.

El uso de REST no nos libra de crearnos una solución cliente para comunicarse con el servidor y la lógica a nivel de servidor para responder a las solicitudes REST.

¿Qué es Google Cloud Platform?

Google cloud platform es la integración de todos los servicios de cloud computing de Google en una misma plataforma, dotándoles de mayor cohesión y homogeneidad. Mediante Google cloud platform vamos a poder construir un backend para una aplicación móvil. Gracias a la infraestructura de Google no vamos a tener que preocuparnos por la escalabilidad, ya que esta será automática. Algunas de las características que nos permite implementar son:

- Nos da soporte para almacenamiento y el procesado de datos fuera de los dispositivos móviles.
- Gestión y envío de notificaciones push a dispositivos Android e IOS.
- Autenticación del usuario a través de oAuth 2.0.

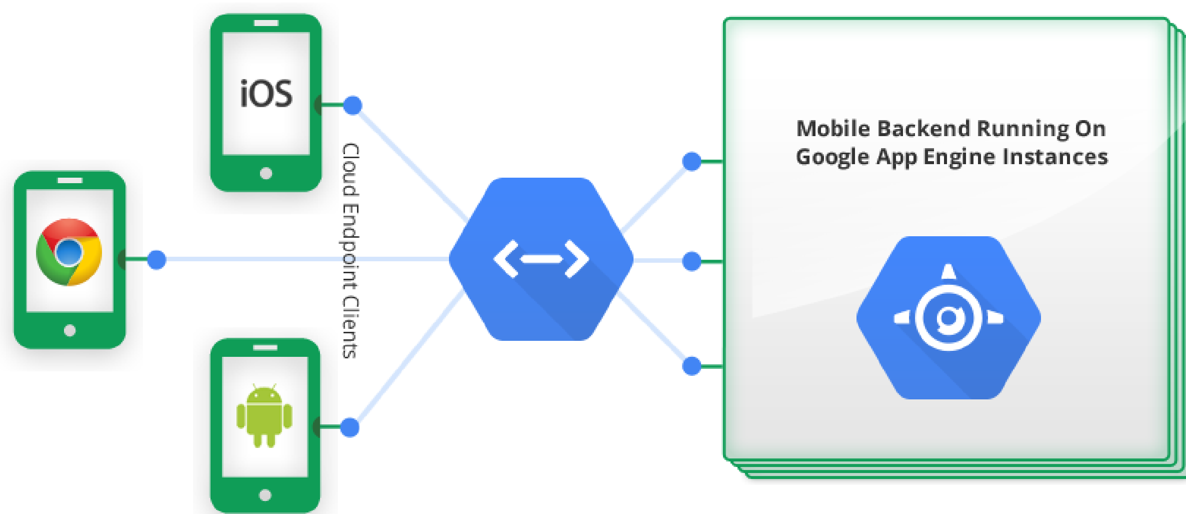


En la imagen podemos ver dos partes cláramente diferenciadas. Por un lado tenemos los dispositivos iOS y Android que se comunican a través de los Cloud Endpoints con la aplicación Mobile Backend que está desplegada en instancias del Google App Engine. Esta aplicación Mobile backend tiene acceso a todas las soluciones cloud de Google, como pueden ser la gestión de colas de tareas (Task Queues) o el memcache (un caché distribuido).

Por estar la aplicación desplegada en una instancia de GAE tenemos disponibles, sin ninguna interacción por nuestra parte, una serie de características, como el auto escalado, el balanceo a través de distintas instancias o la seguridad, rendimiento y seguridad.

Google Cloud EndPoints.

Los Google Cloud EndPoints son un conjunto de herramientas que nos permiten, de forma sencilla e incluso automáticamente, generar las APIs para poder comunicar nuestras aplicaciones clientes (webs y móviles) con una web backend.

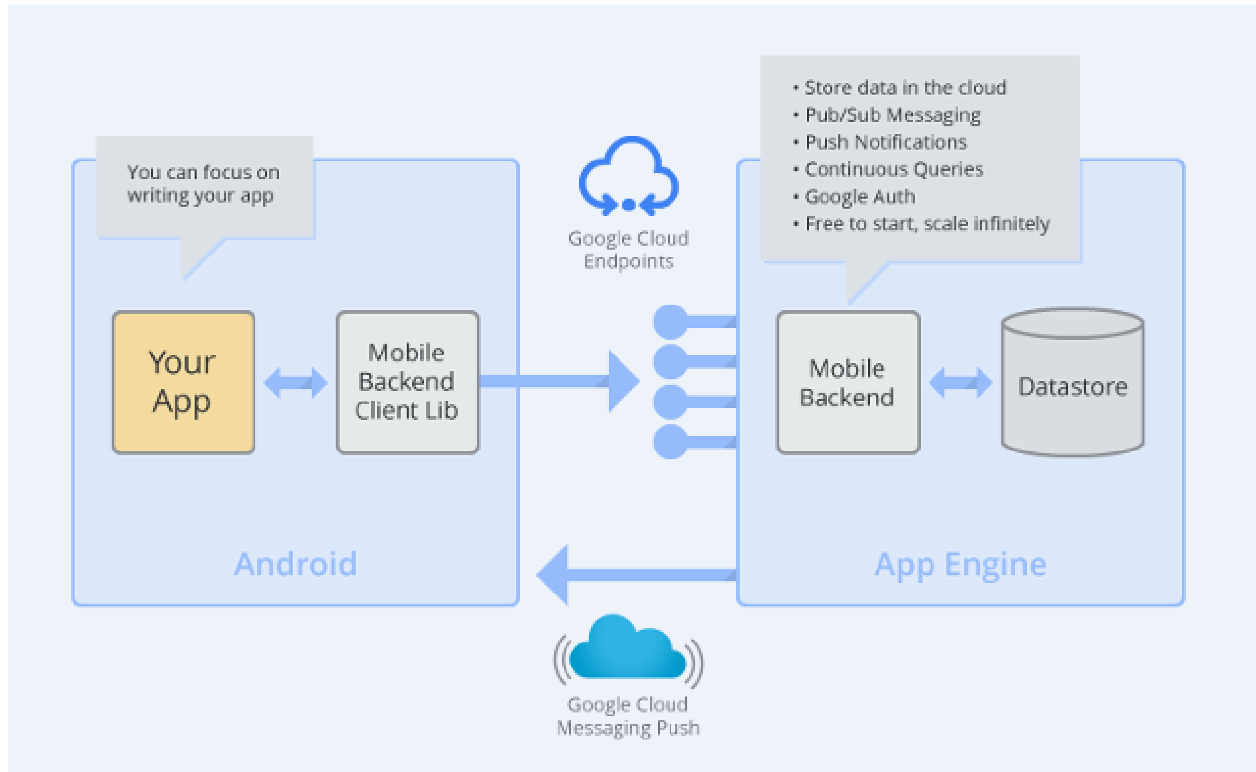


Es decir, con EndPoints vamos a disponer de unos mecanismos sencillos para poder crear y exponer y consumir nuestra API REST con la que compartiremos datos entre nuestras aplicaciones clientes y la parte backend. Toda la parte backend se va a almacenar y gestionar en el Google App Engine (GAP).

Podemos decir, que lo que vamo a tener, van a ser una serie de objetos de datos a los que vamos a poder realizar las típicas operaciones CRUD. Vamos a disponer de métodos para consultas, dar de alta nuevos registros, actualizar y eliminar registros. Y todo ello, se gestionará a través de una API Rest.

EndPoints nos facilita poder utilizar autenticación mediante OAuth 2.0 en nuestra API REST.

Tenemos disponible también con un servicio de notificación push a todos los clientes que registren dicho servicio.



Vamos a poder utilizar Google EndPoints para clientes android, clientes iOS y clientes javascript. Actualmente, falta la integración con clientes windows Phone.

Olvídate de escribir código repetitivo ya que con Google EndPoints vas a poder centrarte en la lógica de negocio de tu aplicación.

Precios.

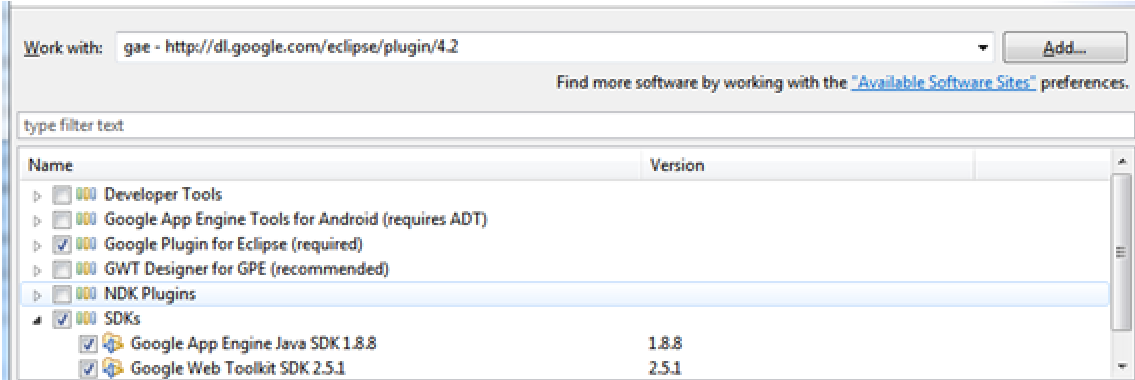
Endpoints es una herramienta gratuita. Si que tendremos que pagar por el resto de los productos del Google cloud platform que usemos, como por ejemplo, el almacenamiento para la persistencia(cloud datastore). Podremos utilizar todas las opciones gratuitamente siempre que estemos dentro de unos márgenes. Estos son bastantes generosos, con lo cual, no tendremos problemas para probar toda la tecnología sin necesidad de hacer un gran desembolso.

Los precios van variando, en general reduciéndose, con lo cual, lo mejor es consultar el precio actual. Puedes encontrar información sobre todos los productos de la plataforma cloud de Google en https://cloud.google.com/products/.

Instalación.

Para el ejemplo que vamos a seguir utilizaremos eclipse. Necesitamos instalar el SDK de Google App Engine y el Plugin para eclipse. Para ello, vamos a "Ayuda->Instalar nuevo software"

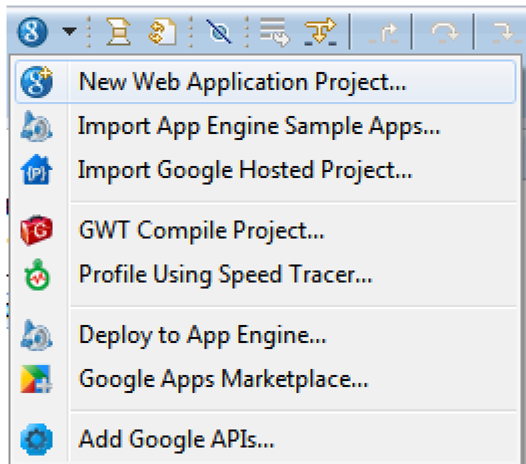
Es necesario instalar al menos el App Engine Java SDK 1.7.5. Para poder generar los Google EndPoints necesitamos también la última versión del Plugin.

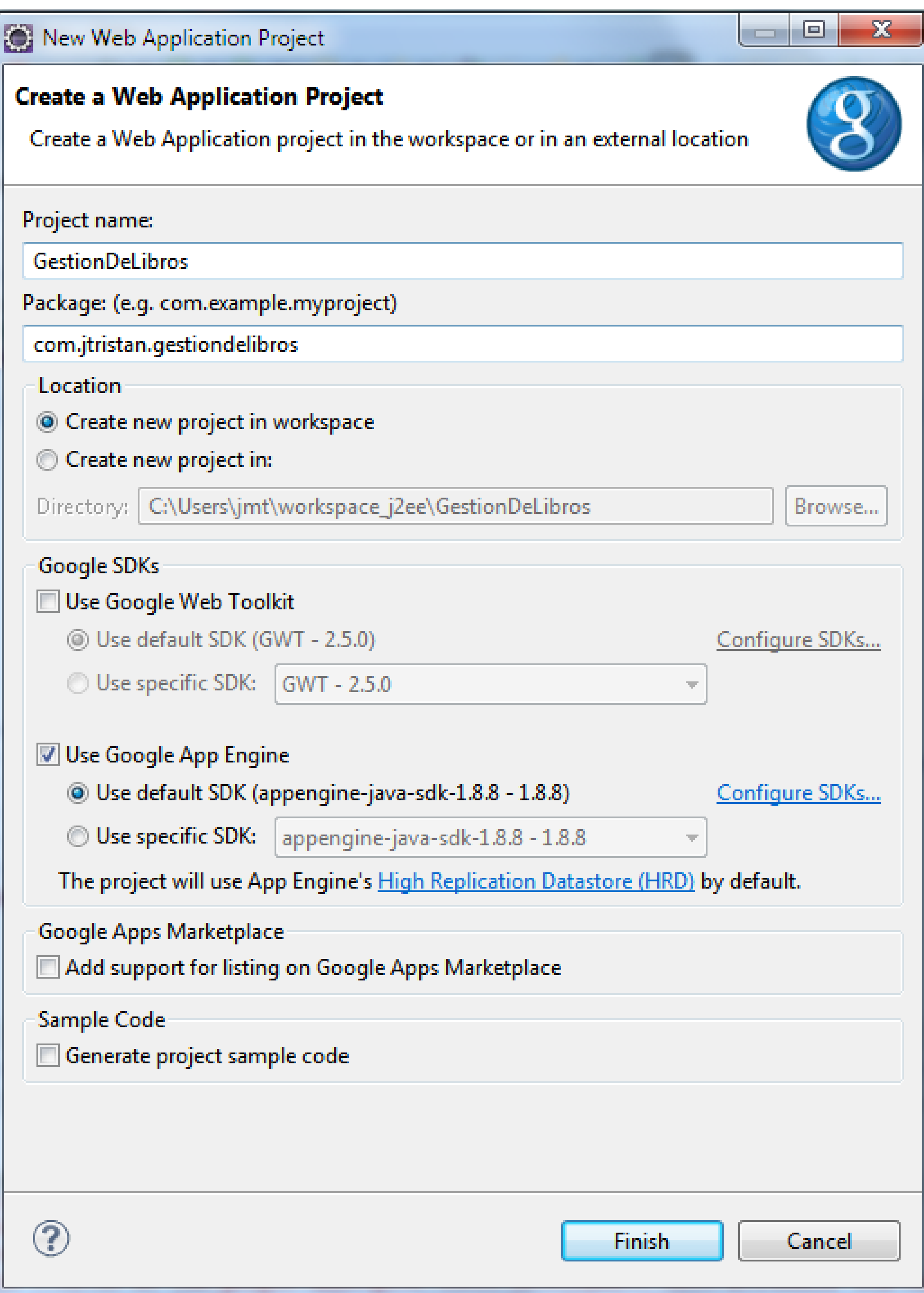


Crear la aplicación backend.

Para crear la aplicación backend vamos al icono de Google y seleccionamos "New web application project".

Introducimos el nombre del proyecto y el paquete. Desmarcamos el check de "Google Web Toolkit", ya que todo el proyecto va a ser a nivel de backend. También desmarcamos el check "Generate project sample code".


New Web Application Project...
Import App Engine Sample Apps...
Import Google Hosted Project...
GWT Compile Project...
Profile Using Speed Tracer...
Deploy to App Engine...
Google Apps Marketplace...
Add Google APIs...

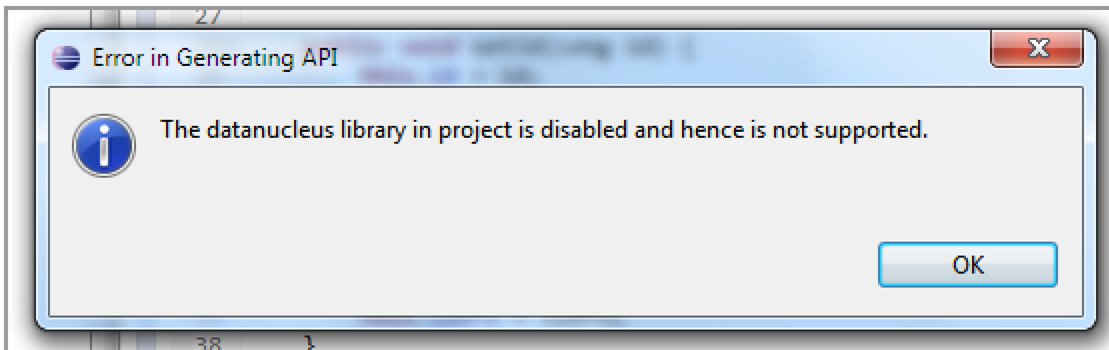


Lo primero que necesitamos es crearnos nuestra clase contenedora de datos. para ello, vamos a utilizar las anotaciones de JDO o JPA para poder persistir esta clase.

Para poder utilizar el plugin eclipse de Google y que nos cree toda la estructura (tanto las anotaciones de la API REST como las operaciones de gestión de la base de datos) es **obligatorio utilizar JDO o JPA a través de su implementación en Datanucleus**. Esto

puede suponer un problema, ya que la curva de aprendizaje de JDO o JPA es elevada y sus carencias en Google DataStore son grandes.

NOTA: Podemos utilizar cualquier otro framework como objectify-appengine, que tiene un API mucho más amigable, o en vez de trabajar contra cloud Datastore utilizar Cloud Sql. En estos casos, tendremos que desarrollar toda la parte backend. Si que podremos generar automáticamente las librerías cliente para comunicarnos con android, iOs o cualquier aplicación que use javascript. Si bien, el plugin de eclipse, está creado para facilitar todo el proceso de backend cuando utilicemos JDO o JPA, los Endpoints y su API son totalmente independientes del método de persistencia que utilicemos.



Mensaje de error en un projecto backend una vez desactivado la opción de "Usar Datanucleus JDO/JPA para acceder al datastore" y cuando se intenta generar automáticamente el Cloud Endpoint para la entidad.

En los siguientes ejemplos vamos a trabajar con JDO.

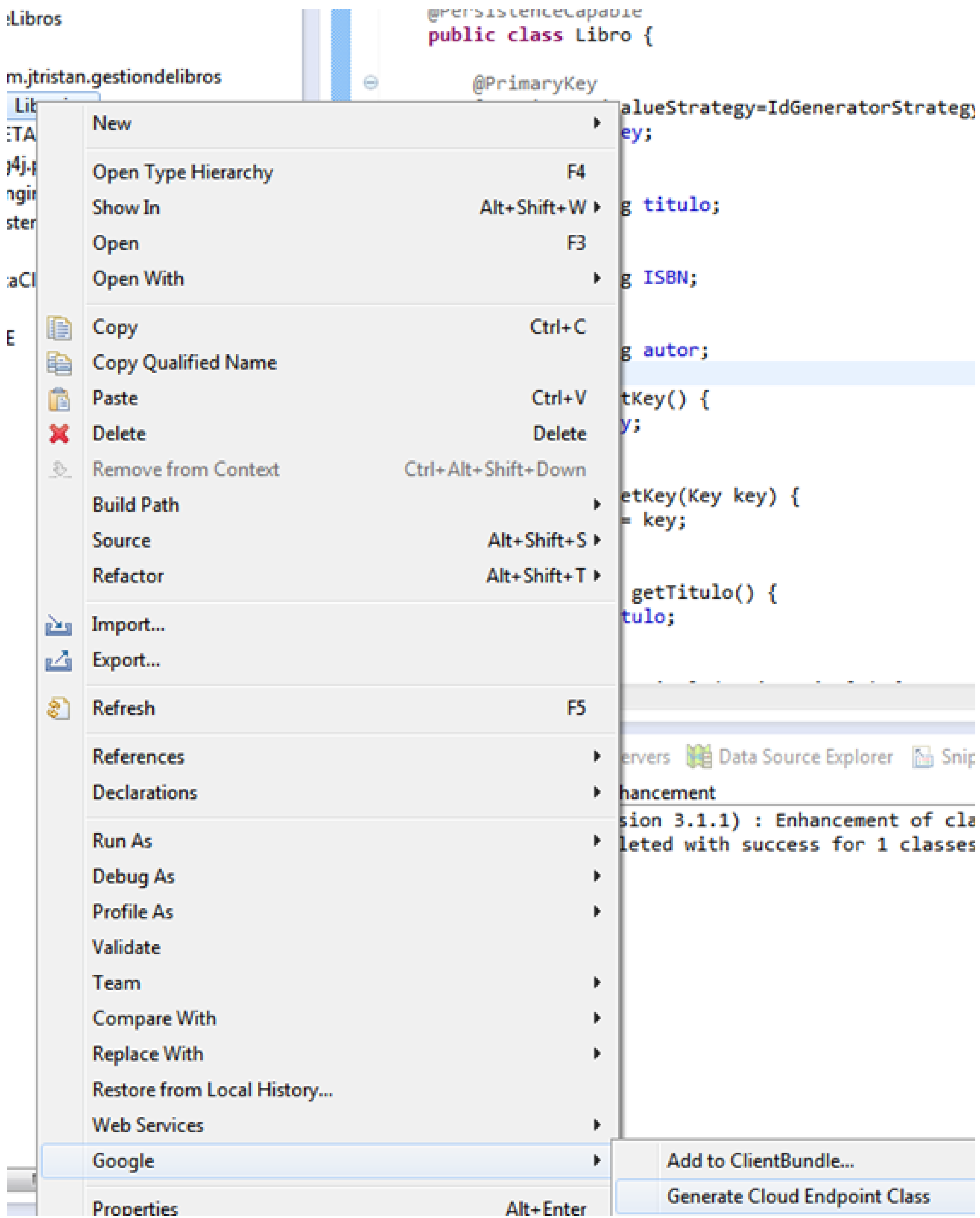
Marcamos la clase como **@PersistenceCapable** para indicar que esta clase va a tener persistencia. Igualmente, tenemos que indicar qué campos deseamos que se almacenen. Esto lo indicamos con la anotación **@Persistent.**

```
@PersistenceCapable
public class Libro {

   @PrimaryKey
   @Persistent(valueStrategy=IdGeneratorStrategy.IDENTITY)
   private Long id;
   @Persistent
   private String titulo;
   @Persistent
   private String autor;
   @Persistent
   private int puntuacion;
```

```
    //Utilizamos el usuario para saber quién graba los libros.
    @Persistent
    private User usuario;

        .... getters and setters

}
```

Cuando trabajamos con JDO y GAE es necesario definir un campo clave. Puede ser de tipo Key, Key codificado como cadena, Long o Cadena. Más adelante veremos con más detenimiento los tipos de claves. El campo debemos anotarle con **@PrimaryKey**.

El atributo **valueStrategy** de la anotación @Persistent indica que podemos tener ciertos valores que van a ser establecidos por el usuario o por DataNucleus. Se utilizan para los campos clave, para poder mantener una clave única. El único valor soportado por GAE es **IDENTITY**. Identity utiliza valores autogenerados.



Y ahora es cuando llega la magia de los EndPoints. Seleccionamos la clase y vamos a “Google->Generate Cloud Endpoint Class”. Esto nos genera dos clases:

- PMF: es la clase que gestiona la solicitud de las conexiones con el datastore.
- LibroEndPoint: es la clase para nuestra API, donde vamos a tener los siguientes métodos:
  - listLibro(): que nos devuelve una colección con todos los libros.

- getLibro(Long id): nos devuelve el libro que tiene el id que indicamos.
- insertLibro(Libro libro).
- updateLibro(Libro libro).
- removeLibro.

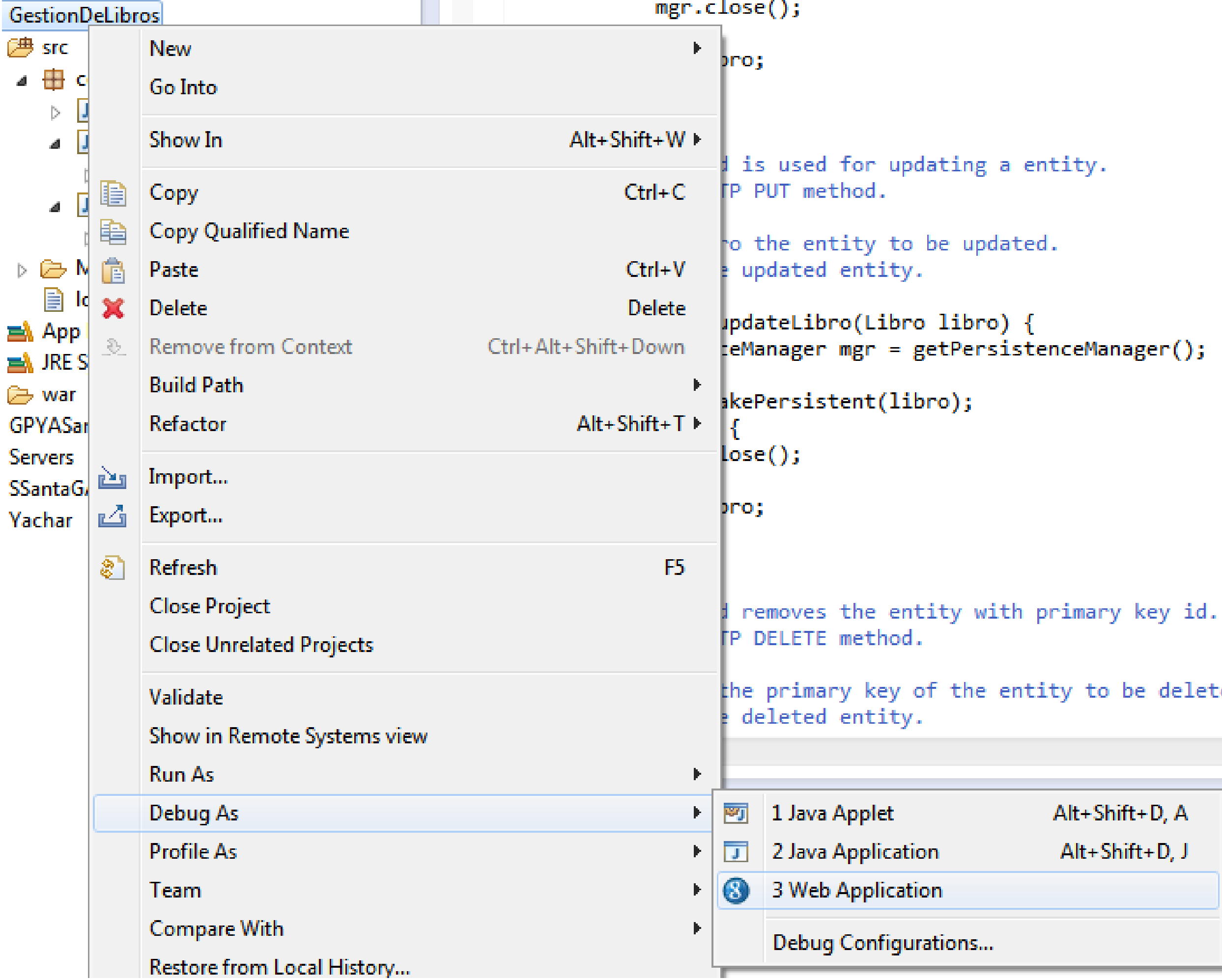
Estos métodos son los equivalentes a los métodos HTTP: GET (para la devolución de información), POST (para insertar), PUT (para actualizar) y DELETE (para eliminar).

Con un sólo paso, hemos creado el API REST.

Con la anotación **@Api** a nivel de clase indicamos cómo deseamos que se llame nuestra API. Cambiamos el nombre por “libro”.

```
@Api(name = "libro")
public class LibroEndpoint {
```

Con este esqueleto de código ya podemos gestionar datos con nuestra aplicación cliente. Vamos a probar la aplicación en local. Para ello, la ejecutamos o depuramos como una “Web Application”. GAE trabaja con Jetty como servidor para poder ejecutar la aplicación en local.



Componer la URL.

Merece la pena detenernos un momento en ver cómo se crea la URL para acceder a nuestra API. Este es el esquema:

http://host:port/ServiceRoot/ApiName/Version/Entity/param1/param2/paramN

Para nuestro ejemplo sería: http://localhost:8888/_ah/api/libros/v1/libro.

Nuestro host en local es localhost y el puerto suele ser el 8888. Podemos ver el puerto en las configuraciones del debug. "_ah" es el namespace reservado por AppEngine. De esta forma evitamos conflictos con cualquier otra aplicación de nuestro servidor.

ApiName: es el nombre del API y corresponde con el atributo de @Api().

Version: es la versión de nuestra API. Podemos tener distintas versiones de API para mantener compatibilidad con aplicaciones clientes que utilicen ya la API. Por defecto, es la 1.

Entity: nombre en minúsculas de la clase que devuelve el método de la API. Para el método eliminar utilizamos el sufijo del método.

Param1...paramN: parámetros que espera recibir el método.

Testear el API.

Para probar el API vamos a utilizar el programa **cURL.** Es una librería a nivel de línea de comando que podemos usar para transferir datos utilizando varios protocolos.

Como no tenemos datos, lo primero que vamos a hacer va a insertar un par de registros.

curl --header "Content-Type:application/json" -X POST --data "{'titulo':'Piensa en Java','ISBN':'9788489660342','autor':'Brucke Eckel','descripcion':'Completo libro sobre java','puntuacion':10}" http://localhost:8888/_ah/api/libro/v1/libro

```
D:\Otros Temas\Programa\Desarrollo\gae\curl-7.34.0-win32\bin>curl --header "Cont
ent-Type:application/json" -X POST --data "{'titulo':'Piensa en Java','ISBN':'97
88489660342','autor':'Brucke Eckel','descripcion':'Completo libro sobre java','p
untuacion':10}" http://localhost:8888/_ah/api/libro/v1/libro
{
  "key" : {
    "kind" : "Libro",
    "appId" : "no_app_id",
    "id" : "5629499534213120",
    "complete" : true
  },
  "titulo" : "Piensa en Java",
  "autor" : "Brucke Eckel",
  "descripcion" : "Completo libro sobre java",
  "puntuacion" : 10
}
```

Vemos que nos devuelve la clave.

curl --header "Content-Type:application/json" -X POST --data "{'titulo':'Principios de diseño de APIs REST','ISBN':'','autor':'Enrique Amodeo','descripcion':'Cómo diseñar tus APIs REST','puntuacion':10}" http://localhost:8888/_ah/api/libro/v1/libro

Si quisiésemos obtener todos los libros simplemente mandamos la URL.

curl http://localhost:8888/_ah/api/libro/v1/libro

```
D:\Otros Temas\Programa\Desarrollo\gae\curl-7.34.0-win32\bin>curl http://localho
st:8888/_ah/api/libro/v1/libro
{
  "items" : [ {
    "key" : {
      "kind" : "Libro",
      "appId" : "no_app_id",
      "id" : "5066549580791808",
      "complete" : true
    },
    "titulo" : "Principios de dise?o de APIs REST",
    "autor" : "Enrique Amodeo",
    "descripcion" : "C?mo dise?ar tus APIs REST",
    "puntuacion" : 10
  }, {
    "key" : {
      "kind" : "Libro",
      "appId" : "no_app_id",
      "id" : "5629499534213120",
      "complete" : true
    },
    "titulo" : "Piensa en Java",
    "autor" : "Brucke Eckel",
    "descripcion" : "Completo libro sobre java",
    "puntuacion" : 10
  } ]
}
```

Ahora probamos a obtener un libro en concreto.

curl http://localhost:8888/_ah/api/libro/v1/libro/5066549580791808

```
D:\Otros Temas\Programa\Desarrollo\gae\curl-7.34.0-win32\bin>curl http://localho
st:8888/_ah/api/libro/v1/libro/5066549580791808
{
  "key" : {
    "kind" : "Libro",
    "appId" : "no_app_id",
    "id" : "5066549580791808",
    "complete" : true
  },
  "titulo" : "Principios de dise?o de APIs REST",
  "autor" : "Enrique Amodeo",
  "descripcion" : "C?mo dise?ar tus APIs REST",
  "puntuacion" : 10
}
```

Vamos a actualizar el libro de Enrique, cambiándole la descripción. Como es una actualización utilizamos el método HTTP PUT.

curl --header "Content-Type:application/json" -X PUT --data "{'titulo':'Principios de diseño de APIs REST','ISBN':'','autor':'Enrique Amodeo','descripcion':'Cambiamos la descripción','puntuacion':10}" http://localhost:8888/_ah/api/libro/v1/libro

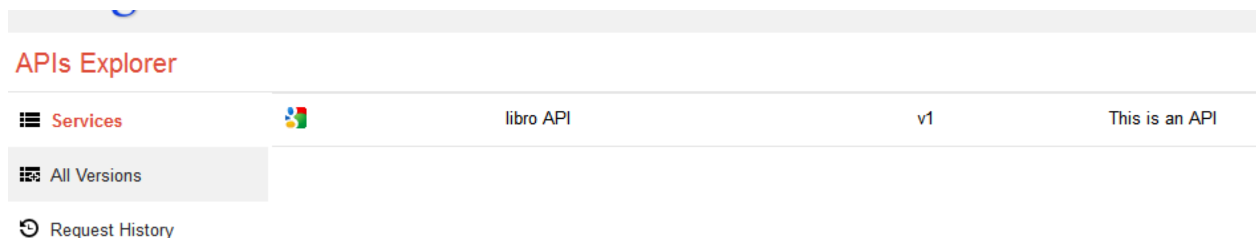
```
D:\Otros Temas\Programa\Desarrollo\gae\curl-7.34.0-win32\bin>curl --header "Cont
ent-Type:application/json" -X PUT --data "{'titulo':'Principios de diseño de API
s REST','ISBN':'','autor':'Enrique Amodeo','descripcion':'Cambiamos la descripci
ón','puntuacion':10}" http://localhost:8888/_ah/api/libro/v1/libro
{
  "key" : {
    "kind" : "Libro",
    "appId" : "no_app_id",
    "id" : "5910974510923776",
    "complete" : true
  },
  "titulo" : "Principios de dise?o de APIs REST",
  "autor" : "Enrique Amodeo",
  "descripcion" : "Cambiamos la descripci?n",
  "puntuacion" : 10
}
```

Finalmente, usaremos el método HTTP DELETE para eliminar el registro que indicamos con la clave que pasamos como parámetro.

curl -X DELETE http://localhost:8888/_ah/api/libro/v1/libro/5066549580791808

Probando el API por consola.

Google CloudEndpoints nos ofrece un sistema más cómodo de poder testear nuestra API. Podemos utilizar una consola a través del navegador desde la que podremos probar todos los métodos HTTP. Para ello introducimos la url: **http://localhost:8888/_ah/api/explorer** y veremos todas las APIs que tenemos registradas.



Nos da acceso a todos los métodos que podemos usar. Si una parte de nuestra API requiere de autentificación podemos “simularla” activando el check “Authorize request using OAuth 2.0”. En este caso, el usuario logueado siempre será example@example.com

Obtenemos una lista con todos los libros.

Services > libro API v1 > libro.gestLibros

fields | Selector specifying which fields to include i
Use fields editor

Execute

libro.gestLibros executed one minute ago time to execute: 2866 ms

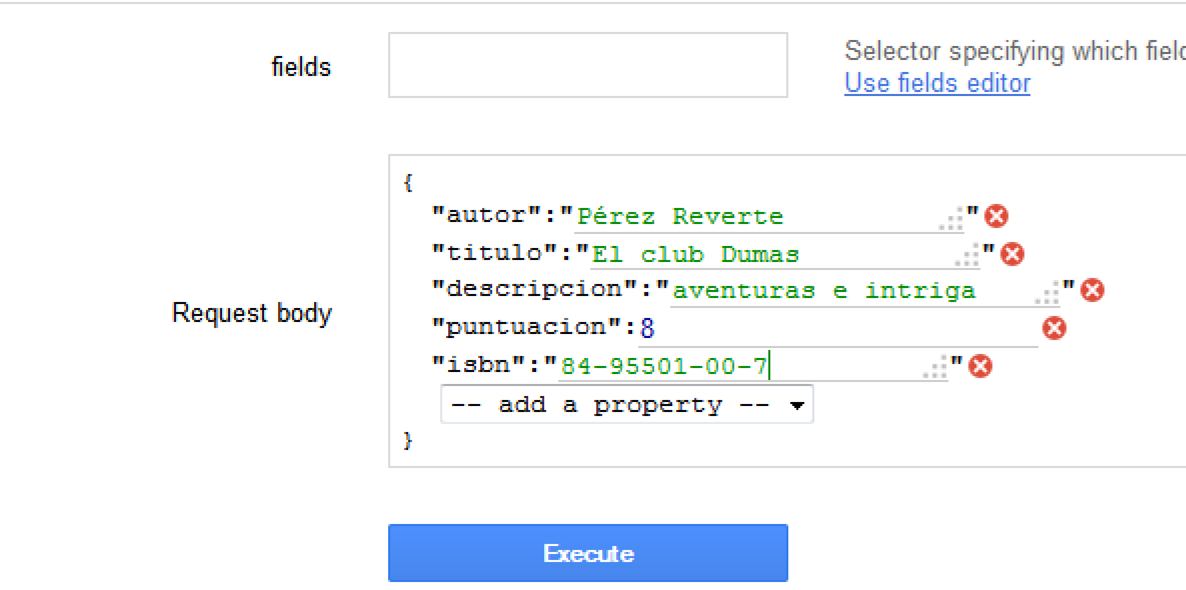
Request

```
GET http://localhost:8888/_ah/api/libro/v1/libreria/libros

X-JavaScript-User-Agent:  Google APIs Explorer
```

Response

```
200 OK

- Show headers -

-{
 -"items": [
  -{
   -"key": {
      "kind": "Libro",
      "appId": "no_app_id",
      "id": "5910974510923776",
      "complete": true
    },
    "titulo": "Principios de dise?o de APIs REST",
    "autor": "Enrique Amodeo",
    "descripcion": "Cambiamos la descripci?n",
    "puntuacion": 10
```

Insertamos un libro. Con el checkbox propiedades vamos indicando las que queremos mandar en la solicitud.



Y aquí tenemos la respuesta de que el proceso ha sido completado correctamente.

libro.libroEndpoint.insertLibro executed one minute ago time to execute: 632 ms

Request

```
POST http://localhost:8888/_ah/api/libro/v1/libro

Content-Type:  application/json
X-JavaScript-User-Agent:  Google APIs Explorer
```


```
-{
  "autor": "Pérez Reverte",
  "titulo": "El club Dumas",
  "descripcion": "es una novela de aventuras e intriga ",
  "puntuacion": 8,
  "isbn": "84-95501-00-7"
 }
```


Response

```
200 OK
```

- Show headers -


```
-{
 -"key": {
   "kind": "Libro",
   "appId": "no_app_id",
   "id": "5348024557502464",
   "complete": true
  },
  "titulo": "El club Dumas",
  "autor": "P�rez Reverte",
  "descripcion": "es una novela de aventuras e intriga ",
  "puntuacion": 8,
  "isbn": "84-95501-00-7"
 }
```

Usando anotaciones.

Como vemos, el código generado por los EndPoints es totalmente válido. Aún así, podemos utilizar distintas anotaciones para tener mayor potencia y posibilidades a la hora de generar nuestra API. En nuestro código ya hemos usado una de ellas, **@Api**, mediante la que indicamos cómo queremos que se llame nuestra API.

A nivel de clase, podemos usar estos atributos en **@Api**:

- name: nombre de la API que será usado como prefijo para el nombre de los métodos y rutas.
- version: para indicar la versión de nuestra API. Por defecto es v1.
- description: breve descripción que se expondrá en la información del servicio.
- documentationLink: link desde el que podemos acceder a la documentación del API.
- audiences: es requerido si la API requiere autenticación y si vas a usarla desde clientes android. Más adelante veremos un ejemplo de API con autentificación y cómo conectarnos desde el cliente.
- clientIds: clientes IDs que van a poder acceder a la API cuando estemos utilizando autenticación. Tanto la anotación audiences y clientIds se pueden utilizar a nivel de clase o a nivel de método, si sólo queremos gestionar con seguridad ciertos métodos.
- namespace: nos permite asignar un espacio de nombre. Si no lo asignamos se utilizará por defecto la información del proyecto en GAE; id_proyecto.appspot.com al revés, es decir, com.appspot.id_proyecto. Asignamos el nuevo espacio de nombres mediante la anotación @ApiNamespace.

```
@Api(
   name = "libros",
   version = "v1",
   namespace=@ApiNamespace(ownerDomain="jtristan.com",ownerName="JTristan")
)
```

- root. le indicamos la URL root bajo la cuál se van a exponer nuestros métodos de la API. Por defecto es https://id_app.appspot.com/_ah/api.

Los valores de estas propiedades se aplican a todos los métodos al menos que especifiquemos valores distintos en ellos.

A nivel de métodos, podemos destacar para la anotación **@ApiMethod** estos atributos:

- path: La URI de acceso al método. Si no indicamos nada, por defecto se utiliza el nombre del método.
- httpMethod: el método HTTP que vamos a usar.

Para indicar qué parámetros vamos a recibir en el método, usamos la anotación **@Named**. Hay varias anotaciones @Named, aquí usamos la de la clase javax.injected.Named. Si el parámetro es opcional, lo marcamos con **@Nullable**.

Vamos a modificar nuestro código para poder usar las anotaciones. Para el método listLibro le indicamos que el path de acceso va a ser libreria/libros y que lo llamaremos mediante un método GET.

```
@ApiMethod(name="gestLibros", path="libreria/libros", httpMethod = HttpMethod.GET)
public List<Libro> listLibro(
        @Nullable @Named("isbn") String isbn) {
        PersistenceManager mgr = getPersistenceManager();
        List<Libro> result = new ArrayList<Libro>();
        try {
            Query query = mgr.newQuery(Libro.class);
            for (Object obj : (List<Object>) query.execute()) {
                result.add(((Libro) obj));
            }
        } finally {
            mgr.close();
        }
        return result;
    }
```

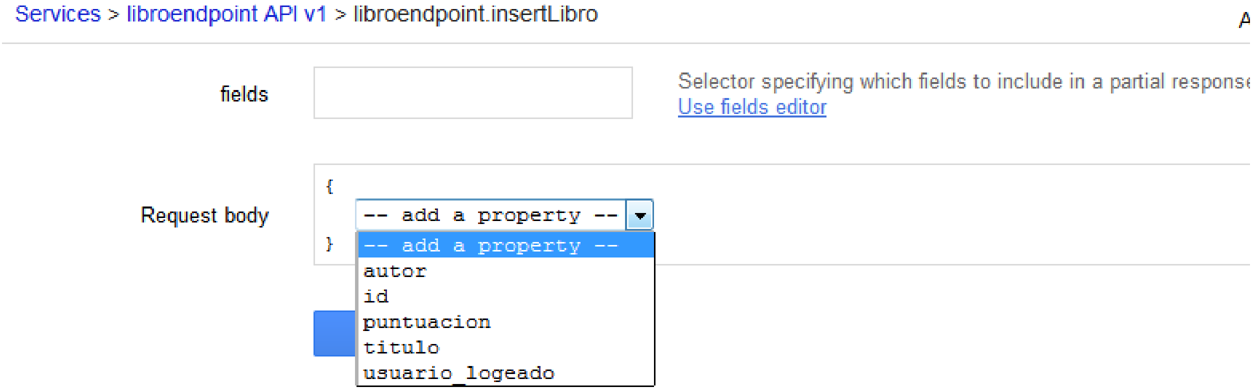
Con la anotación **@ApiResourceProperty** vamos a indicar cómo las propiedades son mostradas. Dispone de dos atributos:

- ignored: Mediante AnnotationBoolean.TRUE indicamos que no queremos que se muestre el campo.
- name: indica el nombre con el que la propiedad va a ser expuesta.

```
@PersistenceCapable
public class Libro {

  @PrimaryKey
  @Persistent(valueStrategy=IdGeneratorStrategy.IDENTITY)
  private Long id;
  @Persistent
  private String titulo;
```

```
    @Persistent
    private String autor;
    @Persistent
    private int puntuacion;
    @ApiResourceProperty(ignored = AnnotationBoolean.TRUE)
    @Persistent
    private String valorOculto;
    @ApiResourceProperty(name = "usuario_logeado")
    @Persistent
    private User usuario;

        .... getters and setters

}
```

Hemos añadido un nuevo campo a nuestra entidad que no deseamos que se visualice en el API: valorOculto. Para ellos la anotamos como ignored. Por otro lado, cambiamos el nombre del método usuario. En la parte backend seguiremos trabajando con la propiedad usuario pero quién haga uso de nuestra API verá que obtiene información del campo usuario_logeado.



Mobile backend starter.

Para probar Google EndPoints en un cliente Google ha diseñado un ejemplo introductorio, el Mobile backend starter. Desde la consola de Google Cloud podremos activar la parte backend y también podremos descargarnos el ejemplo de la librería cliente que contienen todos los procesos necesarios para poder compartir datos entre nuestras aplicaciones móviles clientes y el servidor. Además de poder almacenar datos, también nos ofrece otras características propias de las aplicaciones móviles:

- Notificaciones push: para enviar mensajes a través de Apple Push Notifications o Google Cloud Messaging para Android.
- Continuous queries:
- Autentificación de usuario: a través de la cuenta de Google.

Mobile Backend Starter

La aplicación nos permite mandar mensajes y que cualquier dispositivo registrado reciba los mensajes enviados desde otros dispositivos. Para ello utilizan

Desplegar el backend.

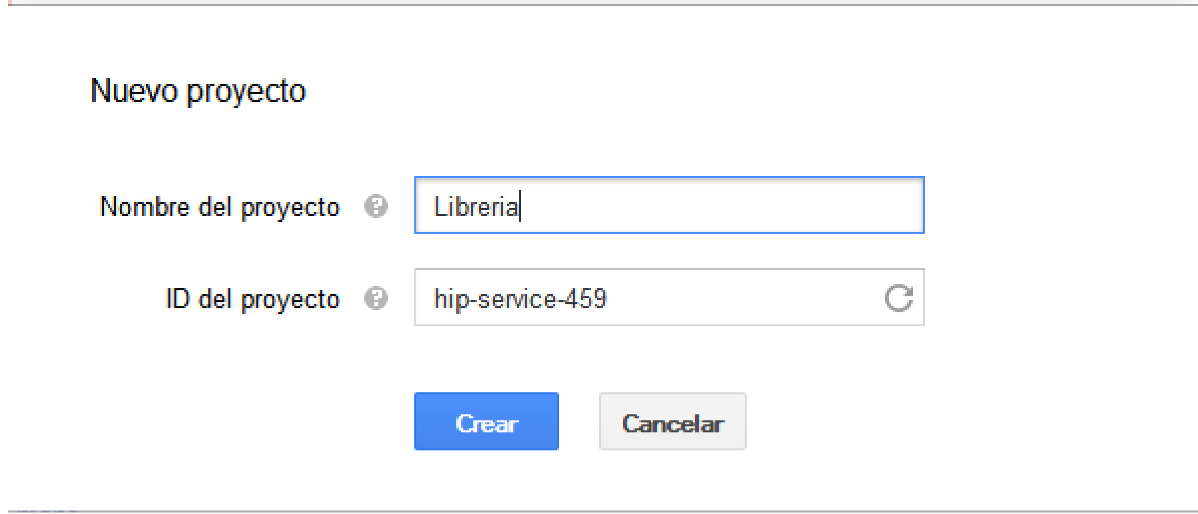
Necesitamos tener una cuenta en Google Cloud Platform (cloud.google.com). Una vez que tenemos la cuenta, entramos en la pestaña “Solutions”, en la opción “Mobile” y “Try it now”.

Vamos a crear un nuevo proyecto. Indicamos el nombre, podemos usar el ID del proyecto que nos dan.

Activar las notificaciones PUSH.

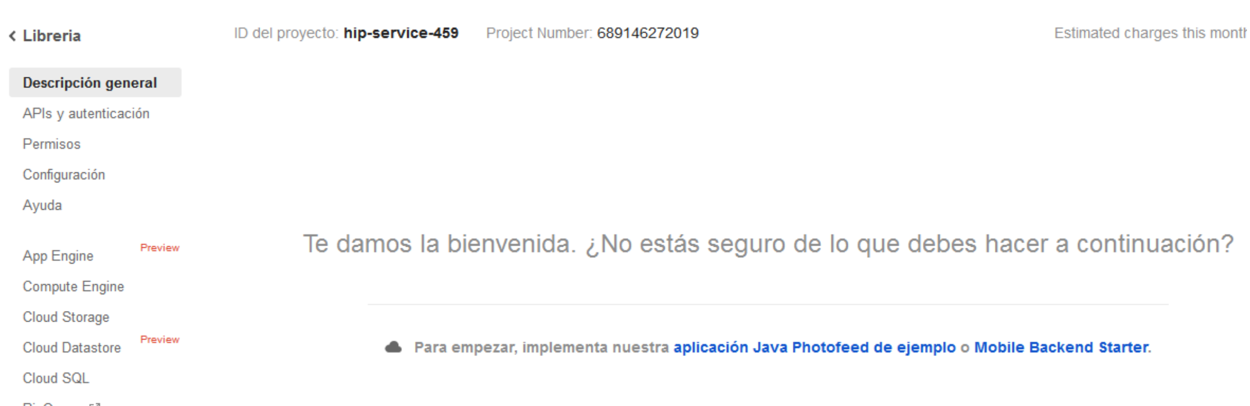
Para activar las notificaciones Push volvemos a la parametrización del Mobile Backend y activamos Google Cloud Messaging and iOS Push Notification. Si la notificación la queremos

hacer para dispositivos Android tenemos que introducir una API key.



Seguimos el link para crear el Mobile Backend Starter.

Get started by deploying our **Photofeed sample app** or **Mobile sample app**



Desplegamos (Botón “Implementar”).

Mobile Backend Starter

> **Nota:** Este proyecto está diseñado para desarrolladores de Android e iOS.

This model project uses Mobile Backend Starter to help you begin building your mobile app with Google Cloud Platform. It provides a general-purpose, ready-to-deploy backend plus client libraries for Android and iOS. (Learn more)

Mobile Backend Starter shows you how to:

- Store objects to the cloud from a mobile device
- Send real-time messages between devices
- Subscribe to changes in data stored on the cloud
- Authenticate users with Google accounts authentication

Pasos de la implementación:

1 Implementar el backend

Implementar

Pasos de la implementación:

1 Implementar el backend

Cloning 5 static files.
Uploading 5 files.
Uploaded 1 files.
Uploaded 2 files.
Initializing precompilation...
Uploaded 4 files.
Uploaded 5 files.
Uploaded 3 files.
Deploying new version.
Sending batch containing 5 file(s) totaling 38KB.

Pasos de la implementación:

1. Implementar el backend

   El backend móvil ya está implementado.
   Volver a implementar

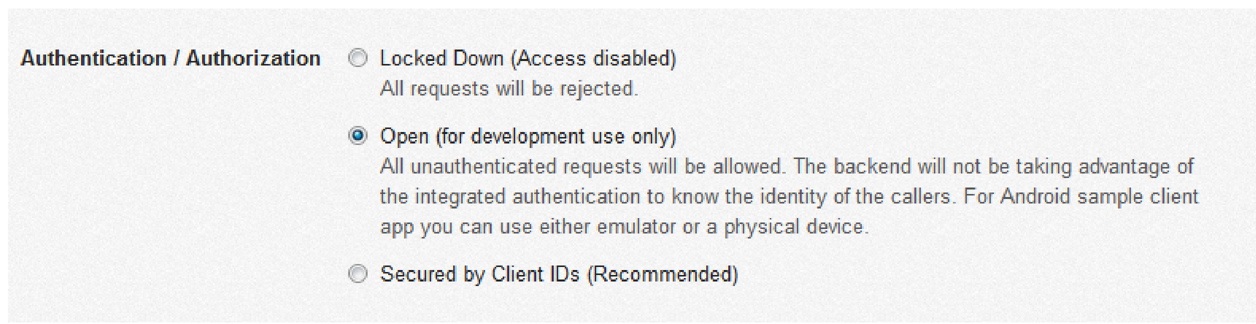
2. Open (for development use only) the backend to accept incoming requests via Settings

3. Abrir un cliente móvil:

   Cliente Android   Cliente iOS

Podemos descargarnos el cliente Android o iOs y también vamos a poder entrar en la parametrización del backend.

En parametrización vamos a poder indicar el tipo de autenticación/autorización y seleccionamos Open (for development use only). De esta forma, podemos utilizar nuestro cliente Android en un emulador o un en dispositivo físico ya que va a permitir todas solicitudes no autenticadas.



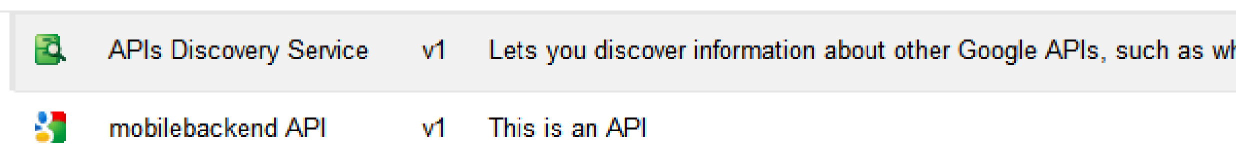
También podemos activar las notificaciones push. Si la activamos, en el caso de Android tendremos que introducir la API ley.

Para comprobar que el backend está funcionando introducimos la URL:

https://tu_id_proyecto.appspot.com/_ah/api/explorer.

Sustituimos tu_id_proyecto por el nombre del proyecto, en nuestro caso quedaría:

https://hip-service-459.appspot.com/_ah/api/explorer.



	APIs Discovery Service	v1	Lets you discover information about other Google APIs, such as wh
	mobilebackend API	v1	This is an API

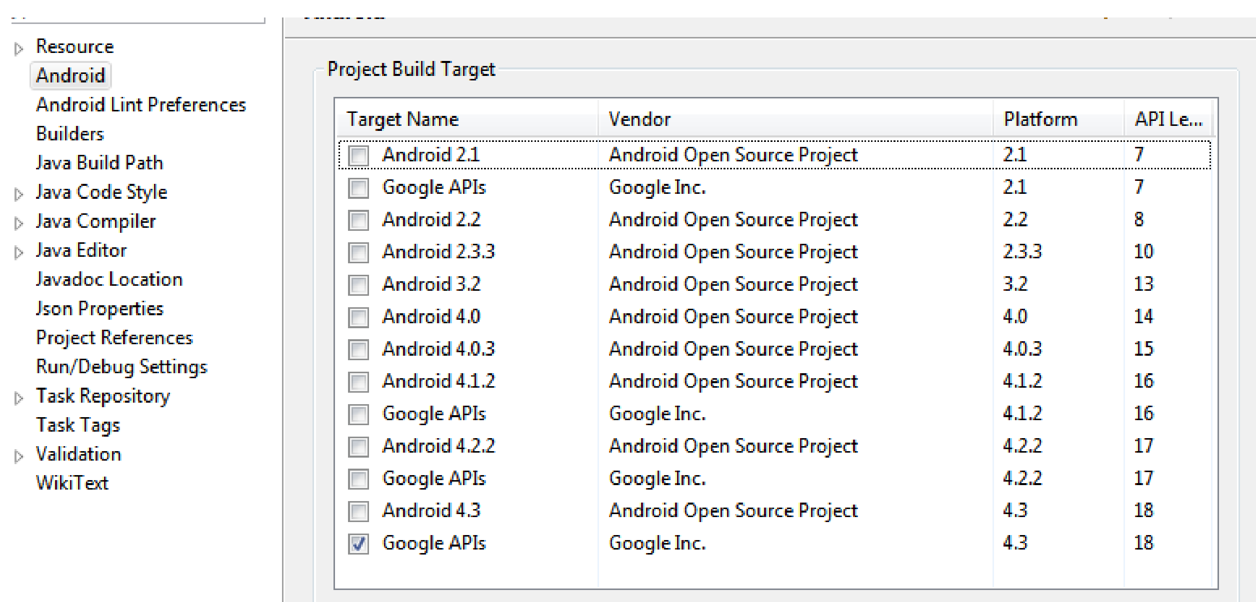
De esta forma podemos ver todo lo que nos va a permitir hacer el API Rest.

Services > mobilebackend API v1

- mobilebackend.blobEndpoint.deleteBlob
- mobilebackend.blobEndpoint.getDownloadUrl
- mobilebackend.blobEndpoint.getUploadUrl
- mobilebackend.endpointV1.delete
- mobilebackend.endpointV1.deleteAll
- mobilebackend.endpointV1.get
- mobilebackend.endpointV1.getAll
- mobilebackend.endpointV1.insert
- mobilebackend.endpointV1.insertAll
- mobilebackend.endpointV1.list
- mobilebackend.endpointV1.patch
- mobilebackend.endpointV1.update
- mobilebackend.endpointV1.updateAll

Librería cliente para Android.

En eclipse importamos el proyecto Android. Nos va a marcar el proyecto con error ya que hay que configurar el API del proyecto para que utilice las Google APIs mayor o igual a la 15.



Necesitaremos también tener importada en eclipse la librería "Google Play Services" ya que el proyecto cliente hace uso de la misma.

Vamos a la clase Const.java y cambiamos la constante **PROJECT_ID** por el ID de nuestro proyecto que se nos asignó cuando creamos el proyecto backend.

```
/**
 * Set Project ID of your Google APIs Console Project.
 */
public static final String PROJECT_ID = "******************";
```

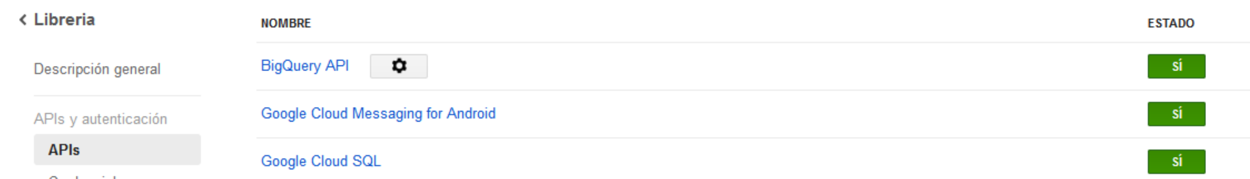
También necesitamos modificar la variable **WEB_CLIENT_ID** con el valor de un ID web.

Para comprobar que llegan los datos podemos ir a la consola de nuestra aplicación y en Datastore Viewer, seleccionamos "By kind" Guestbook y vemos todos los registros que se han guardado.

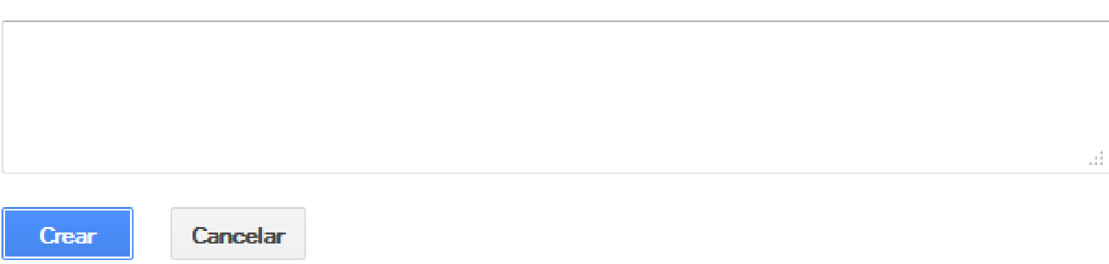
Activando los mensajes Push

Hasta ahora hemos conseguido persistir y recuperar nuestros datos en GAE. Pero también podemos hacer, que cualquier dispositivo que tenga la aplicación reciba una notificación cada vez que otro dispositivo genera un mensaje. Para ello vamos a usar Google Cloud Messaging(GCM).

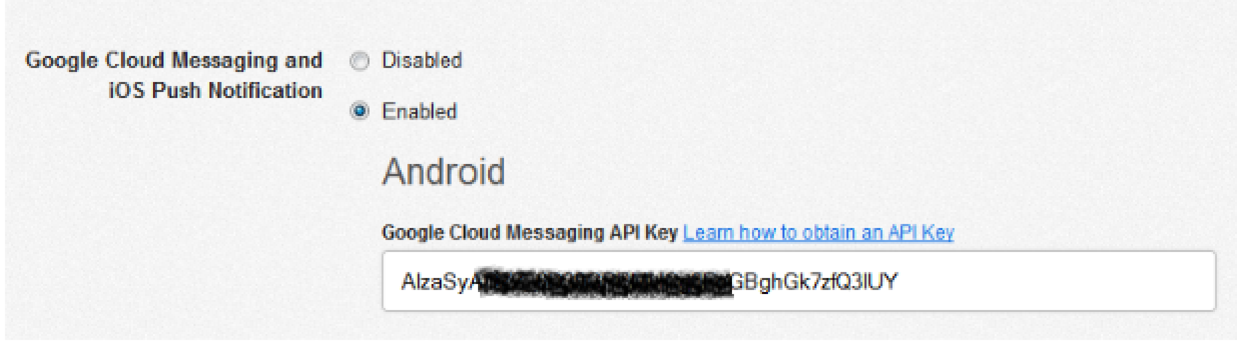
Vamos a la consola de GAE y en APIs buscamos “Google Cloud Messaging for Android” y la activamos.



Necesitamos crearnos una nueva clave de servidor. Para ello, entramos en “Credenciales”, “Acceso API Pública” “Crear nueva Clave”. Le indicamos que es una clave de servidor. No es necesario introducir ninguna IP simplemente crear.



Esta clave la tenemos que pegar en la Configuración de nuestra aplicación backend una vez que hemos activado el “Google Cloud Messaging and iOS Push Notification”.



En el proyecto Android, en la clase “Const.java” tenemos que sustituir la constante **PROJECT_NUMBER** por el número de proyecto de nuestra aplicación GAE.

Ahora ya podemos probarlo. Necesitamos dos dispositivos Android. Desde el primero de ellos mandamos el mensaje y el segundo debe recibirlo.

Estructura del cliente Android.

Cuando a través del plugin generemos la librería cliente se van a generar dos paquetes por cada entidad que tengamos en nuestro backend. Este paquete va a contener clases para poder hacer la llamada a los métodos de la API y para poder convertir los ficheros JSON en objetos planos.

En nuestro caso en el backend tendremos sólo una entidad “EntityDto.java” con información sobre los mensajes (la fecha de creación, quién les crea, el id, etc). La representación de las entidades están en el paquete mobilebackend.model. En el paquete mobilebackend tenemos acceso a todos los métodos de la API.