php avanzado

Implementación de API REST con PHP como back-end

José Antonio Carmona Fombella - G1 IS

Introducción a la Ingeniería del Software y los Sistemas de Información

# TABLA DE CONTENIDO

1. Objetivo, tecnologías y herramientas de desarrollo.
2. Contrato de la API REST.
3. Diseño e Implementación de la Base de Datos.
4. Conexión con la BD mediante objetos PDO
5. API REST con PHP
   1. Comprobación de la URI
   2. Distinción de métodos de la petición
   3. Procesamiento de los diferentes métodos
   4. Representación JSON del recurso
   5. Representación XML del recurso (?)
   6. Manejo de Excepciones (?)
6. Pruebas

TABLA DE CONTENIDO

[TABLA DE CONTENIDO 2](#_Toc486802108)

[OBJETIVO, TECNOLOGÍAS Y HERRAMIENTAS DE DESARROLLO 3](#_Toc486802109)

[ESCENARIO HIPOTÉTICO Y CONTEXTO 5](#_Toc486802110)

[DISEÑO E IMPLEMENTACIÓN DE LA BASE DE DATOS 5](#_Toc486802111)

[CONTRATO DE LA API REST 7](#_Toc486802112)

[Contrato para el recurso ‘Dispositivo’ 7](#_Toc486802113)

[Dispositivos: Ejemplo JSON con los datos a proporcionar en el cuerpo 9](#_Toc486802114)

[Dispositivos: Ejemplo JSON con los datos devueltos (GET) 9](#_Toc486802115)

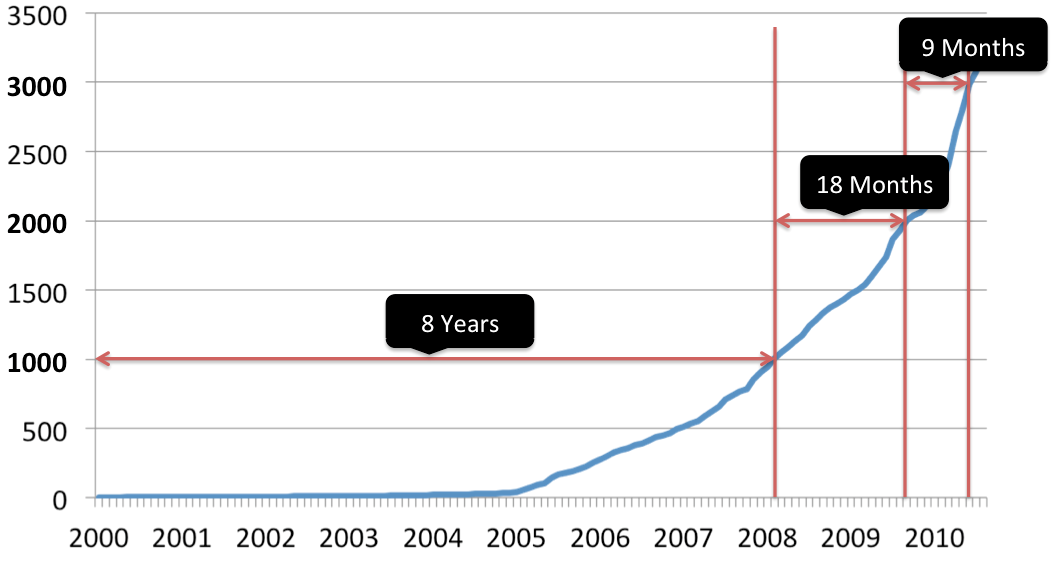
[Contrato para el recurso ‘Fabricantes’ 10](#_Toc486802116)

[Fabricantes: Ejemplo JSON con los datos a proporcionar en el cuerpo 11](#_Toc486802117)

[Fabricantes: Ejemplo JSON con los datos devueltos (GET) 11](#_Toc486802118)

# OBJETIVO, TECNOLOGÍAS Y HERRAMIENTAS DE DESARROLLO

Durante los últimos años, el uso de APIs se ha visto gratamente extendido en toda la web, sobre todo en el campo de los *web services.* Esto hace que el proceso de creación y desarrollo de las mismas sea una etapa fundamental a considerar en cualquier aplicación web.

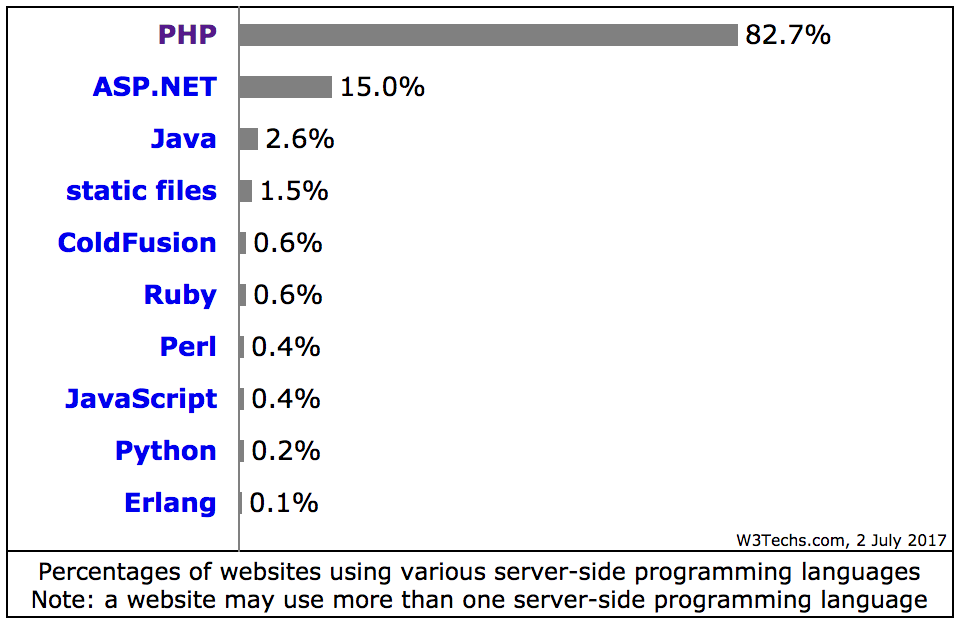


Evolución del número de APIs. En 2013 existían más de 8700 APIs. [1]

Existen varias tecnologías para la implementación de una web API:

* Servicios web RESTful
* Servicios web SOAP
* Javascript
* XML-RPC
* Etc

Nosotros nos centraremos en la primera, ya que es la más usada actualmente debido a su simplicidad y al uso de HTTP como protocolo de aplicación, ampliamente soportado en todas las plataformas.



Market-share de lenguajes de script de servidor [2]

La elección de PHP para implementar esta característica no es una casualidad. Como podemos ver en las estadísticas, de entre todas las opciones disponibles, este lenguaje sigue siendo el predominante para el procesamiento en el lado del servidor. En nuestro caso usaremos PHP v.

Un aspecto fundamental de una API es la autenticación, es decir, el uso de credenciales para restringir el acceso a servicios a ciertos usuarios. Para este cometido usaremos el estándar libre Open Authorization (OAuth) en su versión más reciente (2.0). Este protocolo permite la autorización segura de una API mediante el uso de unos códigos llamados tokens que identifican al usuario sin exponer todos sus datos.

Para implementar la capa de datos (donde guardaremos todos nuestros recursos de la API, así como todo lo relacionado con OAuth2.0) usaremos la base de datos proporcionada por Oracle. Es también una base de datos ampliamente usada, si bien podríamos usar cualquier otra pues su función únicamente será la consulta e inserción de datos.

Durante la fase de desarrollo de la API usaremos varias herramientas:

* XAMPP: Usaremos su servidor web Apache y su intérprete PHP para poder desarrollar en local, sin tener que desplegar cada vez que queramos probar una nueva funcionalidad.
* Oracle Database 11g: Base de datos de Oracle en su versión 11g.
* Oracle SQL Developer: IDE para manejar la BD de Oracle.
* Regex101: Aplicación web que permite la depuración y testeo de expresiones regulares.
* Sublime Text 2: Editor de código multiplataforma con soporte para múltiples lenguajes, entre ellos SQL, PHP, HTML y CSS.
* RESTClient: Plugin para el navegador web Firefox que nos permite depurar y testear de una forma sencilla servicios web RESTful.

# ESCENARIO HIPOTÉTICO Y CONTEXTO

Vamos a desarrollar una API orientada a los principales fabricantes de dispositivos inteligentes de la actualidad. Con ella, se espera que éstos puedan controlar los diferentes terminales que tienen a la venta y las diferentes versiones de los mismos.

Con carácter informativo, cualquier persona debería poder consultar los datos de los fabricantes que existen actualmente en la BD, así como obtener los detalles de un dispositivo en concreto o de todos si lo desea. Sin embargo, operaciones como la modificación, creación o eliminación (aquellas que producen un cambio en el estado de la BD) deben estar restringidas a usuarios registrados. Además, para evitar la manipulación de datos, sólo el propio fabricante debería poder realizar estas operaciones sobre sus datos, así como sobre los dispositivos que tenga asociados.

Con esto en cuenta, se nos pide desarrollar una API RESTful.

# DISEÑO E IMPLEMENTACIÓN DE LA BASE DE DATOS

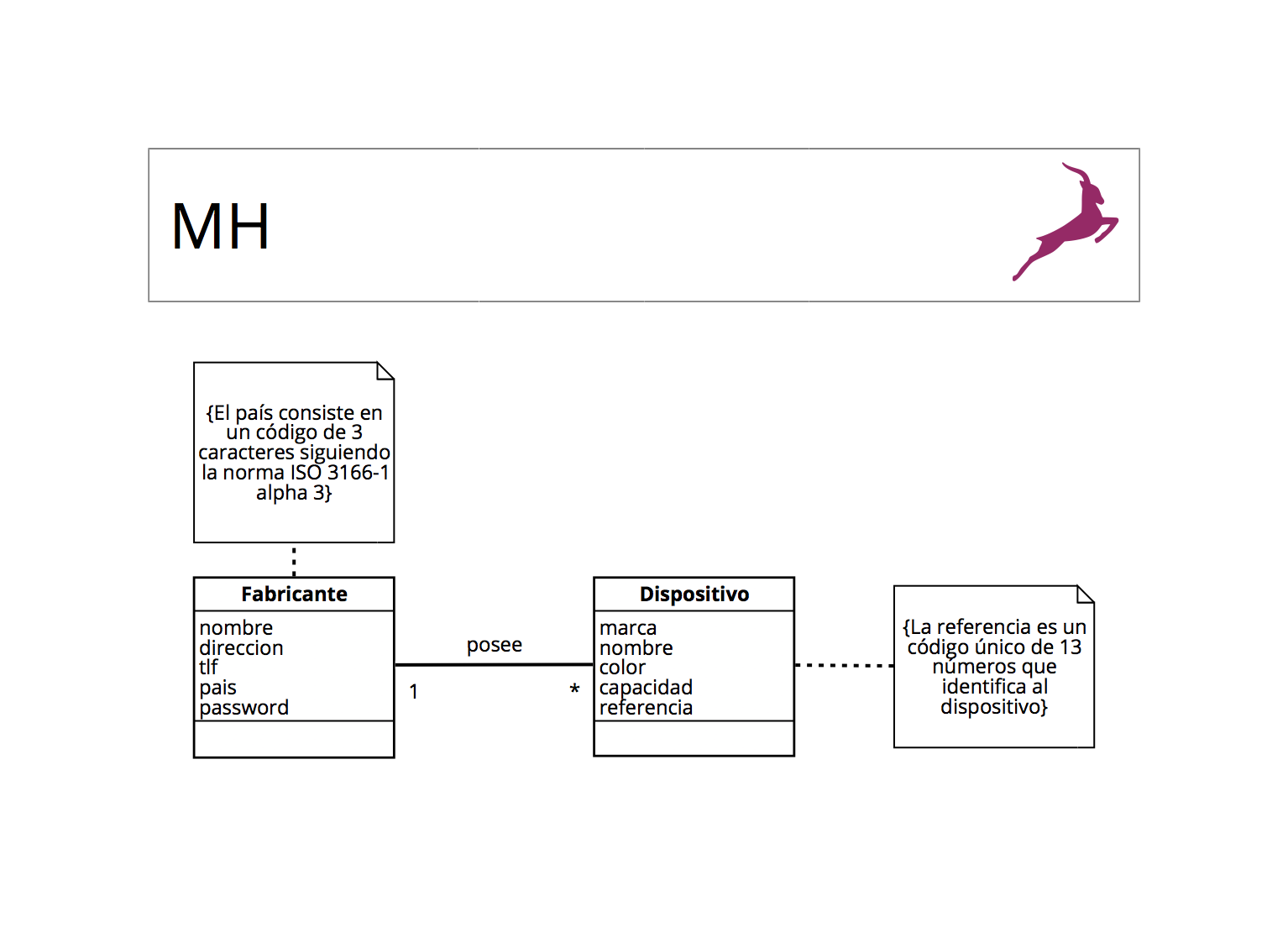
Vamos a suponer que se nos ha pedido (o hemos identificado la necesidad) conocer de cada *Fabricante* los siguientes datos:

* Nombre: El nombre del *Fabricante.* (no se puede repetir)
* Dirección: La dirección del *Fabricante.*
* Teléfono: El número de teléfono del *Fabricante*.
* País: El país donde el *Fabricante* tiene su sede directiva, en formato ISO 3166-1 alpha 3.
* Contraseña: La contraseña que el *Fabricante* desea usar para acceder a los servicios con restricción.

De igual manera, procedemos con los *Dispositivos:*

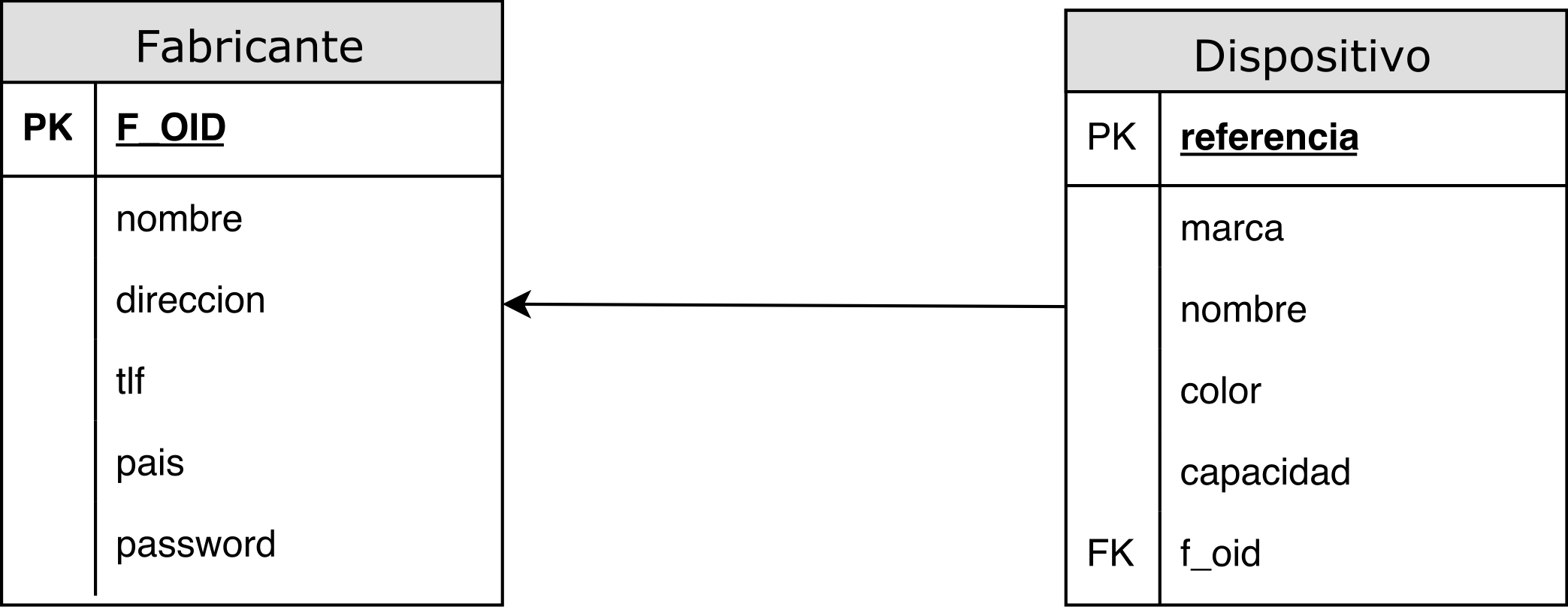
* Marca: El nombre de la marca comercial con la que el *Dispositivo* se lanza al mercado. No necesariamente tiene que ser el nombre del fabricante, pues este puede tener varias submarcas.
* Nombre: El nombre del *Dispositivo*.
* Color: El color de este *Dispositivo.*
* Capacidad: El tamaño de la memoria interna del *Dispositivo*
* Referencia: Código único de 13 números que identifica a un *Dispositivo*. Dos dispositivos con la misma marca y nombre pero con diferente color o capacidad tendrán una referencia diferente.

Teniendo en cuenta estos datos, realizamos un modelo conceptual con los recursos de nuestra API.

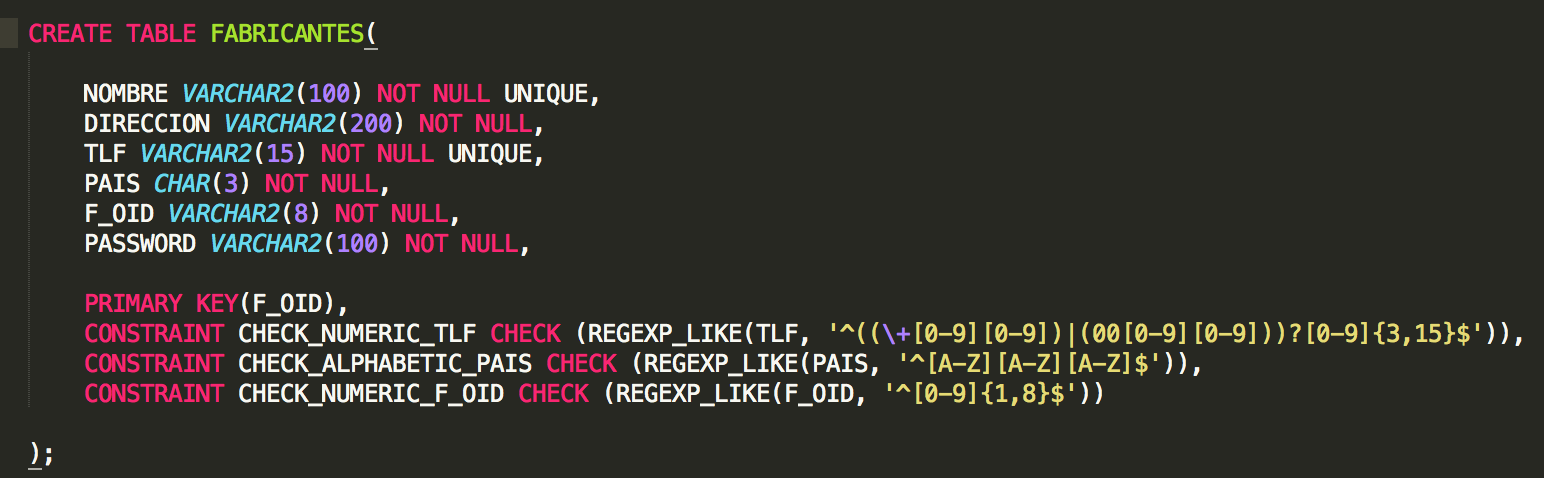


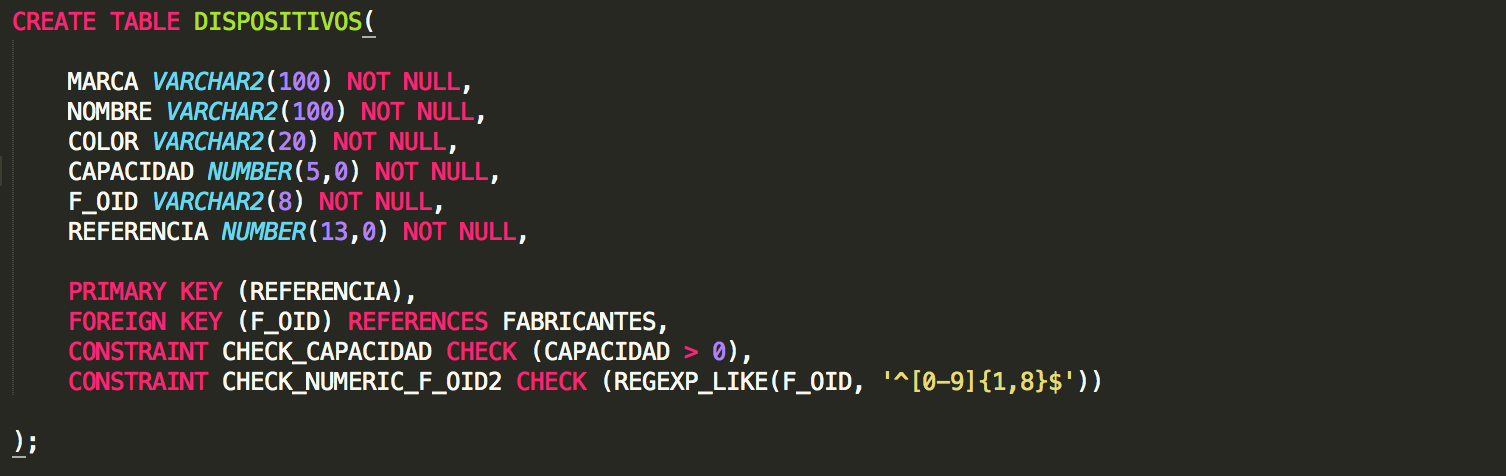
Modelo Conceptual propuesto para la solución al escenario hipotético.

Partiendo de este modelo conceptual, podemos transformarlo en un modelo relacional para su implementación en una BD:



Ya con este modelo en mente, es fácil convertir el problema en sentencias para construir tablas y relaciones SQL en la BD de Oracle:





# CONTRATO DE LA API REST

Un contrato de una API es un documento que la define, exponiendo para ello los recursos a los que podemos acceder, los parámetros, los verbos HTTP y cualquier otra información relevante.

En nuestra API se pretende trabajar con dos recursos: ‘Fabricantes’ y ‘Dispositivos’.

## Contrato para el recurso ‘Dispositivo’

Recurso: *Dispositivo*

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP** | **URI** | **DESCRIPCIÓN** | **PARÁMETROS** |
| GET | /dispositivos | Devuelve todos los *dispositivos* de la BD en formato JSON.   * **200 Successful Operation**. * **404 Not Found**, si no se encuentran dispositivos en la BD. | limit: número de resultados devueltos. Por defecto es 10.  offset: primer elemento devuelto. Por defecto es 1. |
| GET | /dispositivos/{referencia} | Devuelve el *dispositivo* con la referencia en formato JSON.   * **200 Successful Operation**, si el dispositivo se ha encontrado. * **404 Not Found**, si no hay ningún dispositivo con tal referencia. | Ninguno. |
| POST | /dispositivos | Añade el *dispositivo* que se pasa por el cuerpo en formato JSON.   * **201 Created**, si la operación se realiza con éxito. * **400 Bad Request**, si alguno de los campos no es válido. * **401 Unauthorized**, si el token no es válido. | *token:* código temporal que autoriza la operación.  JSON en el cuerpo con los datos del dispositivo. |
| PUT | /dispositivos/{referencia} | Actualiza el *dispositivo* que tiene la *referencia* usando los valores que se envían en el cuerpo en formato JSON.   * **204 No Content**, si se actualiza correctamente. * **400 Bad Request**, si alguno de los campos no es válido. * **401 Unauthorized**, si el token no es válido. * **403 Forbidden**, si el token de usuario no corresponde con el fabricante del dispositivo a actualizar. | *token:* código temporal que autoriza la operación.  JSON en el cuerpo con los datos del dispositivo a actualizar. |
| DELETE | /dispositivos/{referencia} | Borra un *dispositivo* dado su *referencia.*   * **204 No Content**, si se realiza correctamente. * **401 Unauthorized**, si el token no es válido * **403 Forbidden**, si el token de usuario no corresponde con el fabricante del dispositivo a eliminar. | *token:* código temporal que autoriza la operación. |

## Dispositivos: Ejemplo JSON con los datos a proporcionar en el cuerpo

POST y PUT:

1. {
2. "REFERENCIA": "1000000000000",
3. "MARCA": "Apple",
4. "NOMBRE": "iPhone 7 Plus",
5. "COLOR": "Jet Black",
6. "CAPACIDAD": "128",
7. }

## Dispositivos: Ejemplo JSON con los datos devueltos (GET)

Resultado de la consulta a la URI: <http://localhost/PHP_API/dispositivos?limit=2>

1. [
2. {
3. "Total Recursos": 11,
4. "Resultados Consulta": 2
5. },
6. [
7. {
8. "RNUM": "1",
9. "MARCA": "Apple",
10. "NOMBRE": "iPhone 7 Plus",
11. "COLOR": "Jet Black",
12. "CAPACIDAD": "128",
13. "F\_OID": "1",
14. "REFERENCIA": "1000000000000"
15. },
16. {
17. "RNUM": "2",
18. "MARCA": "Apple",
19. "NOMBRE": "iPad Air 2",
20. "COLOR": "Space Gray",
21. "CAPACIDAD": "64",
22. "F\_OID": "1",
23. "REFERENCIA": "1000000000001"
24. }
25. ]
26. ]

## 

## Contrato para el recurso ‘Fabricantes’

Recurso: *Fabricantes*

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP** | **URI** | **DESCRIPCIÓN Y CÓDIGOS** | **PARÁMETROS** |
| GET | /fabricantes | Devuelve todos los *fabricantes* de la BD en formato JSON.   * **200 Successful Operation**. * **404 Not Found**, si no se encuentran fabricantes en la BD. | limit: número de resultados devueltos. Por defecto es 10.  offset: primer elemento devuelto. Por defecto es 1. |
| GET | /fabricantes/{f\_oid} | Devuelve el *fabricante* con la *f\_oid* en formato JSON.   * **200 Successful Operation**, si el fabricante se ha encontrado. * **404 Not Found**, si no hay ningún fabricante con tal *f\_oid*. | Ninguno. |
| PUT | /fabricantes | Actualiza el *fabricante* del token actual, usando los valores que se envían en el cuerpo en formato JSON.   * **204 No Content,** si se actualiza correctamente. * **400 Bad Request**, si alguno de los campos no es válido. * **401 Unauthorized,** si el token no es válido. | *token:* código temporal que autoriza la operación.  JSON en el cuerpo con los datos del dispositivo a actualizar. |

## Fabricantes: Ejemplo JSON con los datos a proporcionar en el cuerpo

PUT:

1. {
2. "NOMBRE": "Apple Inc.",
3. "DIRECCION": "1 Infinite Loop, California",
4. "PAIS": "USA",
5. "TLF": "+1987654320"
6. }

## Fabricantes: Ejemplo JSON con los datos devueltos (GET)

Resultado de la consulta a la URI: <http://localhost/PHP_API/fabricantes?limit=2>

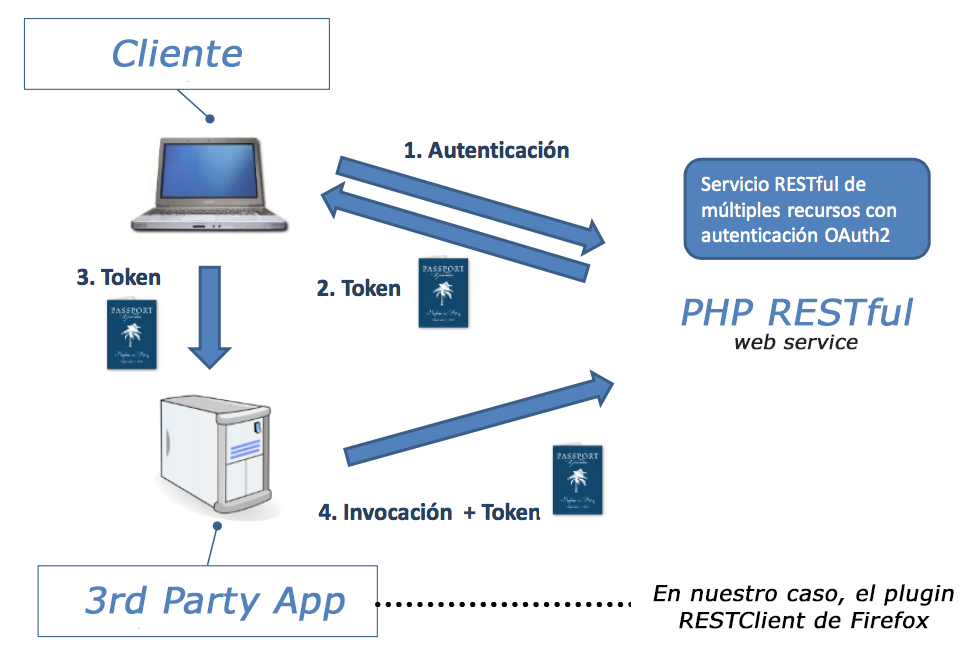
1. [
2. {
3. "Total Recursos": 5,
4. "Resultados Consulta": 2
5. },
6. [
7. {
8. "RNUM": "1",
9. "NOMBRE": "Apple Inc.",
10. "F\_OID": "1",
11. "DIRECCION": "1 Infinite Loop, California",
12. "PAIS": "USA",
13. "TLF": "+1987654320"
14. },
15. {
16. "RNUM": "2",
17. "NOMBRE": "BQ",
18. "F\_OID": "2",
19. "DIRECCION": "Las Rozas, Madrid",
20. "PAIS": "ESP",
21. "TLF": "+34911829384"
22. }
23. ]
24. ]

## 

# OAUTH 2.0

Es un estándar abierto que permite la autorización segura de una API mediante el uso de unos códigos llamados tokens, que identifican al usuario sin exponer todos sus datos. Es ampliamente usado en la web para servicios que requieran el uso de credenciales y el acceso a terceros.

En la siguiente imagen encontraremos un flujo normal de autenticación con OAuth2.0, donde tendremos a nuestra API PHP como proveedor del servicio:



Como vemos, el cliente procede a autenticarse directamente con el proveedor de servicios, no compartiendo ninguna credencial con la aplicación de terceros. Cuando el cliente se identifica, se le proporciona un token temporal que éste suministra a la aplicación que quiere acceder a sus datos, la cual invoca al servicio usando el código proporcionado. Normalmente esta aplicación se ejecuta en otro terminal, aunque en nuestro caso sea en el mismo.

De esta forma, garantizamos que sólo el *Fabricante* en cuestión puede modificar sus datos y los de sus *Dispositivos* asociados, denegando las operaciones a cualquier otro usuario que intente realizar cambios.

Para implementar esta funcionalidad me he basado en el trabajo de Bshaffer [3], el cual no provee con una librería en PHP para abordar este problema.

## Adaptación de la librería al contexto y Oracle Database

La librería proporcionada viene preparada para el su uso en sistemas con BD como MySQL, SQLite, PostgreSQL ó MS SQL Server. Para poder usarla con la BD de Oracle y adaptarla a nuestro problema hay que realizar unos pequeños cambios que veremos a continuación.

En primer lugar debemos crear 7 tablas que contendrán información específica del protocolo OAuth2.0 en nuestra BD, como por ejemplo los códigos tokens, los clientes, los usuarios, los scopes o permisos, …

En OAuth 2.0, denominamos cliente a la aplicación que pide la emisión de un token. Un usuario por el contrario, es la persona o entidad que introduce sus credenciales para tener acceso a los datos.

En la guía de la librería se nos proporciona con el siguiente código para la creación de las tablas:

**CREATE** **TABLE** oauth\_clients (client\_id **VARCHAR**(80) **NOT** **NULL**, client\_secret **VARCHAR**(80), redirect\_uri **VARCHAR**(2000) **NOT** **NULL**, grant\_types **VARCHAR**(80), scope **VARCHAR**(100), user\_id **VARCHAR**(80), **CONSTRAINT** clients\_client\_id\_pk **PRIMARY** **KEY** (client\_id));

**CREATE** **TABLE** oauth\_access\_tokens (access\_token **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** access\_token\_pk **PRIMARY** **KEY** (access\_token));

**CREATE** **TABLE** oauth\_authorization\_codes (authorization\_code **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), redirect\_uri **VARCHAR**(2000), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** auth\_code\_pk **PRIMARY** **KEY** (authorization\_code));

**CREATE** **TABLE** oauth\_refresh\_tokens (refresh\_token **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** refresh\_token\_pk **PRIMARY** **KEY** (refresh\_token));

**CREATE** **TABLE** oauth\_users (username **VARCHAR**(255) **NOT** **NULL**, password **VARCHAR**(2000), first\_name **VARCHAR**(255), last\_name **VARCHAR**(255), **CONSTRAINT** username\_pk **PRIMARY** **KEY** (username));

**CREATE** **TABLE** oauth\_scopes (scope **TEXT**, is\_default **BOOLEAN**);

**CREATE** **TABLE** oauth\_jwt (client\_id **VARCHAR**(80) **NOT** **NULL**, subject **VARCHAR**(80), public\_key **VARCHAR**(2000), **CONSTRAINT** jwt\_client\_id\_pk **PRIMARY** **KEY** (client\_id));

Sin embargo, debemos tener en cuenta que los tipos de dato **TEXT** y **BOOLEAN** no están disponibles en Oracle SQL, por lo que debemos cambiarlos a otros que cumplan su función de forma equivalente.

En nuestro caso, optaremos por **VARCHAR(2000)** y **NUMBER(1)** para reemplazarlos, ya que de todas formas no es algo demasiado relevante al no usar scopes.

**CREATE** **TABLE** oauth\_clients (client\_id **VARCHAR**(80) **NOT** **NULL**, client\_secret **VARCHAR**(80), redirect\_uri **VARCHAR**(2000) **NOT** **NULL**, grant\_types **VARCHAR**(80), scope **VARCHAR**(100), user\_id **VARCHAR**(80), **CONSTRAINT** clients\_client\_id\_pk **PRIMARY** **KEY** (client\_id));

**CREATE** **TABLE** oauth\_access\_tokens (access\_token **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** access\_token\_pk **PRIMARY** **KEY** (access\_token));

**CREATE** **TABLE** oauth\_authorization\_codes (authorization\_code **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), redirect\_uri **VARCHAR**(2000), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** auth\_code\_pk **PRIMARY** **KEY** (authorization\_code));

**CREATE** **TABLE** oauth\_refresh\_tokens (refresh\_token **VARCHAR**(40) **NOT** **NULL**, client\_id **VARCHAR**(80) **NOT** **NULL**, user\_id **VARCHAR**(255), expires **TIMESTAMP** **NOT** **NULL**, scope **VARCHAR**(2000), **CONSTRAINT** refresh\_token\_pk **PRIMARY** **KEY** (refresh\_token));

**CREATE** **TABLE** oauth\_users (username **VARCHAR**(255) **NOT** **NULL**, password **VARCHAR**(2000), first\_name **VARCHAR**(255), last\_name **VARCHAR**(255), **CONSTRAINT** username\_pk **PRIMARY** **KEY** (username));

**CREATE** **TABLE** oauth\_scopes (scope **TEXT**, is\_default **BOOLEAN**);

**CREATE** **TABLE** oauth\_jwt (client\_id **VARCHAR**(80) **NOT** **NULL**, subject **VARCHAR**(80), public\_key **VARCHAR**(2000), **CONSTRAINT** jwt\_client\_id\_pk **PRIMARY** **KEY** (client\_id));

[1] <http://www.slideshare.net/jmusser/open-apis-state-of-the-market-2011> por John Musser. ProgrammableWeb.

[2] <https://w3techs.com/technologies/overview/programming_language/all>

[3] <http://bshaffer.github.io/oauth2-server-php-docs/>