



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica  
Universitat Politècnica de València

# Desarrollo de un ToDo-List/Pending Task integrado en una vivienda inteligente.

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor:** Diego Garín Menéndez

**Tutor:** Joan Josep Fons Cors

Septiembre 2014



# Agradecimientos

---

En primer lugar, quiero mostrar mi agradecimiento a mi tutor del trabajo, Joan Josep Fons Cors, por su inestimable ayuda y por su gran labor como docente.

En segundo lugar quiero agradecer a mi familia, en especial a mis padres su apoyo incondicional durante todos estos años y me gustaría disculparme por los disgustos académicos que les he dado más de una vez.

En tercer lugar quiero expresar mi más sincero agradecimiento a mi compañero Guille, por la paciencia que ha tenido conmigo desde que le conozco y por su amistad, que no tiene precio.

Por último quiero agradecer a la persona que mejor me comprende y siempre ha estado a mi lado, ayudándome en los momentos difíciles y haciendo que crea que puedo con todo. Sara, sin ti nada de esto habría sido posible, un pedacito de esto es tuyo.

# Resumen

---

Esta memoria documenta el desarrollo de una API REST, desarrollada en Java, integrada con una vivienda domótica. Esta aplicación utilizará los servicios que proporciona Google Task y servirá para poder programar las tareas pendientes de los usuarios, y enlazar dichas tareas con acciones que puedan ocurrir en la casa domótica.

Las acciones que ocurren en la vivienda inteligente se obtienen del simulador de una vivienda real, proporcionada por el tutor del proyecto Joan Josep Fons Cors.

Además se desarrolla de forma paralela una interfaz web que de funcionalidad a la aplicación y permita la creación de tareas. Esta interfaz hace posible el acceso a la aplicación desde cualquier lugar.

**Palabras clave:** REST, domótica, Google Task, Java, API.

# Abstract

---

The current report develops an API REST integrated on a home automation and programmed in Java. Google Task services are used in this application in order to organize the remaining tasks of the users and link these tasks with the home automation actions.

The mentioned actions are taken from the real home automation, provided by the project tutor Joan Josep Fons Cors.

In addition, it is developed a web interface which supplies functionality to the application and also, allow creating new tasks for the home automation. This interface gives the chance to use and access the application from anywhere, which provides a higher grade of autonomy.

**Keywords:** REST, home automation, Google Task, Java, API.

# Acrónimos

---

API	Application Programm Interface
COBOL	COmmon Business-Oriented Language
GNU	GNU's Not Unix, Linux OS
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JDT	Java Development Tools
JSON	Javascript Object Notation
PHP	Hypertext Preprocessor
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
XAMPP	X Apache MySQL PHP Perl
XML	eXtensible Markup Language
XSRF	Cross Site Request Forgery



## Contenido

1. Introducción .....	9
1.1.  Ámbito .....	9
1.2.  Objetivo .....	10
2. Análisis .....	11
2.1.  Especificaciones y requisitos .....	11
2.2.  Tecnologías utilizadas .....	13
2.2.1.  REST.....	13
2.2.2.  Google Task .....	14
2.2.3.  OAuth 2.0 .....	15
2.2.4.  CakePHP.....	17
2.2.5.  Smarthome.....	17
2.3.  Herramientas software.....	18
2.3.1.  Eclipse.....	18
2.3.2.  Xampp.....	18
2.3.3.  Navegadores .....	19
3. Diseño .....	20
3.1.  Diseño de la API REST .....	21
3.1.1.  Uso de JSON.....	25
3.1.2.  CrossDomain.....	25
3.2.  Diseño del portal web .....	26
3.3.  Diseño de la base de datos .....	31
3.4.  Integración con la Smarthome .....	34
4. Desarrollo .....	36
4.1.  App.....	37
4.1.1.  Principal.java .....	37
4.1.2.  Aplicación.java.....	38
4.2.  Cliente .....	39
4.2.1.  Login .....	39
4.2.2.  Oauth.java .....	40
4.2.3.  Callback.java .....	41
4.2.4.  Listas.java .....	42
4.2.5.  Lista.java .....	44
4.2.6.  Tareas.java.....	46
4.2.7.  Tarea.java .....	49
4.2.8.  Eventos.java.....	50

4.2.9. ForzadoEventos.java .....	51
4.3. Desarrollo del portal Web. ....	52
5. Mejoras aplicables .....	54
6. Resultados.....	55
7. Conclusiones .....	58
8. Bibliografía.....	59

## Tabla de contenidos

---

## Lista de ilustraciones

---

<i>Ilustración 1: Diagrama de la API REST</i>	12
<i>Ilustración 2: Diagrama de autenticación mediante OAuth</i>	16
<i>Ilustración 3: Estructura de la aplicación</i>	20
<i>Ilustración 4: Listas – Vista principal</i>	26
<i>Ilustración 5: Listas – Nueva Lista</i>	27
<i>Ilustración 6: Listas – Contenido de la Lista</i>	27
<i>Ilustración 7: Tareas – Vista Principal</i>	28
<i>Ilustración 8: Tareas – Nueva Tarea</i>	28
<i>Ilustración 9: Tareas – Contenido de la Tarea</i>	29
<i>Ilustración 10: Usuarios- Vista Principal</i>	29
<i>Ilustración 11: Eventos – Vista Principal</i>	30
<i>Ilustración 12: Diagrama de la Base de Datos</i>	33
<i>Ilustración 13: Contenido de la API REST</i>	36
<i>Ilustración 14: Interfaz web - Listas</i>	52
<i>Ilustración 16: Interfaz web - Eventos</i>	53
<i>Ilustración 15: Interfaz web - Tareas</i>	53
<i>Ilustración 17 : Nueva Tarea</i>	55
<i>Ilustración 19: Nueva Tarea en la Base de Datos</i>	56
<i>Ilustración 20: Nuevo Evento en la Base de Datos</i>	56
<i>Ilustración 18: Nueva Tarea en Google</i>	56
<i>Ilustración 21: Nueva tarea en la Lista</i>	57

## Lista de tablas

---

<i>Tabla 1: API de Google Tasks - Funciones de Listas</i>	21
<i>Tabla 2: API de Google Tasks - Atributos de Listas</i>	22
<i>Tabla 3: API de Google Tasks - Funciones de Tareas</i>	22
<i>Tabla 4: API de Google Tasks - Atributos de Tareas</i>	23
<i>Tabla 5: Dispositivos de Smarthome utilizados</i>	34
<i>Tabla 6: Acciones de Smarthome utilizados</i>	35



# 1. Introducción

---

En este momento la tecnología avanza a pasos agigantados y surgen cada vez hay más recursos puestos a nuestro alcance que nos simplifican las tareas cotidianas. Este es el caso de la domótica o vivienda inteligente. La domótica se podría definir como la integración de la tecnología en el diseño inteligente de un recinto cerrado (Wikipedia - Domótica, 2014).

La domótica doméstica cada día está más presente en nuestras vidas haciéndonos la vida mucho más sencilla en nuestros hogares. Estas viviendas cuentan con avanzados sistemas que aportan servicios tales como la gestión energética, gestión de seguridad y sistemas para mejorar nuestro bienestar entre muchas otras cosas. Todas esas ventajas hacen que cada día aumente el número de viviendas de este tipo.

Por otro lado el tiempo es un bien muy preciado en la sociedad en la que vivimos actualmente, siempre con prisas mirando el reloj y organizando toda nuestra jornada intentando aprovechar al máximo nuestro horario. Pero a pesar de todo siempre hay cosas que olvidamos, por este motivo existen cientos de aplicaciones capaces de programar las tareas que debemos realizar que nos ayudan a poder llegar a todas partes con el tiempo que tenemos disponible.

Este proyecto desarrolla la unión de una casa domótica y un gestor de tareas, haciendo así más funcional cualquier vivienda inteligente.

## 1.1. Ámbito

Antes de empezar a desarrollar todo el proyecto, hay ciertos aspectos que conviene definir sobre la domótica.

El término domótica viene de la unión de las palabras *domus* (que significa casa en latín) y *tica* (de automática, palabra en griego, 'que funciona por sí sola'), consiste en la automatización de los procesos que suceden dentro de un hogar y que pueden ser controlados desde dentro como desde fuera de casa. (Wikipedia - Domótica, 2014)

Los servicios que ofrece la domótica se pueden agrupar en estos aspectos:

-Programación y ahorro energético.

El ahorro energético no es algo tangible, sino un concepto al que se puede llegar de muchas maneras. En muchos casos no es necesario sustituir los aparatos o sistemas del hogar por otros que consuman menos sino una gestión eficiente de los mismos.

El control de toldos y persianas, la climatización inteligente (dependiente de los factores climatológicos externos a la vivienda) o la gestión eléctrica son aspectos fundamentales.

**-Confort.**

El confort conlleva todas las actuaciones que se puedan llevar a cabo que mejoren el confort en una vivienda. En este grupo se pueden incluir la gestión de los elementos de ocio en el hogar o el control de la iluminación de manera automatizada entre otros aspectos.

**-Seguridad.**

Consiste en una red de seguridad encargada de proteger tanto los bienes patrimoniales, como la seguridad personal y la vida. Como alarmas de intrusión, detectores de humo, alertas médicas para personas dependiente, etc.

**-Comunicaciones.**

Son los sistemas o infraestructuras de comunicaciones que posee el hogar. Se podría incluir en este grupo la capacidad de controlar la vivienda desde cualquier lugar con conexión a internet.

**-Accesibilidad.**

Se incluyen las aplicaciones o instalaciones de control remoto del entorno que favorecen la autonomía personal de personas con limitaciones funcionales, o discapacidad.

## **1.2. Objetivo**

El objetivo principal del proyecto es la creación de una API de tipo REST para poder controlar tareas de la vida cotidiana. Este proyecto será integrado con los servicios que proporciona una vivienda inteligente, y se podrán programar tareas relacionadas con los dispositivos que la forman. De este modo aumentamos la funcionalidad de dicha vivienda.

El objetivo secundario, es el desarrollo de un portal web que será la interfaz que conectará al usuario con la API y le ayudará a tener un mayor control sobre las tareas pendientes que tiene que realizar. El portal será diseñado para que resulte muy sencillo e intuitivo su uso, y que no requiera una formación previa para ser utilizado.

Por otro lado este desarrollo puede ser un punto de partida para próximas ampliaciones y mejoras que doten de más funcionalidad a la casa domótica.

## 2. Análisis

---

Antes de empezar a analizar que funcionalidad va a tener la aplicación, se desarrolló un estudio las herramientas que existen en la actualidad en lo referente a la organización de las tareas.

Este estudio se llevó a cabo principalmente para decidir que herramienta ofrecía mas funcionalidades, cual tenía manuales más completos y cual tenía un mayor uso, entre otros factores.

Tras una primera fase de evaluación, había 3 herramientas que se acoplaban a los requisitos básicos que se buscaban, Google Task<sup>1</sup>, Evernote<sup>2</sup> e iCal<sup>3</sup> de Apple. Finalmente el elegido fue Google Task, ya que tiene una API muy bien estructurada, hay infinidad de información en la web sobre la herramienta y, sobretodo no es tan restrictiva como las demás.

### 2.1. Especificaciones y requisitos

La aplicación que se desarrolla tiene que permitir a un usuario controlar las tareas que haya introducido, y darle la posibilidad de que esas tareas se asocien a eventos que ocurren en la vivienda inteligente. Así por ejemplo, si un usuario tiene una tarea que le recuerde llevar unos documentos a su trabajo y asocia dicha actividad al evento “Abrir puerta principal”, cuando el usuario vaya a salir de casa y se active el sensor de apertura de la puerta, recibirá un aviso recordándole que tiene que llevar los documentos.

Por otro lado, el usuario debe ser capaz de realizar múltiples funciones sobre las tareas que ya haya creado o sobre tareas nuevas.

Las tareas básicas que un usuario debe realizar son:

- Crear listas de tareas: El usuario debe de poder crear tantas listas de tareas como desee, pudiendo tener varias listas activas al mismo tiempo.
- Compartir tareas con otros usuarios autorizados: Varios usuarios pueden compartir una tarea siempre y cuando el usuario creador los haya autorizado.
- Eliminar tareas incompletas: Un usuario puede eliminar una tarea que no haya sido completada.
- Que otro usuario complete tareas que hayan sido compartidas. Si un usuario ha sido autorizado a acceder a mis tareas, también debe de poder completarlas.
- El usuario debe de poder acceder a sus tareas desde cualquier lugar con conexión a internet.

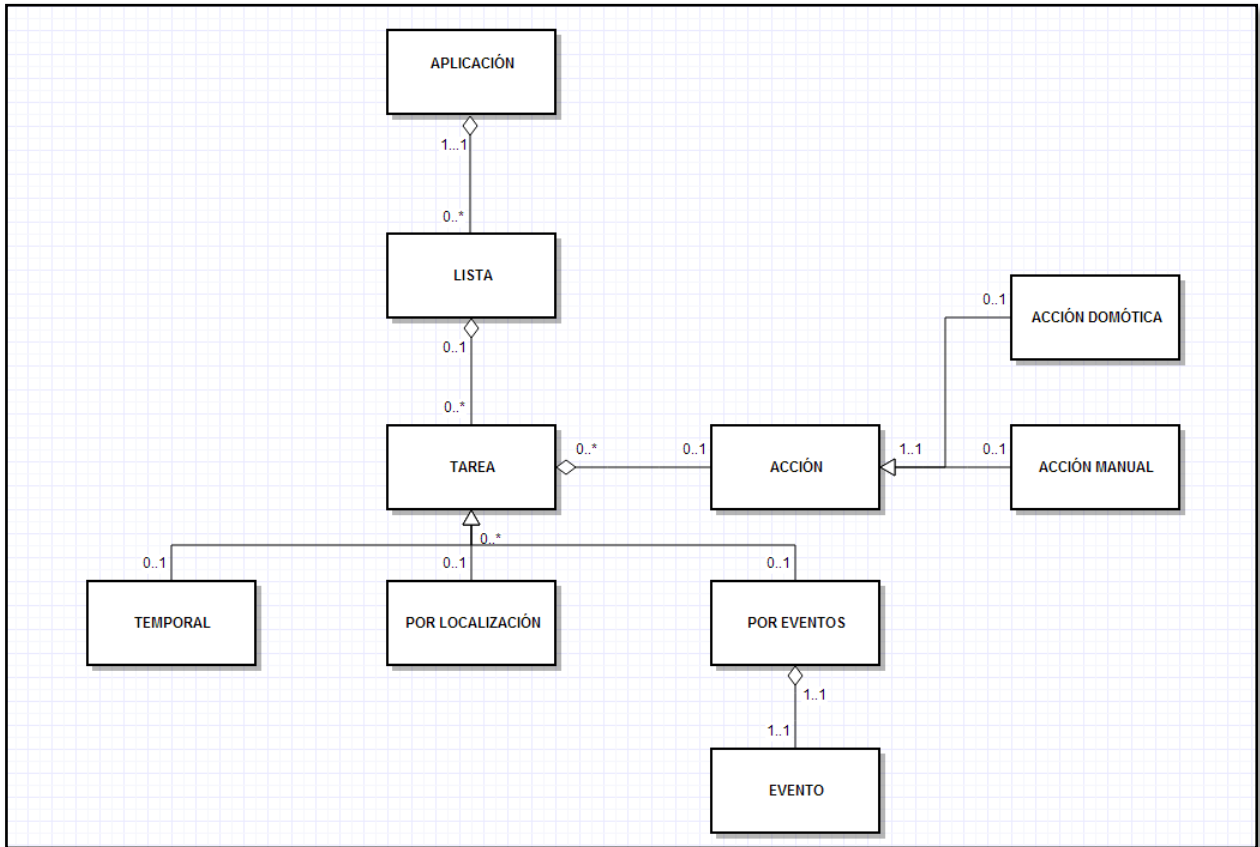
---

<sup>1</sup> <https://www.gmail.com/mail/help/tasks/>

<sup>2</sup> <https://evernote.com/intl/es/>

<sup>3</sup> <http://www.apple.com/es/support/lion/applications/>

A continuación se incluye un diagrama de clases básico sobre la aplicación. En este punto no se entra en detalles, ya que eso se realiza en la parte de *Diseño*, donde se explicará lo que contiene la aplicación de forma pormenorizada.



**Ilustración 1: Diagrama de la API REST**

Del diagrama hay que señalar que a través de la aplicación se podrán crear listas de tareas, estas listas podrán contener un número indefinido de tareas y esas tareas solo podrán ser contenidas por una lista.

Cada tarea puede contener diferentes propiedades, ser activada por eventos de la casa domótica, por encontrarse en una determinada localización o por tener una fecha límite para ser realizada. Las tareas llevan asociadas acciones que pueden ocurrir en la vivienda inteligente o por acciones realizadas manualmente.

## 2.2. Tecnologías utilizadas

En este apartado se va a mencionar que herramientas software y que tecnologías han sido utilizadas durante el desarrollo del proyecto.

### 2.2.1. REST

REST (REpresentational State Transfer) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es más simple y convencional que otras alternativas como SOAP y XML-RPC.

Podríamos considerar REST como un framework para construir aplicaciones web respetando HTTP, lo que lo convierte en el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Existen varias reglas que se deben de cumplir para que la aplicación sea RESTful, que use correctamente REST. Las reglas son las siguientes:

- Uso correcto de las URLs:
  - Las URLs no deben de implicar acciones y deben ser únicas.
  - Las URLs deben ser independientes del formato.
  - Las URLs deben mantener una jerarquía lógica.
- Uso correcto de HTTP.

HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Estas operaciones deben de utilizarse en la API para que siga las reglas de REST.

- Implementar Hypermedia.

Conectar mediante vínculos las aplicaciones clientes con las APIs, permitiendo a dichos clientes despreocuparse por conocer de antemano del cómo acceder a los recursos.



- No almacenar el estado en el servidor.

Toda la información que se requiere para mostrar lo que se solicita debe estar en la consulta por parte del cliente. En el servidor no se debe almacenar ninguna información sobre la consulta realizada. (Marqués, 2013)

## 2.2.2. *Google Task*

Google Task es una aplicación desarrollada por Google, que permite la creación de listas de tareas pendientes. Un aspecto muy positivo es que esta aplicación está integrada con el servicio de correo de Google, lo que hace posible que el acceso a la aplicación se pueda realizar sin necesidad de instalar nada en el equipo.

La API Google Task<sup>4</sup> proporciona a los desarrolladores un potente conjunto de criterios de valoración de la API para la búsqueda, lectura y actualización de contenidos Google Tasks. En esta API se describe cómo usar REST y las funciones que componen la API para acceder y editar los datos de Google Tasks. Está disponible para varios lenguajes de programación (actualmente Java, Python y PHP) y tiene una gran cantidad de tutoriales para familiarizarse con su uso.

Esta API es gratuita, y nos permite un máximo de 5.000 peticiones diarias, pero si por algún motivo superamos esas peticiones, se puede pedir un número mayor de peticiones desde el panel de control de la API.

Google Task proporcionar varios métodos sobre tareas y listas de tareas que nos permiten insertar nuevas tareas o listas, editarlas, eliminarlas, actualizar el estado si han sido completadas, o generar un listado con lo que hay en nuestra cuenta.

Estos métodos se explicarán más adelante en el apartado de *Desarrollo* donde podremos especificar mejor que lleva a cabo cada función de la API.

---

<sup>4</sup> <https://developers.google.com/google-apps/tasks/>

## 2.2.3. OAuth 2.0

OAuth 2<sup>5</sup> es un protocolo abierto de autorización, que posibilita a aplicaciones de terceros solicitar acceso limitado a los recursos de un servicio HTTP sin tener que compartir las contraseñas. En otras palabras, OAuth 2 define como un usuario puede compartir información con un consumidor de ese servicio sin que se efectúe un intercambio de contraseñas entre ellos.

Para ofrecer estas funcionalidades, OAuth 2.0 se basa en la generación de códigos de acceso temporales: código de acceso (*Access Token*), código de autorización (*Authorization Token*) y código de refresco (*Refresh token*). Estos códigos permiten a las aplicaciones de terceros acceder a la información de un usuario de forma y tiempo limitado simplemente incluyendo código en la petición. (INTECO- Instituto Nacional de Tecnologías de Comunicación, 2014)

En el siguiente diagrama podemos ver su funcionamiento.

---

<sup>5</sup> <http://oauth.net/>

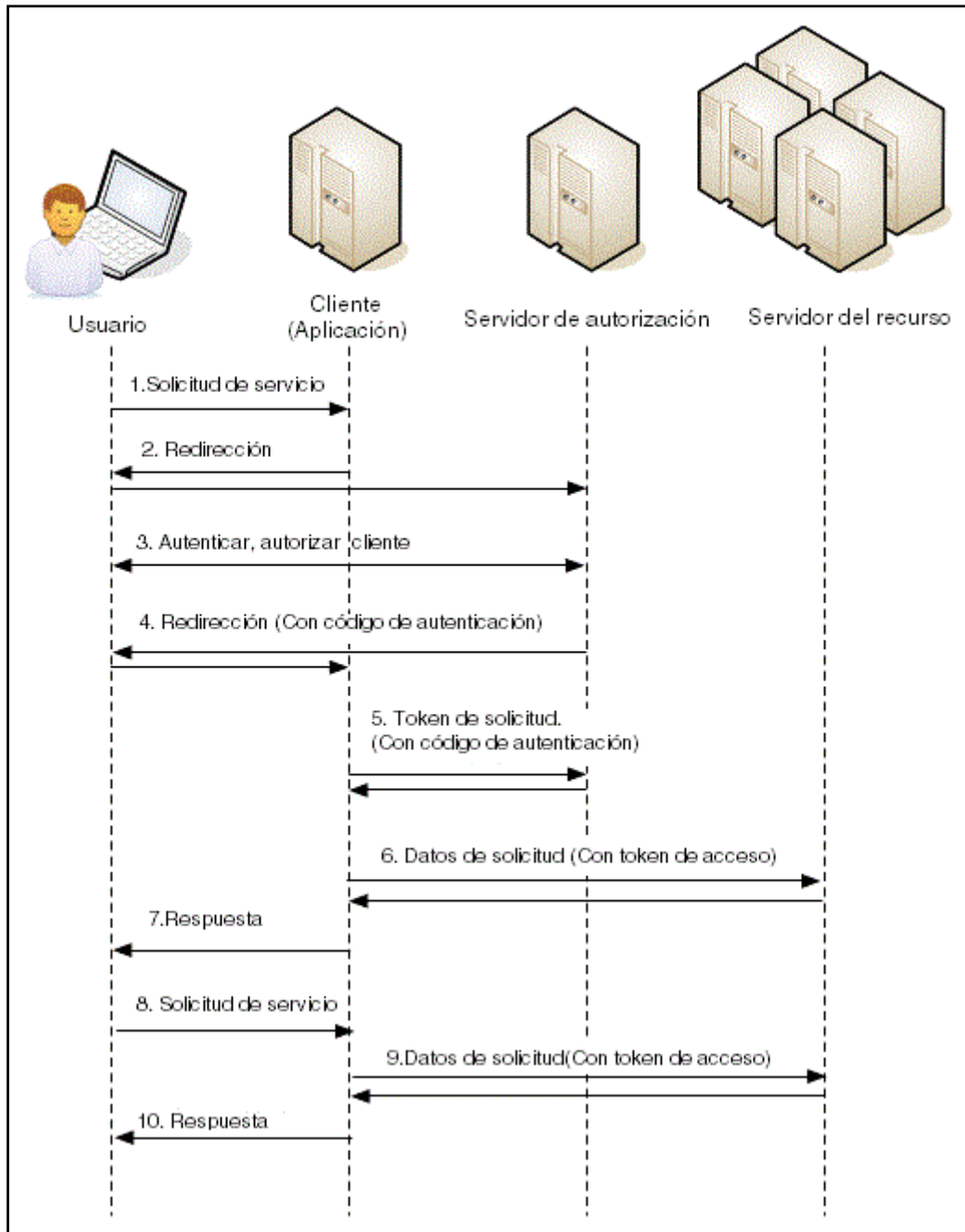


Ilustración 2: Diagrama de autenticación mediante OAuth



## 2.2.4. CakePHP

CakePHP<sup>6</sup> es un marco de desarrollo (Framework) rápido para PHP, libre, de código abierto. Se trata de una estructura que sirve de base a los programadores para que éstos puedan crear aplicaciones Web.

Este framework trabaja utilizando el patrón MVC (Modelo-Vista-Controlador), consiste en separar la aplicación en tres partes principales. El modelo representa los datos de la aplicación, la vista hace una presentación del modelo de datos, y el controlador maneja y enruta las peticiones hechas por los usuarios.

## 2.2.5. Smarthome

Para poder enlazar tareas y acciones que ocurren en una vivienda domótica, el tutor del proyecto Joan Fons Cors, nos proporcionó un simulador que contiene una capa REST con funciones básicas que permiten obtener el estado de los dispositivos de la casa domótica.

En la API de la Smarthome solo podemos hacer los métodos HTTP GET y PUT. El primero permite obtener información sobre el estado de los dispositivos y el segundo permite cambiar dicho estado.

Existen muchos dispositivos en la vivienda inteligente, cada uno de estos dispositivos es de un determinado tipo de recurso. Y para cada recurso hay unas acciones predeterminadas. De este modo, la Luz de la Cocina es un dispositivo de tipo biestable y sus acciones permitidas son encender y apagar.

Más adelante expondremos que dispositivos van a ser utilizados y que métodos podemos consultar.

---

<sup>6</sup> <http://cakephp.org>

## 2.3. Herramientas software

Para la realización de este proyecto se han utilizado dos herramientas ampliamente conocidas como son Eclipse y Xampp. A continuación se explicarán algunas características interesantes que han justificado la elección de este software para el desarrollo

### 2.3.1. *Eclipse*

El entorno de programación Eclipse, proporciona un conjunto de herramientas de código abierto para programación, con soporte para el lenguaje de programación Java, muy útil para la depuración del código y de sus funciones además cuenta con una grna cantidad de plugins que ayudan a la programación, además esta herramienta ha sido ampliamente utilizada durante la carrera y es por lo que se ha elegido como entorno de desarrollo para el proyecto.

En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plugin). Hay plugins para el desarrollo de Java (JDT Java Development Tools) así como para el desarrollo en C/C++, COBOL, PHP etc. (Eclipse (2.1) y Java- Departamento de Informatica- Universidad de Valencia, 2004)

En este caso la versión utilizada es la Juno Service Release 2 <sup>7</sup>. Los plugins para desarrollo Java y para desarrollo en PHP. Para el desarrollo a la parte referente a Google Task usaremos un plugin con su API llamado Task API. Como añadido, también se utiliza el plugin de Google Oauth2 API que nos brindará todas las bibliotecas para poder programar cómodamente.

### 2.3.2. *Xampp*

XAMPP es el entorno más popular de desarrollo con PHP. El nombre proviene del acrónimo de X (para cualquier sistema operativo), Apache, MySQL, PHP, Perl.

El programa está liberado bajo la licencia GNU y actúa como un servidor web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris y Mac OS X. (Wikipedia - XAMPP, 2014)

Gracias a este entorno de desarrollo, es posible crear un servidor web privado para trabajar de manera local. De este modo, se ha podido llevar a cabo el desarrollo de la

---

<sup>7</sup> <http://www.eclipse.org/juno/>

aplicación web que trabaja de interfaz de la API, y también se ha implementado una base de datos MySQL en la que guardar algunos datos de nuestra aplicación.

## 2.3.3. Navegadores

Para poder completar el desarrollo de la interfaz web, se han utilizado los navegadores más utilizados los últimos años (No Two The Same, 2013):

- **Google Chrome**<sup>8</sup>: El navegador desarrollado por Google, es actualmente el más utilizado en el mundo motivo que incentiva su uso, ya que pretendemos que la interfaz este bien optimizada. También cuenta con Chrome Store, lugar donde se pueden encontrar muchísimos programas añadidos al navegador.

*La versión utilizada es la 36.0.1985.143m.*

- **Internet Explorer**<sup>9</sup>: El navegador instalado por defecto en el sistema operativo Windows.

*La versión utilizada es la: 11.0.9600.17239.*

- **Mozilla Firefox**<sup>10</sup>: Otro navegador muy común, que será utilizado para comprobar que esté optimizada la interfaz para su uso.

*La versión utilizada es la 30.0.*

---

<sup>8</sup> [http://www.google.com/intl/es\\_es/chrome/](http://www.google.com/intl/es_es/chrome/)

<sup>9</sup> <http://windows.microsoft.com/es-es/internet-explorer/download-ie>

<sup>10</sup> <https://www.mozilla.org/es-ES/>

### 3. Diseño

Tras el análisis de la aplicación y como paso previo al desarrollo, se ha realizado un diseño global del proyecto para poder planificar las fases que se desarrollarán primero. Asimismo se ha diseñado como va a funcionar la aplicación y en qué lugares van a estar alojados los diferentes servicios que se proporcionan.

El proyecto engloba 3 aplicaciones independientes que se comunican mediante peticiones de tipo REST. El eje central de este proyecto es la API REST que genera las peticiones de las tareas a Google. Para que esta API pueda ser utilizada por el usuario, se ha desarrollado una interfaz web que permite usar las funciones de la API y por último, se utiliza el servidor domótico proporcionado por el tutor.

El funcionamiento básico de la aplicación sería el siguiente:

- Un usuario accede a la web de la casa domótica y realiza alguna acción en ella, por ejemplo crear una tarea asociada a un evento de la casa.
- En el momento en el que se acepta la tarea en la web y se guarda, se envían a la API REST los datos introducidos en la página.
- La API procesa estos datos y los trata de distintas formas. Por un lado, almacena algunos datos en la base de datos que está alojada en el servidor web y por otro lado, crea en Google la tarea con los datos introducidos.
- Por último, al ser una tarea asociada a un evento de la casa domótica, la API REST enviará peticiones al servidor domótico (Smarthome) para comprobar si ese evento ha ocurrido, en ese caso, se lanzaría un aviso a la web.

En la parte inferior podemos ver un diagrama del montaje de la aplicación.

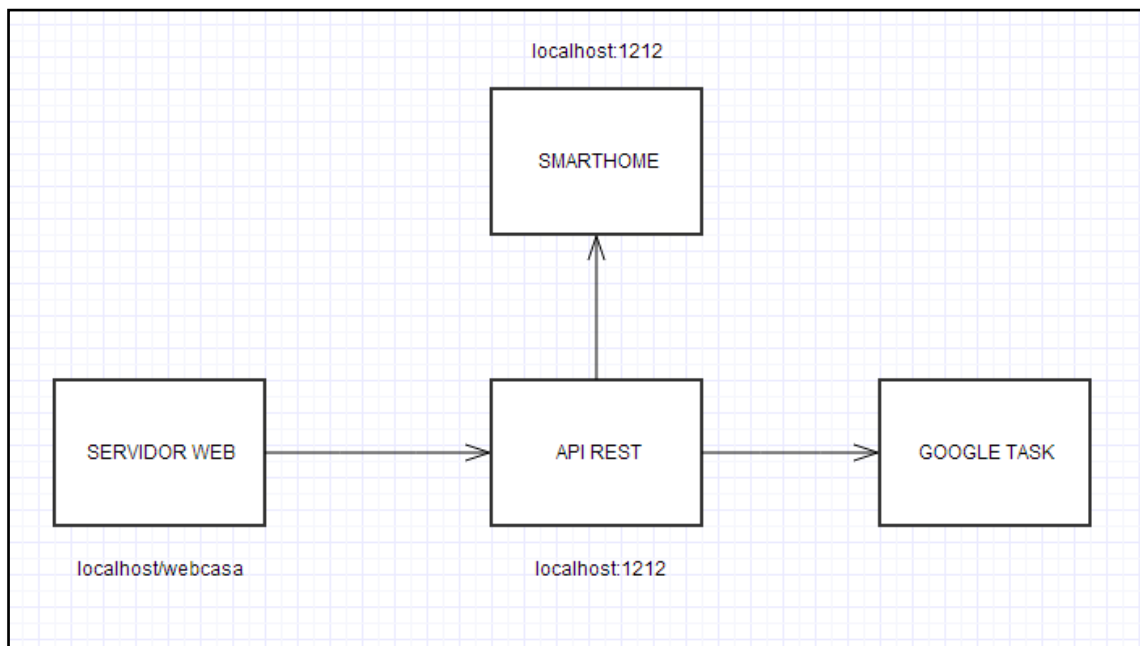


Ilustración 3: Estructura de la aplicación

## 3.1. Diseño de la API REST

Antes de entrar en el diseño de la funcionalidad de la API se diseñó el proceso para que el usuario se autentificara utilizando, como se sugiere en la parte de 'Análisis', OAuth 2. En este paso, se diseñó que la autenticación se encontrara en la API REST y se realizara como primer paso antes de acceder a la funcionalidad de la aplicación.

Para poder entender como se ha diseñado la API hay que entender lo que nos proporciona la API de Google Tasks. En Google Tasks hay dos elementos muy bien diferenciados, las listas de tareas y las propias tareas. Veamos qué métodos y atributos nos ofrece Google.

### Listas (List):

Estas son las funciones que podemos utilizar.

Tabla 1: API de Google Tasks - Funciones de Listas

Método	REST URI *	Descripción
list	GET /users/@me/lists	Devuelve las listas del usuario autenticado.
get	GET /users/@me/lists/ <i>tasklist</i>	Devuelve una lista específica del usuario autenticado
insert	POST /users/@me/lists	Crea una nueva lista y la guarda en el usuario autenticado
update	PUT /users/@me/lists/ <i>tasklist</i>	Actualiza la lista especificada.
delete	DELETE /users/@me/lists/ <i>tasklist</i>	Elimina la lista especificada.
patch	PATCH /users/@me/lists/ <i>tasklist</i>	Actualiza la lista especificada. Con diferente semántica a 'update'

A continuación veremos que atributos tienen las listas.

**Tabla 2: API de Google Tasks - Atributos de Listas**

Nombre	Tipo	Descripción
<b>kind</b>	string	Tipo de recurso, siempre es "tasks#taskList".
<b>id</b>	string	Identificador de la lista de tareas.
<b>etag</b>	string	ETag del recurso.
<b>title</b>	string	Título de la lista.
<b>selfLink</b>	string	URL que apunta a la propia lista.
<b>updated</b>	datetime	Última modificación de la lista (Con formato RFC 3339 timestamp).

- **Tareas (Task):**

Ahora veremos las funciones que nos proporciona la API de Google para trabajar con tareas.

**Tabla 3: API de Google Tasks - Funciones de Tareas**

Método	REST URI *	Descripción
<b>list</b>	GET /lists/ <i>tasklist</i> /tasks	Devuelve todas las tareas de la lista especificada.
<b>get</b>	GET /lists/ <i>tasklist</i> /tasks/ <i>task</i>	Devuelve la tarea especificada.
<b>insert</b>	POST /lists/ <i>tasklist</i> /tasks	Crea una nueva tarea, en la lista especificada.
<b>update</b>	PUT /lists/ <i>tasklist</i> /tasks/ <i>task</i>	Actualiza la tarea.
<b>delete</b>	DELETE /lists/ <i>tasklist</i> /tasks/ <i>task</i>	Elimina la tarea de la lista especificada.
<b>clear</b>	POST /lists/ <i>tasklist</i> /clear	Elimina las tareas completadas de la lista. Las tareas afectadas son marcadas como ocultas, y no serán devueltas por defecto al usuario.
<b>move</b>	POST /lists/ <i>tasklist</i> /tasks/ <i>task</i> /move	Mueve una tarea a otra posición de la lista especificada.
<b>patch</b>	PATCH /lists/ <i>tasklist</i> /tasks/ <i>task</i>	Actualiza la tarea, acepta semántica diferente a 'update'.

Y por último veamos que atributos tienen las tareas en esta API.

**Tabla 4: API de Google Tasks - Atributos de Tareas**

Nombre	Tipo	Descripción
<b>kind</b>	string	Tipo del recurso, siempre es "tasks#task".
<b>id</b>	string	Identificador de la tarea.
<b>etag</b>	etag	ETag del recurso.
<b>title</b>	string	Título de la tarea.
<b>updated</b>	datetime	Última modificación de la tarea (Con formato RFC 3339 timestamp).
<b>selfLink</b>	string	URL que apunta a la propia tarea
<b>parent</b>	string	Identificador de tarea padre. Este campo se omite si se trata de una tarea de nivel superior. Este campo sólo es de lectura.
<b>position</b>	string	Cadena que indica la posición de la tarea entre sus tareas hermanas bajo la misma tarea padre o en el nivel superior.
<b>notes</b>	string	Notas o contenido de la tarea.
<b>status</b>	string	Estado de la tarea, puede ser "needsAction" o "completed".
<b>due</b>	datetime	Fecha de vencimiento de la tarea (Con formato RFC 3339 timestamp).
<b>completed</b>	datetime	Fecha de realización de la tarea (Con formato RFC 3339 timestamp). Vacio si la tarea está pendiente
<b>deleted</b>	boolean	Indica si la tarea ha sido eliminada.
<b>hidden</b>	boolean	Indica si la tarea es oculta.
<b>links[]</b>	list	Colección de enlaces.
<b>links[].type</b>	string	Tipo de enlace. Por ejemplo "email".
<b>links[].description</b>	string	Descripción.
<b>links[].link</b>	string	La URL.

En este caso es necesario hacer una distinción entre listas y tareas por varios motivos, el primero es que cada uno de los elementos tiene unas funciones y atributos diferentes, el segundo es que a la hora de obtener los valores de los elementos las llamadas no se crean de la misma forma, motivo por el cual se crearon clases diferentes para listas y tareas. Además se optó por crear dos clases diferentes para listas (Lista y Listas) y otras dos clases para las tareas (Tareas y Tarea).

Esto vino motivado por la diferencia que hay entre funciones para una sola lista o para varias listas, lo mismo ocurre con las tareas.

La clase listas ofrecerá toda la funcionalidad para un conjunto de listas, y la clase lista definirá las funciones que se pueden realizar sobre una lista específica. De este modo, en la clase listas (*Listas.java*) tendremos las siguientes funciones:



- Obtener listas: Que obtiene todas las listas que existen en la cuenta del usuario que ha accedido.
- Insertar lista: En la que se inserta una lista, con los datos especificados por el usuario y se guarda en Google. Además se guardan en la base de datos algunos campos que no nos permite almacenar Google. Los atributos que se guardaran son el identificador de la lista, 'compartida' y 'compartida con'.

Por otro lado la clase lista (*Lista.java*) tendrá estas funciones:

- Obtener lista: Obtiene todos los datos de una lista identificada. En este paso se deben obtener los atributos almacenados en la cuenta de Google y también los que se ha guardado en la base de datos.
- Eliminar lista: Elimina una lista especificada por el usuario. Se deben borrar los datos de Google así como los guardados en la base de datos.

Del mismo modo que en las listas se crean dos clases. En el caso de que se trate de varias tareas, la clase será tareas (*Tareas.java*) y contará con estas funciones:

- Obtener tareas: Se obtienen todas las tareas que existen en la lista especificada.
- Insertar tarea: En la que se inserta una tarea, con los datos especificados por el usuario y se guarda en Google. Además de los datos que se almacenan en Google, se realizará un guardado en la base de datos de otros atributos como si tiene eventos asociados, que tipo de eventos son, si es compartida, etc.

Además la clase tarea (*Tarea.java*) proporcionará a la API que estamos desarrollando estas funciones:

- Obtener tarea: Obtiene todos los datos de una tarea, esta función devuelve tanto los datos que se almacenan en Google como lo que se almacenan en la base de datos.
- Eliminar tarea: Elimina la tarea seleccionada por el usuario. Esta acción debería eliminar los datos que están en Google y los datos que se almacenaron en la base de datos previamente.

Una vez llegados a este punto, se tomó una decisión acerca del modo de transmisión de los datos entre los diferentes componentes de la aplicación, la API REST, la API de Google Task, la web y el servidor Smarthome. Se acabó optando por utilizar el formato JSON<sup>11</sup>.

---

<sup>11</sup> <http://json.org/>



### 3.1.1. *Uso de JSON*

JSON (JavaScript Object Notation), es un formato ligero para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. Está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, tabla hash.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como vectores, listas o secuencias.

Un ejemplo de datos guardados en JSON sería lo siguiente:

```
var objetoJSON = {"usuario":"user","password":"123456"}
```

Donde el valor del atributo usuario es user, y el valor del atributo password es 123456.

Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías. Este aspecto es el que ha motivado la elección de JSON para nuestra aplicación, ya que se pasarán datos entre diferentes aplicaciones, siendo necesario que todas las partes implicadas puedan leer los mismos datos sin necesidad de traducirlos a los lenguajes de programación de cada aplicación. (Rodríguez, 2013)

### 3.1.2. *CrossDomain*

Se trata de un mecanismo de seguridad de las comunicaciones en los navegadores actuales. Evitan que un script (XMLHttpRequest de AJAX) o una aplicación (Flash, Silverlight) de una página web puedan acceder a un servidor web diferente del que residen. (Salgado, 2008)

Intentan ayudar a evitar dos formas habituales de sabotaje en internet, el (1) Cross Site Request Forgery (XSRF o session riding) y el (2) Cross Site Scripting (XSS).

Dado que la interfaz web y la API REST se encuentran en servidores diferentes, se tendrá en cuenta este mecanismo, a la hora de la implementación.

## 3.2. Diseño del portal web

Para poder utilizar las funciones de la API que se ha desarrollado, hacía falta contar con una interfaz. En este caso se optó por una interfaz web que permitiera utilizar la funcionalidad de nuestra API REST. Como se ha comentado en el análisis, se va a utilizar CakePHP ya que nos ofrece muchas opciones a la hora de desarrollar este tipo de interfaces web.

Se van a crear varias interfaces referentes a listas, tareas usuarios y eventos. A continuación se muestran los bocetos realizados.

### Listas:

- Vista principal:

Este es el diseño de la vista principal de listas de tareas, con una tabla que está llena de contenido referente a cada lista. En la parte derecha se encuentran las acciones para añadir nuevos elementos como listas o tareas.

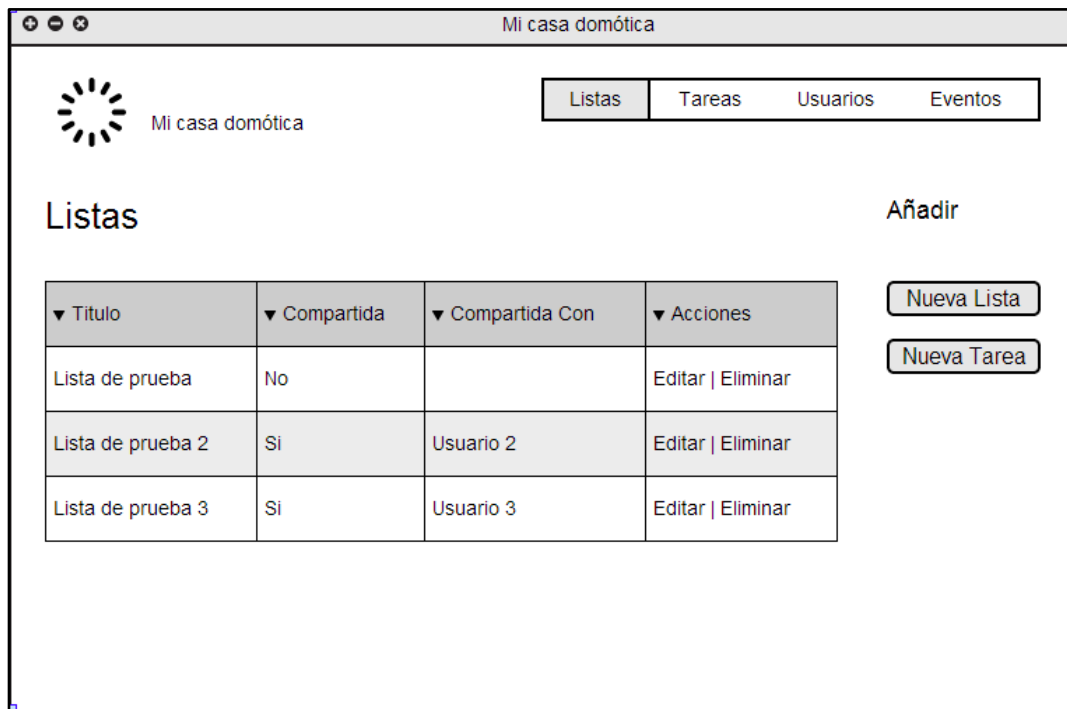
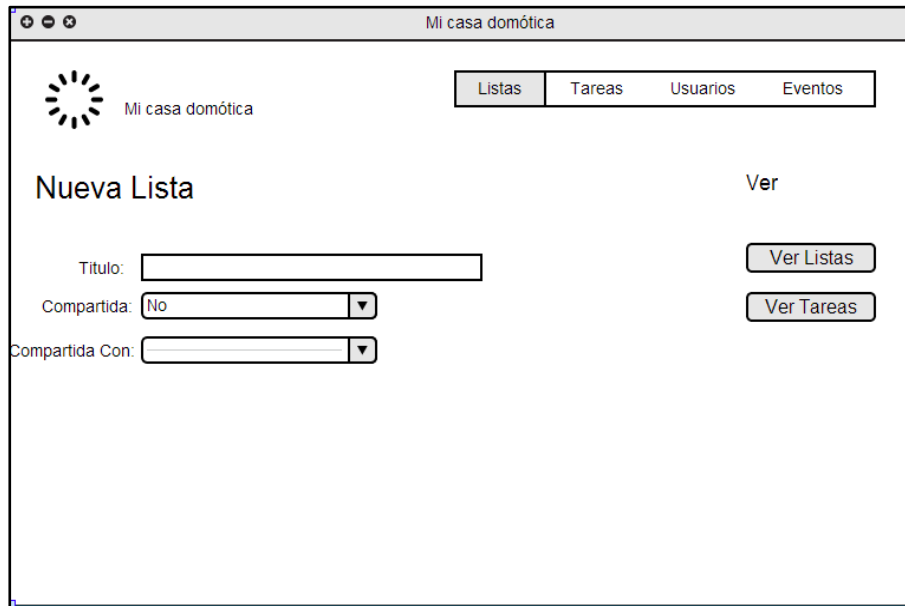


Ilustración 4: Listas – Vista principal

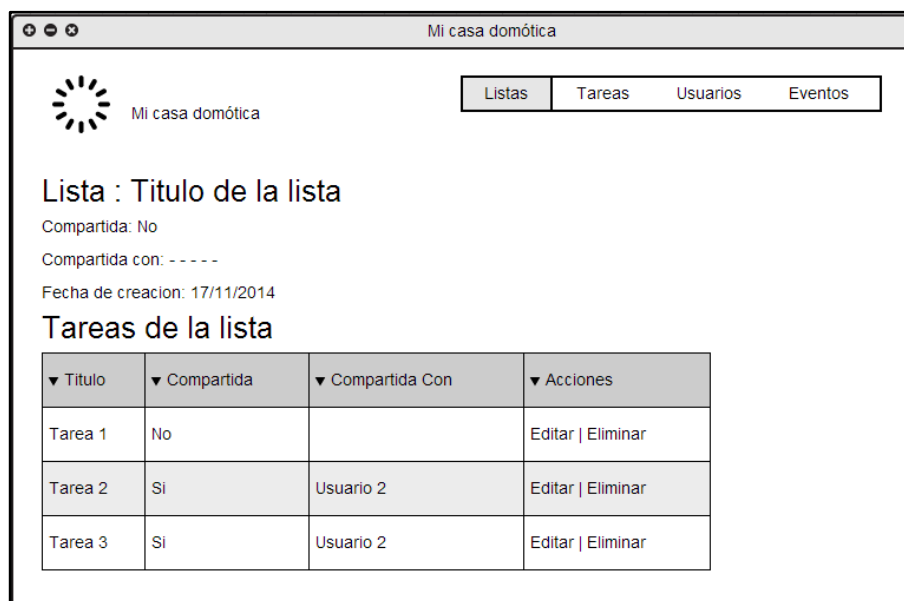
- Nueva Lista: En este caso, se muestra la interfaz que se usaría para crear una nueva lista. Se muestra un pequeño formulario con los campos requeridos, y en la derecha de la pantalla varias opciones para mostrar el contenido ya existente. En este caso listas y tareas.



**Ilustración 5: Listas – Nueva Lista**

- Contenido de la lista:

Por último, esta es la interfaz que aparece al entrar en el contenido de una lista determinada. Donde se muestran los datos relativos a la lista, y en la parte inferior se observa una tabla con las tareas que pertenecen a dicha lista, con sus principales atributos.

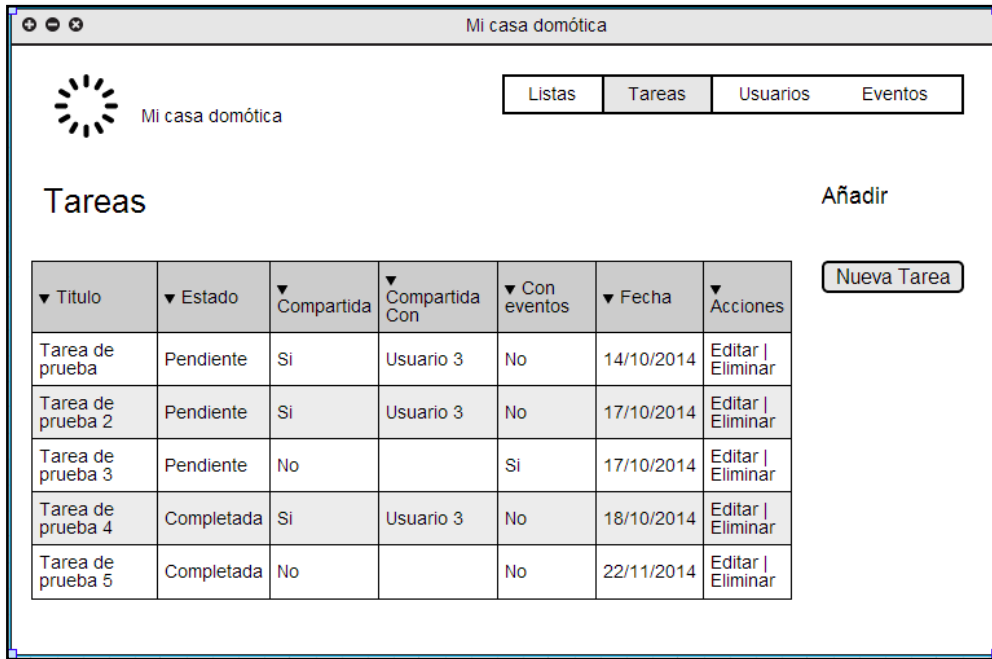


**Ilustración 6: Listas – Contenido de la Lista**

**Tareas:**

- Vista principal:

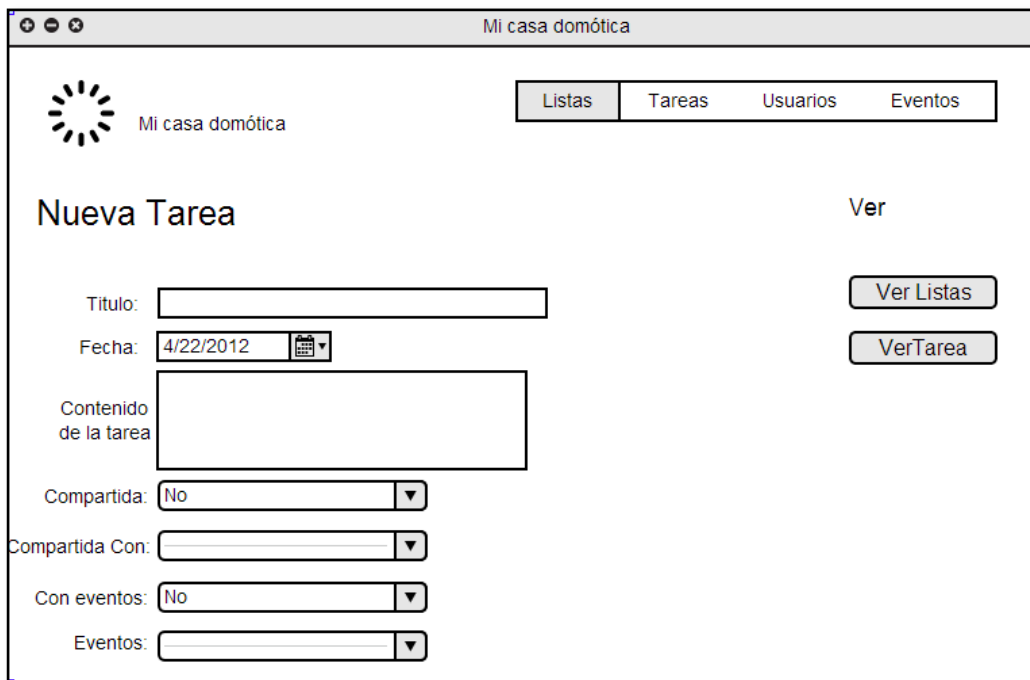
En esta primera interfaz observamos una tabla, con todos los atributos referentes a cada una de las tareas que pertenecen a una lista. Encontramos un botón en la parte derecha de la pantalla para crear nuevas tareas.



**Ilustración 7: Tareas – Vista Principal**

- Nueva Tarea:

A continuación se puede observar la interfaz para la creación de nuevas tareas, es muy similar a la interfaz de creación de listas, pero añadiendo los campos requeridos para una tarea nueva. En la parte derecha se encuentran las opciones



para ver las listas y tareas ya creadas.

- Contenido de la tarea:

Este es el diseño de la interfaz que muestra la información de una tarea determinada. En ella podemos ver los datos que conforman la tarea además de una pequeña tabla con los eventos de la casa domótica asociados a la tarea.



**Ilustración 9: Tareas – Contenido de la Tarea**

### Usuarios:

- Vista principal:

En esta interfaz se pueden ver los usuarios que se encuentran en el sistema, así como los datos de cada uno de ellos.

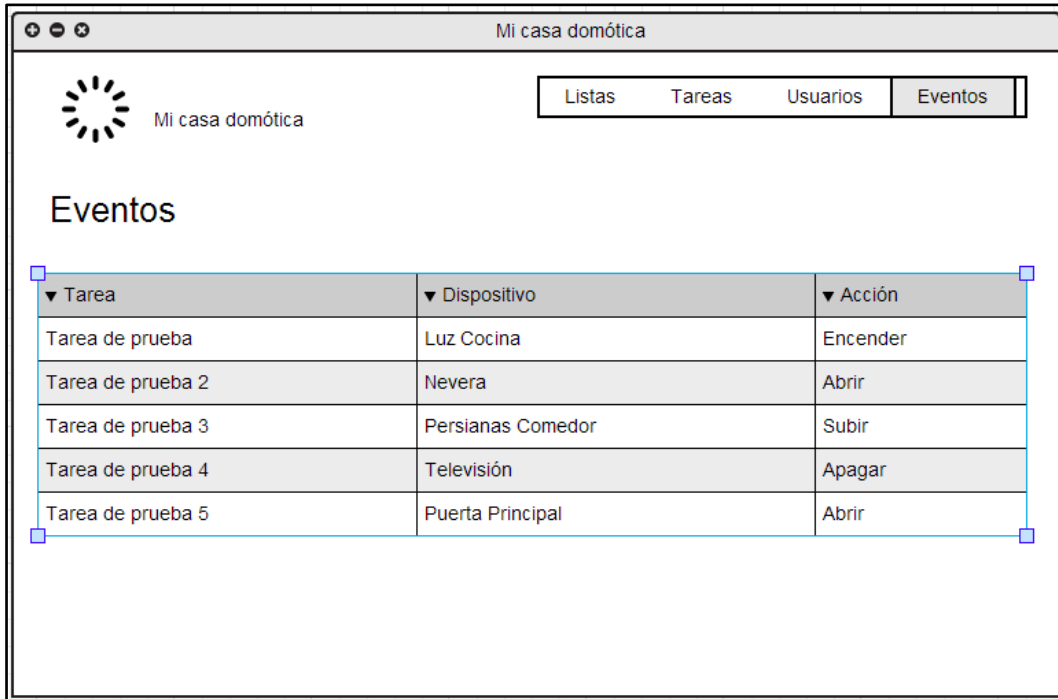


**Ilustración 10: Usuarios- Vista Principal**

## Eventos:

- Vista principal:

La última interfaz es la de eventos, en ella se muestran los eventos que están asociados a las tareas que han sido creadas.



▼ Tarea	▼ Dispositivo	▼ Acción
Tarea de prueba	Luz Cocina	Encender
Tarea de prueba 2	Nevera	Abrir
Tarea de prueba 3	Persianas Comedor	Subir
Tarea de prueba 4	Televisión	Apagar
Tarea de prueba 5	Puerta Principal	Abrir

**Ilustración 11: Eventos – Vista Principal**

### 3.3. Diseño de la base de datos

Tras el análisis realizado al principio del proyecto, la creación de una base de datos en la que almacenar los atributos que no permitía Google Tasks destacó como una necesidad imprescindible.

En la nueva base de datos se contará con 4 tablas: Listas, Tareas, Usuarios y Eventos.

- **Listas:** En esta tabla se almacenarán los atributos 'id', 'compartida' y 'compartida\_con'.
  - *Id* (Entero): Es el identificador de la lista que se obtiene de Google Task al insertar una lista. Este atributo es la clave primaria de la tabla.
  - *Compartida* (Booleano): Indica si la lista está compartida con algún usuario de la base de datos.
  - *Compartida\_con* (Cadena de caracteres): Es el usuario con el que se comparte la lista, en caso de que sea una lista compartida. Este atributo es una clave ajena, que enlaza con el atributo *id* de la tabla de Usuarios y relaciona ambas tablas.
- **Tareas:** En esta tabla se almacenan muchos atributos, ya que la funcionalidad sobre las tareas es mucho mayor que la que hay sobre listas.
  - *Id* (Entero): Es el identificador de la tarea que se obtiene al insertar una tarea en Google. Este atributo es la clave primaria de la tabla.
  - *Id\_lista* (Entero): Es el identificador de la lista a la que pertenece la tarea almacenada. Este atributo es una clave ajena que enlaza con el atributo *id* (Clave primaria) de la tabla Listas.
  - *Estado* (Cadena de caracteres): Muestra si la tarea está en estado 'Pendiente' o 'Completada'.
  - *Compartida* (Booleano): Indica si la tarea está compartida con algún usuario.
  - *Compartida\_con* (Cadena de caracteres): Es el usuario con el que se comparte la lista, en caso de que sea una lista compartida. Del mismo modo que en la tabla listas, este atributo es una clave ajena, que enlaza con el atributo *id* de la tabla de Usuarios.



- *Con eventos* (Booleano): Establece si la tarea almacenada tiene eventos asociados o no.
- **Eventos:** El almacenamiento de los eventos que pueden ocurrir en la vivienda inteligente se realiza en esta tabla.
  - *Id* (Entero): Este atributo identifica al evento que se almacena. Este atributo es la clave primaria de la tabla.
  - *Dispositivo* (Cadena de caracteres): Se almacena que dispositivo de la casa domótica va a ser el que active el evento.
  - *Acción* (Cadena de caracteres): Este atributo establece que acción se debe realizar en el dispositivo para activar el evento.
  - *Id\_tarea* (Entero): Es el identificador de la tarea que tiene el evento asociado. Este atributo es una clave ajena que relaciona esta tabla con el atributo *id* (Clave primaria) de la tabla Tareas.
- **Usuarios:** Por último, en esta tabla, se almacenan todos los usuarios que están creados en el sistema.
  - *Id* (Entero): Este atributo identifica al usuario que se ha guardado en la base de datos. Este atributo es la clave primaria de la tabla.
  - *Nombre* (Cadena de caracteres): Nombre del usuario introducido.
  - *Apellidos* (Cadena de caracteres): Apellidos del usuario introducido.
  - *Email* (Cadena de caracteres): Este atributos establece que email tiene el usuario. Es muy importante, porque es el nombre de usuario de la cuenta de Google. De este modo se acceden a las tareas.
  - *Fecha\_sesion* (Datetime): Indica cuando ha sido el último acceso que ha realizado el usuario a la aplicación.

A continuación se muestra el diagrama de la tabla diseñada, incluyendo los campos necesarios y las relaciones entre las tablas

A continuación se muestra el diagrama de la tabla diseñada, incluyendo los campos necesarios y las relaciones entre las tablas.



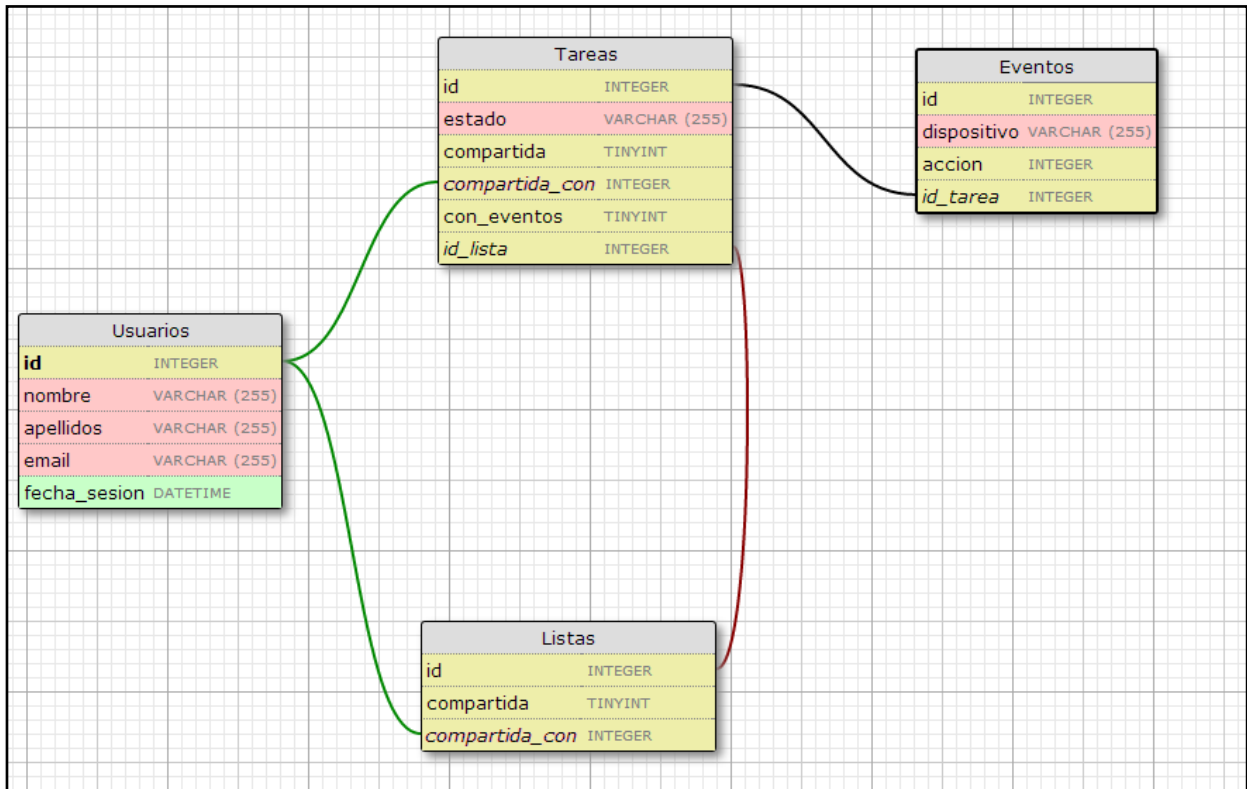


Ilustración 12: Diagrama de la Base de Datos

### 3.4. Integración con la Smarthome

En este apartado se comentará la integración de la casa domótica con la API REST y la web. Para ello, se definen las acciones que se pueden llevar a cabo en la vivienda desde las interfaces web y la funcionalidad que la API puede obtener de la Smathome.

En primer lugar la integración con la casa domótica se realiza de dos maneras, en la API REST, se crea un método que obtiene el estado de los dispositivos que tienen activos la casa, además se creará otro método capaz de producir cambios en el estado de dichos dispositivos.

Por otro lado, en la página web se almacenan algunos de los eventos que pueden ocurrir en la vivienda inteligente. En este caso no se van a introducir todos los posibles eventos y acciones la base de datos, sino que van a introducirse un conjunto determinado de dispositivos y sus posibles acciones.

Para finalizar en la interfaz web se creará, dentro de la vista de eventos, unos desplegables para seleccionar que acción se quiere provocar en la vivienda, posibilitando de este modo que se lancen los avisos de las tareas relacionadas con ese evento.

En la base de datos se van a introducir los dispositivos que se muestran a continuación:

**Tabla 5: Dispositivos de Smarthome utilizados**

Funcionalidad	ID-Funcionalidad	Descripción
<b>bistate (Biestable)</b>	DF-CUINA.NEVERA.EN	Nevera
	DF-CUINA.CAFETERA.EN	Cafetera
<b>togglebistate (Biestable)</b>	DF.CUINA.IL.CENTRAL	Luz Central (Cocina)
	DF.MENJ.EN.ENDOLLS	Enchufes (Comedor).

- *bistate* representa algo que puede estar en dos estados, que habitualmente son encendido o apagado.
- *togglebistate*: Representa lo mismo que *bistate* pero además, añade otra operación que permite conmutar el estado en el que se encuentra.

Para estos dispositivos, se establecen una serie de acciones que pueden realizar. En este caso se introducirán en la base de datos las opciones de estos dispositivos, como se muestra a continuación.

**Tabla 6: Acciones de Smarthome utilizados**

<b>Funcionalidad</b>	<b>Acciones permitidas.</b>
<b>bistate (Biestable)</b>	biaON(Encendido)
	biaOFF(Apagado)
<b>togglebistate (Biestable)</b>	biaON(Encendido)
	biaOFF(Apagado)
	biaToggle(Toggle)

La sintaxis de de las acciones representa que tipo de dispositivo es, si se desarrolla una acción, y que tipo de acción es.

De este modo la acción biaOFF se puede desgranar de la siguiente manera:

bia: Indica que el dispositivo es de tipo bistable.

a: Indica que se va a llevar a cabo una *acción* sobre el dispositivo.

OFF: Indica que acción va a llevarse a cabo.

Con esta explicación se da por terminada la fase de Diseño del proyecto, y se pasará a analizar la implementación realizado

## 4. Desarrollo

---

A continuación se definirá el desarrollo e implementación de la API REST, que es el objetivo principal del proyecto. En este apartado, se explicará la funcionalidad que aporta cada una de las clases creadas y se fundamentarán los ejemplos con parte del código implementado.

Esta API ha sido dividida en dos paquetes (contenedores de clases Java) llamados *App* y *Cliente*. Se realiza esta diferencia, ya que en *App* se establecen los métodos que permiten la autenticación del usuario y los que establecen la ruta para obtener los recursos especificados. Por otro lado, en el paquete *Cliente*, se desarrollan las clases que dotan de funcionalidad a la API, en este caso las relacionadas con las listas y las tareas.

En la siguiente imagen, se muestran los diferentes paquetes, mencionados anteriormente y las clases que componen cada uno de ellos.

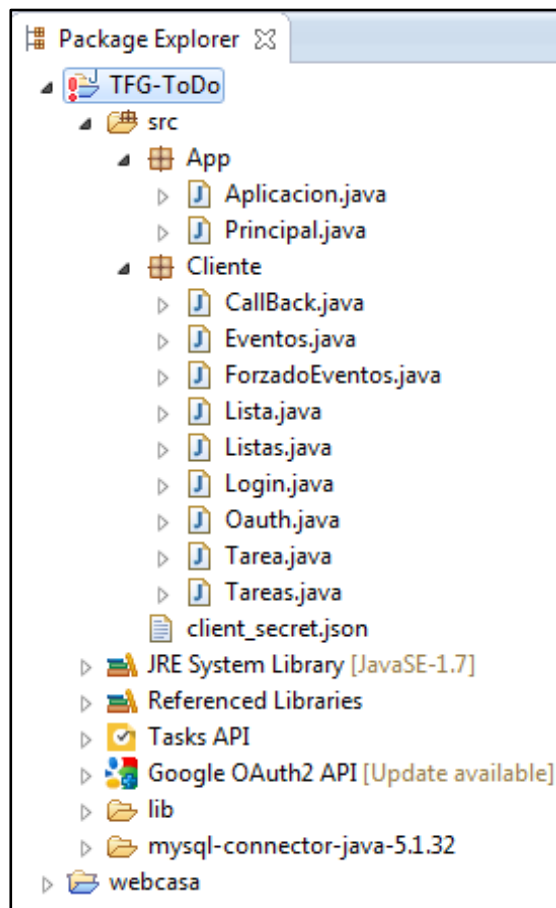


Ilustración 13: Contenido de la API REST

También se definirá brevemente, que desarrollo se ha realizado para implementar el portal web.

## 4.1. App

En este primer paquete, se definen dos clases, *Aplicación.java* y *Principal.java*. A continuación se expondrá el desarrollo de cada una de las clases.

Se mostrará el código más relevante que compone cada una de las clases, y se realizará una explicación del funcionamiento del mismo. Fragmentos código tales como librerías importadas, paquete al que pertenecen las clases, etc. no se incluirán a fin de facilitar la lectura. De igual modo, los fragmentos que se repiten han sido explicados la primera vez que aparecen.

### 4.1.1. *Principal.java*

Esta es la clase más importante de toda la aplicación, ya que es la que arranca el servidor HTTP.

En el código que se muestra a continuación se observa cómo se crea una nueva conexión HTTP, en el puerto 1212, que será el puerto que utiliza la API para resolver las peticiones. Para ello, dirige cualquier petición que recibe a la clase *Aplicación*, como se muestra.

```
public class Principal extends ServerResource {
    /**
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        Component component = new Component();
        // Create the HTTP server and listen on port 1212
    }
}
```

En el código que se muestra a continuación se observa cómo se crea una nueva conexión HTTP, en el puerto 1212, que será el puerto que utiliza la API para resolver las peticiones. Para ello, dirige cualquier petición que recibe a la clase *Aplicación*, como se muestra en la siguiente línea.

```
        component.getServers().add(Protocol.HTTP, 1212);
        component.getDefaultHost().attach("/", new Aplicacion());
        component.start();
    }
}
```



## 4.1.2. Aplicación.java

Esta clase es la que permite redirigir las peticiones efectuadas a la API a un recurso o a otro. A continuación se muestra el código que realiza esta acción.

```
public class Aplicacion extends Application {
    @Override
    public Restlet createInboundRoot() {
```

Se crea un elemento de tipo Router, que es el que permite realizar la redirección dentro de la clase, para acceder al resto de recursos que conforman la aplicación.

```
Router router = new Router(getContext());
```

En las siguientes líneas se enumeran las diferentes clases referenciadas, en función de la llamada que reciba la API.

```
router.attach("listas", Listas.class);
router.attach("listas/{idLista}", Lista.class);
router.attach("listas/{idLista}/tareas", Tareas.class);
router.attach("listas/{idLista}/tareas/{idTarea}", Tarea.class);
router.attach("login", Login.class);
router.attach("oauth2callback", CallBack.class);
router.attach("eventos/{dispositivo}", Eventos.class);
router.attach("forzadoeventos/{dispositivo}", ForzadoEventos.class);
return router;
}
```

De este modo la llamada a la dirección localhost:1212/listas/{id lista}/tareas haría referencia a la clase Tareas. Y de ese modo se podría acceder a todas las funciones disponibles en la clase mencionada.

## 4.2. Cliente

En este paquete se encuentra toda la funcionalidad que proporciona la API, podemos encontrar las clases relacionadas con las listas y tareas, la clase que servirá para simular eventos en la casa domótica, y además las clases Login.java, Oauth.java y Callback.java.

A continuación se llevará a cabo una explicación acerca de cada una de las clases y las funciones que aporta.

### 4.2.1. Login

La principal funcionalidad de esta clase es la de autenticar a un usuario, para que posteriormente pueda acceder a sus tareas.

Esta clase extiende de ServerResource puesto que es una clase base para tratar con los recursos del servidor. Actúa como un contenedor para una llamada determinada, incluyendo la solicitud de entrada y la respuesta saliente. Así que nuestras clases dentro del paquete Cliente, siempre extenderán de esta clase.

```
public class Login extends ServerResource{
    @Get
    public String login(){
        String url = Oauth.AuthorizationURL();
        try {
            Desktop.getDesktop().browse(new URI(url));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (URISyntaxException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "OK";
    }
    else{
        return url;
    }
}
```

Básicamente la función de este código es iniciar la cadena de autenticación de usuarios. Esto se realiza llamando a la función AuthorizationURL() de la clase Oauth, como se aprecia en las primeras líneas de código.



## 4.2.2. Oauth.java

Esta clase se utiliza al igual que Login.java, para autenticar al usuario. Se explicará el funcionamiento de la clase, comentando el siguiente código.

Las funciones creadas se corresponden con los pasos requeridos para la autenticación de un usuario mediante OAuth explicados en el apartado de Análisis, que se muestran en la Ilustración 2.

En esta parte se crean las variables requeridas por la clase Oauth.java, que obtenemos de Google. En este caso son variables necesarias para poder autenticar al usuario posteriormente.

```
public class Oauth {

    private static HttpTransport httpTransport = new NetHttpTransport();
    private static JacksonFactory jsonFactory = new JacksonFactory();
    private static AuthorizationCodeFlow flow;
```

A continuación se introducen los datos que se han obtenido de la Consola de Google APIs<sup>12</sup>, creados específicamente para este proyecto. Estos datos autenticarán un usuario sobre el que trabajar, durante todo el desarrollo.

```
// The clientId and clientSecret are copied from the API Access tab on
// the Google APIs Console
private static String clientId = "919795637691-
412j01b47nlv4i67ufth3a4ndcmb5rmv.apps.googleusercontent.com";
private static String clientSecret = "RxfQTniF4dCK81PmptVEsvYI";
private static String redirectUri =
"http://localhost:1212/oauth2callback";
private static Collection<String> scopes = new ArrayList<String>();
public static GoogleCredential credential;
```

En este fragmento de código se genera un nuevo *GoogleAuthorizationCodeFlow*, para que se puedan insertar en el los valores que hay que comparar con Google. Este método genera una URL con toda la información necesaria para realizar el siguiente paso en la autenticación.

```
public static String AuthorizationURL() {
    System.out.println("Pidiendo URL");
    scopes.add("https://www.googleapis.com/auth/tasks");
    flow = new GoogleAuthorizationCodeFlow.Builder(httpTransport,
jsonFactory,clientId,clientSecret,scopes)
        .setAccessType("offline").build();
    String url =
flow.newAuthorizationUrl().setClientId(clientId).setRedirectUri(redirectUri).s
etScopes(scopes).build();
    System.out.println("URL: " + url );
    return url;
}
```

<sup>12</sup> <https://console.developers.google.com/>



La siguiente función crea una respuesta con el token de solicitud recibido, esto nos hace falta para poder llamar a la función *getCredential()*, que se explicará más adelante.

```
public static TokenResponse TokenRequest(String codigo) throws
IOException{
    TokenResponse token =
flow.newTokenRequest(codigo).setRedirectUri(redirectUri).execute();
    return token;
}
```

Esta última función, obtiene la credencial de Google, con la que poder trabajar. Para ello, se crea una petición con los datos obtenidos en las funciones anteriores, *TokenResponse*, *clientId*, *ClientSecret*, etc.

```
public static GoogleCredential getCredential(TokenResponse response){
    return new GoogleCredential.Builder().setTransport(httpTransport)
        .setJsonFactory(jsonFactory).setClientSecrets(clientId, clientSecret)
        .build().setFromTokenResponse(response);
}
```

### 4.2.3. *Callback.java*

Esta clase, tiene una única función llamada *callback()*, que recibe el código de acceso (*Access Token*) a la API de Google. Tras todo el proceso, se crea una credencial de acceso para que el usuario pueda utilizar los datos de Google.

```
public class CallBack extends ServerResource{

    @Get
    public String callback() throws IOException{
        String code = getQuery().getValues("code");
        TokenResponse token = OAuth.TokenRequest(code);
        GoogleCredential credential = OAuth.getCredential(token);
        OAuth.credential=credential;
        return "Login OK";
    }
}
```



## 4.2.4. Listas.java

Esta clase es la encargada de realizar las funciones *obtenerListas()* e *insertarListas()*. Además se añade una función *doOptions()* que permite evitar el problema de *CrossDomain* tratado en el apartado de *Diseño*.

En primer lugar explicaremos el funcionamiento de *obtenerListas()*, que devolverá todas las listas que el usuario tenga creadas.

La última línea de este fragmento de código es necesaria para evitar el problema de *CrossDomain*, en este caso se establece en las cabeceras de la petición que el control a este método se puede realizar desde cualquier servidor, al haber definido los accesos aceptados como "\*".

```
@Get
public Representation obtenerListas() throws IOException, JSONException{
    GoogleCredential credencial = OAuth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
}
```

En el código posterior, se crea la llamada que devuelve las listas que tiene el usuario.

```
TaskLists taskLists = service.tasklists().list().execute();
```

A continuación, se recorre una lista de listas, y se almacena en un Array de datos todo el contenido de cada una de las listas, para así poder tratarlo de manera independiente. Finalmente se devuelven los datos obtenidos en formato JSON, que como se ha comentado antes, es el seleccionado para el intercambio de datos en toda la aplicación.

```
JSONObject cuerpo = new JSONObject();
JSONArray datos = new JSONArray();

java.util.List<TaskList> listas = taskLists.getItems();

for(TaskList lista : listas){
    JSONObject Objetolista = new JSONObject();
    Objetolista.put("titulo", lista.getTitle());
    Objetolista.put("id", lista.getId());
    datos.put(Objetolista);
}

cuerpo.put("listas", datos);
return new StringRepresentation
(cuerpo.toString(),MediaType.APPLICATION_JSON);
}
```

La función *insertarListas()*, permite la creación de nuevas listas que serán almacenadas en Google, y además guarda en la base de datos, los atributos que se mencionan en el *Diseño*.

En este caso al insertar una nueva lista, se reciben los datos con los que rellenar la lista. Como se puede comprobar, los datos se obtienen del objeto recibido. Una vez se tiene cada dato por separado, se procede a la inserción de la nueva lista en Google.

```

    @Post
public String insertarListas(Representation body ) throws IOException,
JSONException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-
Origin", "*");

    GoogleCredential credencial = Oauth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);

    String datos= body.getText();
    JSONObject cuerpo = new JSONObject(datos);

    String title = cuerpo.getString("titulo");
    String compartida = cuerpo.getString("compartida");
    String compartida_con = cuerpo.getString("compartida_con");

    TaskList taskList = new TaskList();
    taskList.setTitle(title);

    TaskList nuevalista =
service.tasklists().insert(taskList).execute();

    String resultado ="";
    resultado = nuevalista.getId();
}

```

A continuación se muestra la consulta SQL que realiza el guardado de los datos en la base de datos. Uno de estos datos es el identificador de la lista, que se ha obtenido en el momento de su creación.

```

/*     PARTE SQL     */
try {
    Class.forName("com.mysql.jdbc.Driver");
    Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/
portal_casa_domotica", "root", "");
    Statement st = conexion.createStatement();
    String instruccion=("INSERT INTO listas
(`id`,`compartida`,`compartida_con`) VALUES
('"+resultado+"', '"+compartida+"', '"+compartida_con+
"'"));
    st.executeUpdate(instruccion);

}
catch(SQLException s){
    System.out.println("Error: SQL.");
    System.out.println("SQLException: " +
s.getMessage());
}
catch(Exception s){
    System.out.println("Error: Varios.");
    System.out.println("SQLException: " +
s.getMessage());
}
    System.out.println("FIN DE EJECUCIÓN.");
/*     FIN DEL SQL     */

return resultado;
}

```

Como ya se ha comentado, para evitar el *CrossDomain* se ha introducido en todas las clases esta función. En ella establecen las cabeceras adecuadas para poder acceder al servidor desde cualquier otra aplicación.

Estas cabeceras definen qué métodos HTTP están permitidos, los lugares de acceso permitidos y el tiempo de duración del acceso.

```
@Options
public void doOptions(Representation entity) {
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Methods", "POST,OPTIONS,GET,PUT,DELETE");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Headers", "Content-Type");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Credentials", "false");
    getMessageHeaders(getResponse()).add("Access-Control-Max-Age", "60");
}
}
```

## 4.2.5. Lista.java

Esta clase es la encargada de realizar las funciones *obtenerLista()*, *eliminarLista()* y *doOptions()*.

La función siguiente, permite obtener el contenido de una lista específica. Para ello, el identificador de la lista se obtiene de la propia petición a la API. Tras obtener dicho identificador, este método devuelve el objeto lista especificada donde se encuentra todo el contenido de dicha lista.

```
@Get
public Representation obtenerLista() throws IOException, JSONException{

    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
    GoogleCredential credencial = Oauth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);
    String idLista = (String) getRequestAttributes().get("idLista");

    TaskList lista = service.tasklists().get(idLista).execute();
    JSONObject Objetolista = new JSONObject(lista);

    System.out.println(Objetolista.toString());
    return new StringRepresentation
    (Objetolista.toString(),MediaType.APPLICATION_JSON);
}
```

A continuación se muestra la función que permite eliminar una lista especificada. Hay que tener en cuenta que además de eliminar la lista de Google, hay que eliminar el contenido relacionado en la base de datos.

De igual modo que en el código anterior, el identificador de la lista se obtiene de la petición. Primero se realiza el borrado en la base de datos y posteriormente en la cuenta de Google.

Por otro lado nos encontramos con la función *doOptions()* que ya se ha explicado con anterioridad.

```
@Delete
public String eliminarLista() throws IOException{

    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
    GoogleCredential credencial = Oauth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);
    String idLista = (String) getRequestAttributes().get("idLista");

    /*     PARTE SQL     */
    System.out.println("INICIO DE EJECUCIÓN. BORRADO DE LISTA");
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conexion =
            DriverManager.getConnection("jdbc:mysql://localhost/portal_casa_domotica", "root", "");
        Statement st = conexion.createStatement();
        String instruccion=("DELETE FROM listas WHERE id='"+idLista+"'");
        st.executeUpdate(instruccion);
        System.out.println("Parece que funciona y borra lo que toca");

    }
    catch(SQLException s){
        System.out.println("Error: SQL.");
        System.out.println("SQLException: " + s.getMessage());
    }
    catch(Exception s){
        System.out.println("Error: Varios.");
        System.out.println("SQLException: " + s.getMessage());
    }
    System.out.println("FIN DE EJECUCIÓN.");
    /*     FIN DEL SQL     */
    service.tasklists().delete(idLista).execute()
}

@Options
public void doOptions(Representation entity) {
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Methods", "POST,OPTIONS,GET,PUT,DELETE");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Headers", "Content-Type");
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Credentials", "false");
    getMessageHeaders(getResponse()).add("Access-Control-Max-Age", "60");
}
}
```



## 4.2.6. Tareas.java

En esta clase se definen las funciones *obtenerListas()*, *insertarTareas()*, *borraCompletadas()* y *doOptions()*.

Se explicará primero la función encargada de devolver las tareas que pertenecen a una lista. Como ocurren en las funciones de las clases anteriores, se definen las cabeceras y atributos para poder realizar la petición.

El identificador de la lista seleccionada se obtiene de la propia petición y se trabaja para obtener todas las tareas de dicha lista. Finalmente se devuelven los datos obtenidos en formato JSON.

```

@Get
public Representation obtenerTareas() throws IOException, JSONException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-
    Origin", "*");
    GoogleCredential credencial = Oauth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);

    String idLista = (String) getRequestAttributes().get("idLista");
    com.google.api.services.tasks.model.Tasks tasks =
    service.tasks().list(idLista).execute();

    JSONObject cuerpo = new JSONObject();
    JSONArray datos = new JSONArray();
    java.util.List<Task> tareas = tasks.getItems();

    for(Task tarea : tareas){
        JSONObject Objetolista = new JSONObject();
        Objetolista.put("titulo", tarea.getTitle());
        Objetolista.put("id", tarea.getId());
        Objetolista.put("contenido", tarea.getNotes());

        datos.put(Objetolista);
    }

    cuerpo.put("tareas", datos);
    return new StringRepresentation
    (cuerpo.toString(),MediaType.APPLICATION_JSON);
}

```

La siguiente función es la que permite insertar una determinada tarea en una determinada lista. Además de almacenar la tarea en Google, se realizará la inserción en la base de datos. Para ello necesitamos el identificador de la lista, que se obtiene como en ejemplos anteriores.

La función recibe un conjunto de datos, que se separan para poder realizar la creación de la tarea. Una vez introducida la tarea en el perfil de Google del usuario se obtiene su identificador y posteriormente, se guarda junto con otros datos en la base de datos.

La tarea puede tener eventos asociados a acciones que ocurren en la casa domótica, por lo que además habría que almacenar el evento al que hace referencia en la tabla *Eventos* de la base de datos.

```

    @Post
    public String insertarTareas(Representation body ) throws IOException,
    JSONException{
        getMessageHeaders(getResponse()).add("Access-Control-Allow-
        Origin", "*");
        GoogleCredential credencial = Oauth.credencial;
        Tasks service = new Tasks(httpTransport,jsonFactory,credencial);

        String datos= body.getText();
        JSONObject cuerpo = new JSONObject(datos);

        String idLista= cuerpo.getString("id");
        String titulo = cuerpo.getString("titulo");
        String contenido = cuerpo.getString("contenido");
        String compartida = cuerpo.getString("compartida");
        String compartida_con = cuerpo.getString("compartida_con");
        String con_eventos = cuerpo.getString("con_eventos");
        String dispositivo = cuerpo.getString("dispositivo");
        String accion = cuerpo.getString("accion");
        String fecha = cuerpo.getString("fecha");
        Task task = new Task();
        task.setTitle(titulo);
        task.setNotes(contenido);
        task.setDue(fecha);
        Task nuevatarea = service.tasks().insert(idLista,task).execute();

        String resultado ="";
        resultado = nuevatarea.getId();
        /*     PARTE SQL     */
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conexion =
                DriverManager.getConnection("jdbc:mysql://loc
                alhost/portal_casa_domotica", "root", "");
            Statement st = conexion.createStatement();
            if(con_eventos.compareTo("1")==0){
                String instruccion=("INSERT INTO tareas (`id`, `estado`,
                `compartida`, `compartida_con`, `con_eventos`, `id_evento`, `id_lista`) VALUES
                ('"+resultado+"', 'Pendiente', '"+compartida+"', '"+compartida_con+",
                '"+con_eventos+"', '0', '"+idLista+"')");
                st.executeUpdate(instruccion);
                String instruccion_evento=("INSERT INTO eventos
                (`dispositivo`, `accion`, `id_tarea`) VALUES ('"+dispositivo+"', '"+accion+",
                '"+resultado+"')");
                st.executeUpdate(instruccion_evento);
            }else{
                String instruccion=("INSERT INTO tareas (`id`, `estado`,
                `compartida`, `compartida_con`, `con_eventos`, `id_evento`, `id_lista`) VALUES
                ('"+resultado+"', 'Pendiente', '"+compartida+"', '"+compartida_con+",
                '"+con_eventos+"', '0', '"+idLista+"')");
                st.executeUpdate(instruccion);
            }
        }
        catch(SQLException s){
            System.out.println("Error: SQL.");
            System.out.println("SQLException: " +
            s.getMessage());
        }
        catch(Exception s){
            System.out.println("Error: Varios.");
            System.out.println("SQLException: " +
            s.getMessage());
        }
        /*     FIN SQL     */
        return resultado;
    }

```



La función que se explica a continuación, elimina de la cuenta de Google las tareas que hayan sido completadas. Esta opción no las borra del todo, ya que permanecen vinculadas a la cuenta del usuario pero no están visibles.

Para que se pueda realizar esta acción, se obtiene el identificador de la lista y se ejecuta la acción de borrado sobre todas las tareas de dicha lista.

```

public String borrarCompletadas (Representation body ) throws
IOException, JSONException{
    getMessageHeaders (getResponse ()).add ("Access-Control-Allow-
Origin", "*");

    GoogleCredential credencial = Oauth.credencial;

    Tasks service = new Tasks (httpTransport,jsonFactory,credencial);

    String datos= body.getText ();
    JSONObject cuerpo = new JSONObject (datos);
    String idLista= cuerpo.getString ("idLista");

    service.tasks ().clear (idLista).execute ();

    return "Las tareas completadas de la lista"+idLista+" se han borrado
correctamente";
}

```

Por último, se añadiría la función *doOptions()* que en este caso no se muestra en el código al haber sido explicada con anterioridad.



## 4.2.7. Tarea.java

La clase que se describe a continuación consta de tres funciones, *obtenerTarea()*, *eliminarTarea* y *doOptions()*. Seguidamente se explicará el funcionamiento de cada una de ellas.

En primer lugar, la función *obtenerTarea()* devuelve todo el contenido de una sola tarea, que ha sido especificada por el usuario. Para que se lleve a cabo esta operación se obtienen los identificadores de la lista y la tarea y se procede a la obtención de los datos.

Una vez se obtienen los datos, son convertidos a JSON y enviados a quien formuló la petición.

```
@Get
public Representation obtenerTarea() throws IOException,JSONException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-
Origin", "*");

    GoogleCredential credencial = Oauth.credencial;

    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);

    String Lista = (String) getRequestAttributes().get("idLista");
    String Tarea = (String) getRequestAttributes().get("idTarea");
    Task tarea = service.tasks().get(Lista, Tarea).execute();

    JSONObject Objetotarea = new JSONObject(tarea);
    return new StringRepresentation
(Objetotarea.toString(),MediaType.APPLICATION_JSON);
}
```

A continuación se muestra la función que permite eliminar una tarea de una lista especificada. Hay que tener en cuenta que además de eliminar la tarea de la cuenta de Google, hay que eliminar el contenido relacionado en la base de datos (dentro de *tareas* y *eventos*)

El identificador de la lista y el de la tarea se obtienen de la petición. Primero se lleva a cabo el borrado en las tablas afectadas en la de base de datos y posteriormente se borra el contenido existente en Google.

Además se incluye la función *doOptions()* que no se muestra al haber sido explicada con anterioridad.

```
@Delete
public String eliminarTarea() throws IOException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin", "*");
    GoogleCredential credencial = Oauth.credencial;
    Tasks service = new Tasks(httpTransport,jsonFactory,credencial);
    String Lista = (String) getRequestAttributes().get("idLista");
    String Tarea = (String) getRequestAttributes().get("idTarea");
```

```

        /*     PARTE SQL     */

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/portal_
                casa_domotica", "root", "");
            Statement st = conexion.createStatement();
            String instruccion=("DELETE FROM tareas WHERE id='"+Tarea+"'");
            st.executeUpdate(instruccion);
        }
        catch(SQLException s) {
            System.out.println("Error: SQL.");
            System.out.println("SQLException: " + s.getMessage());
        }
        catch(Exception s) {
            System.out.println("Error: Varios.");
            System.out.println("SQLException: " + s.getMessage());
        }
        System.out.println("FIN DE EJECUCIÓN.");
        /*     FIN DEL SQL     */

        service.tasks().delete(Lista,Tarea).execute();

        return "La lista con id :"+Lista+"ha sido eliminada
        correctamente";
    }

```

## 4.2.8. *Eventos.java*

En la siguiente clase veremos la función *obtenerEstado()* que devuelve el estado de un dispositivo determinado.

La función recibe el nombre del dispositivo en la llamada. Posteriormente, desde la función se realiza una llamada de tipo GET a la Smarthome, y se almacena su respuesta. Para finalizar se devuelven los datos, que llegaron con formato JSON, al que genero la llamada a nuestra API.

También se encuentra en esta clase la función *doOptions()* mencionada anteriormente.

```

@Get
public Representation obtenerEstado() throws IOException,
JSONException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-
    Origin", "*");
    String dispositivo = (String)
    getRequestAttributes().get("dispositivo");

    ClientResource resource = new
    ClientResource("http://localhost:8182/devFunc/"+dispositivo);

    Representation datos = resource.get();
    return datos;
}

```

## 4.2.9. ForzadoEventos.java

Esta clase se utiliza para poder generar eventos en la casa domótica, y provocar que haya cambios en el estado de un determinado dispositivo.

Se obtiene en la llamada a la función el nombre del dispositivo, mientras que la acción es recibida en el *body*. Se realiza la llamada a la dirección donde se encuentra el servidor, indicando a que dispositivo hacemos referencia. Y se le pasa en el cuerpo del mensaje el *body* donde se encuentra la acción.

También se encuentra en esta clase la función *doOptions()* mencionada anteriormente.

```
@Put
public void forzarEvento(Representation body) throws IOException,
JSONException{
    getMessageHeaders(getResponse()).add("Access-Control-Allow-Origin",
    "");
    String dispositivo = (String)getRequestAttributes().get("dispositivo");
    JSONObject cuerpo = new JSONObject(datos);

    String dispositivo = cuerpo.getString("dispositivo");

    ClientResource resource = new
    ClientResource("http://localhost:8182/devFunc/"+dispositivo);

    resource.put(body);
}
```

## 4.3. Desarrollo del portal Web.

El desarrollo de la interfaz web se ha realizado siguiendo al máximo los bocetos realizados durante el *Diseño*.

Se han realizado cambios en la interfaz de eventos, debido a la necesidad de crear en esta interfaz un modo de forzar eventos en la casa domótica.

En este apartado no se va a hacer una explicación del código como se realizó en la implementación de la API REST, puesto que no es la finalidad principal de este proyecto. En cambio se van a mostrar la interfaz desarrollada y sus diferentes vistas.

### Listas:

Este es el aspecto final que tendrá la interfaz de listas, donde se mostrarán todas las listas que tenga un usuario.

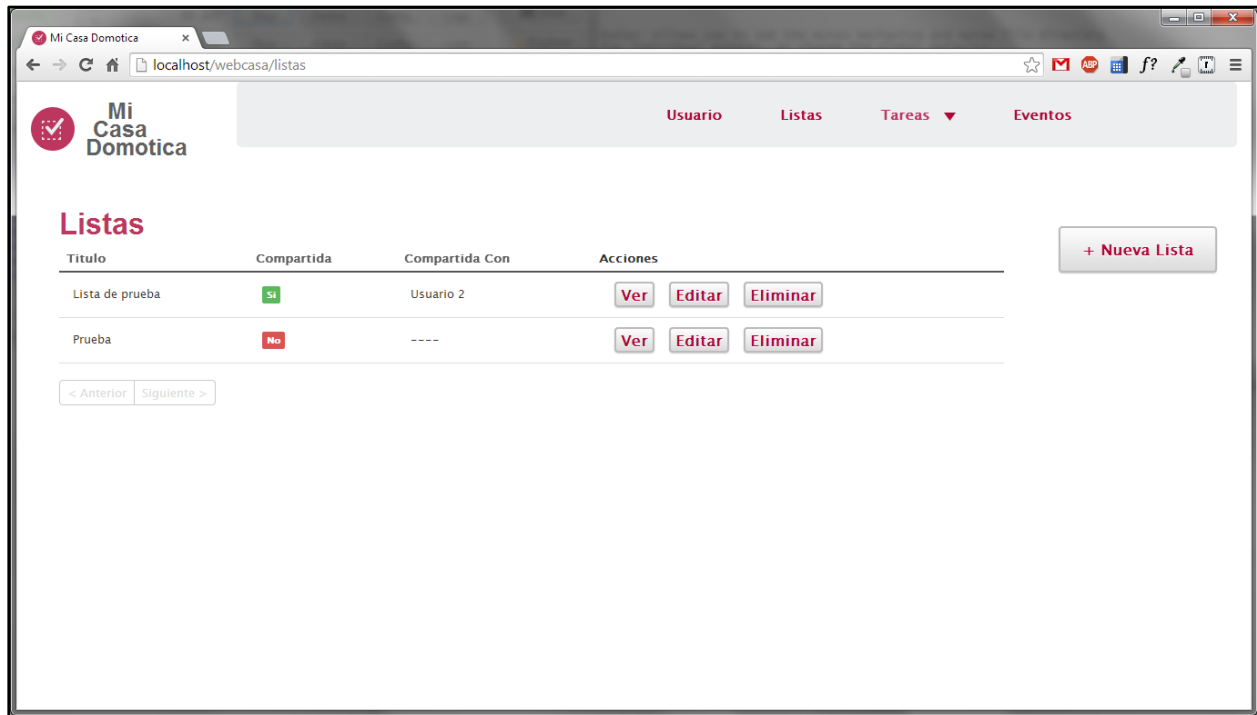


Ilustración 14: Interfaz web - Listas

### Tareas:

En la siguiente imagen se puede apreciar el formato que tendrá la interfaz de tareas, en la cual se crean tantas tablas como listas diferentes tenga el usuario.

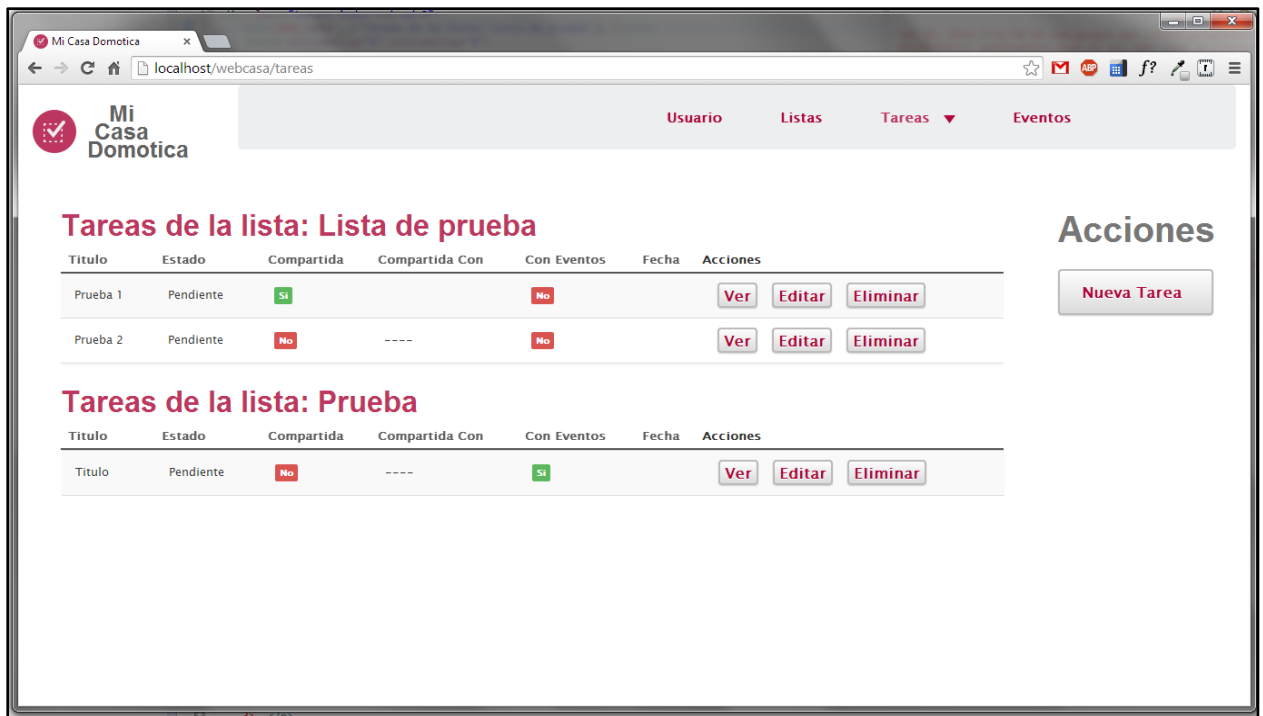


Ilustración 15: Interfaz web - Tareas

### Eventos:

Por último, se muestra la interfaz que finalmente tendrá el apartado de eventos. Como se comenta anteriormente, se ha introducido un selector en la parte derecha que permita la generación de eventos en la vivienda inteligente.

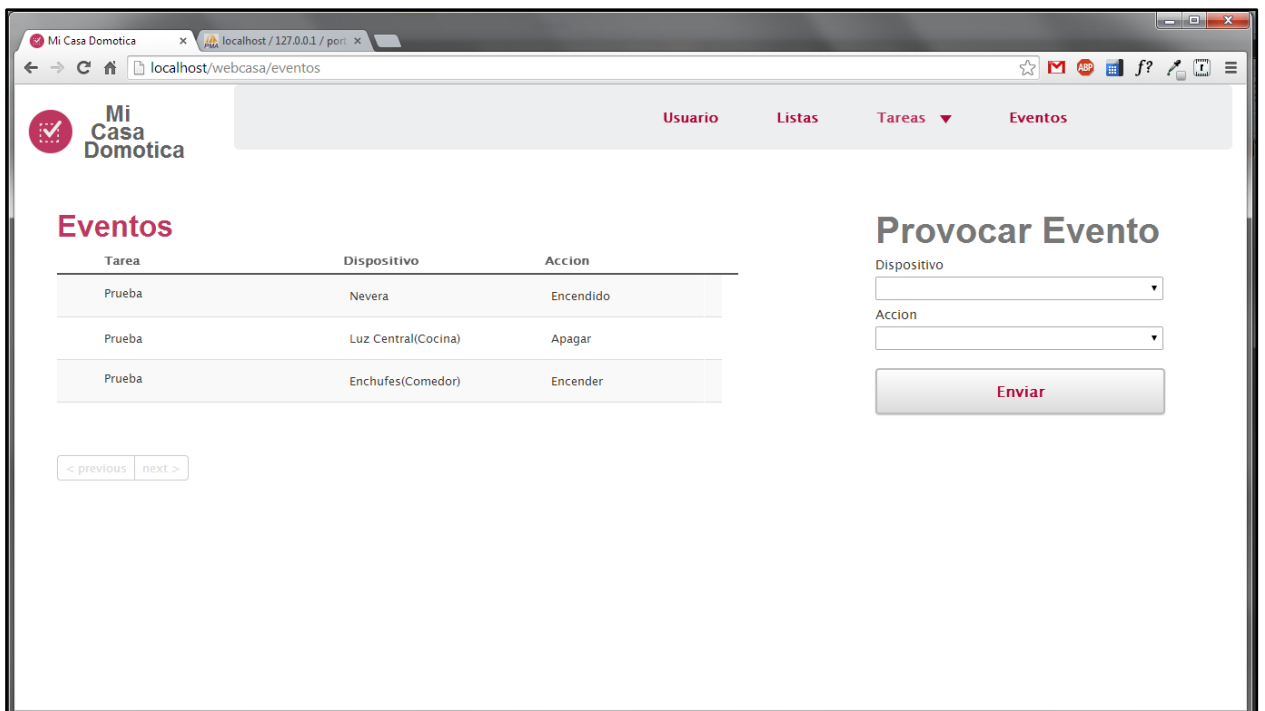


Ilustración 16: Interfaz web - Eventos

## 5. Mejoras aplicables

---

Al trabajo que se ha llevado a cabo se le podrían realizar mejoras que añadirían mayor funcionalidad al proyecto que se ha desarrollado. Entre las posibles ideas, se han seleccionado las que pueden tener mayor utilidad:

### **Asociar tareas a posiciones geográficas.**

Esta opción podría permitir al usuario la creación de nuevas tareas, que tengan asociadas posiciones o lugares. De este modo cuando el usuario se encontrase cerca de la posición, se le enviaría un aviso indicando que tiene una tarea pendiente de realizar cerca de su posición actual. Además, se obtendría la posición del usuario a través de su teléfono móvil.

### **Compartir tareas con varios usuarios.**

Un usuario crearía una tarea, y la compartiría con los usuarios que decidiera. En este caso, cualquiera puede realizar la tarea compartida, y cuando se diera por concluida se enviaría un aviso a todos los usuarios que comparten la tarea.

### **Crear etiquetas de tareas.**

Una buena idea sería la creación de diferentes etiquetas o categorías para almacenar en las tareas. Por ejemplo se podrían añadir etiquetas como Casa, Trabajo, Ejercicio, etc. A la hora de crear la tarea para que el usuario las categorice.

Así a simple vista tendría una idea acerca de las tareas pendientes que le quedan, y sobre que tratan.

### **Mejorar la interfaz web.**

La interfaz web puede mejorarse añadiendo por ejemplo un tablón donde compartir comentarios con otros usuarios. Esto resulta muy útil por ejemplo si el sistema se comparte con la familia, ya que cada miembro puede dejar notas de manera virtual.

## 6. Resultados

A continuación se presentará una traza de uso de la aplicación en la que se va a insertar una nueva tarea en una lista ya existente. Se mostrará además que se guardan los datos necesarios tanto en la base de datos como en la cuenta de Google.

Se creará una nueva tarea en la lista 'Lista de prueba', que llamaremos 'Tarea Resultado.'

En primer lugar se definirán los datos que forman parte de la tarea, como se muestra a continuación.

The screenshot displays the 'Mi Casa Domotica' application interface. At the top, there is a navigation bar with a home icon and the title 'Mi Casa Domotica'. To the right of the title are four menu items: 'Usuario', 'Listas', 'Tareas' (with a dropdown arrow), and 'Eventos'. Below the navigation bar, the main content area is divided into two sections. On the left, under the heading 'Nueva Tarea', there is a form with several fields: 'Seleccionar Lista' (a dropdown menu with 'Lista de prueba' selected), 'Titulo' (a text input field containing 'Tarea Resultado'), 'Fecha' (a text input field containing '01/09/2014 15:30'), and 'Contenido de la tarea' (a large text area containing 'Contenido de la tarea de prueba.'). Below these fields are sections for 'Compartir Tarea' (with 'Compartida' set to 'No' and 'Compartida Con' set to 'Usuario 2') and 'Eventos asociados' (with 'Con Eventos' set to 'Si', 'Dispositivo' set to 'Cafetera', and 'Accion' set to 'Encendido'). At the bottom left of the form is a red 'Guardar' button. On the right side of the form, under the heading 'Acciones', there are two buttons: 'Ver Tareas' and 'Ver Listas'.

Ilustración 17 : Nueva Tarea

A continuación se muestra la tarea introducida en Google.

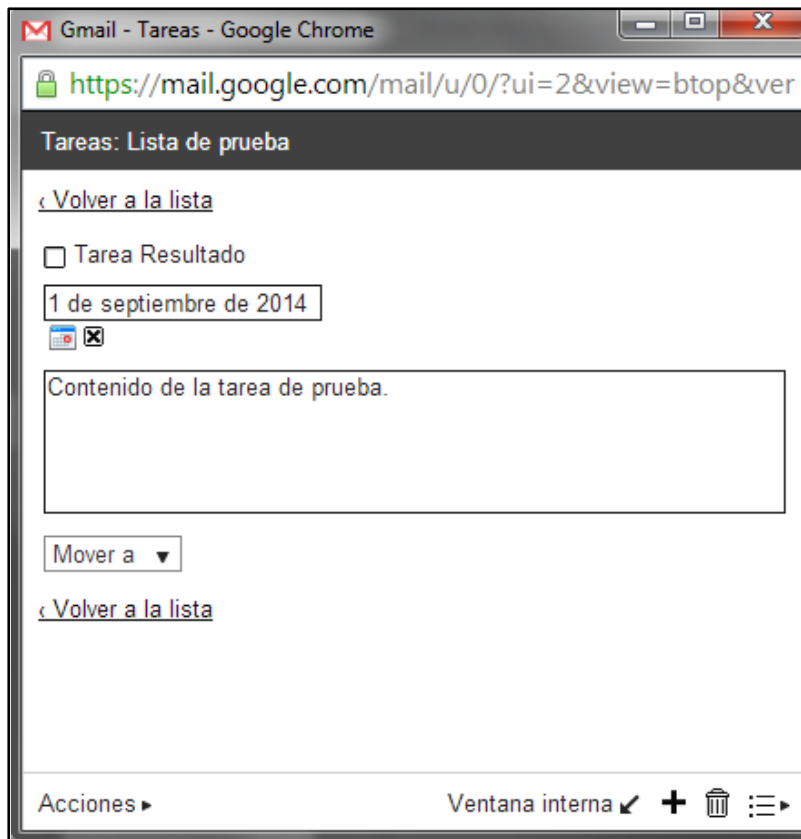


Ilustración 18: Nueva Tarea en Google

También se muestra como quedaría la tabla *tareas* una vez insertada la nueva tarea. En este caso la tarea insertada sería la primera de la lista.

id	id_lista	estado	compartida	compartida_con	con_eventos	id_evento
MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDoxMzU3MDQ0MzI4	MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDow	Pendiente	0	0	1	1
MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDoxNTI2ODU0ODAw	MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDow	Pendiente	1	3	0	0
MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDoxNzQ3MjcyMzk2	MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDow	Pendiente	0	0	0	0
MDk5NDAzNjkyMzMyNzMwMzgzMTg6MTkxODg0NDY3NDoxN	MDk5NDAzNjkyMzMyNzMwMzgzMTg6MTkx	Pendiente	0	0	0	0

Ilustración 19: Nueva Tarea en la Base de Datos


Al crear una nueva tarea con evento, se almacena en la tabla *Eventos* una nueva fila con la información introducida.

	id	dispositivo	accion	id_tarea
<input type="checkbox"/> <span>✎</span> Editar <span>📄</span> Copiar <span>🗑️</span> Borrar	1	Cafetera	Encendido	MDk5NDAzNjkyMzMyNzMwMzgzMTg6MDoxMzU3MDQ0MzI4

Ilustración 20: Nuevo Evento en la Base de Datos



Por último se muestra como queda la lista con la nueva tarea añadida, el resultado de la inserción se observa en la tabla de la parte inferior.

**Mi Casa Domótica**UsuarioListasTareas ▼Eventos

### Lista

Titulo: Lista de Prueba

Compartida: Si

Compartida Con: Usuario 3

Fecha de creacion: 15/08/2014

### Tareas de la lista

Titulo	Compartida	Compartida Con	Fecha	Acciones
Tarea Resultado	No	----	1/09/2014	<a data-bbox="879 792 919 813">Ver</a> <a data-bbox="943 792 999 813">Editar</a> <a data-bbox="1023 792 1114 813">Eliminar</a>
Prueba 1	Si	3		<a data-bbox="879 846 919 866">Ver</a> <a data-bbox="943 846 999 866">Editar</a> <a data-bbox="1023 846 1114 866">Eliminar</a>
Prueba 2	No	----		<a data-bbox="879 900 919 920">Ver</a> <a data-bbox="943 900 999 920">Editar</a> <a data-bbox="1023 900 1114 920">Eliminar</a>

### Listas

Nueva Lista

Editar Lista

Eliminar

Ver Listas

### Tareas

Ver Tareas

Nueva Tarea

Ilustración 21: Nueva tarea en la Lista

## 7. Conclusiones

---

Tras el desarrollo del proyecto y la elaboración de esta memoria, las conclusiones obtenidas que se destacan son las siguientes:

- Se han cumplido el objetivo primario que se estableció al principio del proyecto, la creación de la API REST completamente funcional, así como el objetivo secundario, que pretendía desarrollar la interfaz web.
- El proyecto desarrollado permite que se de continuidad a la implementación y se puedan realizar mejoras posteriores.
- Todo el software utilizado en este desarrollo es de código libre, por lo que la implementación se puede repetir de manera gratuita.
- La plataforma web desarrollada resulta accesible desde cualquier navegador, la interfaz es sencilla y de diseño simple. Lo que hace que sea fácil de utilizar y entender.
- El objetivo de escalabilidad ha sido estudiado durante todo el desarrollo, y se ha cumplido que sea escalable, aunque se puede mejorar la implementación para que tenga mayor escalabilidad.
- Como punto final, se ha aprendido a trabajar con APIs para obtener los servicios que proporcionan. Y se ha asentado el conocimiento previo en Java, PHP y en HTML.

## 8. Bibliografía

---

*Eclipse (2.1) y Java- Departamento de Informatica-Universidad de Valencia.* (2004).  
Obtenido de [http://www.uv.es/~jgutier/MySQL\\_Java/TutorialEclipse.pdf](http://www.uv.es/~jgutier/MySQL_Java/TutorialEclipse.pdf).

*INTECO- Instituto Nacional de Tecnologías de Comunicación.* (20 de Mayo de 2014).  
Obtenido de  
[http://www.inteco.es/blogs/post/Seguridad/BlogSeguridad/Articulo\\_y\\_comentarios/Seguridad\\_OAuth\\_2\\_0](http://www.inteco.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/Seguridad_OAuth_2_0).

Louvel, J., Templier, T., & Bolileau, T. (2013). *Restlet in Action*. Shelter Island, NY: Manning Publications Co.

Marqués, A. (11 de Abril de 2013). *Asier Marqués*. Obtenido de  
<http://asiermarques.com/2013/conceptos-sobre-apis-rest/>.

No Two The Same. (2013). *No Two The Same*. Obtenido de  
<http://notwothesame.com/articles/2013-webstats/>.

Richardson, L., & Ruby, S. (2007). *RESTful Web Services*. Sebastopol, CA: O'Reilly.

Rodríguez, A. E. (13 de Octubre de 2013). *geekytheory.com/*. Obtenido de  
<http://geekytheory.com/json-i-que-es-y-para-que-sirve-json/>.

Salgado, D. (20 de Agosto de 2008). *\\Brain\Backup*. Obtenido de  
<http://blogs.msdn.com/b/davidsalgado/archive/2008/08/20/las-dichosas-cross-domain-calls.aspx>.

Sandoval, J. (2009). *RESTful Java Web Services*. Birmingham: Packt Publishing.

*Wikipedia - Domótica.* (26 de Agosto de 2014). Obtenido de  
<http://es.wikipedia.org/wiki/Dom%C3%B3tica>

*Wikipedia - XAMPP.* (20 de Agosto de 2014). Obtenido de  
<http://es.wikipedia.org/wiki/XAMPP>.