

TEMA 8

CONTENIDO

1.- Reutilización de código e información.	2
2.- Características de las aplicaciones web híbridas.	4
3.- Utilización de repositorios de información.	5
3.1.- OAuth2.	6
3.2.- JSON y XML.	9
4.- Creación de aplicaciones web híbridas.	11
4.1.- Yahoo! PlaceFinder.	12
4.1.1- Yahoo! PlaceFinder (II).	13
4.2.- Google Geocoding.	14
4.3.- Aplicación web híbrida de geocodificación.	16
form.php	16
ubicar.js	18
estilos.css	19
4.4.- Google Directions.	20
4.5.- Google Tasks.	21
4.5.1- Google Tasks (II).	22
4.6.- Aplicación web híbrida de gestión de repartos.	23
4.6.1.- Aplicación web híbrida de gestión de repartos (II).	24
4.6.2.- Aplicación web híbrida de gestión de repartos (III).	24
4.6.3.- Aplicación web híbrida de gestión de repartos (IV).	25
repartos.php	26
ajaxmaps.php	29
código.js	30
estilos.css	32

Aplicaciones web híbridas.

Caso práctico

Después de semanas de trabajo, **Carlos** da por finalizado el desarrollo de la aplicación en la que ha estado trabajando. Ha tenido que reescribir el código en varias ocasiones, cambiar la apariencia de algunas pantallas del interfaz, y retocar el esquema de la base de datos que habían diseñado en un principio, pero finalmente parece que el trabajo ha dado sus frutos.

Habla con **Juan** y entre los dos revisan el resultado. **Juan** está muy contento con lo que ve, y le propone hablar con **Ada**, para mostrarle la aplicación web. Aunque la directora ha seguido el progreso de la misma, en gran parte ha sido en las conversaciones que ha mantenido con **Juan**, y hace ya tiempo que no le informan directamente sobre los últimos avances.

1.- Reutilización de código e información.

Caso práctico

Ada revisa la aplicación web y les propone retocar un par de detalles, sobre todo relativos a la apariencia. Con los años de experiencia que tiene en el desarrollo de aplicaciones, les ofrece también algunos consejos sobre la usabilidad de los interfaces, y le indica que es muy importante seguir algunas normas básicas que garanticen la accesibilidad de la aplicación.

Por último, tienen que hablar con su amigo **Esteban** para concretar los detalles de la implantación de la aplicación en las dependencias del cliente. Parece que están a punto de finalizar el proyecto.

La unidad 6 de este módulo tenía por título "Servicios Web". En ella aprendiste a crear y utilizar servicios web, empleando una arquitectura orientada a servicios (SOA). Los principales temas que practicaste en esa unidad son:

- ✓ A utilizar el protocolo SOAP para comunicarte con un servicio web. Con ayuda de la clase `SoapClient`, intercambiabas peticiones y respuestas en formato SOAP con un servicio web existente.
- ✓ A crear tu propio servicio web. Mediante la clase `SoapServer` podías publicar tus propias funciones para que fueran accesibles como servicio web mediante SOAP.
- ✓ A procesar los documentos WSDL de descripción de los servicios web, y a crear los documentos WSDL de descripción de tus propios servicios.

Los servicios web permiten a tus aplicaciones comunicarse con otras utilizando la web (*el protocolo HTTP*) como medio de transmisión. Sin embargo, los mecanismos que has utilizado hasta ahora no son la única forma de implementar y utilizar servicios web. Desde hace un tiempo han ido apareciendo servicios web que utilizan arquitecturas basadas en **REST**.

REST hace referencia a un conjunto de normas que se pueden utilizar para establecer mecanismos de comunicación con servicios web. Concretamente, un servicio web implementado mediante REST debería al menos:

- ✓ Utilizar una estructura de URIs para acceder a los recursos gestionables mediante el servicio web:

```
/articulo/KSTDTG332GBR
/tienda/CENTRAL
```

- ✓ Usar los distintos métodos HTTP (*El protocolo HTTP 1.1 define 9 métodos que puede usar el cliente para identificar el tipo de acción que desea realizar. El método más conocido y utilizado es GET, que indica que se quiere obtener una representación de un recurso del servidor. Los 8 métodos restantes son HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT y PATCH*) para las peticiones. Por ejemplo, se podría utilizar el método HTTP `GET` para obtener información de un artículo:

```
GET /articulo/KSTDTG332GBR
```

Y el método HTTP `DELETE` para borrarlo:

```
DELETE /articulo/KSTDTG332GBR
```

- ✓ No almacenar información sobre el estado: todas las peticiones se tratarán de forma independiente y deben incluir toda la información necesaria para poder atenderla.

- ✓ Utilizar XML o JSON en sus comunicaciones (o incluso ambos).

REST no es un estándar, pero muchos servicios web actuales se implementan utilizando arquitecturas de tipo REST. Existen en Internet varios documentos sobre REST y ejemplos de su utilización que conviene revisar.

REST y Servicios Web.

<http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>

Serie de artículos sobre REST.

<http://eamodeorubio.wordpress.com/category/webservices/>

En la presente unidad crearás aplicaciones que utilicen diversos servicios web.

2.- Características de las aplicaciones web híbridas.

Caso práctico

En poco tiempo, **Carlos** modifica los detalles que había apreciado **Ada**. Está contento con el resultado obtenido, pero con la experiencia que ha adquirido en programación web durante su desarrollo, sabe que habría algunos detalles que se podrían añadir a la aplicación y que influirían positivamente en la experiencia del usuario.

Ada le llama y le comenta que la próxima semana **Esteban** vendrá a BK Programación a conocer la aplicación. **Carlos** guarda una copia de la aplicación en su estado actual, y decide tomarse esa semana para intentar mejorarla en algunos aspectos. Hace una lista de los detalles que le podría añadir, y al revisarla se da cuenta de que otras aplicaciones que conoce ya los implementan; pero en muchos casos, no son desarrollos propios sino que se basan en servicios ofrecidos por terceros: mapas de Google, imágenes de Flickr, ... ¿Podrá aprovechar algún servicio existente e integrarlo en su propia aplicación?

Una aplicación web híbrida, también conocida por su nombre en inglés (*mashup*), se caracteriza por combinar datos y/o funcionalidades procedentes de diversos orígenes para formar un nuevo tipo de aplicación o servicio.

Los tipos de fuentes de información más habituales que se utilizan en una aplicación web híbrida son:

- ✓ Información proveniente de servicios web, disponible mediante diversos protocolos y estructurada utilizando formatos de intercambio como JSON o XML.

En ocasiones el proveedor del servicio ofrece también un interface de programación (API) para facilitar el acceso a los datos. Es el caso de las API de compañías como Google, Yahoo!, Flickr, Microsoft o Amazon.

En otras ocasiones los datos se ofrecen de forma pública utilizando protocolos de redifusión web (*también conocido como sindicación web es una manera sencilla de poner ciertos contenidos de un sitio web a disposición de otros*) como **RSS** o **Atom**, y puede ser necesario procesarlos para extraer la información necesaria.

Redifusión web.

http://es.wikipedia.org/wiki/Redifusi%C3%B3n_web

- ✓ Información generada y gestionada por el propietario de la aplicación web híbrida, como pueden ser datos internos de una empresa.

De forma menos habitual, podemos encontrarnos aplicaciones web que utilicen técnicas de ingeniería inversa para extraer los datos que se muestran en algunos sitios web, como puede ser el caso de los precios de los productos en las tiendas web. Estas técnicas se conocen por su nombre en inglés: **web scraping**.

Por ejemplo, podrías montar una aplicación web híbrida que utilice la API de Google Maps, e información de ubicación geográfica de las franquicias de una empresa para mostrar la localización de las tiendas en un mapa.

En esta unidad, vas a programar una aplicación web híbrida para la tienda web con la que has venido trabajando. En este caso se trata de facilitar la gestión de los envíos de las compras.

Las siglas REST hacen referencia a un estándar que se utiliza en la implementación de servicios web.



Verdadero.



Falso.

Aunque el enunciado pueda parecer correcto, REST no es un estándar sino un conjunto de normas.

3.- Utilización de repositorios de información.

Caso práctico

Carlos se informa sobre los servicios web que pueden serle útiles, y decide añadir a la aplicación una funcionalidad que no tiene: un servicio de gestión de envíos para los productos que se vendan. En realidad nadie ha solicitado esa funcionalidad, pero cree que si es capaz de programarla la empresa de **Esteban** la llegará a utilizar. En ocasiones los clientes no piden una característica simplemente porque desconocen que es posible realizarla. Y cuando comenzó este proyecto en particular, no había nadie en BK Programación con la experiencia suficiente como para orientar correctamente al cliente.

Se pone manos a la obra. Si en la semana que tiene le da tiempo a finalizarla, se la mostrará a **Esteban**. Y si no le da tiempo, echará mano de la versión anterior. De cualquier modo, no es tiempo perdido. Este proyecto le está sirviendo para adquirir experiencia que a buen seguro aprovechará en el futuro inmediato.

Cuando utilices servicios de terceros para desarrollar una aplicación web híbrida, deberás tener en cuenta que en ocasiones existen condiciones y límites al uso que puedes hacer del mismo.

La mayoría de las grandes compañías que proveen servicios web al usuario, como Google o Yahoo!, requieren un registro previo y ofrecen unas condiciones para su uso gratuito que dependen del servicio al que necesites acceder. Algunos de estos servicios ofrecen una versión adicional de pago con mejores condiciones.

Por ejemplo, actualmente Google ofrece los siguientes límites de acceso gratuito para los siguientes servicios:

- ✓ **Google Tasks:**5.000 consultas diarias.
- ✓ **Google Maps:**25.000 consultas diarias.
- ✓ **Google Custom Search:**100 consultas diarias.

En muchas ocasiones, el proveedor del servicio (aunque también puede ser un tercero) ofrece además librerías que facilitan la utilización del servicio desde un lenguaje de programación determinado y ejemplos de utilización del mismo. Por ejemplo, si quieres utilizar la API de *Google Tasks* existen librerías de programación para los lenguajes Java, Python, PHP y para la plataforma .Net de Microsoft.

Para que se pueda verificar la utilización que hace cada usuario de un servicio determinado, es necesario incluir dentro del código que interactúa con el mismo una clave personal que le identifica en el sistema. Por ejemplo, si quieres utilizar la API de *Google Books*, necesitarás indicar tu código de desarrollador al hacer una consulta de forma similar a:

```
$client->setDeveloperKey('la clave de desarrollador va aquí');
```

De la misma forma, si quieres acceder al servicio de geocodificación (*Asignación de coordenadas geográficas como latitud / longitud a una dirección*) *PlaceFinder* de Yahoo!, tendrás que indicar en la consulta un identificador de aplicación.

Existen ciertos servicios web que nos permiten acceder a información privada que almacenan de sus usuarios. Por ejemplo, la API de *Google Calendar* posibilita gestionar la información que cada usuario mantiene en sus calendarios personales. Si vas a usar un servicio de este tipo (como *Google Tasks*), necesitarás que tu aplicación solicite permiso a los propietarios de la información antes de poder acceder a la misma. Para ello muchos proveedores de servicios web utilizan un protocolo llamado **OAuth** (la versión actual es la 2.0).



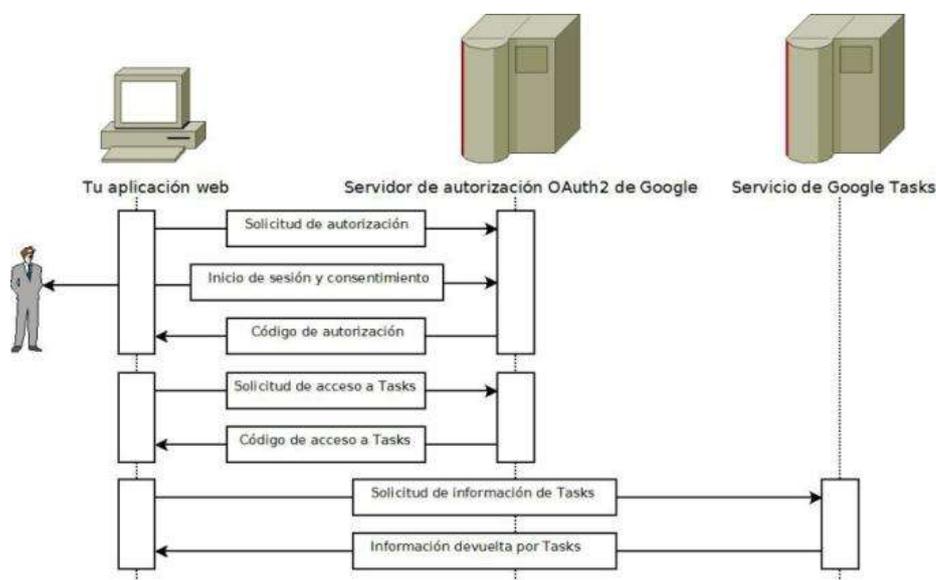
3.1.- OAuth2.

El protocolo estándar de autorización **OAuth2**, permite a una aplicación externa obtener acceso a información de carácter privado a través de un servicio web. Para ello establece un acuerdo de acceso a la misma entre la aplicación externa, el servicio web y el propietario de los datos a los que se solicita el acceso.

Por ejemplo, si una aplicación "X" solicita a Google acceso a los calendarios del usuario "dwes", Google pedirá a "dwes" permiso indicándole qué aplicación es la que solicita el acceso y a qué información. Si el usuario "dwes" otorga permiso, la aplicación "X" podrá acceder a los datos que solicitó a través del servicio de Google.

En uno de los videotutoriales anteriores, en el que creaste los identificadores necesarios para utilizar las API de Yahoo!, vimos un ejemplo de utilización de OAuth. Al utilizar una cuenta de Google para registrarnos como desarrolladores, Yahoo! necesitaba utilizar los servicios de Google para acceder a parte de nuestra información. Para ello, remitió una solicitud a Google, y éste a su vez nos mostró una pantalla informando de que Yahoo! solicitaba información nuestra y pidiéndonos permiso para darle acceso.

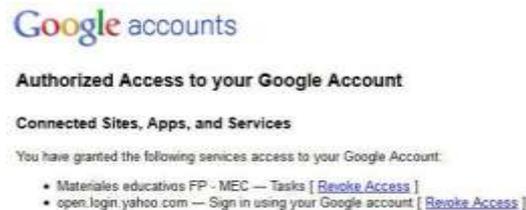
OAuth2 funciona de forma similar pero ligeramente distinta dependiendo de quién solicite acceso a la información. En nuestro caso supondremos que el solicitante será siempre una aplicación web. Veamos por ejemplo qué sucede cuando nuestra aplicación necesita acceder a información personal del usuario a través del servicio de Google Tasks. En este caso, los pasos que se seguirán son los siguientes:



- ✓ La aplicación web se comunica con el servidor de autorización OAuth2, indicando la información a que quiere acceder y el tipo de acceso a la misma.
- ✓ El servidor de autorización OAuth2 requiere al usuario de la aplicación web a que inicie sesión con su cuenta de Google (si aún no lo ha hecho), y le redirige a una página en la que le pide su consentimiento para otorgar acceso a su información privada.



- ✓ Si el usuario da su consentimiento, el servidor de autorización OAuth2 devuelve a la aplicación web un código de autorización.
- ✓ Antes de poder acceder a la información privada del usuario, la aplicación web debe intercambiar ese código de autorización por otro código de acceso.
- ✓ Utilizando el código de acceso, la aplicación puede utilizar el servicio de Google Tasks para gestionar la información privada del usuario, dentro de los límites de acceso que se han otorgado.
- ✓ Los códigos de acceso tienen un tiempo de vida limitado. Cuando caducan, la aplicación ha de comunicarse de nuevo con el servidor de autorización OAuth2 para obtener un código de refresco.



http://www.youtube.com/watch?feature=player_embedded&v=ssrhy1pAlak

Utilizar OAuth2 con Google

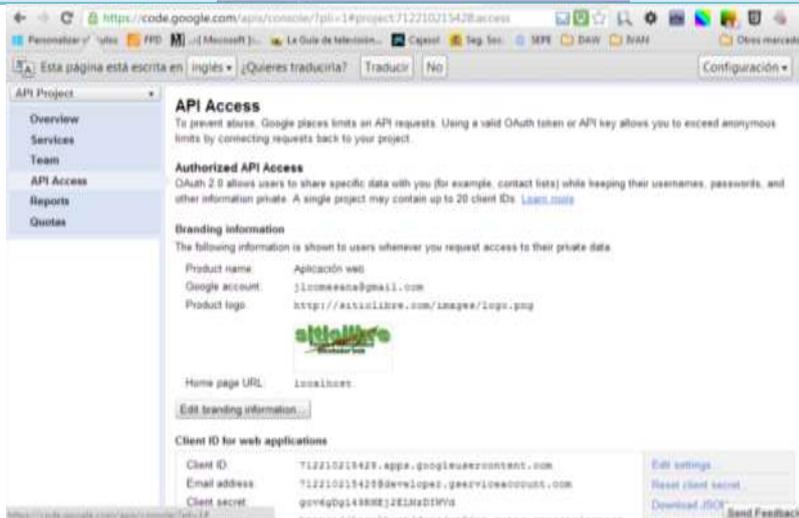
Iniciamos sesión en Google con nuestra cuenta de gmail y accedemos a <http://code.google.com/apis/console>

Seleccionamos la opción API Access y creamos un identificador de cliente OAuth2 para nuestra aplicación web

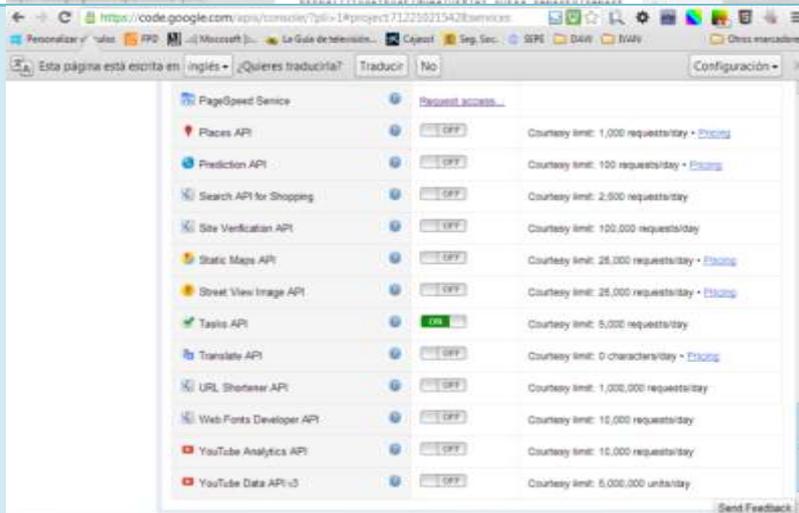
Rellenamos la información relativa a nuestra aplicación y pulsamos sobre Create Client ID.



Es importante ajustar correctamente las URIs de redirección, porque es a dónde podrá dirigirse el navegador tras el consentimiento del usuario. Siempre podremos cambiar los datos pulsando sobre *Edit setting*



Uno de los servicios que requiere OAuth2 es Google Tasks. Puedes activarlo para crear una aplicación que lo utilice, para lo que seleccionaremos la opción *Services* del menú de Google Apis, y pulsaremos sobre el botón OFF para activarlo, lo cual tiene efecto tras aceptar los términos de uso. Y ya podrás crear tu aplicación web utilizando OAuth2 con los servicios de Google Task



Existe una extensión PHP para programar las partes cliente y servidor del protocolo OAuth. **Extensión OAuth.** <http://es.php.net/manual/es/book.oauth.php>

Al utilizar OAuth2, cuando tu aplicación solicite información privada de un usuario, deberá acreditar su autorización utilizando:



Un código de autorización.



Un código de acceso.

Antes de acceder a información privada de un usuario, tu aplicación deberá estar en posesión de un código de autorización, pero tendrá que utilizarlo para conseguir un código de acceso, que será el que finalmente utilice

3.2.- JSON y XML.

Muchas de las operaciones que se llevan a cabo cuando utilizas un servicio web implican la obtención de cierta información por parte del mismo. La información obtenida puede ser bastante sencilla o tener cierto grado de complejidad. Por ejemplo, cuando utilizas un servicio de geocodificación para averiguar las coordenadas concretas de una dirección, obtienes básicamente esas coordenadas. Pero cuando utilizas un servicio como **Google Directions** para averiguar la ruta a seguir entre dos puntos, la respuesta que obtienes es una ruta compuesta por una serie de indicaciones a seguir para llegar al destino.

Los formatos más utilizados por los servicios web para dar formato a esas respuestas son dos: **JSON** y **XML**. En algunos casos tendrás que adaptar tu código al tipo de respuesta que ofrezca el servicio. Otros servicios permiten que escojas el tipo de respuesta que prefieras. Veamos cómo se pueden procesar de forma sencilla desde PHP mensajes en ambos formatos.

A partir de la versión 5.2 de PHP, se incluye por defecto la extensión JSON. Su funcionamiento es muy sencillo. Incorpora dos funciones para tratar con cadenas de texto en notación JSON:

Extensión JSON. <http://es.php.net/manual/es/book.json.php>

- ✓ **json_decode.** Decodifica una cadena de texto JSON y la transforma en un objeto PHP (opcionalmente también se podría convertir en un array).

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json));
var_dump(json_decode($json, true));
?>
```

Salida:

```
object(stdClass) [1]
  public 'a' => int 1
  public 'b' => int 2
  public 'c' => int 3
  public 'd' => int 4
  public 'e' => int 5
array
  'a' => int 1
  'b' => int 2
  'c' => int 3
  'd' => int 4
  'e' => int 5
```

- ✓ **json_encode.** Función inversa a la anterior. Devuelve una cadena de texto en notación JSON a partir del contenido de una variable PHP.

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
echo json_encode($arr);
?>
```

Salida:

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

Así como las opciones para trabajar con JSON desde PHP están bien definidas, para utilizar XML hay más variedad de herramientas para escoger. En PHP4 podías utilizar dos formas para procesar documentos en formato XML: **DOM** y **SAX**. La extensión **SimpleXML**, habilitada por defecto a partir de PHP 5.1.2, facilita la tarea de extraer información de un documento XML. Vamos a ver su funcionamiento.

Extensión SimpleXML. <http://es.php.net/manual/es/book.simplexml.php>

La extensión convierte un documento XML en un objeto de la clase **SimpleXMLElement**. Puedes cargar el documento:

- ✓ desde una cadena de texto, utilizando la función **simple_load_string**.

```
$xml = simplexml_load_string($cadena);
```

- ✓ desde un fichero, utilizando la función `simplexml_load_file`. Puedes utilizar una dirección URI como origen, por ejemplo:

```
$xml = simplexml_load_file('http://localhost/dwes/ut8/config.xml');
```

Los nodos y atributos del documento XML se convierten en atributos. Los nodos pasan a ser arrays que contienen a su vez como elementos los atributos y subnodos del documento XML. Por ejemplo, para acceder al resumen (`summary`) de una ruta en formato XML obtenida a partir del servicio **Google Directions**, podrías utilizar:

```
$ruta = simplexml_load_file('ruta al servicio');  
$ruta->route[0]->summary
```

Ruta de Google Directions.

<http://code.google.com/intl/es-ES/apis/maps/documentation/directions/#XML>

Es importante que conozcas cómo procesar documentos XML para extraer información de los mismos. En el manual de PHP puedes encontrar información sobre la utilización de la extensión SimpleXML.

Extensión SimpleXML.

<http://www.php.net/manual/es/book.simplexml.php>

4.- Creación de aplicaciones web híbridas.

Caso práctico

Pasa la semana y **Carlos** consigue tener a punto la nueva versión de la aplicación. Para la presentación, pone las dos versiones en funcionamiento, y no le comenta a nadie los nuevos cambios. Si hay algún problema con las últimas modificaciones, echará mano de la versión anterior. Cuando en BK Programación muestran a **Esteban** la aplicación, éste queda asombrado por el resultado. El nuevo servicio de gestión de envíos sorprende a todos positivamente, especialmente a **Ada** y a **Juan**, que empiezan a darse cuenta de las nuevas capacidades que ha adquirido **Carlos** en las últimas semanas. **Esteban** comenta que muy posiblemente les sea útil, especialmente al poder consultar la información de los envíos desde un dispositivo móvil. Pasarán un par de meses probando la aplicación, y a continuación plantea la posibilidad de tener una nueva reunión para hablar de posibilidades de ampliación y de otros proyectos. Ha quedado muy contento con el trabajo de **Carlos** y quiere seguir contando con él próximamente.

Para crear aplicaciones web híbridas, necesitarás que tu aplicación web acceda a diversos servicios para obtener información. En muchas ocasiones esos servicios estarán accesibles mediante HTTP. En otras ocasiones, especialmente cuando accedas a información privada de un usuario, necesitarás utilizar un protocolo seguro como HTTPS.

Tanto si vas a generar tu propio código para acceder a un servicio web, como si utilizas una API ya programada, es posible que necesites utilizar la librería **cURL**.

Manual de PHP.

<http://es.php.net/manual/es/book.curl.php>

cURL es una librería (y también una herramienta para utilizar desde la línea de comandos), que permite utilizar de forma sencilla varios protocolos de comunicaciones para transmitir información. Soporta, entre otros, los protocolos HTTP, HTTPS, FTP, TFTP, TELNET, LDAP, SMTP, POP3 e IMAP. Es importante que te asegures de que tu instalación de PHP incluye dicha librería. Si no la tienes activada en tu instalación de PHP, en Ubuntu puedes instalarla ejecutando:

```
sudo apt-get install php5-curl"
```

```
smr@ubuntu-profe:~$ sudo apt-get install php5-curl
[sudo] password for smr:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  libapache2-mod-php5 php5-cli php5-common php5-gd php5-mysql
Paquetes sugeridos:
  php5-subsin
Se instalarán los siguientes paquetes NUEVOS:
  php5-curl
Se actualizarán los siguientes paquetes:
  libapache2-mod-php5 php5-cli php5-common php5-gd php5-mysql
5 actualizados, 1 se instalarán, 8 para eliminar y 284 no actualizados.
Necesito descargar 6325kB de archivos.
Se utilizarán 119kB de espacio de disco adicional después de esta operación.
¿Desea continuar [S/n]?
```

Recuerda reiniciar apache tras la instalación, y comprueba que la ejecución de `phpinfo()` muestra algo como lo siguiente.

curl	
CURL support	enabled
CURL Information	7.13.1
Age	1
Features	
AsynchDNS	no
Debug	no
GSS-Negotiate	no
IDN	no
IPv6	no
Language	no
NTLM	no
SPNEGO	no
SSL	no
SSPI	no
libSASL	no
libz	no
WinCache	no
Protocols	dict, ftp, ftps, http, https, imap, imaps, irc, ircs, rtsp
Proxy	no
SSL Version	OpenSSL 1.0.1
SSL Module	no
SSL Version	1.1.2.1

En Windows la instalación necesita que añadas otras dos librerías:

`libeay32.dll` y `ssleay32.dll`.

En el manual de PHP y en otras páginas web tienes más información sobre la instalación de cURL en este sistema operativo.

Instalación de cURL en Windows.

<http://dy3g0.wordpress.com/2008/12/09/activar-curl-en-windows/>

Aunque la instalación y utilización de cURL en sistemas Linux/Ubuntu es inmediata, no sucede lo mismo con los sistemas Windows. En Windows la librería cURL se incluye en la instalación estándar de PHP, y no incluye una lista de autoridades de certificación.

Esta lista de autoridades de certificación es imprescindible para establecer conexiones seguras utilizando el protocolo HTTPS. Los certificados de los sitios web seguros deben estar firmados por una autoridad de certificación confiable, y su firma se debe poder comprobar. Si no tenemos una lista de autoridades de certificación confiables, no se podrá realizar esta comprobación y obtendremos un mensaje como:

```
CURL Error 60: SSL certificate problem, verify that the CA cert is OK.
```

Para solventar este problema, desde la web de cURL puedes descargar la lista adaptada de autoridades de certificación que incluye Mozilla en su navegador Firefox. Al utilizar la librería cURL desde tu código, tendrás que indicarle que utilice esa lista de autoridades de certificación, añadiendo una línea como la siguiente:

```
curl_setopt($ch, CURLOPT_CAINFO, 'ruta a la lista');
```

Lista de autoridades de certificación.

<http://curl.haxx.se/docs/caextract.html>

Si utilizas Windows para los ejemplos de aplicaciones web híbridas de este tema, tendrás que modificar el código de la API de Google para no tener problemas de certificados con cURL.

La librería cURL debe utilizar una lista válida de autoridades de certificación:



Para poder acceder a servidores web utilizando HTTPS.



Para poder acceder a servidores web utilizando HTTP.

Para poder certificar la validez del certificado digital que envía el servidor cuando se utiliza HTTPS, cURL necesita una lista válida de autoridades de certificación

4.1.- Yahoo! PlaceFinder.

Existen muchas fuentes de datos disponibles en Internet que puedes utilizar para construir una aplicación web híbrida. En esta unidad vamos a centrarnos en la información accesible a través de servicios web, concretamente en los que ofrecen las empresas Google y Yahoo!.



Comenzaremos creando una aplicación web híbrida sencilla que haga uso de los servicios de geocodificación que ofrecen ambos. El de Yahoo! se llama PlaceFinder. Puedes encontrar toda la información necesaria sobre su utilización en la web de desarrollo de Yahoo!.

Yahoo! PlaceFinder.

<http://developer.yahoo.com/geo/placefinder/>

El servicio **Yahoo! PlaceFinder** se basa en REST y los datos relativos a la petición se envían mediante parámetros **GET**. Es necesario indicar como mínimo un parámetro de localización.

La forma más sencilla de indicar una localización es utilizando el parámetro `location` (o su equivalente `q`).

Existe otro tipo de parámetros, los de control, que permiten indicar otra información no directamente relacionada con la localización. Es obligatorio el parámetro `appid` para indicar el ID de tu aplicación web. Otros parámetros de control son:

Parámetro de control	Significado
appid	ID de la aplicación que utiliza el servicio. Es obligatorio.
locale	Código del lenguaje y país. Por defecto "en_US". En nuestro caso deberíamos utilizar "es_ES".
count	Número máximo de respuestas que se devolverán. Por defecto 10.
flags	Cadena de caracteres que especifica qué datos se obtendrán. Por ejemplo: J – Indica que la información se devuelva en formato JSON (por defecto se utiliza XML). P – Indica que la información se devuelva en formato PHP serializado. G – Devuelve información global, no específica de los Estados Unidos.
gflags	Cadena de caracteres que especifica cómo se realizará la búsqueda. Por ejemplo: L – Buscar sólo en el país que se indica. R – Realizar una búsqueda inversa: obtener la dirección a partir de una latitud y longitud.

En la documentación del servicio tienes una lista completa de los parámetros de localización y de control que puedes emplear.

[Parámetros de localización.](#)

<http://developer.yahoo.com/geo/placefinder/guide/requests.html>

Por ejemplo, una petición simple podría tener la siguiente forma:

http://where.yahooapis.com/geocode?location=Plaza+de+la+Peregrina,Pontevedra,Spain&locale=es_ES&flags=G&count=1&appid=tuID

Recuerda incluir en las peticiones a los servicios de Google y Yahoo!, tus propios identificadores (los que te han asignado al registrarte) allí donde sea necesario.

4.1.1- Yahoo! PlaceFinder (II).

Las respuestas obtenidas al utilizar el servicio, contienen los siguientes elementos (indiferentemente de si se indica XML, JSON o PHP serializado):



Elemento	Significado
ResultSet	Elemento de nivel superior que contiene al resto.
Error	Código de error (0 si no se ha producido ningún error). En la documentación del servicio se incluye una tabla con los posibles códigos de error y su significado.
ErrorMessage	Mensaje de error. "No error" o "Sin errores" si no se ha producido ninguno.
Locale	Código del lenguaje utilizado en la respuesta.
Quality	Estimación de la calidad de la respuesta obtenida.
Found	Número de resultado obtenidos.
Result	Cada uno de los resultados obtenidos.

En la documentación del servicio tienes toda la información sobre los elementos que conforman los resultados obtenidos.

[Formato de las respuestas de PlaceFinder.](#)

<http://developer.yahoo.com/geo/placefinder/guide/responses.html>

Por ejemplo, como respuesta a la petición anterior podíamos obtener el siguiente documento en formato XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet version="1.0">
  <Error>0</Error>
  <ErrorMessage>Sin Errores</ErrorMessage>
  <Locale>es_ES</Locale>
  <Quality>87</Quality>
  <Found>1</Found>
  <Result>
    <quality>72</quality>
    <latitude>42.430283</latitude>
    <longitude>-8.643625</longitude>
    ...
  </Result>
</ResultSet>
```

Si quisieras efectuar una consulta inversa (obtener una dirección a partir de unas coordenadas de latitud y longitud), podrías realizar una consulta utilizando `gflags=R`, como:

```
http://where.yahooapis.com/geocode?location=42.430283,-8.643625&flags=G&locale=es_ES&gflags=R&appid=tuID
```

Y obtendrías:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet version="1.0">
  <Error>0</Error>
  <ErrorMessage>Sin errores</ErrorMessage>
  <Locale>es_ES</Locale>
  <Quality>99</Quality>
  <Found>1</Found>
  <Result>
    <quality>99</quality>
    <latitude>42.430283</latitude>
    <longitude>-8.643625</longitude>
    ...
    <line1>praza da Peregrina, 9</line1>
    <line2>36001 Pontevedra</line2>
    <line3></line3>
    <line4>España</line4>
    ...
  </Result>
</ResultSet>
```

Relaciona las palabras con su significado relativo al servicio Yahoo! PlaceFinder:

Palabra	Relación	Significado
<code>appid</code>	3	1. Número de resultados que incluye la respuesta.
<code>Found</code>	1	2. Parámetro que se utiliza en una petición para indicar el número máximo de respuestas que se devolverán.
<code>count</code>	2	3. Parámetro que se utiliza en una petición para indicar el identificador de la aplicación.
<code>Quality</code>	4	4. Estimación de la calidad de la respuesta obtenida.

En la pregunta figuran tanto parámetros que se utilizan al hacer las peticiones, como elementos que se incluyen en las respuestas.

4.2.- Google Geocoding.

El servicio de Google Geocoding, que forma parte de los servicios web de Google Maps, es muy similar a PlaceFinder de Yahoo!. También se basa en peticiones REST, indicando a continuación el tipo de respuesta que queremos obtener:



Google Geocoding.

<https://developers.google.com/maps/documentation/geocoding/>

Google Geocoding respuestas en formato JSON (recomendado por Google).

<http://maps.google.com/maps/api/geocode/json>

Google Geocoding para respuestas en formato XML

<http://maps.google.com/maps/api/geocode/xml>

Como parte de la petición, puedes utilizar entre otros los siguientes parámetros:

Parámetro	Significado
address	Dirección de la que quieres obtener las coordenadas. Es obligatorio, salvo para peticiones inversas.
latlng	Coordenadas a partir de las cuales quieres obtener una dirección. Es obligatorio solo para peticiones inversas.
language	Idioma en el que se devuelven los resultados. Nosotros utilizaremos "es". Es opcional.
sensor	Indica si la solicitud proviene de un dispositivo con sensor de localización. Sus posibles valores son true y false. Es obligatorio.

Para usar el servicio Google Geocoding no necesitas indicar tu ID de desarrollador registrado en Google. Tienes más información sobre el mismo en su página de documentación.

Documentación de Google Geocoding.

<http://code.google.com/intl/es-ES/apis/maps/documentation/geocoding/>

Así, por ejemplo, las solicitudes siguientes son equivalentes a las que realizaste anteriormente al servicio Yahoo! PlaceFinder, pero devolviendo información en formato JSON en lugar de XML:

```
http://maps.google.com/maps/api/geocode/json?address=plaza+de+la+peregrina,pontevedra,spain&language=es&sensor=false
http://maps.google.com/maps/api/geocode/json?latlng=42.430283,-8.643625&language=es&sensor=false
```

La respuesta obtenida a la primera petición es:

```
{
  "results" : [
    {
      "address_components" : [
        ...
      ],
      "formatted_address" : "Plaza de la Peregrina, 36001 Pontevedra, España",
      "geometry" : {
        ...
        "location" : {
          "lat" : 42.43080090,
          "lng" : -8.644160399999999
        },
        ...
      },
      "types" : [ "route" ]
    }
  ],
  "status" : "OK"
}
```

En este caso, la información que nos interesa es la que contiene el elemento `location`. En la petición inversa, para obtener la dirección podremos usar los elementos `formatted_address` o `address_components`, que nos ofrecen la misma información de forma compacta o desglosada respectivamente.

De nuevo puedes recurrir a la documentación del servicio para obtener información sobre cada uno de los elementos que forman la respuesta.

El servicio Google Geocoding devuelve la información en un formato u otro:



Dependiendo de la URL que se utilice en la petición.



En función de un parámetro GET que se debe utilizar en la petición.

La parte final de la URL puede ser json o xml en función del formato en que necesites la respuesta.

4.3.- Aplicación web híbrida de geocodificación.

Vas a crear una aplicación sencilla que utilice los dos servicios web que acabas de ver de Google y Yahoo!. Se trata simplemente de ver cómo se pueden utilizar desde PHP. Para ello crearemos un interfaz como el siguiente:

El formulario está dividido en dos zonas. En la superior, el usuario podrá introducir unas coordenadas y en la inferior los datos de una dirección. Se trata de utilizar los dos botones del formulario para realizar consultas a ambos servicios de geocodificación, de tal forma que:

- ✓ Al utilizar el botón "Ver dirección con Yahoo!", se realiza una consulta inversa al servicio **PlaceFinder** y se cubren los datos de la zona inferior del formulario con la respuesta obtenida.
- ✓ Al pulsar sobre el botón "Ver coordenadas con Google" se lanza una petición a **Google Geocoding** y se cubren las coordenadas de la parte superior del formulario con las que se reciben.

Para realizar las llamadas mediante Ajax utilizarás la librería Xajax, vista en la unidad anterior. La función que se encarga de realizar la llamada a Yahoo! PlaceFinder incluye el siguiente código:

```
$respuesta = new xajaxResponse();
$search =
'http://where.yahooapis.com/geocode?location='.$coordenadas['latitud'].'+ '.$coordenadas['longitud'].'&flags=G&locale=es_ES&gflags=R&appid=tuID';

$xml = simplexml_load_file($search);
$respuesta->assign("calle", "value", (string) $xml->Result[0]->street . " " . $xml->Result[0]->house);
...
return $respuesta;
```

Y la que utiliza Google Geocoding para obtener las coordenadas de una dirección (también en formato XML):

```
$respuesta = new xajaxResponse();
$search =
'http://maps.google.com/maps/api/geocode/xml?address='.$coordenadas['calle'].'+ '.$coordenadas['ciudad'].'+ '.$coordenadas['pais'].'&language=es&sensor=false';
$xml = simplexml_load_file($search);

$respuesta->assign("latitud", "value", (string) $xml->result[0]->geometry->location->lat);
$respuesta->assign("longitud", "value", (string) $xml->result[0]->geometry->location->lng);
```

Examina el código completo de la aplicación que se incluye en el siguiente fichero. Asegúrate de ajustar la ruta a la librería Xajax y de configurar tu propio ID de aplicación para el acceso a Yahoo! PlaceFinder.

form.php

```
<?php
/**
 * Desarrollo Web en Entorno Servidor
 * Tema 8 : Aplicaciones web híbridadas
```

```

* Ejemplo geocodificación: form.php
*/

// Incluimos la librería Xajax
require_once("../../libs/xajax_core/xajax.inc.php");

// Creamos las funciones que van a ser llamadas
// desde JavaScript

function ubicaryahoo($coordenadas){
    $respuesta = new xajaxResponse();
    // Hacemos una búsqueda inversa (gflags=R), esto es
    // obtenemos una dirección a partir de unas coordenadas
    // Indicamos también búsqueda global (flags=G)
    // y localización española para el lenguaje (locale=es_ES)
    $search =
'http://where.yahooapis.com/geocode?location='.$coordenadas['latitud'].'+'.$coordenadas['longitud'].'&flags=G&locale=es_ES&gflags=R&appid=tuID';
    $xml = simplexml_load_file($search);

    $respuesta->assign("calle", "value", (string) $xml->Result[0]->street . " " . $xml->Result[0]->house);
    $respuesta->assign("ciudad", "value", (string) $xml->Result[0]->level3 . " " . $xml->Result[0]->level2 . " " . $xml->Result[0]->level1);
    $respuesta->assign("pais", "value", (string) $xml->Result[0]->level0);
    $respuesta->assign("cp", "value", (string) $xml->Result[0]->uzip);

    $respuesta->assign("enviarcoordenadas","value","Ver dirección");
    $respuesta->assign("enviarcoordenadas","disabled",false);
    $respuesta->assign("enviardireccion","disabled",false);
    return $respuesta;
}

function ubicargoogle($coordenadas){
    $respuesta = new xajaxResponse();
    // Indicamos idioma español (language=es) y
    // que no estamos usando un sensor de localización (sensor=false)
    $search =
'http://maps.google.com/maps/api/geocode/xml?address='.$coordenadas['calle'].'+'.$coordenadas['ciudad'].'+'.$coordenadas['pais'].'&language=es&sensor=false';
    $xml = simplexml_load_file($search);

    $respuesta->assign("latitud", "value", (string) $xml->result[0]->geometry->location->lat);
    $respuesta->assign("longitud", "value", (string) $xml->result[0]->geometry->location->lng);

    $respuesta->assign("enviardireccion","value","Ver coordenadas");
    $respuesta->assign("enviarcoordenadas","disabled",false);
    $respuesta->assign("enviardireccion","disabled",false);
    return $respuesta;
}

// Creamos el objeto xajax
$xajax = new xajax();

// Registramos las funciones que vamos a llamar desde JavaScript
$xajax->register(XAJAX_FUNCTION,"ubicaryahoo");
$xajax->register(XAJAX_FUNCTION,"ubicargoogle");

// Y configuramos la ruta en que se encuentra la carpeta xajax_js
$xajax->configure('javascript URI','../../libs/');

// El método processRequest procesa las peticiones que llegan a la página
// Debe ser llamado antes del código HTML
$xajax->processRequest();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 8: Geolocation con jQuery4PHP</title>
<link rel="stylesheet" href="estilos.css" type="text/css" />
<?php
// Le indicamos a Xajax que incluya el código JavaScript necesario
$xajax->printJavascript();
?>
<script type="text/javascript" src="ubicar.js"></script>

```

```

<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>
<script type="text/javascript">
  // 
    $(document).ready(function() {
      navigator.geolocation.getCurrentPosition(
        function(posicion) {
          $("#latitud").val(posicion.coords.latitude);
          $("#longitud").val(posicion.coords.longitude);
        }
      );
    });
  // ]]&gt;
&lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
  &lt;div id='form'&gt;
    &lt;form id='datos' action="javascript:void(null);"&gt;
      &lt;fieldset&gt;
        &lt;legend&gt;Servicios de geocodificación&lt;/legend&gt;
        &lt;div class='campo'&gt;
          &lt;label for='latitud' &gt;Latitud:&lt;/label&gt;
          &lt;input type='text' name='latitud' id='latitud' /&gt;
        &lt;/div&gt;
        &lt;div class='campo'&gt;
          &lt;label for='longitud' &gt;Longitud:&lt;/label&gt;
          &lt;input type='text' name='longitud' id='longitud' /&gt;
        &lt;/div&gt;

        &lt;div class='campo'&gt;
          &lt;input type='submit' id='enviarcoordenadas' name='enviar' value='Ver dirección con
Yahoo!' onclick="enviarCoordenadas();" /&gt;
          &lt;input type='submit' id='enviardireccion' name='enviar' value='Ver coordenadas con
Google' onclick="enviarDireccion();" /&gt;
        &lt;/div&gt;
        &lt;div class='campo'&gt;
          &lt;label for='calle' &gt;Calle:&lt;/label&gt;
          &lt;input type='text' name='calle' id='calle' /&gt;
        &lt;/div&gt;
        &lt;div class='campo'&gt;
          &lt;label for='ciudad' &gt;Ciudad:&lt;/label&gt;&lt;br /&gt;
          &lt;input type='text' name='ciudad' id='ciudad' /&gt;
        &lt;/div&gt;
        &lt;div class='campo'&gt;
          &lt;label for='pais' &gt;País:&lt;/label&gt;&lt;br /&gt;
          &lt;input type='text' name='pais' id='pais' /&gt;
        &lt;/div&gt;
        &lt;div class='campo'&gt;
          &lt;label for='cp' &gt;CP:&lt;/label&gt;&lt;br /&gt;
          &lt;input type='text' name='cp' id='cp' maxlength="5" /&gt;
        &lt;/div&gt;
      &lt;/fieldset&gt;
    &lt;/form&gt;
  &lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>
</div>
<div data-bbox="91 692 169 709" data-label="Section-Header">
<h3>ubicar.js</h3>
</div>
<div data-bbox="91 708 588 904" data-label="Text">
<pre>
function enviarCoordenadas() {
  // Se cambia el botón de Enviar y se deshabilita
  // hasta que llegue la respuesta
  xajax.$('enviarcoordenadas').disabled=true;
  xajax.$('enviardireccion').disabled=true;
  xajax.$('enviarcoordenadas').value="Un momento...";

  // Aquí se hace la llamada a la función registrada de PHP
  xajax_ubicaryahoo (xajax.getFormValues("datos"));

  return false;
}

function enviarDireccion() {
  // Se cambia el botón de Enviar y se deshabilita
  // hasta que llegue la respuesta
  xajax.$('enviarcoordenadas').disabled=true;
  xajax.$('enviardireccion').disabled=true;
}
</pre>
</div>
<div data-bbox="91 938 136 955" data-label="Page-Footer">- 18 -</div>
```

```
xajax.$('enviardireccion').value="Un momento...";

// Aquí se hace la llamada a la función registrada de PHP
xajax_ubicargoogle (xajax.getFormValues("datos"));

return false;
}
```

estilos.css

```
#form {
  position: absolute;
  left: 50%;
  top: 50%;
  width: 340px;
  margin-left: -170px;
  height: 450px;
  margin-top: -210px;
  padding:10px;
  border:1px solid #ccc;
  background-color: #eee;
}

legend, h3 {
  font-family : Arial, sans-serif;
  font-size: 1.3em;
  font-weight:bold;
  color:#333;
}

#form .campo {
  margin-top:8px;
  margin-bottom: 10px;
  margin-left: 10px;
}

#form label {
  font-family : Arial, sans-serif;
  font-size:0.8em;
  font-weight: bold;
}

#form input[type="text"] {
  font-family : Arial, Verdana, sans-serif;
  font-size: 0.8em;
  line-height:140%;
  color : #000;
  padding : 3px;
  border : 1px solid #999;
  height:20px;
  width:280px;
}

#form input[type="submit"] {
  width:200px;
  height:30px;
  padding-left:0px;
}

.oculto {
  display:none;
}
```

HTML5 incluye entre sus nuevas características una API de geolocalización, que permite a las aplicaciones web utilizar código JavaScript para obtener las coordenadas en que se encuentra el usuario. La aplicación del código anterior utiliza esa funcionalidad. Para conocer más detalles sobre el funcionamiento de la geolocalización en HTML5 y ver cómo se puede integrar con Google Maps, puedes consultar la siguiente página web.

Geolocalización con HTML5. <http://blog.atruiweb.com/html5/geolocalizacion-con-html5>

4.4.- Google Directions.

Google Directions es un servicio web de Google, que al igual que Google Geocoding también forma parte de Google Maps, y cuya principal utilidad es el cálculo de rutas para llegar desde una ubicación origen a otra ubicación destino. Las rutas pueden incluir además puntos intermedios (hitos), y tanto ellos como el origen o el destino pueden especificarse mediante direcciones reconocibles por el usuario o mediante coordenadas (latitud y longitud).



Documentación de Google Directions.

<http://code.google.com/intl/es-ES/apis/maps/documentation/directions/>

Google Directions respuestas en formato JSON (recomendado por Google).

<http://maps.google.com/maps/api/directions/json>

Google Directions para respuestas en formato XML.

<http://maps.google.com/maps/api/directions/xml>

Los únicos parámetros que deben figurar obligatoriamente en una petición son:

- ✓ El punto origen (`origin`).
- ✓ El punto destino (`destination`).
- ✓ La indicación de si se utiliza o no un dispositivo de localización (`sensor`).

Entre los parámetros opcionales están:

- ✓ El idioma en que se devuelven los resultados (`language`).
- ✓ El medio de transporte para el cálculo de las rutas (`mode`).
- ✓ Los puntos intermedios o hitos (`waypoints`). Se deben separar unos de otros utilizando una barra vertical "|". Opcionalmente se puede incluir "`optimize:true`" como primer argumento, lo que provocará que se reordenen los puntos intermedios para optimizar la ruta. En este caso, en el elemento `waypoint_order` de la respuesta se mostrará el orden resultante de optimizar la ruta.

Por ejemplo, una consulta a Google Directions podría tener la siguiente forma:

```
http://maps.google.com/maps/api/directions/json?origin=42.430283,-8.643625&destination=42.402497,-8.812001&language=es&sensor=false
```

La respuesta obtenida en formato JSON mostrará las indicaciones necesarias para llegar desde el origen al destino. El código necesario en PHP para utilizar el servicio será:

```
// Con una URL como la anterior almacenada en la variable $url:
$json = file_get_contents($url);
$respuesta = json_decode($json);
```

Una vez decodificado el formato JSON obtenido, el acceso a la información se realiza utilizando los elementos del objeto `$respuesta`. Revisa la documentación del servicio para ver cómo se estructuran las respuestas. Por ejemplo:

```
$resumen = $respuesta->routes[0]->summary;
```

Se utiliza la función `file_get_contents` de PHP para almacenar en una variable la respuesta obtenida del servicio Google Directions.

Función `file_get_contents`. <http://es.php.net/manual/es/function.file-get-contents.php>

Cuando crees una aplicación que utilice servicios proporcionados por terceros, deberás tener en cuenta siempre sus **licencias de uso**. Por ejemplo, los resultados obtenidos al usar tanto Google

Directions como Google Geocoding han de acompañarse obligatoriamente de los resultados de visualización de un mapa de Google. Además, en algunos casos las respuestas incluyen advertencias e información sobre derechos de autor que deberán mostrarse a los usuarios.

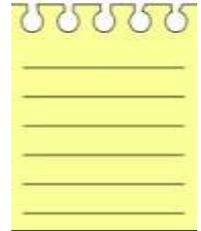
En una petición al servicio Google Directions, deben figurar obligatoriamente los parámetros:

 **origin, destination y sensor.**  **origin, destination, mode y sensor.**

El parámetro mode indica el medio de transporte a utilizar pero es opcional (se supone automóvil por defecto)

4.5.- Google Tasks.

Los servicios que has utilizado hasta ahora recibían peticiones por parte del usuario (en nuestro caso una aplicación web) y generaban, y devolvían, respuestas a las mismas en formatos JSON o XML. El servicio Google Tasks necesita, además, una autorización para devolver información privada de un usuario. Para obtener esa autorización, nuestra aplicación deberá usar el protocolo OAuth2.



El servicio de Google Tasks nos permite gestionar las tareas personales del usuario.

Servicio de Google Tasks.

<http://code.google.com/intl/es-ES/apis/tasks/>

Básicamente existen dos tipos de recursos:

- ✓ **Listas de tareas.** Una de ellas es la lista de tareas por defecto; existe siempre y no se puede eliminar.
- ✓ **Tareas.** Es cada uno de los elementos que contiene una lista de tareas. Puede contener información como el título de la tarea, notas, o fechas.

Hay dos formas de utilizar el servicio:

- ✓ Mediante llamadas REST directamente, al igual que hicimos cuando utilizamos los servicios anteriores.
- ✓ Mediante una librería cliente, disponible para múltiples lenguajes (entre ellos PHP).

En la documentación del servicio tienes información y ejemplos sobre la utilización del servicio desde cualquiera de estos dos métodos. Para poder utilizar la librería cliente deberás, en primer lugar, descargarla en su versión para PHP.

Librerías cliente para Google Tasks.

<http://code.google.com/intl/es-ES/apis/tasks/libraries.html>

Una vez descomprimida y ubicada en una ruta accesible al servidor web, deberás incluirla en las aplicaciones que la utilicen. Existen un fichero común, y otro específico según el tipo de servicio al que necesites acceder. Por ejemplo, para poder utilizar el servicio Google Tasks tendrás que añadir las siguientes líneas en tu código PHP:

```
session_start();
require_once 'google-api-php-client/src/apiClient.php';
require_once 'google-api-php-client/src/contrib/apiTasksService.php';
```

Asegúrate de ajustar correctamente la ruta a la librería.

Será necesario crear dos objetos; uno de tipo `apiClient`, que utilizará OAuth2 para gestionar las autorizaciones de acceso a los servicios.

```
// Creamos el objeto de la API de Google
$cliente = new apiClient();

// Y lo configuramos con los nuestros identificadores
$cliente->setClientId('Aquí irá tu identificador de cliente');
$cliente->setClientSecret('Aquí tu clave secreta');
```

```
$cliente->setRedirectUri('http://localhost/dwes/ut8/tareas.php');
$cliente->setDeveloperKey('Aquí irá tu clave de la API de Google');
```

Es importante señalar que la URI de redirección debe ser válida y estar dada de alta como tal en la configuración de la aplicación web, tal y como se mostró en el videotutorial sobre OAuth2 (por ejemplo, puede ser la misma dirección de la página que estamos programando).

También necesitamos crear otro objeto del tipo específico según el servicio al que accedamos, en nuestro caso `apiTasksService`.

```
// Creamos un objeto para manejar las listas y sus tareas
$apitareas = new apiTasksService($cliente);
```

4.5.1- Google Tasks (II).

El primer paso para acceder al servicio es autenticarse utilizando el método `authenticate` de la clase `apiClient`. La clave de acceso obtenida se debe almacenar utilizando la función `setAccessToken`. Puede almacenarse en una variable de sesión para futuras llamadas al servicio.

```
if (isset($_SESSION['clave_acceso'])) {
    $cliente->setAccessToken($_SESSION['clave_acceso']);
} else {
    $cliente->setAccessToken($cliente->authenticate());
    $_SESSION['clave_acceso'] = $cliente->getAccessToken();
}
```



Una vez autenticado, puedes emplear el objeto de la clase `apiTasksService` para gestionar las listas de tareas y las tareas del usuario.

- ✓ Para crear una nueva lista de tareas:

```
$nuevalista = new TaskList();
$nuevalista->setTitle('titulo_lista');
$apitareas->tasklists->insert($nuevalista);
```

- ✓ Para crear una nueva tarea y agregarla a una lista de tareas:

```
$nuevatarea = new Task();
$nuevatarea->setTitle('titulo_tarea');
$nuevatarea->setNotes('notas_tarea');
$apitareas->tasks->insert('id_lista_tareas', $nuevatarea);
```

- ✓ Para eliminar una lista de tareas:

```
$apitareas->tasklists->delete('id_lista_tareas');
```

- ✓ Para eliminar una tarea de una lista:

```
$apitareas->tasks->delete('id_lista_tareas', 'id_lista');
```

- ✓ Para recorrer las listas de tareas y sus tareas:

```
// Para recorrer las listas de tareas
$listas = $apitareas->tasklists->listTasklists();
foreach ($listas['items'] as $lista) {
    // Para recorrer las tareas de cada lista
    $tareas = $apitareas->tasks->listTasks($lista['id']);
    foreach ($tareas['items'] as $tarea) {
        ...
    }
}
```

Para que la librería funcione correctamente en sistemas Windows, deberás solucionar el problema de cURL con las autoridades de certificación. Una vez descargado el fichero de autoridades tal y como se mencionó anteriormente, deberás indicar que se utilice. Para ello, en el fichero `src/io/apiCurlIO.php`, tendrás que modificar la función `makeRequest` añadiendo la siguiente línea después del conjunto de llamadas a `curl_setopt`:

```
curl_setopt($ch, CURLOPT_CAINFO, 'ruta a la lista');
```

Para utilizar el servicio Google Tasks desde PHP, puedes emplear:

La API que ofrece Google.



La API que ofrece Google o llamadas REST.

*Puedes utilizar uno u otro método, pero la API que ofrece Google es el más sencillo de ambos.***4.6.- Aplicación web híbrida de gestión de repartos.**

Vamos a ver cómo puedes crear una aplicación web híbrida que utilice los servicios que acabas de ver, utilizando como punto de partida la aplicación web con la que has estado trabajando en unidades anteriores. El supuesto del que se parte es el siguiente:

Como la zona de influencia de la tienda aún es pequeña, y pensando también en mantener la relación con los clientes, se ha decidido hacer reparto directo de los productos que se compran en la tienda online. Para ello se ha pensado en crear una aplicación web híbrida con las siguientes características:

- ✓ Se utilizará la API del servicio de tareas de Google (Google Tasks) para almacenar como listas de tareas la información de los repartos. De esta forma la información podrá ser consultada desde cualquier lugar utilizando un dispositivo con conexión a Internet. Cada lista de tareas se corresponde en la aplicación con una lista de reparto, y cada una de sus tareas con un envío. Para diferenciar una lista de otra, se le pone como parte del título la fecha del día en que se hará el reparto.
- ✓ Para cada producto que se reparte se creará una tarea en la lista correspondiente. Esa tarea almacenará la dirección de envío y sus coordenadas. Para obtenerlas, y para mostrar su ubicación en un mapa, en el momento en que se introduzca la dirección se utilizará el servicio de geocodificación de Google (Google Geocoding).
- ✓ Para optimizar la ruta que se ha de recorrer, se utilizará Google Directions. La idea es reorganizar de forma automática el orden de los productos que se van a repartir cada día de forma que se minimice la distancia recorrida.

La apariencia de la aplicación será:



Cuando se cree una nueva tarea (un nuevo envío), se pedirá la dirección y se mostrará una pantalla como la siguiente para que el usuario complete los datos necesarios.

Se utiliza también Google Maps para mostrar en una nueva ventana el mapa correspondiente a las coordenadas de envío de los productos.

4.6.1.- Aplicación web híbrida de gestión de repartos (II).

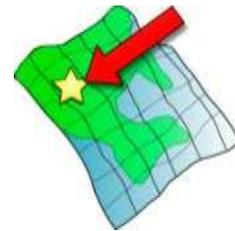
Para gestionar (crear y eliminar) las listas de tareas y sus tareas, puedes utilizar parámetros **GET** y recargar la misma página. Si está presente un parámetro **'accion'**, se realizará un procesamiento determinado. Por ejemplo:

```
switch ($_GET['accion']) {
  case 'nuevalista':
    if (!empty($_GET['nuevotitulo'])) {
      // Crear una nueva lista de reparto
      try {
        $nuevalista = new TaskList();
        $nuevalista->setTitle($_GET['nuevotitulo']);
        $apitareas->tasklists->insert($nuevalista);
      } catch (Exception $e) {
        $error="Error al crear un nuevo reparto.";
      }
    }
}
```

Para abrir una ventana que muestre ciertas coordenadas en un mapa, puedes utilizar una función Javascript como la siguiente:

```
function abrirMaps(coordenadas) {
  var url = "http://maps.google.com/maps?hl=es&t=h&z=17&output=embed&ll=";

  if(!coordenadas)
    // Cogemos las coordenadas del diálogo
    url+=$("#latitud").val()+"-"+$("#longitud").val();
  else
    // Si hay coordenadas, las usamos
    url+=coordenadas;
  window.open(url, 'nuevaventana', 'width=425,height=350');
}
```



Para obtener las coordenadas de una dirección concreta, una solución es utilizar Xajax para llamar a Google Geocoding, de forma similar a como ya hiciste en la aplicación web de geocodificación anterior.

4.6.2.- Aplicación web híbrida de gestión de repartos (III).

Un caso especial es el método a utilizar para optimizar las distintas entregas de un pedido (las tareas de una lista). Una solución es realizar el proceso en dos pasos. En primer lugar puedes utilizar Xajax para llamar al servicio Google Directions y obtener el orden óptimo de entrega. Al pulsar en el botón **'Ordenar'** de una lista, se ejecuta la siguiente función JavaScript:

```
function ordenarReparto(idReparto) {
  // Utilizamos jQuery para obtener una lista con las coordenadas
  // de los puntos intermedios que debemos ordenar
  var paradas = $('#'+idReparto+' li').map(function() {
    return this.title;
  }).get().join('|');

  // Modo síncrono (esperamos a obtener una respuesta)
  var respuesta = xajax.request({xjxfun:"ordenarReparto"}, {mode:'synchronous', parameters:
[paradas]});
  if (respuesta) {
    // Cogemos la URL base del documento actual
    var url = document.location.href.replace(/\?.*/,'');
    // Añadimos el código de la lista de reparto
    url += '?accion=ordenarEnvios&reparto='+idReparto;
    // Y las nuevas posiciones que deben ocupar los envíos
    for(var r in respuesta) url += '&pos[]='+respuesta[r];

    window.location = url;
  }
}
```

Esta función ejecuta mediante AJAX el siguiente código PHP, que obtiene el orden óptimo de reparto:

```
function ordenarReparto($coordenadasPuntosIntermedios){
    // Coordenadas del "almacén" (punto de origen y destino)
    // Se podría añadir código para permitir al usuario indicarlo
    $coordenadasOrigen = "42.402497,-8.812001";
    $url =
'http://maps.google.es/maps/api/directions/json?origin='.$coordenadasOrigen.'&destination='.$c
ordenadasOrigen;

    // Se añaden los puntos intermedios, indicando que optimice
    $url .= '&waypoints=optimize:true|';
    $url .= $coordenadasPuntosIntermedios.'&sensor=false';

    // Como el resultado es JSON, lo procesamos de la siguiente forma
    $json = file_get_contents($url);
    $respuesta = json_decode($json);
    $orden = $respuesta->routes[0]->waypoint_order;

    // Y devolvemos ese array
    $respuesta = new xajaxResponse();
    $respuesta->setReturnValue($orden);
    return $respuesta;
}
```

El array obtenido se envía en parámetros **GET** a la misma página, que lo debe procesar para ordenar las tareas según indica.

Para obtener una ruta optimizada utilizando el servicio Google Directions, debes utilizar `optimize:true` en el parámetro `waypoints`, y al procesar la respuesta recibida:



Revisar el orden de los elementos `step` recibidos.



Revisar el orden que contiene el elemento `waypoint_order`.

El elemento `waypoint_order` contiene el orden óptimo de los puntos intermedios que tu indicas al realizar la petición

4.6.3.- Aplicación web híbrida de gestión de repartos (IV).

Una vez obtenido el array que indica el orden óptimo de los puntos intermedios, se vuelve a cargar la página y en ese momento se deberán procesar los parámetros **GET** recibidos que indican cómo reordenar las tareas de la lista.

```
switch ($_GET['accion']) {
case 'ordenarEnvios':
    if (!empty($_GET['reparto']) && !empty($_GET['pos'])) {
        // Reordenar las tareas de envío según el orden que se recibe
        try {
            // Obtenemos todas las tareas de la lista de reparto
            $tareas = $apitareas->tasks->listTasks($_GET['reparto']);

            // Y las movemos según se indica en el array 'pos'
            // 'pos' indica qué posición debe tener cada tarea
            // $pos[0] = 3 significa que la 1ª tarea (la de índice 0)
            // debe ponerse en la 4ª posición (la de índice 3)

            // Lo convertimos en el array 'orden' que contiene
            // las tareas que debe haber en cada posición de la lista
            // $orden[3] = 0 significa que en la 4ª posición
            // debemos poner la 1ª tarea
            $orden = array_flip($_GET['pos']);

            // Recorremos el array en orden inverso
            // En cada paso ponemos una tarea en la primera posición
            for($i=count($orden)-1; $i>=0; $i--){
                $apitareas->tasks->move($_GET['reparto'], $tareas['items'][$orden[$i]]['id']);
            }
        } catch (Exception $e) {
            $error="Se ha producido un error al intentar ordenar los envíos del reparto.";
        }
    }
break;
```

Intenta completar por ti mismo la programación de la aplicación web. Puedes descargar y revisar el código de la solución completa en el siguiente enlace. Recuerda que debes ajustar el código para indicar las rutas correctas a las librerías de Xajax y de Google, y teclear en el mismo tus propias claves de uso de los servicios web de Google y la ruta de redirección, que debe estar registrada.

repartos.php

```
<?php
/**
 * Desarrollo Web en Entorno Servidor
 * Tema 8 : Aplicaciones web híbridas
 * Ejemplo Rutas de reparto: repartos.php
 */

session_start();

// Incluimos la API de Google
require_once '../libs/google-api-php-client/src/apiClient.php';
require_once '../libs/google-api-php-client/src/contrib/apiTasksService.php';

// y la librería Xajax
require_once("../libs/xajax_core/xajax.inc.php");

// Creamos el objeto xajax
$xajax = new xajax('ajaxmaps.php');

// Configuramos la ruta en que se encuentra la carpeta xajax_js
$xajax->configure('javascript URI','../libs/');

// Y registramos las funciones que vamos a llamar desde JavaScript
$xajax->register(XAJAX_FUNCTION,"obtenerCoordenadas");
$xajax->register(XAJAX_FUNCTION,"ordenarReparto");

// Creamos el objeto de la API de Google
$cliente = new apiClient();

// Y lo configuramos con los nuestros identificadores
$cliente->setClientId('Aquí irá tu identificador de cliente');
$cliente->setClientSecret('Aquí tu clave secreta');
$cliente->setRedirectUri('http://localhost/dwes/ut8/ej_rutas_reparto/repartos.php');
$cliente->setDeveloperKey('Aquí irá tu clave de la API de Google');

// Creamos también un objeto para manejar las listas y sus tareas
$apitareas = new apiTasksService($cliente);

// Comprobamos o solicitamos la autorización de acceso
if (isset($_SESSION['clave_acceso'])) {
    $cliente->setAccessToken($_SESSION['clave_acceso']);
} else {
    $cliente->setAccessToken($cliente->authenticate());
    $_SESSION['clave_acceso'] = $cliente->getAccessToken();
}

// Comprobamos si se debe ejecutar alguna acción
if (isset($_GET['accion'])) {
    switch ($_GET['accion']) {
        case 'nuevalista':
            if (!empty($_GET['nuevotitulo'])) {
                // Crear una nueva lista de reparto
                try {
                    $nuevalista = new TaskList();
                    $nuevalista->setTitle($_GET['nuevotitulo']);
                    $apitareas->tasklists->insert($nuevalista);
                }
                catch (Exception $e) {
                    $error="Se ha producido un error al intentar crear un nuevo reparto.";
                }
            }
            break;
        case 'nuevatarea':
            if (!empty($_GET['nuevotitulo']) && !empty($_GET['idreparto']) &&
!empty($_GET['latitud']) && !empty($_GET['longitud'])) {
                // Crear una nueva tarea de envío
                try {
                    $nuevatarea = new Task();
                    $nuevatarea->setTitle($_GET['nuevotitulo']);
```

```

        if (isset($_GET['direccion'])) $nuevatarea->setTitle($_GET['nuevotitulo']." - ".$_GET['direccion']);
        else $nuevatarea->setTitle($_GET['nuevotitulo']);
        $nuevatarea->setNotes($_GET['latitud']." ".$_GET['longitud']);
        // Añadimos la nueva tarea de envío a la lista de reparto
        $apitareas->tasks->insert($_GET['idreparto'], $nuevatarea);
    }
    catch (Exception $e) {
        $error="Se ha producido un error al intentar crear un nuevo envío.";
    }
}
break;
case 'borrarlista':
    if (!empty($_GET['reparto'])) {
        // Borrar una lista de reparto
        try {
            $apitareas->tasklists->delete($_GET['reparto']);
        }
        catch (Exception $e) {
            $error="Se ha producido un error al intentar borrar el reparto.";
        }
    }
    break;
case 'borrartarea':
    if (!empty($_GET['reparto']) && !empty($_GET['envio'])) {
        // Borrar una tarea de envío
        try {
            $apitareas->tasks->delete($_GET['reparto'],$_GET['envio']);
        }
        catch (Exception $e) {
            $error="Se ha producido un error al intentar borrar el envío.";
        }
    }
    break;
case 'ordenarEnvios':
    if (!empty($_GET['reparto']) && !empty($_GET['pos'])) {
        // Reordenar las tareas de envío según el orden que se recibe en el array
'pos'
        try {
            // Primero obtenemos todas las tareas de la lista de reparto
            $tareas = $apitareas->tasks->listTasks($_GET['reparto']);

            // Y después las movemos según la posición recibida en el array 'pos'
            // El array 'pos' indica la posición que debe tener cada tarea de la lista
            // $pos[0] = 3 significa que la 1ª tarea (la de índice 0)
            // debe ponerse en la 4ª posición (la de índice 3)
            //
            // Lo convertimos en el array 'orden' que contiene las tareas que debe
haber
            // en cada posición de la lista
            // $orden[3] = 0 significa que en la 4ª posición debemos poner la 1ª tarea
            $orden = array_flip($_GET['pos']);

            // Recorremos el array en orden inverso, esto es, empezando por la tarea
            // que debería figurar en última posición de la lista
            // En cada paso ponemos una tarea en la primera posición de la lista
            for($i=count($orden)-1; $i>=0; $i--)
                $apitareas->tasks->move($_GET['reparto'],
                $tareas['items'][$orden[$i]]['id']);
        }
        catch (Exception $e) {
            $error="Se ha producido un error al intentar ordenar los envíos del
reparto.";
        }
    }
    break;
}
}

// Obtenemos el id de la lista de tareas por defecto, para no mostrarla
$listapordefecto = $apitareas->tasklists->get('@default');
$id_defecto = $listapordefecto['id'];
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />

```

```

<title>Ejemplo Tema 8: Rutas de reparto</title>
<link href="estilos.css" rel="stylesheet" type="text/css" />
<?php
// Le indicamos a Xajax que incluya el código JavaScript necesario
$xajax->printJavascript();
?>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>
<script type="text/javascript" src="codigo.js"></script>
</head>

<body>
<div id="dialogo">
  <a id="cerrarDialogo" onclick="ocultarDialogo();">x</a>
  <h1>Datos del nuevo envío</h1>
  <form id="formenvio" name="formenvio" action="<?php echo $_SERVER['PHP_SELF'];?>"
method="get">
    <fieldset>
      <div id="datosDireccion">
        <p>
          <label for='direccion' >Dirección:</label>
          <input type='text' size="60" name='direccion' id='direccion' />
        </p>
        <input type='button' id='obtenerCoordenadas' value='Obtener coordenadas'
onclick="getCoordenadas();" /><br />
      </div>
      <div id="datosEnvio">
        <p>
          <label for='latitud' >Latitud:</label>
          <input type='text' size="10" name='latitud' id='latitud' />
        </p>
        <p>
          <label for='longitud' >Longitud:</label>
          <input type='text' size="10" name='longitud' id='longitud' />
        </p>
        <p>
          <label for='nuevotitulo' >Título:</label>
          <input type='text' size="40" name='nuevotitulo' id='titulo' />
        </p>
        <input type='hidden' name='accion' value='nuevatarea' />
        <input type='hidden' name='idreparto' id='idrepartoactual' />
        <input type='submit' id='nuevoEnvio' value='Crear nuevo Envio' />
        <a href="#" onclick="abrirMaps();" >Ver en Google Maps</a><br />
      </div>
    </fieldset>
  </form>
</div>
<div id="fondonegro" onclick="ocultarDialogo();"></div>
<div class="contenedor">
  <div class="encabezado">
    <h1>Ejemplo Tema 8: Rutas de reparto</h1>
    <form id="nuevoreparto" action="<?php echo $_SERVER['PHP_SELF'];?>" method="get">
      <fieldset>
        <input type='hidden' name='accion' value='nuevalista' />
        <input type='submit' id='crearnuevotitulo' value='Crear Nueva Lista de Reparto' />
        <label for='nuevotitulo' >con título:</label>
        <input type='text' name='nuevotitulo' id='nuevotitulo' />
      </fieldset>
    </form>
  </div>
  <div class="contenido">
    <?php
    $repartos = $apitareas->tasklists->listTasklists();
    // Para cada lista de reparto
    foreach ($repartos['items'] as $reparto) {
      // Excluyendo la lista por defecto de Google Tasks
      if($reparto['id'] == $id_defecto) continue;

      print '<div id="' . $reparto['id'] . '>';
      print '<span class="titulo">' . $reparto['title'] . '</span>';
      $idreparto = "" . $reparto['id'] . "";
      print '<span class="accion">(<a href="#"
onclick="ordenarReparto(' . $idreparto . ');">Ordenar</a></span>';
      print '<span class="accion">(<a href="#" onclick="nuevoEnvio(' . $idreparto . ');">Nuevo
Envío</a></span>';

```

```

        print '<span class="accion">(a
href="'.$_SERVER['PHP_SELF'].'?accion=borrarlista&reparto='.$reparto['id'].'">Borrar</a></spa
n>';

        print '<ul>';
        // Cogemos de la lista de reparto las tareas de envío
        $envios = $apitareas->tasks->listTasks($reparto['id']);

        // Por si no hay tareas de envío en la lista
        if (!empty($envios['items']))
            foreach ($envios['items'] as $envio) {
                // Creamos un elemento para cada una de las tareas de envío
                $idenvio = "". $envio['id']. "";
                print '<li title="'. $envio['notes']. ' id="'. $idenvio. '>'. $envio['title']. '
('.$envio['notes'].')';
                $coordenadas = "". $envio['notes']. "";
                print '<span class="accion"> (<a href="#"
onclick="abrirMaps('.$coordenadas.');">Ver mapa</a></span>';
                print '<span class="accion"> (<a
href="'.$_SERVER['PHP_SELF'].'?accion=borrartarea&reparto='.$reparto['id'].'&envio='.$envio['i
d'].'">Borrar</a></span>';
                print '</li>';
            }
        print '</ul>';
        print '</div>';
    }
?>
</div>
<div class="pie">
    <?php print $error; ?>
</div>
</div>
</body>
</html>

```

ajaxmaps.php

```

<?php
/**
 * Desarrollo Web en Entorno Servidor
 * Tema 8 : Aplicaciones web híbridas
 * Ejemplo Rutas de reparto: ajaxmaps.php
 */

// Incluimos la librería Xajax
require_once("../libs/xajax_core/xajax.inc.php");

// Creamos el objeto xajax
$xajax = new xajax();

// Y registramos la función que vamos a llamar desde JavaScript
$xajax->register(XAJAX_FUNCTION,"obtenerCoordenadas");
$xajax->register(XAJAX_FUNCTION,"ordenarReparto");

// El método processRequest procesa las peticiones que llegan a la página
// Debe ser llamado antes del código HTML
$xajax->processRequest();

function ordenarReparto($coordenadasPuntosIntermedios)
{
    // Indicamos las coordenadas del almacén de donde sale la mercancía
    // Se podría añadir código para permitir al usuario indicarlo
    // o incluso coger por defecto la ubicación actual del usuario
    $coordenadasOrigen = "42.402497,-8.812001";

    // Se comienza y finaliza la ruta de reparto en el almacén
    $url = 'http://maps.google.es/maps/api/directions/json?origin='.$coordenadasOrigen;
    // Para obtener el resultado en XML en lugar de JSON, podríamos hacer:
    // $url = 'http://maps.google.es/maps/api/directions/xml?origin='.$coordenadasOrigen;
    $url .= '&destination='.$coordenadasOrigen;

    // Y se añaden los puntos de envío, indicando que optimice el recorrido
    $url .= '&waypoints=optimize:true|';
    $url .= $coordenadasPuntosIntermedios;
    $url .= '&sensor=false';

    // Como el resultado es JSON, lo procesamos de la siguiente forma
    $json = file_get_contents($url);
    $respuesta = json_decode($json);
}

```

```

    $orden = $respuesta->routes[0]->waypoint_order;

    // Si obtuviéramos un resultado en XML, habría que procesarlo de la siguiente forma
    /*
    $xml = simplexml_load_file($url);
    // Guardamos el recorrido óptimo calculado en un array
    foreach($xml->route[0]->waypoint_index as $parada) {
        $orden[] = (integer) $parada;
    }
    */

    // Y devolvemos el array obtenido
    $respuesta = new xajaxResponse();
    $respuesta->setReturnValue($orden);
    return $respuesta;
}

function obtenerCoordenadas($parametros)
{
    $respuesta = new xajaxResponse();
    $search =
'http://maps.google.com/maps/api/geocode/xml?address='.$parametros['direccion'].'&sensor=false
&appid=z9hiLa3e';
    $xml = simplexml_load_file($search);

    $respuesta->assign("latitud", "value", (string) $xml->result[0]->geometry->location->lat);
    $respuesta->assign("longitud", "value", (string) $xml->result[0]->geometry->location-
>lng);

    $respuesta->assign("obtenerCoordenadas", "value", "Obtener coordenadas");
    $respuesta->assign("obtenerCoordenadas", "disabled", false);

    return $respuesta;
}
?>

```

código.js

```

/**
 * Desarrollo Web en Entorno Servidor
 * Tema 8 : Aplicaciones web híbridadas
 * Ejemplo Rutas de reparto: codigo.js
 */

// Indica si se está mostrando o no el diálogo de dirección / coordenadas
// para la introducción de un nuevo envío
var estadoDialogo = false;

function nuevoEnvio(idReparto) {
    $('#idrepartoactual').val(idReparto);
    mostrarDialogo();
}

function ordenarReparto(idReparto) {
    // Utilizamos jQuery para obtener una lista con las coordenadas de los puntos intermedios
    // que debemos ordenar
    // También se podrían haber almacenado por ejemplo en la sesión del usuario
    var paradas = $('#'+idReparto+' li').map(function() {
        return this.title;
    }).get().join('|');

    // Esta vez utilizamos el modo síncrono (esperamos a obtener una respuesta)
    var respuesta = xajax.request({xjxfun:"ordenarReparto"}, {mode:'synchronous', parameters:
[paradas]});
    if (respuesta) {
        // Si obtuvimos una respuesta, reordenamos los envíos del reparto
        // Cogemos la URL base del documento, quitando los parámetros GET si los hay
        var url = document.location.href.replace(/\?.*/,'');
        url = url.replace(/#$/, '');
        // Añadimos el código de la lista de reparto
        url += '?accion=ordenarEnvios&reparto='+idReparto;
        // Y un array con las nuevas posiciones que deben ocupar los envíos
        for(var r in respuesta) url += '&pos[]='+respuesta[r];

        window.location = url;
    }
}

```

```

    }
}

function getCoordenadas() {
    // Comprobamos que se haya introducido una dirección
    if($("#direccion").val().length < 10) {
        alert("Introduzca una dirección válida.");
        return false;
    }

    // Se cambia el botón de Enviar y se deshabilita
    // hasta que llegue la respuesta
    xajax.$('obtenerCoordenadas').disabled=true;
    xajax.$('obtenerCoordenadas').value="Un momento...";

    // Aquí se hace la llamada a la función registrada de PHP
    xajax_obtenerCoordenadas (xajax.getFormValues("formenvio"));

    return false;
}

function abrirMaps(coordenadas) {
    var url = "http://maps.google.com/maps?hl=es&t=h&z=17&output=embed&ll=";

    if(!coordenadas) {
        // Cogemos las coordenadas del diálogo
        url+=$("#latitud").val()+","+$("#longitud").val();
    }
    else {
        // Si hay coordenadas, las usamos
        url+=coordenadas;
    }

    window.open(url, 'nuevaventana', 'width=425,height=350');
}

function mostrarDialogo() {
    //Centramos en pantalla
    var anchoVentana = document.documentElement.clientWidth;
    var altoVentana = document.documentElement.clientHeight;
    var altoDialogo = $("#dialogo").height();
    var anchoDialogo = $("#dialogo").width();

    $("#dialogo").css({
        "position": "absolute",
        "top": altoVentana/2-altoDialogo/2,
        "left": anchoVentana/2-anchoDialogo/2
    });

    //Para IE6
    $("#fondonegro").css({"height": altoVentana});

    //Si no está visible el diálogo
    if(!estadoDialogo){
        // Se muestra el fondo negro
        $("#fondonegro").css({"opacity": "0.7"});
        $("#fondonegro").fadeIn("slow");
        // y el diálogo
        $("#dialogo").fadeIn("slow");

        $("#datosenvio").hide();
        estadoDialogo = true;
    }
}

function ocultarDialogo() {
    // Si está visible
    if(estadoDialogo){
        // Se oculta el fondo y el diálogo
        $("#fondonegro").fadeOut("slow");
        $("#dialogo").fadeOut("slow");
        estadoDialogo = false;
    }
}
}

```

estilos.css

```
@charset "utf-8";
body {
  font: 100%/1.4 Verdana, Arial, Helvetica, sans-serif;
  background: #699;
  margin: 0;
  padding: 0;
  color: #000;
}

h1, h2, h3, h4, h5, h6 {
  margin-top: 0;
  padding-left: 15px;
}

a:link {
  color:#414958;
  text-decoration: underline;
}
a:visited {
  color: #4E5869;
  text-decoration: underline;
}
a:hover, a:active, a:focus {
  text-decoration: none;
}

.accion {
  font-size: x-small;
}

.contenedor {
  min-width: 780px;
  background: #FFF;
  margin: 0 auto;
}

.encabezado {
  padding-top: 10px;
  background-color: #699;
}

.contenido {
  padding: 10px 0;
}

.contenido ul {
  padding: 0 15px 15px 40px;
  margin: 0;
}

.pie {
  color: #F00;
  padding: 10px 0;
  background: #699;
  position: relative;
  clear: both;
}

.titulo {
  padding-top: 5px;
  padding-right: 10px;
  padding-bottom: 0px;
  padding-left: 20px;
  font-size: large;
  font-weight: bold;
}

.clearfloat {
  clear:both;
  height:0;
  font-size: 1px;
  line-height: 0px;
}

#dialogo {
  display:none;
  position:fixed;
  _position:absolute; /* para IE6*/
```

```
height:320px;
width:408px;
background:#FFFFFF;
border:2px solid #cecece;
z-index:2;
padding:12px;
font-size:13px;
}
h1 {
color:#039;
font-size:24px;
font-weight:700;
padding-bottom:2px;
}

#cerrarDialogo {
font-size:14px;
line-height:14px;
right:6px;
top:4px;
position:absolute;
color:#6fa5fd;
font-weight:700;
display:block;
cursor: pointer;
text-decoration:none;
}

#fondonegro {
display:none;
position:fixed;
_position:absolute; /* para IE6*/
height:100%;
width:100%;
top:0;
left:0;
background:#000000;
border:1px solid #cecece;
z-index:1;
}
```