

Programación Web Dinámica

2022

Grupo N° 7

Arce Verónica Lorena

Terrazas Bruno

PHP Trabajo de Investigación

Deberá investigar y probar el uso (con ejemplos) de alguna librería en PHP, que resuelva un problema específico.

- Se debe exponer y explicar la implementación del uso de la clase seleccionada.
- Se entregarán las fuentes, una descripción de la clase y todo el material que puedan reunir.

Librería seleccionada

PHPGeo

PHP Trabajo Práctico Librerías Útiles

PHP Geo - Biblioteca de ubicación geográfica

phpgeo es una biblioteca simple para calcular distancias entre coordenadas geográficas con alta precisión. Esto funcionará muy bien en aplicaciones que utilizan datos de ubicación. Para obtener las coordenadas, puede usar la API de ubicación de HTML5, la API de Yahoo (o ambas). <https://tutorialzine.com/2013/02/24-cool-php-libraries-you-should-know-about>

phpgeo - Una biblioteca geográfica simple para PHP

phpgeo proporciona abstracciones de coordenadas geográficas (incluido soporte para diferentes elipsoides) y le permite calcular distancias geográficas entre coordenadas con alta precisión. <https://github.com/mjaschen/phpgeo>



The screenshot shows the homepage of the phpgeo project. The browser address bar displays 'phpgeo.marcusjaschen.de'. The page has a red header with the 'phpgeo' logo and a search bar. Below the header, a large orange banner contains the text 'Una biblioteca geográfica simple pero poderosa para PHP'. Underneath the banner are two buttons: 'VER EN GITHUB' and 'VER DOCUMENTACIÓN'. The main content area is white and features the heading '¿Qué es phpgeo?'. The text explains that phpgeo is a small PHP library for geographic calculations, including support for different ellipsoids, polylines, polygons, and more. It mentions that the library is developed by Marcus Jaschen and other contributors, and it is licensed under the MIT license. Links are provided for the GitHub project page and the issue tracker. A 'Privacidad' (Privacy) section mentions a 'Datenschutzerklärung' (Privacy Policy). The footer is red and contains links to 'Descargar' (Download), 'Repositorio de GitHub' (GitHub Repository), and 'Ayuda/Soporte/Errores' (Help/Support/Errors).

phpgeo.marcusjaschen.de

phpgeo Búsqueda...

Una biblioteca geográfica simple pero poderosa para PHP

VER EN GITHUB VER DOCUMENTACIÓN

¿Qué es *phpgeo*?

phpgeo es una pequeña biblioteca de PHP que proporciona abstracciones de coordenadas geográficas (incluido el soporte para diferentes elipsoides), polilíneas ("Pistas de GPS"), polígonos, límites y más. *phpgeo* le permite realizar diferentes cálculos con estas abstracciones, como distancias, longitudes de pistas, etc.

phpgeo está desarrollado por [Marcus Jaschen](#) y todos los [colaboradores](#).

phpgeo está licenciado bajo la [Licencia MIT](#).

El proyecto está alojado en Github:

- [Sitio del proyecto Github](#)
- [Rastreador de problemas](#)

Privacidad

La declaración de privacidad de este sitio de documentación se puede encontrar aquí: [Datenschutzerklärung](#)

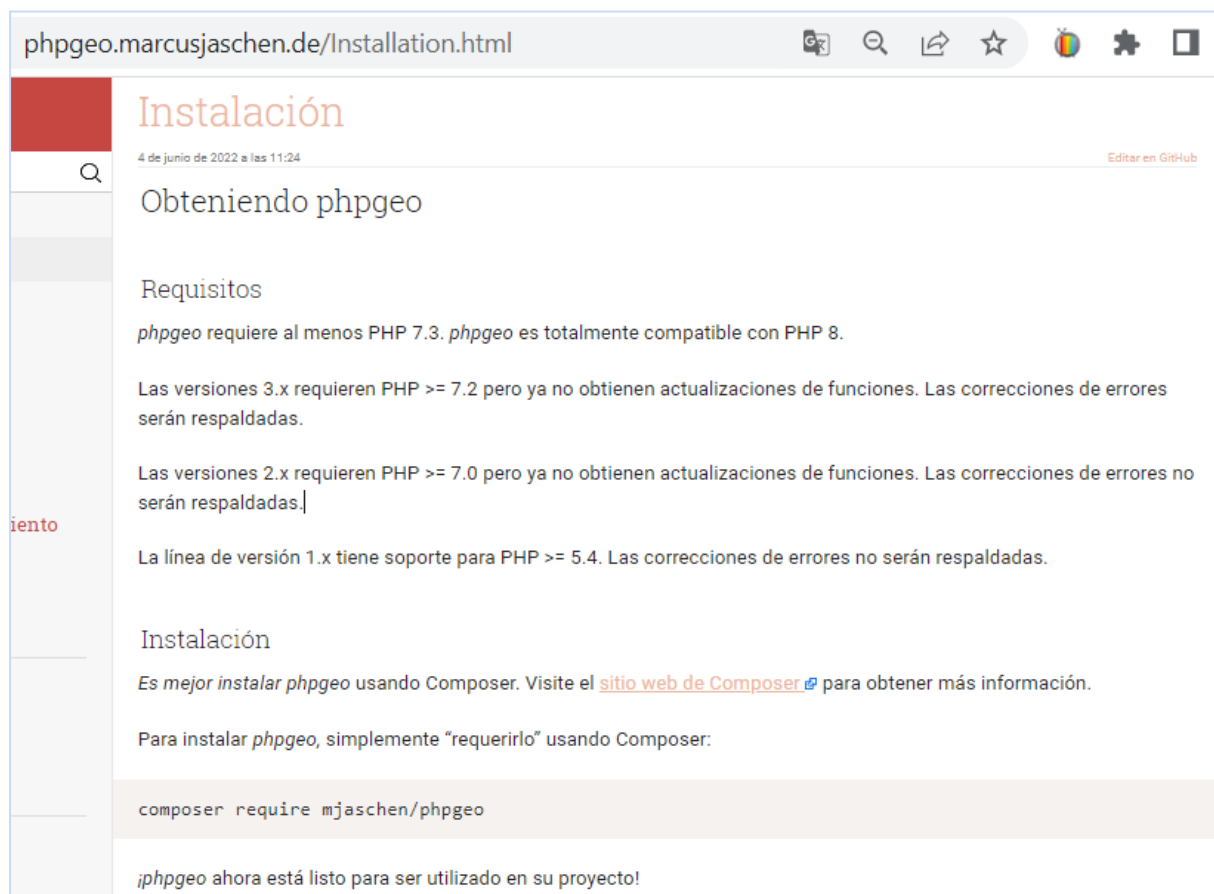
- Descargar
- Repositorio de GitHub
- Ayuda/Soporte/Errores

https://es.wikipedia.org/wiki/Licencia_MIT

La **licencia MIT** es una **licencia de software** que se origina en el **Instituto Tecnológico de Massachusetts** (MIT, Massachusetts Institute of Technology).

Esta licencia es una **licencia de software libre permisiva** lo que significa que impone muy pocas limitaciones en la reutilización y por tanto posee una excelente **Compatibilidad de licencia**. La licencia MIT permite reutilizar software dentro de **Software propietario**. Por otro lado, la licencia MIT es compatible con muchas licencias **copyleft**, como la **GNU General Public License**¹ (el software con licencia MIT puede integrarse en software con licencia GPL, pero no al contrario).

Documentación para instalación

The image is a screenshot of a web browser displaying the installation documentation for phpgeo. The browser's address bar shows the URL 'phpgeo.marcusjaschen.de/Installation.html'. The page has a red header bar with the title 'Instalación' in white. Below the header, there is a search bar and a timestamp '4 de junio de 2022 a las 11:24'. The main content area is titled 'Obteniendo phpgeo' and includes sections for 'Requisitos' and 'Instalación'. The 'Requisitos' section states that phpgeo requires at least PHP 7.3 and is compatible with PHP 8. It also mentions that versions 3.x require PHP >= 7.2 and versions 2.x require PHP >= 7.0. The 'Instalación' section recommends using Composer and provides a code block for the command 'composer require mjaschen/phpgeo'. The page also includes a link to the GitHub repository and a note about the version 1.x support for PHP >= 5.4.

phpgeo.marcusjaschen.de/Installation.html

Instalación

4 de junio de 2022 a las 11:24 [Editar en GitHub](#)

Obteniendo phpgeo

Requisitos

phpgeo requiere al menos PHP 7.3. *phpgeo* es totalmente compatible con PHP 8.

Las versiones 3.x requieren PHP >= 7.2 pero ya no obtienen actualizaciones de funciones. Las correcciones de errores serán respaldadas.

Las versiones 2.x requieren PHP >= 7.0 pero ya no obtienen actualizaciones de funciones. Las correcciones de errores no serán respaldadas.

La línea de versión 1.x tiene soporte para PHP >= 5.4. Las correcciones de errores no serán respaldadas.

Instalación

Es mejor instalar *phpgeo* usando Composer. Visite el [sitio web de Composer](#) para obtener más información.

Para instalar *phpgeo*, simplemente "requerirlo" usando Composer:

```
composer require mjaschen/phpgeo
```

phpgeo ahora está listo para ser utilizado en su proyecto!

Siguiendo los pasos para la instalación descargamos Composer en <https://getcomposer.org/>

Composer es una herramienta para la gestión de dependencias en PHP. Le permite declarar las bibliotecas de las que depende su proyecto y las administrará (instalará/actualizará) por usted.

Para realizar bien la instalación de la librería además seguimos los pasos del siguiente video:

Cómo instalar Composer en Windows 10 para PHP

<https://www.youtube.com/watch?v=NGvfsCOVzwo>

Abrimos CMD, utilizamos los comandos `cd..` y `cd` para ubicarnos dentro de la carpeta donde quedará ubicada la librería y ejecutamos el comando:

```
composer require mjaschen/phpgeo
```

```

C:\>cd xampp

C:\xampp>cd htdocs

C:\xampp\htdocs>cd ProyectoGEO

C:\xampp\htdocs\ProyectoGEO>composer require mjaschen/phpgeo
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^4.2 for mjaschen/phpgeo
./composer.json has been created
Running composer update mjaschen/phpgeo
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking mjaschen/phpgeo (4.2.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading mjaschen/phpgeo (4.2.0)
  - Installing mjaschen/phpgeo (4.2.0): Extracting archive
Generating autoload files
No security vulnerability advisories found

```

En este caso la instalación no requirió ninguna modificación del PHP.ini pero de ser necesaria alguna modificación en el video de instalación de Composer hay algunas recomendaciones.

Para probar el funcionamiento de la librería, realizamos una prueba con el siguiente código, para ver si formateaba la salida en Grados, Minutos y Segundos, pero no funcionó.

```

use Location\Coordinate;
use Location\Formatter\Coordinate\DMS;

$coordinate = new Coordinate(18.911306, -155.678268); // South Point,
HI, USA

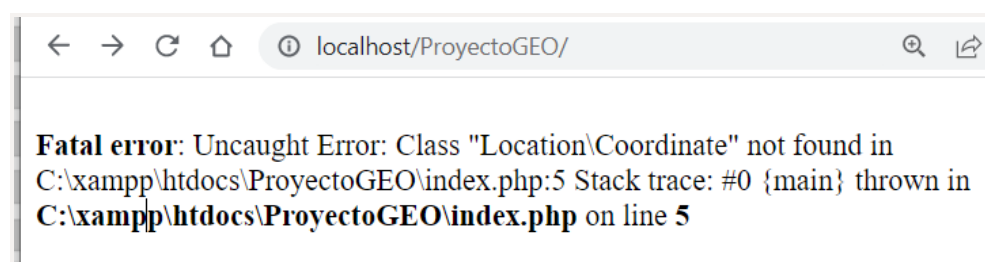
$formatter = new DMS();

echo $coordinate->format($formatter) . PHP_EOL;

$formatter->setSeparator(', ')
->useCardinalLetters(true)
->setUnits(DMS::UNITS_ASCII);

echo $coordinate->format($formatter) . PHP_EOL;

```



Así que regresamos a la documentación, resultó que debíamos completar otra actualización:

Actualización de phpgeo 3.x a phpgeo 4.x

```
C:\xampp\htdocs\ProyectoGEO>composer require mjaschen/phpgeo:^4.0
The "4.0" constraint for "mjaschen/phpgeo" appears too strict and will likely not match what you want. See https://getcomposer.org/constraints
./composer.json has been updated
Running composer update mjaschen/phpgeo
Loading composer repositories with package information
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies
Lock file operations: 0 installs, 1 update, 0 removals
  - Downgrading mjaschen/phpgeo (4.2.0 => 4.0.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 0 installs, 1 update, 0 removals
  - Downloading mjaschen/phpgeo (4.0.0)
  - Downgrading mjaschen/phpgeo (4.2.0 => 4.0.0): Extracting archive
Generating autoload files
No security vulnerability advisories found
```

Volvió a dar otro error, pero ahí nos dimos cuenta que al código le faltaba:

```
require "vendor/autoload.php";
```

Incorporado, el código de prueba devolvió lo esperado.

Funcionalidades de la Librería

Geometrías

phpgeo proporciona varias clases de geometría:

- [Coordinate](#)
- [Line](#)
- [Polyline](#)
- [Polygon](#)

Una Coordenada representa una ubicación geográfica, es decir, contiene una latitud y una longitud, junto con el denominado Elipsoide.

Una línea consta de dos coordenadas, mientras que las polilíneas y los polígonos se construyen a partir de dos o más coordenadas.

- **Coordinate**

La **Coordinate** clase es la clase más importante de *phpgeo* y proporciona la base para todas las funciones. Es una representación de una ubicación geográfica y consta de tres partes:

- Latitud geográfica
- Longitud geográfica
- elipsoide

Los valores geográficos de latitud y longitud son números flotantes entre -90,0 y 90,0 (grados de latitud) y -180,0 y 180,0 (grados de longitud).

El elipsoide es una representación de una forma aproximada de la tierra y se abstrae en su propia [Ellipsoid](#) clase de enlace

- **Line**

Una línea consta de dos puntos, es decir, instancias de la **Coordinate** clase.

Longitud

La **Line** clase proporciona un método para calcular su propia longitud. El método espera una instancia de una clase que implemente el **DistanceInterface**.

Haversine y **Vincenty** son las dos clases disponibles actualmente para el cálculo de distancia.

Punto medio

El punto medio de una línea se calcula siguiendo el Gran Círculo (definido por los dos extremos) y dividiendo la línea en dos mitades.

Punto intermedio

Similar al cálculo del punto medio, pero divide la línea en la fracción dada (entre 0,0 y 1,0; pero los valores fuera de ese rango también funcionan).

Llevando

El rumbo de una instancia se puede calcular utilizando el `getBearing()` método. `BearingInterface` Se debe proporcionar una instancia de como argumento del método.

`BearingEllipsoidal` es una de las dos clases disponibles actualmente para el cálculo de rodamientos. El otro se llama `BearingSpherical`.

Este es el llamado *rumbo inicial*. Existe otro ángulo de rumbo, llamado *rumbo final*. También se puede calcular:

Consulte [Rumbo](#) entre dos puntos @TODO Enlace para obtener más información sobre rodamientos.

- **Polyline**

Una polilínea consta de una lista ordenada de ubicaciones, es decir, instancias de la `Coordinate` clase.

Crear una polilínea

Para crear una polilínea, simplemente cree una instancia de la clase y agregue puntos.

Es posible agregar puntos al final de la polilínea en todo momento con el `addPoint()` método.

Use `addUniquePoint()` para agregar puntos únicos, es decir, puntos que aún no existen en la polilínea.

Segmentos

Es posible obtener una lista de segmentos de polilínea. Los segmentos se devuelven como una matriz de `Line` instancias.

Longitud

El cálculo de la longitud se describe en la sección [Distancia y longitud](#) .

Punto Promedio

El `getAveragePoint()` método devuelve un punto cuya latitud y longitud es el promedio de los valores de latitud/longitud de todos los puntos de polilínea.

PRECAUCIÓN: Actualmente, este método devuelve valores incorrectos si la polilínea cruza la línea de fecha en 180/-180 grados de longitud.

Dirección contraria

Es posible obtener una nueva instancia con dirección invertida mientras la polilínea original permanece sin cambios:

- **Polygon**

Un polígono consta de una lista ordenada de ubicaciones, es decir, instancias de la `Coordinate` clase. Es muy similar a una polilínea, pero sus puntos inicial y final están conectados.

Crear un polígono

Para crear un polígono, simplemente cree una instancia de la clase y agregue puntos

Obtener lista de puntos

`getPoints()` se utiliza para obtener la lista de puntos, el número de puntos se puede recuperar llamando a `getNumberOfPoints()`.

Segmentos

Es posible obtener una lista de segmentos de polígonos. Los segmentos se devuelven como una matriz de `Line` instancias.

Longitud/Perímetro

El cálculo de la longitud se describe en la sección [Distancia y longitud](#).

Área

Es posible calcular el área de un polígono. El resultado se da en metros cuadrados (m²).

ADVERTENCIA: El cálculo da resultados inexactos. Para polígonos relativamente pequeños, el error debe ser inferior al 1 %.

Geocerca

Es posible verificar si un objeto de geometría (punto, línea, polilínea, polígono) se encuentra dentro de un polígono. La documentación se puede encontrar en la sección <<Geofence>> @TODO.

Dirección contraria

Es posible obtener una nueva instancia con dirección invertida mientras el polígono original permanece sin cambios

- **Bounds**

Los límites describen un área que está definida por sus puntos noreste y suroeste.

Todas las geometrías de *phpgeo*, excepto la `Coordindate` clase, proporcionan un `getBounds()` método a través de `GetBoundsTrait`.

La `Bounds` clase tiene un método para calcular el punto central del objeto de los límites (también funciona correctamente para los límites que cruzan la línea de fecha a 180/-180 grados de longitud).

Crear límites para un punto central dado y distancia a las esquinas

- **Ellipsoid**

Un elipsoide es una aproximación definida matemáticamente de la superficie terrestre.

Un elipsoide se define por dos parámetros:

- el semieje mayor *a* (radio ecuatorial)
- el eje semi-menor *b* (radio polar)

a y *b* juntos definen el aplanamiento del elipsoide *f* :

$$f = (a - b) / a$$

NOTA: los elipsoides de *phpgeo* están definidos por *a* y *1/f* en lugar de *a* y *b*. Eso no es un problema porque cada uno de los tres valores se puede calcular a partir de los otros dos.

phpgeo admite elipsoides arbitrarios. WGS-84 se usa por defecto cuando no se proporciona ningún otro elipsoide. Para los cálculos diarios no es necesario preocuparse por los elipsoides en la mayoría de los casos.

Es posible crear una instancia de la clase *Ellipsoid* especificando un nombre o proporcionando los tres parámetros *name*, *a* y *1/f*.

El primer elipsoide se crea a partir de una de las configuraciones predeterminadas. El segundo se crea proporcionando un nombre y los valores de *a* y *1/f*.

Cálculos

- **Distancia y Longitud**

Distancia entre dos puntos (fórmula de Vincenty)

Utilice el objeto de la calculadora directamente:

```
$calculator->getDistance($coordinate1, $coordinate2);
```

o llame al `getDistance()` método de una *Coordinate* instancia inyectando una instancia de calculadora:

```
$coordinate1->getDistance($coordinate2, new Vincenty());
```

Distance Between Two Points (Haversine Formula)

There exist different methods for calculating the distance between two points. The [Haversine formula](#) is much faster than Vincenty's method but less precise.

```
$coordinate1->getDistance($coordinate2, new Haversine());
```

Longitud de una polilínea

phpgeo tiene una implementación de polilíneas que se puede usar para calcular la longitud de un track GPS o una ruta. Una polilínea consta de al menos dos puntos. Los puntos son instancias de la *Coordinate* clase.

Para más detalles sobre polilíneas/pistas GPS ver la [Polyline](#) sección.

```
$track->getLength(new Vincenty());
```

Perímetro de un polígono

El perímetro se calcula como la suma de la longitud de todos los segmentos. El resultado se da en metros. `$polygon->getPerimeter(new Vincenty());`

- **Distancia cardinal entre dos puntos**

Las distancias que hay que recorrer hacia un punto cardinal y finalmente hacia otro para llegar al segundo punto *P 2* desde el primero *P 1* se denominan distancias cardinales.

En el siguiente ejemplo, las distancias cardinales están etiquetadas como *N* y *E*:

Con *phpgeo* hay dos formas de calcular las Distancias Cardinales:

Uso de la Calculator instancia

```
$result =  
$cardinalDirectionDistancesCalculator->getCardinalDirectionDistances(  
$coordinate1, $coordinate2, $calculator);  
  
echo 'Cardinal Distances: north=' . $result->getNorth()  
. ' m; east=' . $result->getEast()  
. ' m; south=' . $result->getSouth()  
. ' m; west=' . $result->getWest() . ' m.';
```

Usando el `getCardinalDirectionDistances()` método de una instancia de `Coordinate`

Distancia perpendicular

La *distancia perpendicular* se define como la distancia más corta entre un punto y una línea (en el plano bidimensional) respectivamente entre un punto y un [gran círculo](#) en una superficie esférica.

Con *phpgeo* es posible calcular la distancia perpendicular entre un punto (instancia de la [Coordinate](#) clase) y un Gran Círculo, que está definido por un [Line](#). Una línea está definida por un par de coordenadas.

```
$pdCalc->getPerpendicularDistance($point, $line)
```

Geocerca

phpgeo tiene una implementación de polígono que se puede usar para determinar si una geometría (punto, línea, polilínea, polígono) está contenida o no. Un polígono consta de al menos tres puntos.

ADVERTENCIA: El cálculo da resultados erróneos si los polígonos cruzan el meridiano de 180/-180 grado

```
$geofence->contains($Point)
```

API DE GOOGLE MAPS

Usamos la API de Google Maps para visualizar las gráficas, se requiere la key de Google Maps.

Para obtener la key de Google Maps debemos seguir los siguientes pasos:

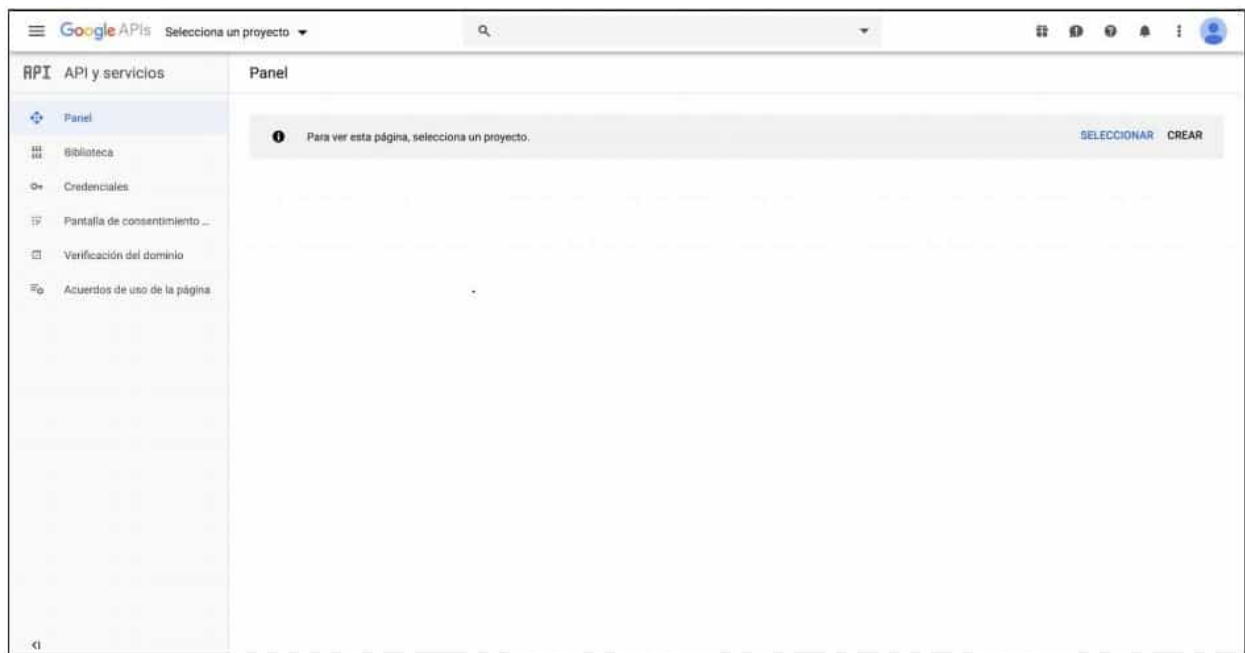
1 – Inicia sesión o crea una cuenta de Gmail

Primero, para obtener una API Key de Google Maps, deberás tener una cuenta de Gmail.

2: Accede al panel de Google API Console

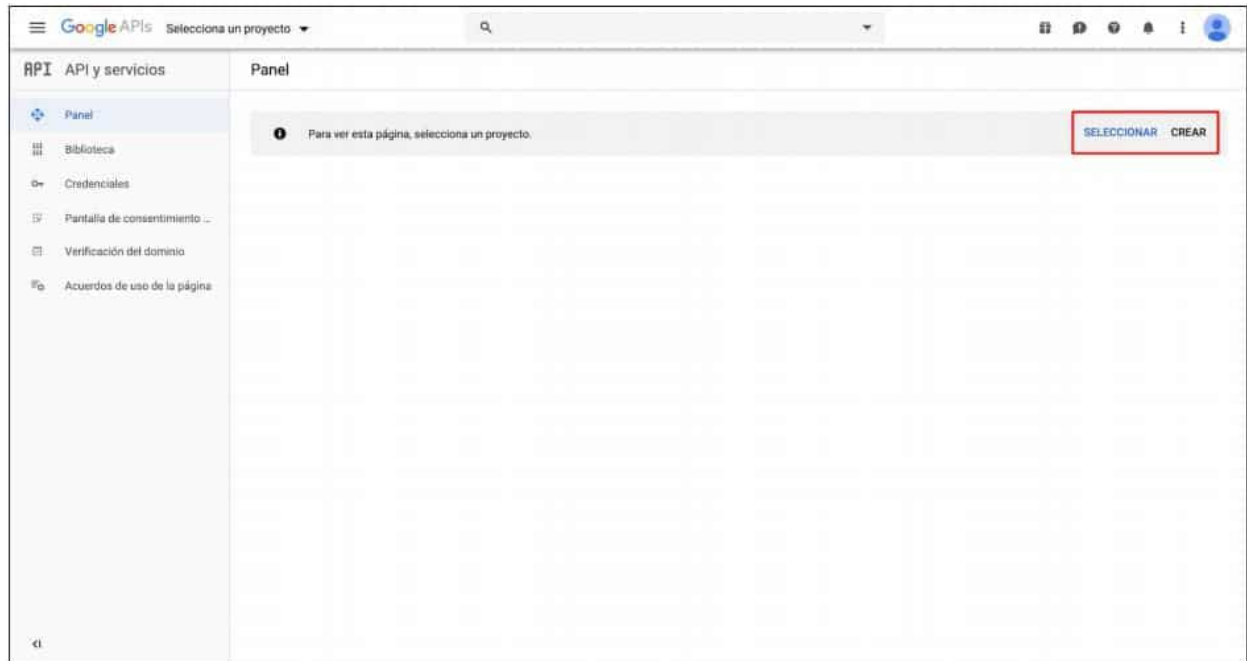
Luego, habiendo iniciado sesión en tu cuenta de Gmail, debes ir al panel del API Console de Google. Ahí encontrarás una pantalla similar a la imagen a continuación:

<https://console.cloud.google.com/apis/dashboard>

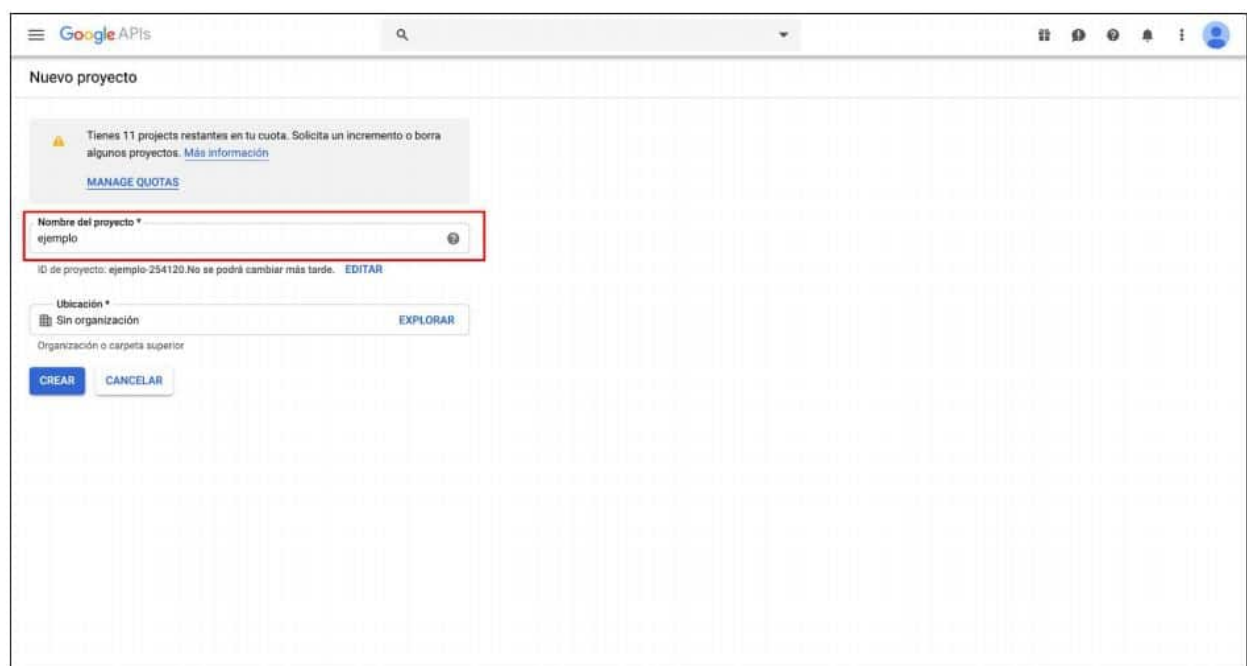


3: Selecciona un proyecto existente o crea uno nuevo

Solo se podrá obtener una API Key de Google Maps si tenemos un proyecto en la plataforma. Si ya has creado uno, aparecerán dos opciones al lado derecho de la pantalla: “SELECCIONAR” y “CREAR”. En caso que aún no tengas ninguno, solo aparecerá la opción «CREAR», en la que debes hacer clic, como se muestra a continuación:



4: Colocar un nombre del proyecto



4: Seleccionar API y servicios

Google Cloud proyecto geophp

Buscar Productos, recursos, documentos (/)

API y servicios

+ HABILITAR API Y SERVICIOS

API y servicios habilitados

Biblioteca

Credenciales

Pantalla de consentimiento ...

Verificación del dominio

Acuerdos de uso de páginas

Tráfico

Errores

Filtro

Nombre	Unidad	Errores (%)	1ª estadística mediana (ms)	95ª estadística (ms)
--------	--------	-------------	-----------------------------	----------------------

5: Definir credenciales

Google Cloud proyecto geophp

Buscar Productos, recursos, documentos (/)

API y servicios

Credenciales + CREAR CREDENCIALES BORRAR

API y servicios habilitados

Biblioteca

Credenciales

Pantalla de consentimiento ...

Verificación del dominio

Acuerdos de uso de páginas

Crea credenciales para acceder a tus API habilitadas. [Más información](#)

⚠ Recuerda configurar la pantalla de consentimiento de OAuth con información sobre tu app.

Claves de API

<input type="checkbox"/>	Nombre	Fecha de creación ↓
<input type="checkbox"/>	⚠ Maps API Key	20 oct 2022
<input type="checkbox"/>	⚠ Clave de API 1	14 oct 2022

Luego seleccionamos Clave de API

Google Cloud proyecto geophp

Buscar Productos, recursos, documentos (/)

API y servicios

Credenciales + CREAR CREDENCIALES BORRAR

API y servicios habilitados

Biblioteca

Credenciales

Pantalla de consentimiento ...

Verificación del dominio

Acuerdos de uso de páginas

Crea credenciales para acceso

Recuerda configuración

Claves de API

Clave de API
Identifica tu proyecto con una clave de API simple para verificar la cuota y el acceso

ID de cliente de OAuth
Solicita el consentimiento del usuario para que tu app pueda acceder a sus datos

Cuenta de servicio
Habilita la autenticación de servidor a servidor en el nivel de la app mediante cuentas robot

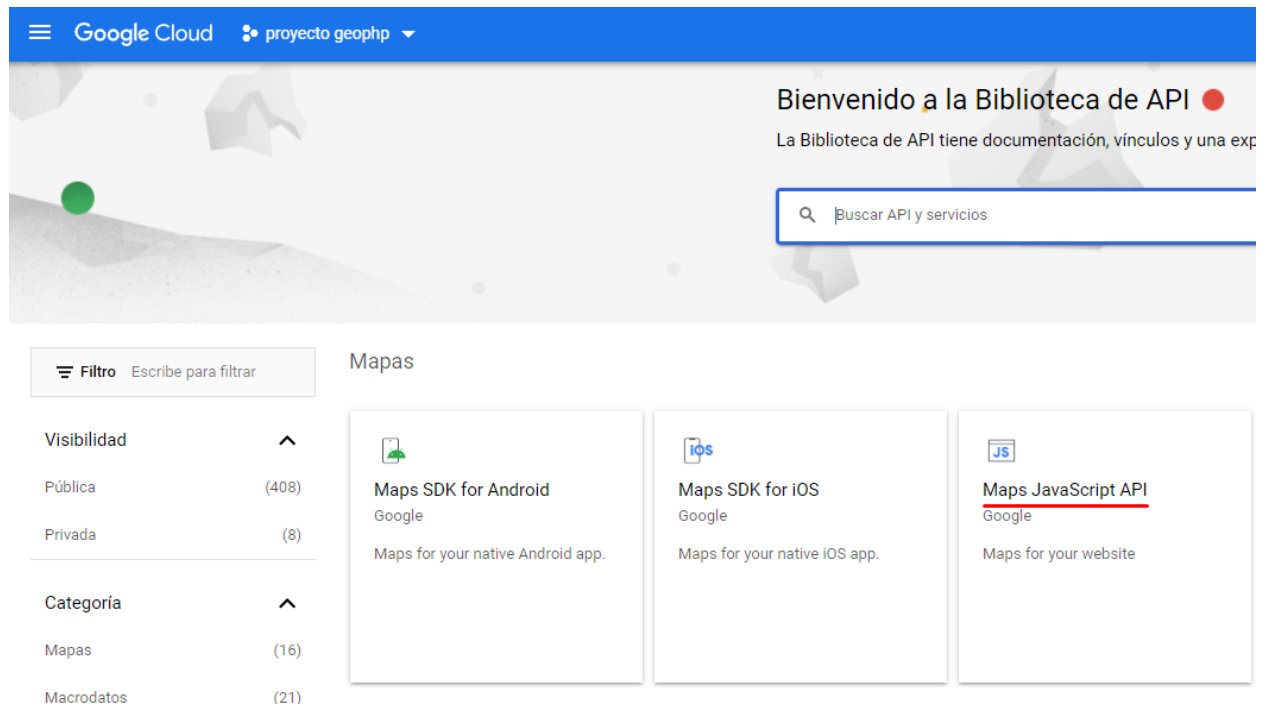
Ayúdame a elegir
Responde algunas preguntas para decidir qué tipo de credencial usar

Nombre

Clave de API



6: Elegir API y servicios (en este caso usamos Maps Javascript API), para nuestro sitio web.



Funciones de Google Maps

- Crear mapa

```
var miMapa = new google.maps.Map(document.getElementById('mapa'), {  
    center: { lat: parseFloat(latitud), lng:  
parseFloat(longitud) },  
    zoom: 6  
});
```

- Dibujar una coordenada

```
var coordenada = new google.maps.Coordinate({  
  
    path: verticeCoordenada,  
  
    map: miMapa,  
  
    strokeColor: 'rgb(255, 0, 0)',  
  
    fillColor: 'rgb(255, 255, 0)',  
  
    strokeWeight: 4,  
  
});
```

- Dibujar una línea y polilínea

```
var polilinea = new google.maps.Polyline({  
  
    path: verticesLinea,  
  
    map: miMapa,  
  
    strokeColor: 'rgb(255, 0, 0)',  
  
    fillColor: 'rgb(255, 255, 0)',  
  
    strokeWeight: 4,  
  
});  
  
var marcadorPuntoMedio = new google.maps.Marker({  
  
    position: {lat: parseFloat(latPuntoMedio), lng:  
parseFloat(lonPuntoMedio)},  
  
    map: miMapa,  
  
    title: 'Punto medio: lat'+parseFloat(latPuntoMedio)+' lng: '+  
parseFloat(lonPuntoMedio)  
  
});
```



```
var polilinea = new google.maps.Polyline({  
    path: verticesLinea,  
    map: miMapa,  
    strokeColor: 'rgb(255, 0, 0)',  
    fillColor: 'rgb(255, 255, 0)',  
    strokeWeight: 4,  
});
```

- Dibujar un polígono

```
var poligono = new google.maps.Polygon({  
    path: verticesPoligono1,  
    map: miMapa,  
    strokeColor: 'rgb(255, 0, 0)',  
    fillColor: 'rgb(255, 255, 0)',  
    strokeWeight: 4,  
});
```