Documentación de la API de Google Maps

Mapa

```
mapa = new google.maps.Map(document.getElementById("mapa"), {...});
```

La clase JavaScript que representa un mapa es la clase `Map`. Los objetos de esta clase definen un solo mapa en una página. (Puedes crear más de una instancia de esta clase; cada objeto definirá un mapa separado en la página). Se crea una nueva instancia de esta clase usando el operador de JavaScript `new`.

Al crear una nueva instancia de un mapa, se especifica en la página un elemento `<div>` de HTML como contenedor para dicho mapa. Los nodos HTML son hijos del objeto `document`

de JavaScript, y se obtiene una referencia a este elemento a través del método `document.getElementById()`.

Este código define una variable (llamada `mapa`) y la asigna a un nuevo objeto `Map`. La función `Map()` es conocida como *constructor* y a continuación se muestra su definición:

Map(mapDiv:Node,opts?:*MapOptions*): Crea un nuevo mapa dentro del contenedor HTML en cuestión (que normalmente es un elemento DIV) mediante la transferencia de parámetros (opcionales).

Hay dos opciones obligatorias para todos los mapas: `center` y `zoom`.

El centro del mapa se especifica en el campo center mediante un valor de tipo LtnLng(coordenadas)

La resolución inicial con la que debe mostrarse el mapa se configura a través de la propiedad `zoom`, en la cual `0` corresponde a un mapa de la Tierra con el máximo zoom de alejamiento, y con niveles de zoom más altos se aplica zoom de acercamiento a una mayor resolución. Especifica el nivel de zoom como un número entero.

Geocodificación

Solicitudes de geocodificación

El acceso al servicio de geocodificación es asincrónico, ya que la Google Maps API debe realizar una llamada a un servidor externo. Por esta razón, debes pasar un método *callback* para la ejecución al completarse la solicitud. Este método callback procesa los resultados. Tené en cuenta que el geocodificador puede devolver más de un resultado.

Puedes acceder al servicio de geocodificación de la Google Maps API dentro de tu código a través del objeto `google.maps.Geocoder`. El método `Geocoder.geocode()` inicia una solicitud dirigida al servicio de geocodificación y pasarle un literal de objeto `GeocoderRequest` que contenga los términos introducidos y un método callback para la ejecución al recibir la respuesta.

El literal de objeto `GeocoderRequest` contiene un único parámetro obligatorio:

- address: dirección que deseas geocodificar.

Hay más campos que podés agregar, pero con el campo dirección ya podrás hacer una búsqueda de coordenadas.
Para ver los demás métodos podés visitar la documentación oficial de la API de Google Maps.

Respuestas de geocodificación

El servicio de geocodificación exige que se ejecute un método callback al recuperarse resultados del geocodificador. Este callback debe pasar dos parámetros para contener `results` y un código `status`, en este orden.

Resultados de geocodificación

El objeto `GeocoderResult` representa un solo resultado de geocodificación. Una solicitud de geocódigo puede devolver varios objetos de resultados:

Cada objeto dentro de resultados[] tiene (entre otras cosas) un campo llamado geometry con su ubicación.

```
results[]: {
 geometry: {
   location: LatLng,
   location_type: GeocoderLocationType
   viewport: LatLngBounds,
   bounds: LatLngBounds
 }
}
```

Estos campos se explican a continuación:

- `geometry` contiene la siguiente información:
  - `location` contiene el valor de latitud y longitud geocodificado. Tené en cuenta que esta ubicación se devuelve como un objeto `LatLng`, no como una cadena con formato. Para entender cómo es el objeto LatLng hacé clic acá.

El geocodificador devolverá direcciones con la configuración de idioma preferida del navegador o el idioma especificado al cargar el código de JavaScript de la API mediante el parámetro `language`. (Para obtener más información, consulta Localización).

Códigos de estado

El código `status` puede devolver uno de los siguientes valores:

- `"OK"` indica que no ocurrieron errores, que la dirección se analizó correctamente y que se devolvió al menos un geocódigo.
- `ZERO_RESULTS` indica que el geocódigo fue exitoso, pero no devolvió resultados. Esto puede ocurrir si se pasa un valor `address` inexistente al geocodificador.
- `"OVER_QUERY_LIMIT"` indica que excediste tu cuota.
- `"REQUEST_DENIED"` indica que se rechazó tu solicitud.
- `"INVALID_REQUEST"` generalmente indica que falta la consulta (`address`, `components` o `latlng`).
- `"UNKNOWN_ERROR"` indica que no se pudo procesar la solicitud por un error en el servidor. La solicitud puede tener éxito si realizas un nuevo intento.

Para más información del geocodificador podés entrar a https://developers.google.com/maps/documentation/javascript/geocoding?hl=es-419

Clase LatLng

Un objeto `LatLng` representa un punto en coordenadas geográficas: latitud y longitud.

- La latitud oscila entre -90 y 90 grados, inclusive. Los valores por encima o por debajo de este rango se sujetarán a la gama [-90, 90]. Esto significa que si el valor especificado es menor que -90, se establecerá en -90. Y si el valor es mayor que 90, se establecerá en 90.
- La longitud oscila entre -180 y 180 grados, inclusive. Los valores por encima o por debajo de este rango se envolverán para que caigan dentro del rango. Por ejemplo, un valor de -190 se convertirá a 170. Un valor de 190 se convertirá en -170. Esto refleja el hecho de que las longitudes se envuelven alrededor del globo.

Aunque la proyección de mapa por defecto asocia la longitud con la coordenada x del mapa y la latitud con la coordenada y, la coordenada de latitud siempre se escribe primero, seguida por la longitud.

Observe que no puede modificar las coordenadas de un `LatLng`. Si desea calcular otro punto, debe crear uno nuevo.

La mayoría de los métodos que aceptan objetos `LatLng` también aceptan un objeto `LatLngLiteral`, de modo que los siguientes son equivalentes:

```
map.setCenter(new google.maps.LatLng(-34, 151));

map.setCenter({lat: -34, lng: 151});
```

El constructor también acepta objetos literales y los convierte a instancias de LatLng:

```
miLatLng = new google.maps.LatLng({lat: -34, lng: 151});
```

Marcadores

Los marcadores identifican una ubicación en un mapa. De manera predeterminada, los marcadores llevan una imagen estándar. En estos pueden mostrarse imágenes personalizadas. En este caso, generalmente reciben la denominación de “iconos”. Los marcadores y los iconos son objetos del tipo `Marker`. Puedes configurar un icono personalizado dentro del constructor del marcador, o bien llamando al método `setIcon()` en el marcador. A continuación, puedes obtener más información sobre la personalización de imágenes de marcadores.

En términos generales, los marcadores son una clase de superposición. Para obtener información sobre otros tipos de superposiciones, consulta Cómo dibujar en el mapa.

Los marcadores están diseñados para ser interactivos. Por ejemplo, de manera predeterminada reciben eventos `'click'` a fin de que puedas agregar un receptor de eventos para activar una ventana de información en la que se muestren datos personalizados. Puedes permitir que los usuarios muevan un marcador del mapa fijando la propiedad `draggable` en el valor `true`. Para obtener más información acerca de los marcadores que pueden arrastrarse, consulta la sección siguiente.

Agregar un marcador

El constructor `google.maps.Marker` toma un único literal de objeto *Marker options*, que especifica las propiedades iniciales del marcador.

Los campos siguientes tienen particular importancia y normalmente se configuran al construir un marcador:

- `position` (obligatorio) especifica un objeto `LatLng` que identifica la ubicación inicial del marcador. Una manera de recuperar un `LatLng` consiste en usar el servicio de geocodificación.
- `map` (opcional) especifica el `Map` en el cual debe ubicarse el marcador. Si no especificas el mapa al construir el marcador, este último se crea y no se adjunta al mapa (ni se muestra en él). Puedes agregar el marcador posteriormente llamando al método `setMap()` de este.

En el ejemplo siguiente se agrega un marcador simple a un mapa en Uluru, en la región central de Australia:

```
function initMap() {
  var miLatLng = {lat: -25.363, lng: 131.044};

  var mapa = new google.maps.Map(document.getElementById('map'), {
    zoom: 4,
    center: miLatLng
  });

  var marcador = new google.maps.Marker({
    position: myLatLng,
    map: mapa,
    title: 'Hello World!'
  });
}
```

La propiedad `title` del marcador aparecerá como información sobre herramientas.

Si no deseas pasar *Marker options* en el constructor del marcador, como alternativa pasa un objeto `{}` vacío en el último argumento del constructor.

Ver el ejemplo (marker-simple.html).

Animar un marcador

Puedes aplicar animación a los marcadores para que exhiban movimiento dinámico en varias circunstancias diferentes. Para especificar la manera en que se anima un marcador, usa su propiedad `animation`, del tipo `google.maps.Animation`. Se admiten los siguientes valores `Animation`:

- `DROP` indica que el marcador debe desplazarse hacia abajo, desde la parte superior del mapa hasta su ubicación final, al disponerse en él por primera vez. La animación se detendrá una vez que el marcador quede en reposo y se restablecerá el valor `null` de `animation`. Este tipo de animación generalmente se especifica durante la creación de `Marker`.
- `BOUNCE` indica que el marcador debe rebotar en el lugar. Los marcadores que rebotan continuarán haciéndolo hasta que se fije de manera explícita la propiedad `animation` en el valor `null`.

Autocompletar

- `Autocomplete` agrega un campo de ingreso de texto a tu página web y controla el ingreso de caracteres de este campo. A medida que el usuario introduce texto, el autocompletado devuelve predicciones de sitios con la forma de una lista de selección desplegable. Cuando el usuario selecciona un sitio de la lista, se devuelve al objeto Autocomplete información sobre el sitio y tu aplicación puede recuperarla. Consulta la información detallada a continuación.

Agregar autocompletado para sitios y direcciones

`Autocomplete` crea un campo de entrada de texto en tu página web, proporciona predicciones de sitios en una lista de selección de IU y devuelve información detallada sobre lugares en respuesta a una solicitud de `getPlace()`. Cada entrada de la lista de selección corresponde a un lugar (según lo definido en la Google Places API).

El constructor de `Autocomplete` toma dos argumentos, pero solo uno es requerido:

- Un elemento `input` de HTML del tipo `text`. El servicio de autocompletado controlará este campo de entrada y le adjuntará sus resultados.

Para restringir la búsqueda del Autocomplete existe el método setBounds(bounds:LatLngBounds|LatLngBoundsLiteral):
Establece el área preferida dentro de la cual se devuelven los resultados del lugar. Los resultados están sesgados hacia, pero no se limitan a, esta área.

Para obtener las coordenadas de límites alrededor de una ubicación se puede crear una forma geométrica como un círculo y obtener los límites de esta.
A continuación vemos cómo construir un círculo con la API.

Círculo

El constructor del círculo se llama con

```
new google.maps.Circle( {
  center: centroDelCirculo,
  radius: radioEnMetros
})
```

Un círculo tiene dos propiedades adicionales que definen su forma:

- `center` especifica el objeto `google.maps.LatLng` del centro del círculo.
- `radius` especifica el radio del círculo en metros.

Buscar sitios cercanos

Una búsqueda de sitios cercanos te permite buscar sitios dentro de un área especificada a través de palabras claves o tipos. En una búsqueda de sitios cercanos siempre debe incluirse una ubicación, que puede especificarse con dos métodos:

- un objeto LatLngBounds;
- un área circular definida como la combinación de la propiedad `location` (que especifica el centro del círculo como un objeto `LatLng`) y un radio, medido en metros.

Una búsqueda de sitios cercanos se inicia con una llamada al método `nearbySearch()` de PlacesService. Este método devolverá un arreglo de objetos PlaceResult.

```
servicio = new google.maps.places.PlacesService(map);

servicio.nearbySearch(request, callback);
```

Este método toma una solicitud con los siguientes campos:

- Ya sea:
  - `bounds`; debe ser un objeto `google.maps.LatLngBounds` que debe definir el área de búsqueda rectangular; o
  - `location` y `radius`; el primero toma un objeto `google.maps.LatLng` y el último toma un elemento íntegro simple, que representa el radio del

círculo en metros. El radio máximo permitido es de 50 000 metros. Tené en cuenta que cuando `rankBy` se fija en DISTANCE, debes especificar una `location`, pero no puedes especificar `radius` ni `bounds`.

- `keyword` (*opcional*): un término para coincidir con todos los campos disponibles, incluidos, entre otros, nombre, tipo y dirección, como así también revisiones del cliente y otro contenido de terceros.
- `minPriceLevel` y `maxPriceLevel` (*opcional*): restringe los resultados a solo dichos sitios dentro del rango especificado. Rango de valores válidos entre 0 (más asequible) y 4 (más costoso), inclusive.
- `name` (*opcional*): un término para coincidir con los nombres de los sitios, separados por un carácter de espacio. Los resultados se restringirán a los que contienen el valor "name" aprobado. Tené en cuenta que un sitio puede tener nombres adicionales asociados, más allá de su nombre indicado. La API intentará hacer coincidir el valor `name` aprobado con todos los de estos nombres. Como resultado, los sitios se pueden devolver en los resultados cuyos nombres *indicados* no coinciden con el término de búsqueda, pero cuyos nombres asociados sí coinciden.
- `types`: restringe los resultados a sitios que coinciden con el tipo especificado. Solo se puede especificar un tipo (si se proporciona más de un tipo, se ignoran todos los tipos que siguen a la primera entrada). Consulta la lista de tipos admitidos.

También debes pasar un método callback a `nearbySearch()` para administrar el objeto de resultados y una respuesta de `google.maps.places.PlacesServiceStatus`.

Servicio de Street View

Google Street View proporciona vistas panorámicas a 360 grados de ubicaciones designadas en su área de cobertura. La cobertura de la API de Street View es igual a la que ofrece la aplicación de Google Maps (`https://maps.google.com/`). La lista de ciudades admitidas de Street View se encuentra disponible en el sitio web de Google Maps.

La Google Maps JavaScript API proporciona un servicio de Street View para obtener y administrar las imágenes usadas en Street View de Google Maps. Este servicio de Street View tiene compatibilidad nativa con el navegador.

Uso de mapas de Street View

Aunque Street View puede usarse dentro de un elemento de DOM independiente, su mayor utilidad se manifiesta cuando se indica una ubicación en un mapa. De manera predeterminada, Street View se habilita a los mapas y el *control del Pegman*

aparece integrado dentro de los controles de navegación (zoom y desplazamiento). Puedes ocultar este control dentro del objeto `MapOptions` fijando el valor de `streetViewControl` en `false`. También puedes cambiar la posición predeterminada del control de Street View configurando la propiedad `streetViewControlOptions.position` del objeto `Map` en una nueva `ControlPosition`.

El control del Pegman de Street View te permite ver panoramas de Street View de manera directa dentro del mapa. Cuando el usuario mantiene presionado el botón del mouse sobre el Pegman, el mapa se actualiza y muestra contornos azules alrededor de las calles con Street View activado. Esto ofrece al usuario una experiencia similar a la que brinda la aplicación de Google Maps.

Cuando el usuario suelta el marcador del Pegman sobre una calle, el mapa se actualiza y muestra un panorama de Street View de la ubicación indicada.

Panoramas de Street View

Se admiten imágenes de Street View a través del objeto `StreetViewPanorama`, que proporciona una interfaz de API a un “visor” de Street View. Cada mapa contiene un panorama predeterminado de Street View que puedes recuperar llamando al método `getStreetView()` del mapa. Al agregar un control de Street View al mapa fijando su opción `streetViewControl` en `true`, el control del Pegman se conecta automáticamente a este panorama predeterminado de Street View.

También puedes crear tu propio objeto `StreetViewPanorama` y configurar el mapa, a fin de que lo use en lugar del valor predeterminado, fijando su propiedad `streetView` de manera explícita en ese objeto construido. Probablemente desees invalidar el panorama predeterminado si deseas modificar el comportamiento predeterminado, como el uso compartido de superposiciones entre el mapa y el panorama. (Consulta Superposiciones dentro de Street View, a continuación).

Contenedores de Street View

Como alternativa, es posible que desees mostrar un `StreetViewPanorama` dentro de un elemento de DOM element separado; a menudo, un `<div>`. Simplemente, pasa el elemento de DOM dentro del constructor de `StreetViewPanorama`. Para que la visualización de imágenes se óptima, recomendamos un tamaño mínimo de 200 por 200 píxeles.

Ubicaciones y punto de vista (POV) de Street View

El constructor de `StreetViewPanorama` también te permite configurar la ubicación y el punto de vista de Street View usando el parámetro `StreetViewOptions`. Puedes llamar a `setPosition()` y `setPov()` en el objeto después de la construcción para cambiar su ubicación y POV.

La ubicación de Street View define la ubicación del foco de la cámara de una imagen, pero no establece la orientación de la cámara para dicha imagen. Para este propósito, el objeto `StreetViewPov` define dos propiedades:

- `heading` (el valor predeterminado es `0`) define el ángulo de rotación alrededor del sitio de la cámara en grados respecto del norte geográfico. Los encabezados se miden en sentido horario (el punto de 90 grados representa el este verdadero).
- `pitch` (el valor predeterminado es `0`) define la variación de ángulo "hacia arriba" o "hacia abajo" a partir de la inclinación inicial predeterminada de la cámara, que a menudo (no siempre) es horizontal y plana. (Por ejemplo, una imagen tomada en una colina posiblemente exhiba una inclinación predeterminada que no es horizontal). Los ángulos de inclinación se miden con valores positivos que apuntan hacia arriba (hasta +90 grados en línea recta hacia arriba y ortogonal respecto de la inclinación predeterminada) y valores negativos que apuntan hacia abajo (hasta -90 grados en línea recta hacia abajo y ortogonales respecto de la inclinación predeterminada).

El objeto `StreetViewPov` se usa con mayor frecuencia para determinar el punto de vista de la cámara de Street View. También puedes determinar el punto de vista del fotógrafo (normalmente, la dirección en que apuntaba el auto o vehículo "trike") con el método `StreetViewPanorama.getPhotographerPov()`.

En el ejemplo siguiente aparece un mapa de Boston con una vista inicial del Fenway Park. Si se selecciona el Pegman y se lo arrastra hasta una ubicación admitida en el mapa, cambiará el panorama de Street View:

```
function initialize() {
  var fenway = {lat: 42.345573, lng: -71.098326};
  var mapa = new google.maps.Map(document.getElementById('map'), {
    center: fenway,
    zoom: 14
  });
  var panorama = new google.maps.StreetViewPanorama(
      document.getElementById('pano'), {
        position: fenway,
        pov: {
          heading: 34,
          pitch: 10
        }
      });
  mapa.setStreetView(panorama);
}
```

Servicio de indicaciones

Puedes calcular indicaciones (usando varios métodos de transporte) con el objeto `DirectionsService`. Este objeto se comunica con el servicio de indicaciones de la Google Maps API, el cual recibe solicitudes de indicaciones y devuelve resultados computados.

Puedes administrar estos resultados de indicaciones por ti mismo o usar el objeto DirectionsRenderer para representarlos.
Al especificar el origen o el destino en una consulta de indicaciones, puedes especificar una cadena de consulta (por ejemplo, “Obelisco, Argentina” o “Darwin, NSW, Australia”), un valor LatLng o un objeto google.maps.Place.
El servicio de indicaciones puede devolver indicaciones de varias partes usando una serie de waypoints. Las indicaciones se muestran como una polilínea que dibuja la ruta en un mapa o, adicionalmente, como una serie de descripciones textuales dentro de un elemento (p. ej., “Doble a la derecha en la rampa del puente de Williamsburg”).

Solicitudes de Indicaciones

El acceso al servicio de indicaciones es asincrónico, ya que la Google Maps API debe realizar una llamada a un servidor externo. Por esta razón, debes pasar un método *callback* para la ejecución al completarse la solicitud. Este método *callback* debe procesar los resultados. Tené en cuenta que el servicio de indicaciones puede devolver más de un itinerario posible como un conjunto de routes[]separadas.

Para usar las indicaciones de la Google Maps JavaScript API, crea un objeto del tipo DirectionsService, llama a DirectionsService.route() para iniciar una solicitud dirigida al servicio de indicaciones y pásale un literal de objeto DirectionsRequest que contenga los términos introducidos y un método *callback* para la ejecución al recibir la respuesta.

El literal de objeto DirectionsRequest contiene los siguientes campos:

```
{
  origin: LatLng | String | google.maps.Place,
  destination: LatLng | String | google.maps.Place,
  travelMode: TravelMode,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
 }
```

Estos campos se explican a continuación:

- origin (obligatorio) especifica la ubicación inicial a partir de la cual deben calcularse las indicaciones. Este valor puede especificarse como un String (p. ej., “Chicago, IL”), un valor LatLng o un objeto google.maps.Place. Si usas un objeto google.maps.Place, puedes especificar un id. de sitio, una cadena de consulta o una ubicación LatLng. Puedes recuperar id. de sitios de los servicios de geocodificación, búsqueda de sitios y Autocompletado de sitios, en la Google Maps JavaScript API. Para hallar un ejemplo en el que se usen los id. de sitio del servicio de autocompletado de sitios, consulta Autocompletado de sitios e indicaciones.

- destination (obligatorio) especifica la ubicación final para la cual deben calcularse las indicaciones. Las opciones son las mismas que para el campo origin descrito antes.
- travelMode (obligatorio) especifica el modo de transporte que debe usarse al calcular indicaciones. En Modos de viaje, a continuación, se especifican valores válidos
- waypoints[] (opcional) especifica un conjunto de DirectionsWaypoint. Los waypoints modifican un trayecto haciendo que pase por las ubicaciones especificadas. Un waypoint se especifica como un literal de objeto con los campos que se muestran a continuación:
    - location especifica la ubicación del waypoint, como un LatLng, un objeto google.maps.Place o un String que llevará geocodificación.
    - stopover es un booleano que indica que el waypoint es un punto de detención en la ruta, el cual tiene el efecto de dividirla en dos.
- (Para obtener más información sobre waypoints, consulta Cómo usar waypoints en rutas a continuación).
- optimizeWaypoints (opcional) especifica que la ruta en la que se usan los waypointsproporcionados puede optimizarse si se ordenan estos waypoints con mayor eficacia. Si el valor es true, el servicio de indicaciones devolverá los waypoints reordenados en un campo waypoint_order. (Para obtener más información, consulta Cómo usar waypoints en rutas a continuación).

Hay más campos que se pueden especificar, pero nos concentraremos en estos que son los que consideramos más importantes. Para más información:

https://developers.google.com/maps/documentation/javascript/directions?hl=es-419

A continuación, se muestra un ejemplo de DirectionsRequest:

```
{
  origin: 'Chicago, IL',
  destination: 'Los Angeles, CA',
  waypoints: [
    {
      location: 'Joplin, MO',
      stopover: false
    },{
      location: 'Oklahoma City, OK',
      stopover: true
    }],
  provideRouteAlternatives: false,
  travelMode: 'DRIVING',
 }
```

Modos de viaje

Al calcular indicaciones, debes especificar el modo de transporte que se usará. Actualmente, se admiten los siguientes modos de viaje:

- DRIVING (predeterminado) establece indicaciones de manejo estándar por la red de carreteras.
- BICYCLING solicita indicaciones para el traslado en bicicleta por ciclovías y calles preferidas.
- TRANSIT solicita indicaciones por rutas de transporte público.
- WALKING solicita indicaciones de traslado a pie por sendas peatonales y veredas.

Consulta los datos de cobertura de Google Maps para determinar el punto hasta el cual un país admite indicaciones. Si solicitas indicaciones para una región en la cual el tipo de indicaciones en cuestión no está disponible, en la respuesta se devolverá `DirectionsStatus="ZERO_RESULTS"`.

Representación de indicaciones

Si se desea iniciar una solicitud de indicaciones para `DirectionsService` con el método `route()`, es necesario pasar un *callback* que se ejecute al completarse la solicitud de servicio. Este *callback* devolverá un `DirectionsResult` y un código `DirectionsStatus` en la respuesta.

Solicitud de estado of indicaciones

`DirectionsStatus` puede devolver los siguientes valores:

- `OK` indica que la respuesta contiene un `DirectionsResult` válido.
- `NOT_FOUND` indica que no se pudo geocodificar al menos a una de las ubicaciones especificadas en el origen, el destino o los waypoints de la solicitud.
- `ZERO_RESULTS` indica que no fue posible hallar una ruta entre el origen y el destino.
- `MAX_WAYPOINTS_EXCEEDED` indica que se proporcionaron demasiados campos `DirectionsWaypoint` en `DirectionsRequest`. Consulta la sección sobre límites de waypoints más adelante.
- `INVALID_REQUEST` indica que el `DirectionsRequest` proporcionado no fue válido. Estos códigos de error se deben con mayor frecuencia a la falta un origen o un destino en las solicitudes, o bien a la inclusión de waypoints en las mismas.
- `OVER_QUERY_LIMIT` indica que la página web ha enviado demasiadas solicitudes dentro del período de tiempo permitido.
- `REQUEST_DENIED` indica que la página web no puede usar el servicio de indicaciones.
- `UNKNOWN_ERROR` indica que no se pudo procesar una solicitud de indicaciones debido a un error en el servidor. La solicitud puede tener éxito si realizas un nuevo intento.

Debes asegurarte de que la solicitud de indicaciones devuelva resultados válidos verificando este valor antes de procesar el resultado.

Visualización de DirectionsResult

DirectionsResult contiene el resultado de la solicitud de indicaciones. Puedes administrarlo por ti mismo o pasarlo a un objeto DirectionsRenderer, el cual puede administrar en forma automática la visualización de dicho resultado en un mapa.
Para visualizar un DirectionsResult usando un DirectionsRenderer, simplemente debes hacer lo siguiente:

1. Crea un objeto DirectionsRenderer.
2. Llama a setMap() en el representador para vincularlo al mapa transferido.
3. Llama a setDirections() en el representador y pásale el DirectionsResult, como se indicó antes. Debido a que el representador es un MVCObject, detectará en forma automática los cambios realizados en sus propiedades y actualizará el mapa cuando sus indicaciones asociadas se hayan modificado.

Guía parte 1: Creá tu mapa y empezá tu aplicación
Preparación
Para este proyecto vas a tener que trabajar sobre los archivos de los recursos descargables. Te sugerimos revisar el código de los archivos antes de empezar y trabajar de manera ordenada sobre cada uno.

Te recomendamos leer la documentación de la API oficial cuando quieras incorporar nuevas funcionalidades. Recordá que si te perdés podés ver la documentación creada por el equipo de contenidos de Acámica.

Paso 1: Obtener la Clave API
Para poder usar la API de Google Maps en tu proyecto vas a tener que agregar al código una clave API que es un identificador único que sirve para autenticarse y poder utilizar las funcionalidades de la herramienta.

Para hacerlo sólo tenés que ir al archivo index.html y reemplazar “CLAVEAPI” por la clave que generaste en la clase anterior.

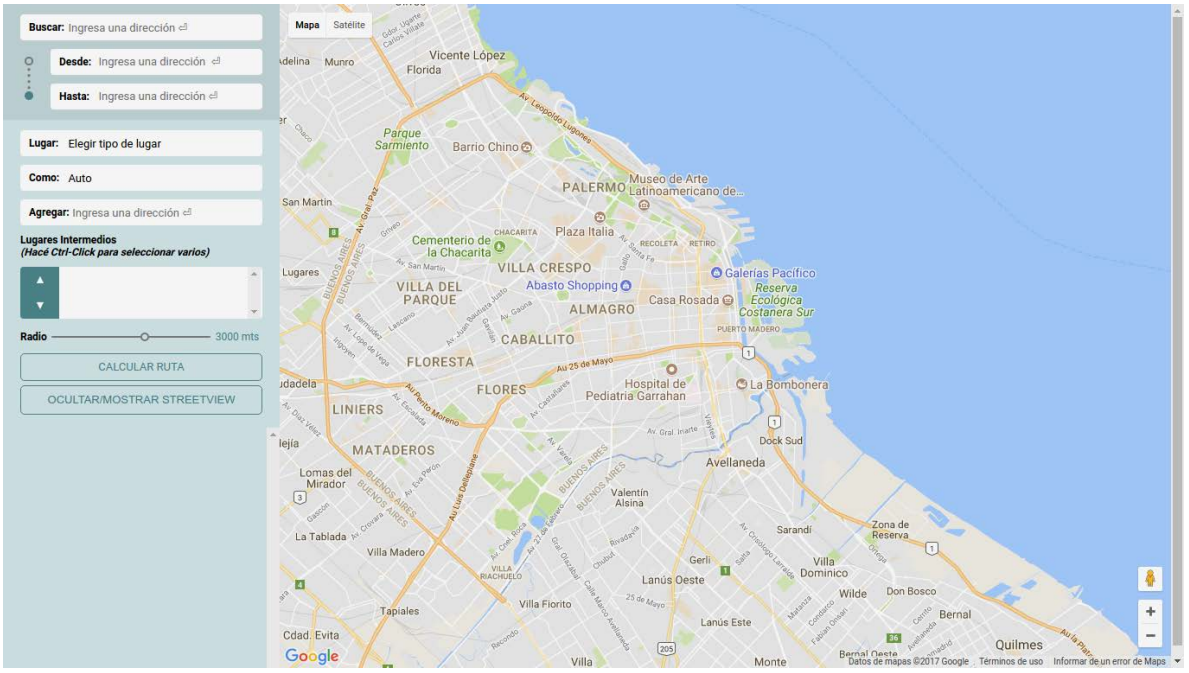
Paso 2: Crear Mapa
Vamos a comenzar completando la función inicializarMapa(), utilizando JavaScript. Esta función agregará el mapa para que se pueda mostrar en la página e inicializará los otros módulos.

A nivel lógico, llamará al constructor Map() con sus parámetros: el elemento del DOM con id = map y un diccionario con la posicionCentral y el zoom. Para esto tendrás que seguir los siguientes pasos:

Creá una variable global llamada posicionCentral con la siguiente forma: {lat: cord1, lng: cord2} donde cord1 y cord2 deberán ser reemplazadas por las coordenadas correspondientes. Estas serán donde vas a centrar tu mapa inicialmente.
En inicializarMapa(), el mapa construido con Map() y los parámetros correspondientes deberás asignarlo a tu variable mapa.
Ahora vas a poder ver un mapa con centro en las coordenadas que fijaste.
Pista: Tenés que tener en cuenta el elemento HTML del mapa ya creado en index.html. La palabra clave “new” es importante.

Nota: Para obtener las coordenadas a partir de una dirección podés usar la API de geocodificación, o a través de esta página.

Recomiendan los/as Pro: Aprovechá la documentación de las APIs. En ella se comentan las funciones que están implementadas, cuales son los parámetros necesarios, etc.



Guía Parte 2: Las primeras funcionalidades de tu mapa
Preparación
Para trabajar sobre la tercer parte de la consigna de este proyecto, necesitás abrir los siguientes archivos que se encuentran en la carpeta js :

geocodificador.js
lugares.js
marcadores.js
Recordá que los cambios que hagas sobre este proyecto se van a ver abriendo en tu navegador el archivo index.html.

Paso 1: Obtener las Coordenadas
Ahora ya podemos visualizar un mapa. La primera funcionalidad que vas a agregar es la de mostrar marcadores cuando se selecciona una dirección en el mapa. Para esto, primero es necesario que tu aplicación traduzca una dirección a una serie de coordenadas. Sin las coordenadas, el programa no entenderá dónde tiene que mostrar el marcador.

Entonces, vamos a empezar por programar una función en el archivo geocodificador.js que, dada una dirección, obtenga las coordenadas de esta. Para eso deberás completar la función usaDireccion(geocodificador, direccion, funcionAllamar) que usará el geocodificador para averiguar las coordenadas de dirección y llamará a funcionAllamar.

Ojo: Por como está implementada la función funcionALlamar siempre se llama con los siguientes parámetros
https://developers.google.com/maps/documentation/javascript/geocoding?hl=es-419

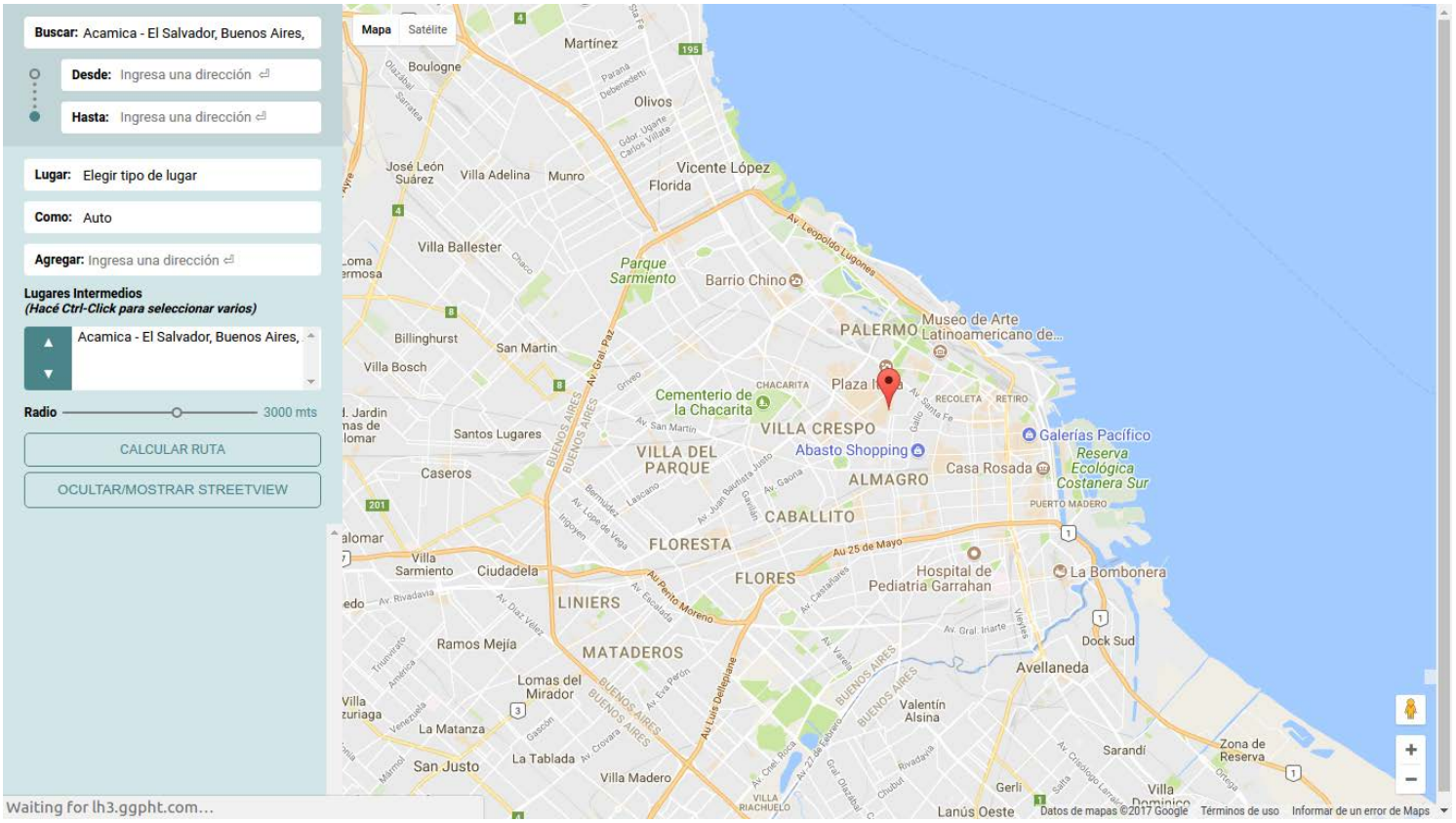
```
 var geocoder;
 var map;
 function initialize() {
  geocoder = new google.maps.Geocoder();
  var latlng = new google.maps.LatLng(-34.397, 150.644);
  var mapOptions = {
   zoom: 8,
   center: latlng
  }
  map = new google.maps.Map(document.getElementById('map'), mapOptions);
 }

 function codeAddress() {
  var address = document.getElementById('address').value;
  geocoder.geocode( { 'address': address}, function(results, status) {
   if (status == 'OK') {
    map.setCenter(results[0].geometry.location);
    var marker = new google.maps.Marker({
        map: map,
        position: results[0].geometry.location
    });
   } else {
    alert('Geocode was not successful for the following reason: ' + status);
   }
  });
 }

<body onload="initialize()">
 <div id="map" style="width: 320px; height: 480px;"></div>
 <div>
  <input id="address" type="textbox" value="Sydney, NSW">
  <input type="button" value="Encode" onclick="codeAddress()">
 </div>
</body>
```

Dirección: la dirección pasada por parámetro.
Coordenada: la ubicación de tipo google.maps.LatLng
Pista: Recordá que el geocodificador da como respuestas un arreglo de resultados, cada uno con su ubicación. Para ver como es la respuesta del geocodificador podés usar la consola.

Paso 2: Mostrar un marcador
Para mostrar el marcador hay que completar la función mostrarMiMarcador(ubicacion) ubicado en el archivo marcador.js, que cree un elemento con un título y una animación .

Pista: Para crear un marcador vas a necesitar usar un constructor de la API. Revisá la documentación y encontrá la función necesaria.

Una vez implementada mostrarMiMarcador(ubicacion), deberías ver el mapa así:
https://developers.google.com/maps/documentation/javascript/markers?hl=es-419#icons



Paso 3: Buscar lugares cercanos
Vamos a empezar por agregar la funcionalidad de buscar lugares de algún tipo en especial, alrededor de la ubicación de un marcador. Los tipos de lugares son una categoría de cada establecimiento que los mismos locales definen en Google Maps, pudiendo ser restaurantes, estaciones de servicio, cines, etc. La idea es que nuestro mapa filtre solo los lugares del tipo que el usuario quiere encontrar, cerca del marcador que definió previamente.

Completá la función buscarCerca(posicion), de modo que realice una búsqueda de los lugares cercanos a una ubicación dada. La función deberá mostrar solo los elementos con el tipo de lugar y el radio especificado por el usuario.

Luego, utilizá la función marcarLugares del marcadorModulo para marcar los lugares obtenidos.
buscarCerca es utilizada desde el archivo marcador.js, en particular en la función marcar.

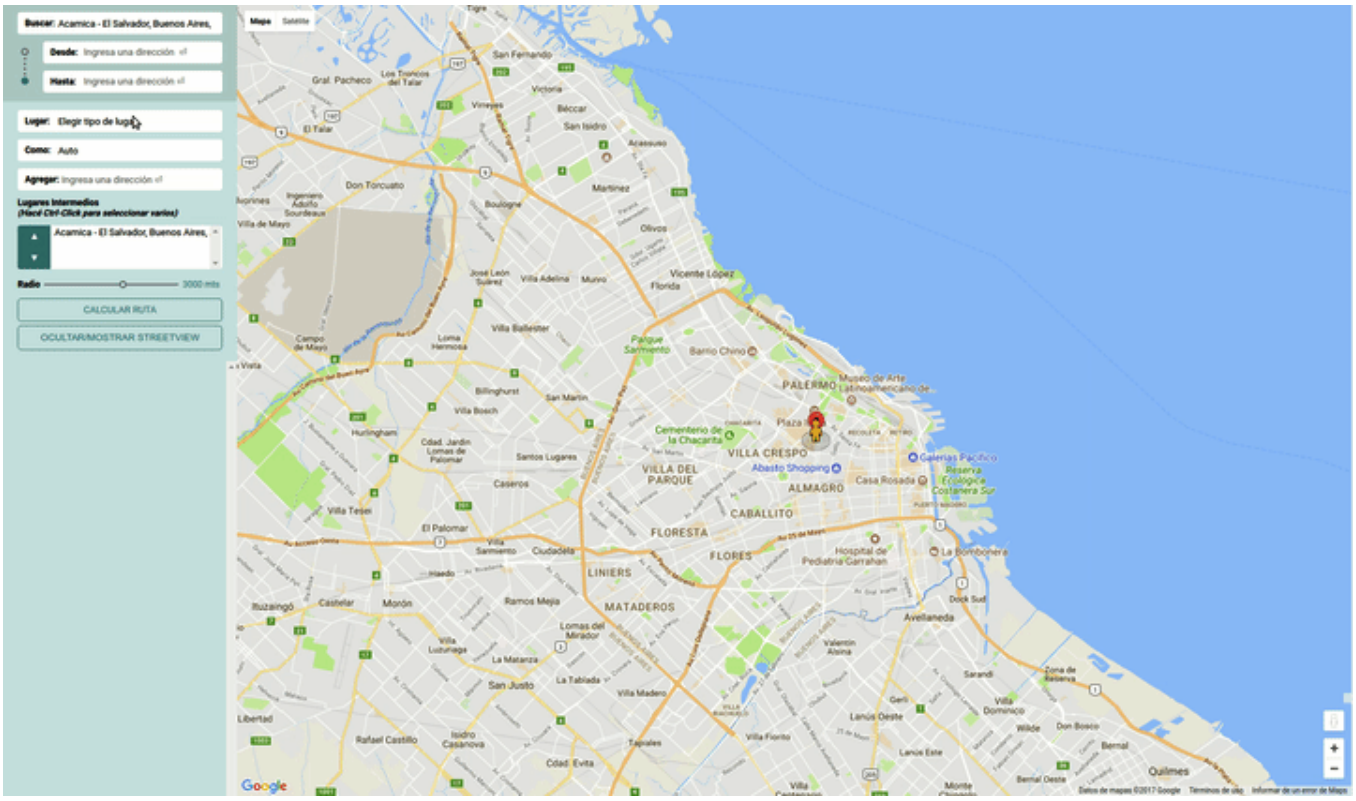
Pista: Necesitás obtener el valor de los elementos tipoDeLugar y radio.

Ahora ya podés buscar lugares cerca. La página debería verse así:

```
var map;
var service;
var infowindow;

function initialize() {
  var pyrmont = new google.maps.LatLng(-33.8665433,151.1956316);

  map = new google.maps.Map(document.getElementById('map'), {
      center: pyrmont,
      zoom: 15
    });

  var request = {
    location: pyrmont,
    radius: '500',
    type: ['restaurant']
  };

  service = new google.maps.places.PlacesService(map);
  service.nearbySearch(request, callback);
}

function callback(results, status) {
  if (status == google.maps.places.PlacesServiceStatus.OK) {
    for (var i = 0; i < results.length; i++) {
      var place = results[i];
      createMarker(results[i]);
    }
  }
}
```

https://developers.google.com/maps/documentation/javascript/places

https://developers-dot-devsite-v2-prod.appspot.com/maps/documentation/javascript/examples/place-search



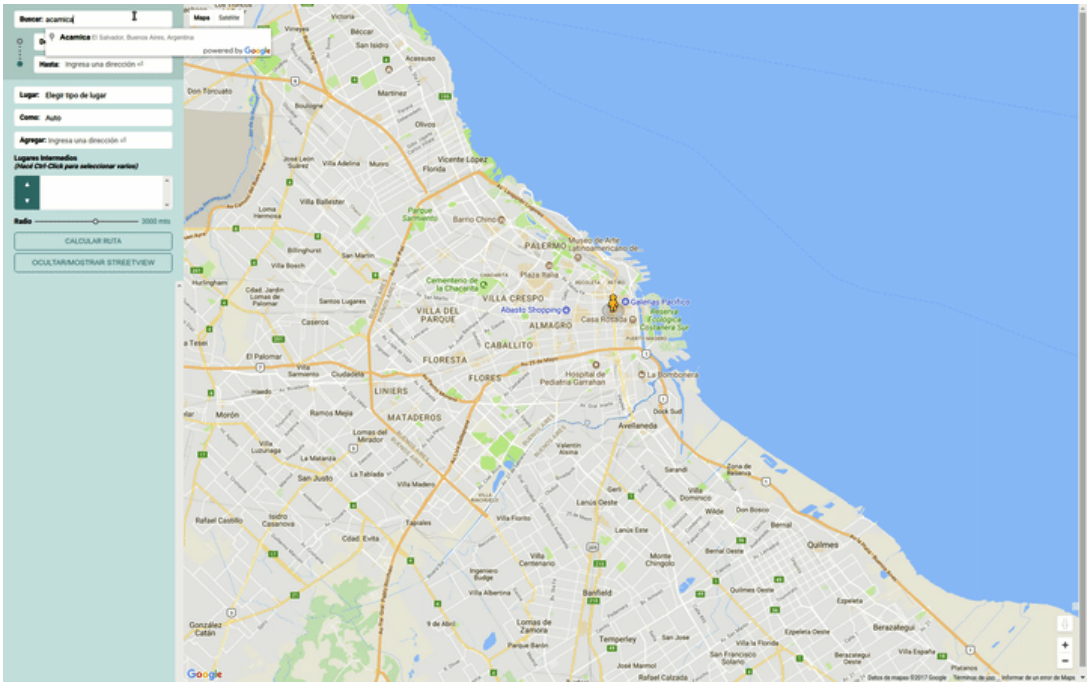
Paso 4: Crear la función para autocompletar dirección
Ahora vamos a incorporar la función para autocompletar las direcciones con los datos que tenemos disponibles en la API. Para esto, vas a tener que implementar la función autocompletar(), ubicada en el archivo en lugares.js. Esta se ejecutará cada vez que el usuario empiece a escribir cualquier dirección y hará que el campo de texto se autocomplete.

Para limitar el campo de búsqueda y evitar que la aplicación haga sugerencias innecesarias, vamos a establecer un límite territorial dentro del cual va a buscar para autocompletar. Estos límites los podés crear haciendo un círculo (de Google Maps) con un cierto radio. Recomendamos que este sea 20000 metros para buscar sobre toda una ciudad, aunque lo podés modificar a tu gusto.

Nota: Este círculo tiene que crearse sin que el usuario final lo vea, por lo que tendría que estar oculto.

Pista: Ver en la documentación la clase Autocomplete de google.maps.places.

Una vez realizado los pasos, deberías poder buscar lugares o calles más fácilmente:



```
var defaultBounds = new google.maps.LatLngBounds(
  new google.maps.LatLng(-33.8902, 151.1759),
  new google.maps.LatLng(-33.8474, 151.2631));

var input = document.getElementById('searchTextField');
var options = {
  bounds: defaultBounds,
  types: ['establishment']
};

autocomplete = new google.maps.places.Autocomplete(input, options);
```

Change the bounds of an existing Autocomplete
Call setBounds() to change the search area on an existing Autocomplete.

```
// Bias the autocomplete object to the user's geographical location,
// as supplied by the browser's 'navigator.geolocation' object.
function geolocate() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position) {
      var geolocation = {
        lat: position.coords.latitude,
        lng: position.coords.longitude
      };
      var circle = new google.maps.Circle(
          {center: geolocation, radius: position.coords.accuracy});
      autocomplete.setBounds(circle.getBounds());
    });
  }
}
```

View example.Set the bounds to the map's viewport
Use bindTo() to bias the results to the map's viewport, even while that viewport changes.

```
autocomplete.bindTo('bounds', map);
```

Restrict the search to the bounds
Set the strictBounds option to restrict the results to the given bounds, even while the viewport changes.

```
var input = document.getElementById('searchTextField');
var options = {
  types: ['(cities)'],
  componentRestrictions: {country: 'fr'}
};
autocomplete = new google.maps.places.Autocomplete(input, options);
```

https://developers.google.com/maps/documentation/javascript/places-autocomplete?hl=es-419#add_autocomplete
Set the bounds to the map's viewport
Use bindTo() to bias the results to the map's viewport, even while that viewport changes.

```
autocomplete.bindTo('bounds', map);
```

Restrict the search to the bounds
Set the strictBounds option to restrict the results to the given bounds, even while the viewport changes.

```
autocomplete.setOptions({strictBounds: true})
```

Preparación
En esta guía vas a trabajar con los archivos streetView.jsy direcciones.js. Estos archivos ya están vinculados en el HTML.

Recordá tener abierto el archivo index.html en el navegador o en “HTML Preview” si estás usando Atom para poder ir viendo los cambios que vas realizando.

Recomiendan los/as pro: Releé la documentación de la API oficial cuando quieras incorporar nuevas funcionalidades. Recordá que si te perdés podés ver la documentación creada por el equipo de contenidos de Acamica.

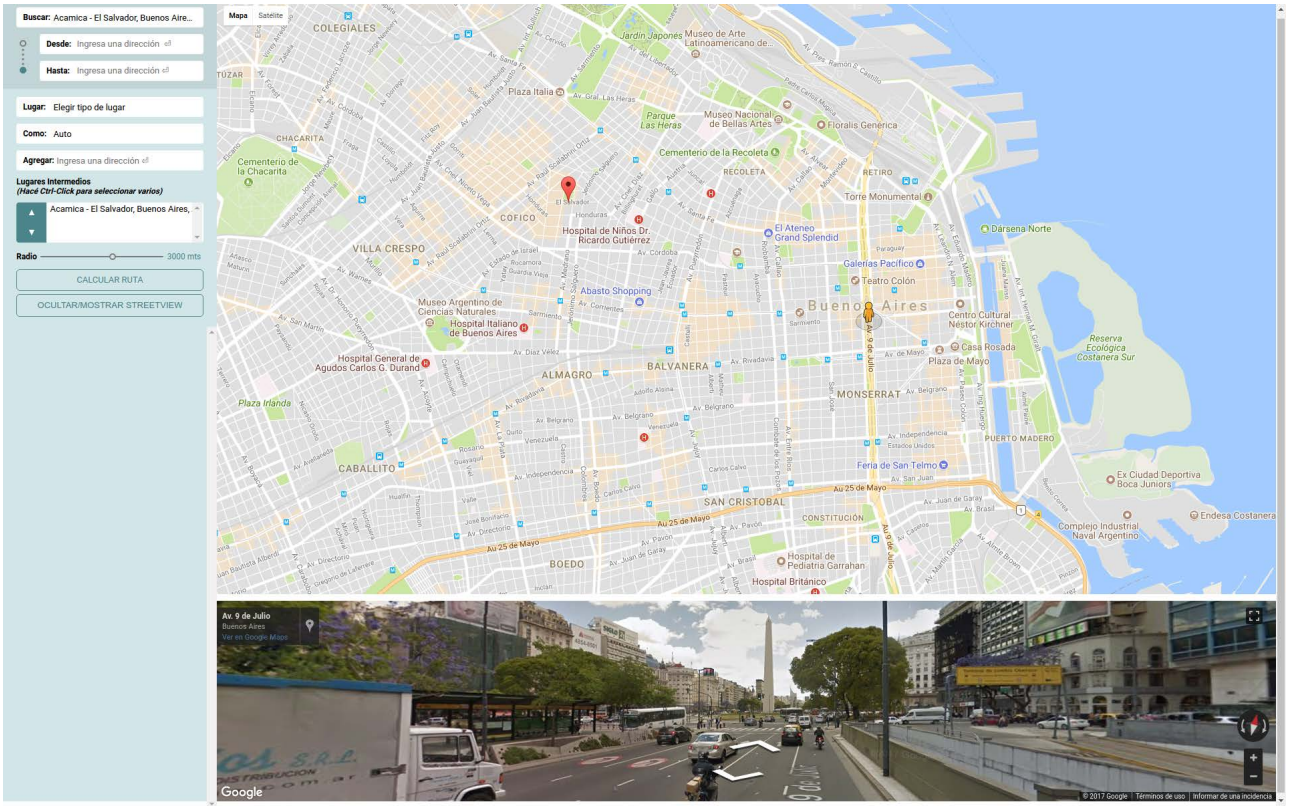
Paso 1: Agregar imágenes panorámicas con StreetView
Ahora vamos a agregar imágenes panorámicas para poder ver los lugares que nos interesan.

Para esto, dentro del archivo streetView.js. encontrarás dos funciones que deberás completar:

inicializar(): Debe crear una instancia de StreetViewPanorama con una posición, el elemento HTML donde se mostrará el mapa.
fijarStreetView(ubicacion): Actualiza la posición de la variable panorama y cambia el mapa de modo tal que se vea el streetView de la posición actual.
Ojo: En los recursos ya está implementado el botón para ocultar o mostrar el Street View. Es posible que tengas que hacer clic en el botón para visualizar la imagen panorámica.

Pista: La clase StreetViewPanorama y un método de la clase Map te servirán para cumplir la consigna. Lee sobre estos en la documentación

Ahora el mapa muestra la ubicación con una imagen 360:



Paso 2: Calcular las rutas entre dos puntos
Vamos a usar la API de Google Maps para hacer que nuestro programa calcule la ruta entre dos posiciones y la muestre en el mapa. De esta manera le damos a nuestro mapa un valor agregado para guiar al usuario a su destino deseado

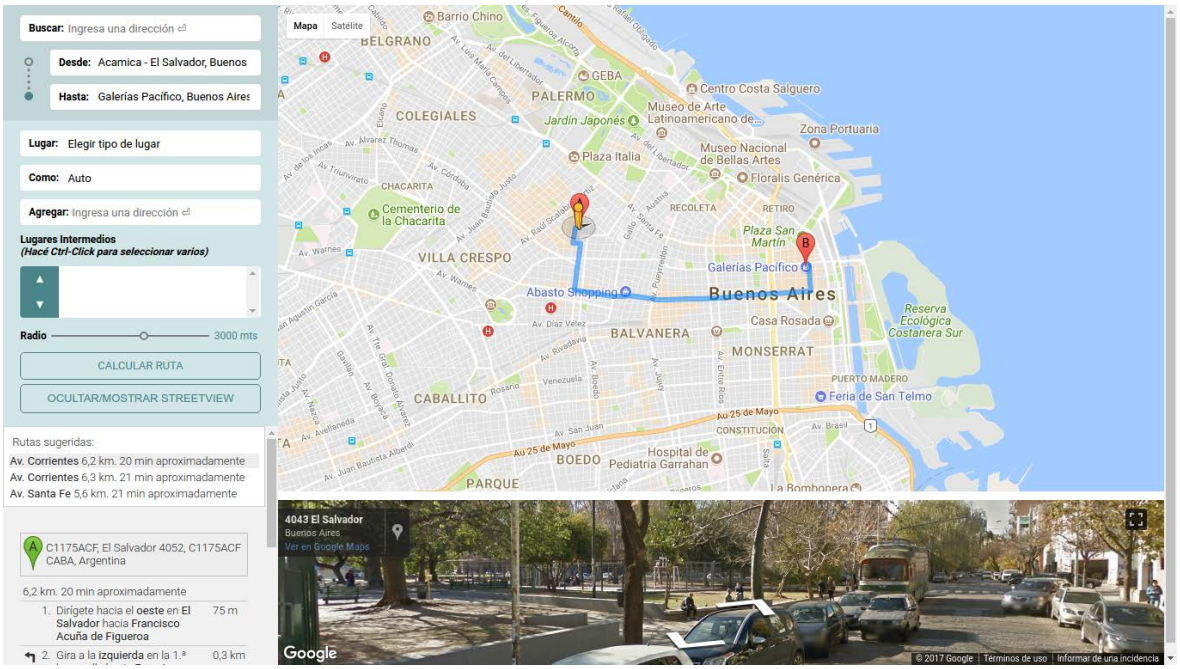
```
function initMap() {
  var directionsService = new google.maps.DirectionsService();
  var directionsRenderer = new google.maps.DirectionsRenderer();
  var chicago = new google.maps.LatLng(41.850033, -87.6500523);
  var mapOptions = {
    zoom:7,
    center: chicago
  }
  var map = new google.maps.Map(document.getElementById('map'), mapOptions);
  directionsRenderer.setMap(map);
}

function calcRoute() {
  var start = document.getElementById('start').value;
  var end = document.getElementById('end').value;
  var request = {
    origin: start,
    destination: end,
    travelMode: 'DRIVING'
  };
  directionsService.route(request, function(result, status) {
    if (status == 'OK') {
      directionsRenderer.setDirections(result);
    }
  });
}
```

```
<div>
<strong>Start: </strong>
<select id="start" onchange="calcRoute();">
  <option value="chicago, il">Chicago</option>
  <option value="st louis, mo">St Louis</option>
  <option value="joplin, mo">Joplin, MO</option>
  <option value="oklahoma city, ok">Oklahoma City</option>
  <option value="amarillo, tx">Amarillo</option>
  <option value="gallup, nm">Gallup, NM</option>
  <option value="flagstaff, az">Flagstaff, AZ</option>
  <option value="winona, az">Winona</option>
  <option value="kingman, az">Kingman</option>
  <option value="barstow, ca">Barstow</option>
  <option value="san bernardino, ca">San Bernardino</option>
  <option value="los angeles, ca">Los Angeles</option>
</select>
<strong>End: </strong>
<select id="end" onchange="calcRoute();">
  <option value="chicago, il">Chicago</option>
  <option value="st louis, mo">St Louis</option>
  <option value="joplin, mo">Joplin, MO</option>
  <option value="oklahoma city, ok">Oklahoma City</option>
  <option value="amarillo, tx">Amarillo</option>
  <option value="gallup, nm">Gallup, NM</option>
  <option value="flagstaff, az">Flagstaff, AZ</option>
  <option value="winona, az">Winona</option>
  <option value="kingman, az">Kingman</option>
  <option value="barstow, ca">Barstow</option>
  <option value="san bernardino, ca">San Bernardino</option>
  <option value="los angeles, ca">Los Angeles</option>
</select>
</div>
```

Para agregar esta funcionalidad deberás completar la función calcularYMostrarRutas() que calcula la ruta entre dos posiciones ingresadas, dependiendo de si el usuario elige ir caminando, en transporte público, o en auto. Luego, la misma función deberá mostrar la ruta en el mapa y un resumen en formato de texto en el panel.

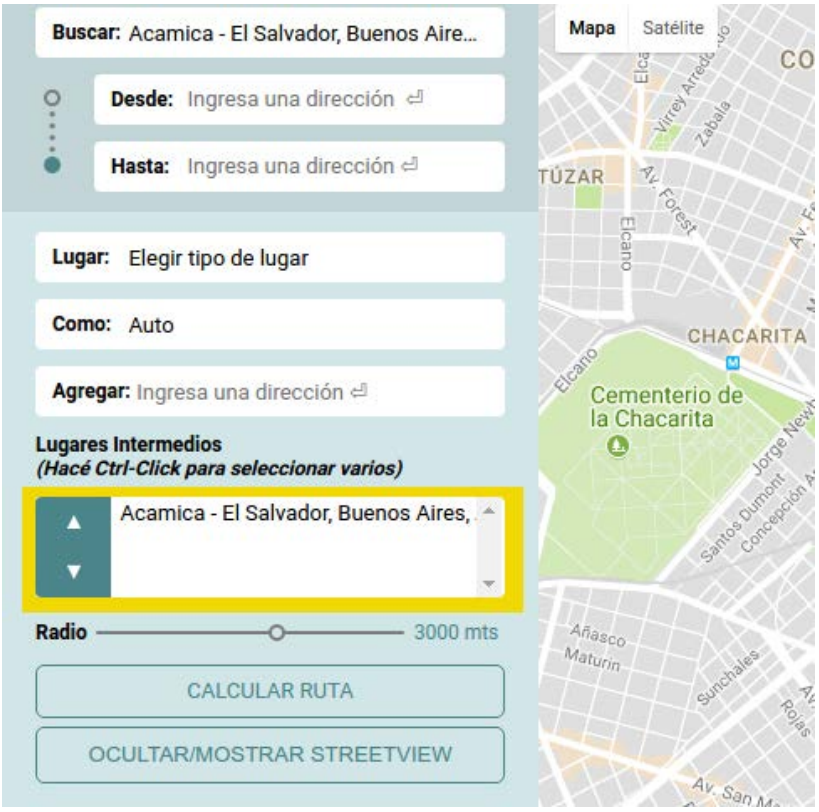
Nota: No te olvides de identificar los elementos HTML de index.html que se usan para calcular las rutas.

Pista: Para calcular una ruta necesito el origen, el destino, la forma de ir y los puntos intermedios. Las variables mostradorDirecciones y servicio Direcciones te serán útiles.



Paso 3: Calcular las rutas con puntos intermedios
Ya hiciste la función para calcular la ruta entre dos puntos. Ahora vamos a agregar la funcionalidad de agregar puntos intermedios para que el usuario pueda crear su propia ruta de viaje con todas las paradas que quiera.

Para esto deberás completar la función calcularYMostrarRutas(), que calcule la ruta con los puntos intermedios seleccionados. Los usuarios eligen los puntos intermedios del selector múltiple ubicado en el archivo index.html, y estos son los que deben ser agregados para calcular la ruta.



```
function initMap() {
  var directionsService = new google.maps.DirectionsService();
  var directionsRenderer = new google.maps.DirectionsRenderer();
  var haight = new google.maps.LatLng(37.7699298, -122.4469157);
  var oceanBeach = new google.maps.LatLng(37.7683909618184, -122.51089453697205);
  var mapOptions = {
    zoom: 14,
    center: haight
  }
  var map = new google.maps.Map(document.getElementById('map'), mapOptions);
  directionsRenderer.setMap(map);
}

function calcRoute() {
  var selectedMode = document.getElementById('mode').value;
  var request = {
      origin: haight,
      destination: oceanBeach,
      // Note that JavaScript allows us to access the constant
      // using square brackets and a string value as its
      // "property."
      travelMode: google.maps.TravelMode[selectedMode]
  };
  directionsService.route(request, function(response, status) {
    if (status == 'OK') {
      directionsRenderer.setDirections(response);
    }
  });
}


<div>
<strong>Mode of Travel: </strong>
<select id="mode" onchange="calcRoute();">
  <option value="DRIVING">Driving</option>
  <option value="WALKING">Walking</option>
  <option value="BICYCLING">Bicycling</option>
  <option value="TRANSIT">Transit</option>
</select>
</div>
```

https://developers.google.com/maps/documentation/javascript/directions?hl=es-419

*** Pista: La documentación de la API llama DirectionsWaypoint a los puntos intermedios. Notar que en el código hay una instancia de DirectionsRequest llamada servicioDirecciones. Recomendamos mirar la especificación del objeto DirectionsRequest en la documentación.

Controlá tu Mapa Interactivo

Para controlar que tu proyecto este terminado antes de entregarlo, podés usar este checklist de evaluación. Vas a encontrar todos los requerimientos que necesita cumplir tu proyecto para ser corregido y para ser aprobado.

para corregir el proyecto Mapa interactivo

Para que un proyecto apruebe, tiene que cumplir con todas las condiciones del checklist correspondiente.

Condiciones para entregar _

● Archivos mínimos entregados: En la carpeta entregada se incluyen como mínimo los archivos descargados con los recursos del proyecto *.html , *.css y *js en las carpetas correspondientes. Pueden incluirse otros archivos extra si la decisión está justificada.

● Enlace al HTML: El HTML, CSS y JS están separados en diferentes archivos y están enlazados correctamente.

Condiciones para aprobar _

● Creación del mapa: Se crea y muestra el mapa correctamente.

● Obtener coordenadas de dirección: Utilizando el módulo geocodificador se obtienen las coordenadas de una dirección correctamente.

● Mostrar Marcador: Se implementa y muestra correctamente un marcador en el mapa.

● Buscar lugares cercanos: La funcionalidad de buscar lugares cercanos funciona correctamente y se marcan en el mapa.

● Autocompletar dirección: Cuando el usuario comience a escribir deberá autocompletarse la información pertinente a un radio establecido.

● Street view: Se agregan imágenes panorámicas con streetview.

● Calcular rutas: Se calcula la ruta entre dos puntos del mapa elegidos y se la muestra en el mapa. También se implementa el cálculo de rutas con puntos intermedios