Mª DEL MAR JUSTICIA Y JOSÉ L. LECHADO 2ºDAW

PRÁCTICA DE GEOLOCALIZACIÓN

La presente práctica presenta una aplicación que permite calcular la distancia entre dos ubicaciones elegidas, al igual que el consumo de combustible y el costo de éste en función del vehículo seleccionado.

El propósito de la misma ha sido poner en práctica todos los conceptos trabajados en el presente curso en la asignatura Desarrollo Web de Entorno Cliente.

En ella, con dicho propósito, hemos tratado de aplicar todos los conceptos de Javascript aprendidos hasta la fecha; así por ejemplo, en lugar de utilizar una base de datos y php, hemos optado por crear un objeto Coche y almacenar este objeto en un array.

Vamos a explicar a continuación la parte de geolocalización de nuestro proyecto.

Aunque hemos usado distintas fuentes, la información la hemos sacado básicamente de https://developers.google.com/maps/documentation/javascript/directions

Servicio llegar

Nuestro proyecto, como ya hemos explicado se centra en el cálculo de distancias que se recorren con un coche, para ello es básico saber conseguir calcular la distancia real entre un origen y un destino.

Se pueden calcular direcciones mediante el uso del objeto `DirectionsService`.

Este objeto se comunica con el Servicio llegar API de Google Maps, que recibe peticiones de dirección y devuelve los resultados calculados.

El objeto `DirectionsRenderer` es el que nos ayuda a reproducir estos resultados.

Al especificar el origen o destino en una solicitud de ruta,podemos hacerlo de varias formas.

Bien se le puede pasar una cadena (por ejemplo, "C/Arabial Granada"), un valor, o una longitud/latitud o n objeto `google.maps.Place`

Sea como sea, el servicio puede llegar devolver direcciones de varias partes utilizando una serie de waypoints

Las indicaciones se muestran como una línea poligonal en nuestro mapa, en el que puede verse la ruta, o también como una serie de descripciones textuales en un `<div>` elemento (por ejemplo, "Gire a la derecha en la C/Arabial").

Solicitud de direcciones

El acceso al servicio de rutas es asíncrono, ya que la API de Google Maps necesita realizar una llamada a un servidor externo.

Por esa razón, es necesario aprobar una *devolución de llamada*método para ejecutar una vez finalizada la petición. Este método de devolución de llamada debe procesar la respuesta.

Para utilizar las direcciones en la API de JavaScript de Google Maps, hay que crear un objeto de tipo `DirectionsService` y llamar a `DirectionsService.route()` para iniciar una solicitud para el servicio de rutas, pasándole un objeto literal `DirectionsRequest.`

DirectiosRequest contiene los términos entrada y un método callback para recibir de la respuesta y ejecutarla.

El objero `DirectionsRequest` contiene los siguientes campos:

```
{
  origin: LatLng | String | google.maps.Place,
  destination: LatLng | String | google.maps.Place,
  travelMode: TravelMode,
  transitOptions: TransitOptions,
  drivingOptions: DrivingOptions,
  unitSystem: UnitSystem,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
  provideRouteAlternatives: Boolean,
  avoidHighways: Boolean,
  avoidTolls: Boolean,
```

```
  region: String
}
```

Estos campos se explican a continuación:

- `origen` (*requerido*) especifica la ubicación de inicio de partida para el cálculo de las direcciones.

  Utilizar un objeto `google.maps.Place` te permite especificar el origen mediante:un ID del lugar, una cadena de consulta o unos valores de longitud-latitud.

  (Puede recuperar los ID de lugar de la Place Autocompletar utilizando los servicios de autocompletado en la API de JavaScript de Google Maps.)

- `destino` (*requerido*) especifica la ubicación final de la cual se calculan las direcciones.Las opciones son los mismos que para el `origen` de campo descrito anteriormente.
- `travelMode` (requerido) especifica qué medio de transporte a utilizar en el cálculo de las direcciones. Los valores válidos están especificados en los modos de Viaje. En nuestra aplicación nos interesa el modo driving puesto que queremos viajar en coche
- `transitOptions` (*opcional*) especifica los valores que se aplican sólo a las solicitudes donde `travelMode` es `google.maps.TravelMode.TRANSIT`
- `drivingOptions` (*opcional*) especifica los valores que se aplican sólo a las solicitudes donde `travelMode` se `google.maps.TravelMode.DRIVING` . Los valores válidos se describen en opciones de ruta , a continuación.
- `unitSystem` (*opcional*) especifica el sistema de unidades a utilizar cuando se muestran los resultados. Los valores válidos se especifican en Sistemas Unidad de abajo.

Modos de viaje

Al calcular las direcciones, es necesario especificar qué modo de transporte a utilizar. En nuestro caso, al interesarnos el modo de viaje en coche usaremos:

`google.maps.TravelMode.DRIVING` (**por defecto**) indica las direcciones de conducción estándar utilizando la red de carreteras.

Sistemas de unidades

En nuestro proyecto nos interesa el METRIC ya que queremos los valores en kilómetros, pero como este campo solo afecta a cómo va a visualizar la información el usuario, dejamos también la opción de que pueda verla en millas. (El resultado de las las distancias siempre viene expresado en la respuesta en metros)

- `UnitSystem.METRIC` especifica el uso del sistema métrico. Las distancias se muestran utilizando kilómetros.
- `UnitSystem.IMPERIAL` especifica el uso del sistema imperial (Inglés). Las distancias se muestran utilizando millas.

Directions.Route

`DirectionsRoute` contiene un único resultado desde el origen y el destino especificado. Esta ruta puede consistir en una o más patas (de tipo `DirectionsLeg) en función de si se especificaron los waypoints.` A su vez, la ruta también contiene información de derechos de autor y la advertencia que debe ser mostrada al usuario, además de la información de enrutamiento.

El `DirectionsRoute` es un objeto literal con los siguientes campos:

- `legs []` contiene un conjunto de `DirectionsLeg` objetos, cada uno de los cuales contiene información sobre un tramo de la ruta, a partir de dos puntos de la ruta determinada. Una leg separada estará presente para cada waypoint o destino especificado, Cada leg consiste en una serie de `DirectionStep` s.
- `waypoint_order` contiene una matriz que indica el orden de los hitos en la ruta calculada. Esta matriz puede contener un orden alterado si el `DirectionsRequest`fue aprobada `optimizeWaypoints: true` .
- `overview_path` contiene una matriz de `LatLng` s que representan una trayectoria aproximada (suavizada) de las direcciones resultantes.
- `overview_polyline` contiene un solo objeto `puntos` que contiene una representación de la ruta polinimizada. Esta es una polilínea (suavizada) que proporciona la ruta de las direcciones resultantes aproximada.

- `límites` contiene un `LatLngBounds` que indican los límites de la polilínea a lo largo de esta ruta determinada.
- `los derechos de autor` contiene el texto que se mostrará derechos de autor para esta ruta.
- `advertencias []` contiene un conjunto de avisos que se mostrarán cuando se muestran estas direcciones.
- `la tarifa` incluye la tarifa total (es decir, los costos de las entradas totales) en esta ruta.

Inspección de elementos de Directions.Route

Es interesante comentar cómo hemos conseguido obtener la distancia de la ruta recorrida y la duración del viaje. Ya que hay una opción que te saca los datos en pantalla imprimientolos en un panel de manera automática, usando directionsDisplay.setPanel(panel); Pero nosotros no queríamos simplemente visualizarlos, sino tratarlos, para ellos necesitábamos acceder a esos datos. Veamos nuestro código:

En nuestro código para procesar nuestra respuesta usamos:

```
directionsService.route(request, function (response, status) {

  if (status == google.maps.DirectionsStatus.OK) {

        directionsDisplay.setDirections(response);

        distancia=(response.routes[0].legs[0].distance.value)/1000;

        //obtenemos el tiempo en segundos de la primera ruta

        tiempo=response.routes[0].legs[0].duration.value;llega response
```

.....}

Mediante el inspector de elementos estuvimos analizando que campos había en response, ya que queríamos recogernos la distancia que había entre origen y destino y el tiempo que se tardaba en llegar. Y vimos que estaba aquí:

distancia=(response.routes[0].legs[0].distance.value)/1000;

routes es un array que contenía las distintas rutas alternativas para el viaje, nosotros nos quedamos con la más corta que era la primera, es decir con routes[0].

Leg[0] nos da la información de esa ruta y con .distance.value podíamos acceder a la distancia que estábamos buscando.

Para el tiempo que se emplea en la ruta se procede de forma similar pero en vez de .distance.value

CÁLCULO DEL CONSUMO DEL COCHE

Una vez obtenidos los valores de distancia y duración explicados en el apartado anterior, procedemos al cálculo de combustible empleado en realizar esa ruta.

```
coche=buscarCoche(marcaCoche,modeloCoche,combustibleCoche,potenciaCoche);
if(coche==null){
    alert('No existen coches con los parametros especificados');
}else{
    var dinero=(calcularConsumo(coche,distancia,0.956)).toFixed(2);
    var gastoCombustible=(calcularCombustible(coche,distancia)).toFixed(2);
    var divDetalles=document.getElementById('detalles');
    divDetalles.textContent='DETALLES DEL VIAJE';
    var divDistancia=document.getElementById('detallesDistancia');
    divDistancia.textContent='Distancia: '+distancia+' km';
    var divConsumo=document.getElementById('detallesConsumo');
    divConsumo.textContent='Consumo: '+gastoCombustible+' litros';
    var divCosto=document.getElementById('detallesCosto');
    divCosto.textContent='Costo: '+dinero+' euros';
    var divDespedida=document.getElementById('despedida');
    divDespedida.textContent='BON VOYAGE!!!!!!';
    var cuadro=document.getElementById('cuadro');
    cuadro.setAttribute('class','marco');
```

Primero tenemos que saber si el objeto coche especificado existe en nuestro array. Para ello usamos el método buscar coche

```
function buscarCoche(marca,modelo,combustible,potencia){
    for(var i=0;i<coches.length;i++){

if(coches[i].marca==marca&&coches[i].modelo==modelo&&coches[i].combustible==combustible&&coches[i].potencia==potenc
ia){
                return coches[i];
            }
    }
    return null;
}
```

Después, si existe, procedemos a calcular el consumo que tiene ese coche.

```
function calcularConsumo(coche,dist,precio){
    var consumo=coche.consumo;
    var costo=((consumo*dist)/100)*precio;
    return costo;
}
```

Y finalmente el gasto de combustible

```
function calcularCombustible(coche,dist){
    var consumo=coche.consumo;
    var comb=(consumo*dist)/100;
    return comb;
}
```