ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

“CAPTURA Y VISUALIZACIÓN DE DATOS EN UN NAVEGADOR GPS”

Andrés Goñi Gallego

Jesús Villadangos Alonso

Pamplona, 5/10/2011

*I*NDICE DEL DOCUMENTO

1- RESUMEN

La utilización de sistemas de localización forma parte de nuestra vida cotidiana. A menudo, necesitamos situar un lugar en un mapa o conocer la ruta que debemos seguir para llegar a un lugar concreto. Estos nos permiten ahorrar tiempo, optimizando el camino entre nuestro origen y un futuro destino, nos dan grandes cantidades de información acerca de estos caminos y nos facilitan mucho nuestros desplazamientos.

El manejo de estos sistemas es algo conocido para la mayoría de personas, ya que en la mayoría de ellos presentan interfaces similares a la de cualquier página Web o aplicación de dispositivos móviles, con las cuales estamos acostumbrados a trabajar.

El propósito general de estos dispositivos es dar información muy detallada y precisa sobre una ubicación concreta, una ruta a seguir y sus diferentes incidencias, cuáles son las diferentes opciones que tenemos para desplazarnos desde un lugar a otro dependiendo de una serie de variables como pueden ser el coste o el tiempo que queramos invertir en dicho desplazamiento.

Para hacer uso de estos dispositivos no es necesario disponer de grandes medios técnicos ni conocimientos acerca de cómo funcionan internamente. Se han creado de forma que nos muestren una interfaz clara, que cualquier persona sea capaz de entender y utilizar, y que devuelvan unos resultados que el usuario interprete rápidamente. Así, sin necesidad de esforzarnos mucho podremos saber la distancia que nos separa de nuestro destino y el tiempo que invertiremos en llegar hasta él, siguiendo ciertos pasos tan solo rellenando unos pequeños detalles de ubicación.

Partiendo de este conocimiento previo de los sistemas de localización, nos adentramos en el proyecto, en el cuál se analizará cómo se puede crear uno de estos sistemas a partir de la aplicación ya existente, Google Maps, que proporciona Google.

2- INTRODUCCIÓN AL PROYECTO

Este proyecto consiste en analizar cómo se puede crear un sistema de localización utilizando una herramienta que actualmente existe y nos proporciona Google, como es Google Maps.

Esta herramienta nos muestra un mapa en el cual nosotros podemos solicitar que nos ubique una dirección concreta que nosotros le proporcionemos, que nos devuelva cómo podemos desplazarnos entre dos lugares y que además nos devuelva datos sobre dicha ruta. Esto es algo que aparentemente resulta muy sencillo para el usuario que solamente debe proporcionar una dirección, o los lugares de partida y destino.

En este proyecto lo que se intenta ver es el funcionamiento interno de estos sistemas. Es decir desde que el usuario debe introducir los datos hasta que se le devuelve un respuesta a su solicitud, cómo se utilizan dichos datos, la forma en la que se va a trabajar con ellos y procesarlos para obtener los resultados y las diferentes posibilidades de trabajo que se tienen para realizar estas tareas.

El motor de trabajo analizado ha sido Google Maps, un sistema muy complejo creado por Google que, con una cantidad mínima de información, devuelve al usuario muchos detalles anteriormente ya nombrados. Teniendo en cuenta que actualmente es utilizado por muchas personas, es interesante adentrarse a conocer cómo trabaja internamente para conseguirlo.

Una vez que conozcamos su funcionamiento, intentaremos ver cómo podemos añadirle diferentes funcionalidades para asemejarlo a un sistema de localización existente y en uso como los que podremos ver a continuación.

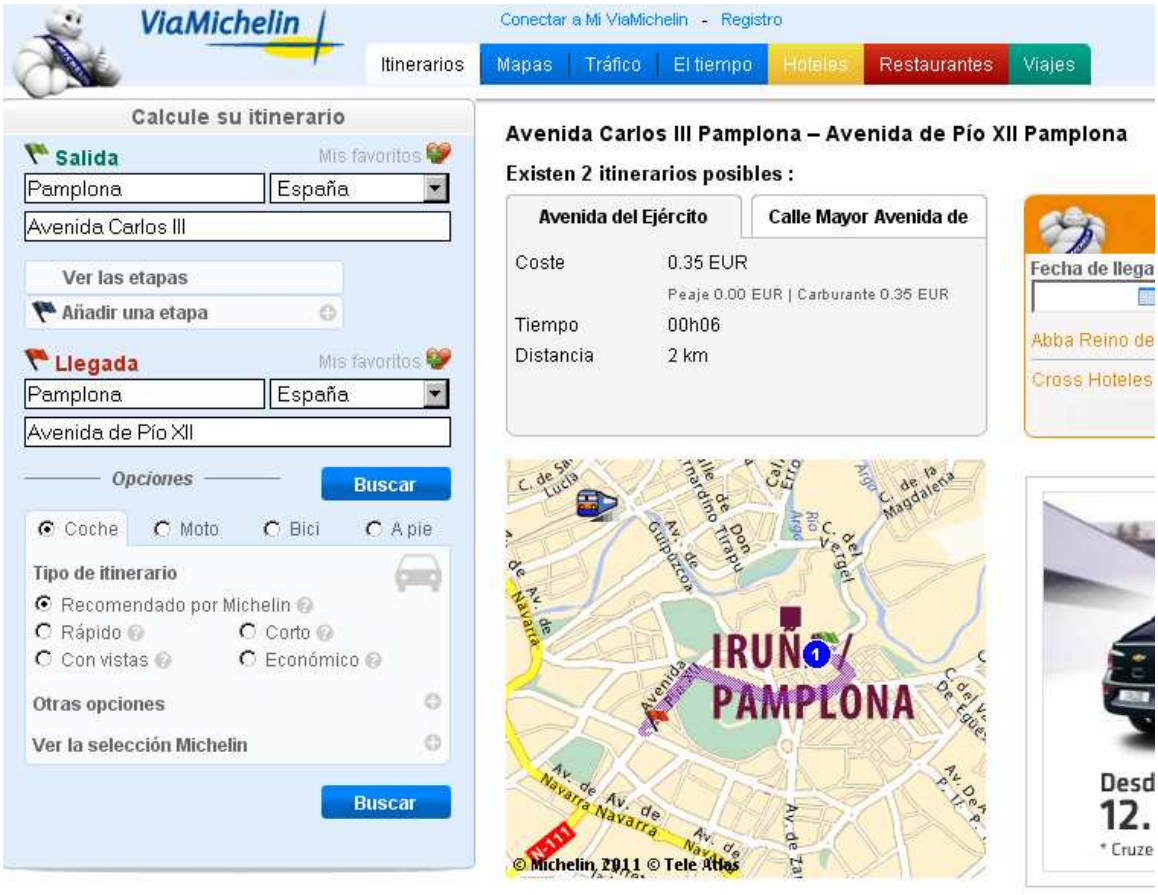
2.1- ESTADO ACTUAL

En la actualidad existen diferentes sistemas de navegación que facilitan a los usuarios trazar sus rutas, organizar sus vacaciones y planificar sus trayectos.

Antes de comenzar el proyecto necesitamos ver todo lo que nos ofrecen los sistema de navegación actuales, cuáles son las diferencias entre ellos, cuáles nos permiten añadir funcionalidades que queremos y cuáles nos permiten desarrollar nuestro propio sistema de navegación.

A continuación, vamos a estudiar los sistemas más utilizados en la actualidad y haremos una comparativa de ellos.

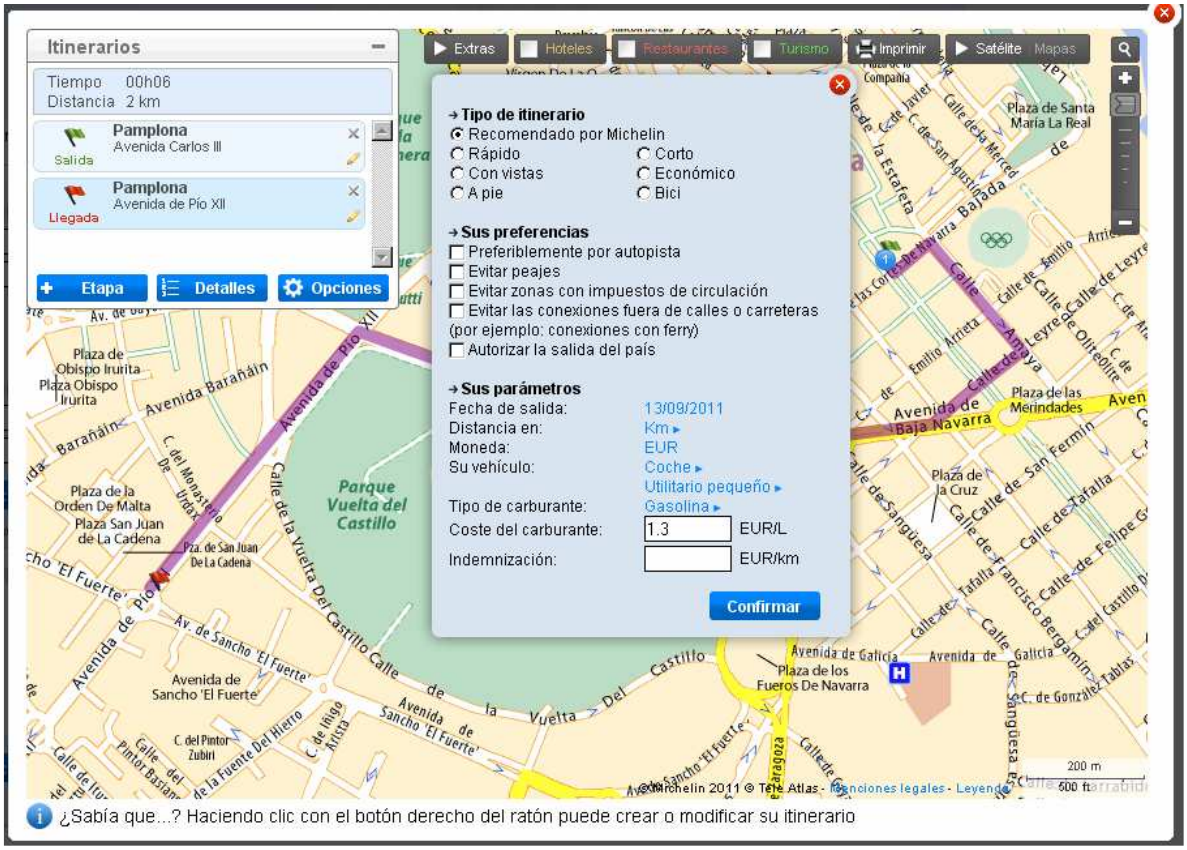
- VÍA MICHELÍN: www.**viamichelin**.es



Vía Michelín nos ofrece la posibilidad de introducir una dirección concreta de una ciudad desde la que inicia la ruta y otra a la que se llega. Además, ofrece itinerarios para ir en coche, bici, en moto o a pie y la posibilidad de elegir el tipo de itinerario (económico, rápido, corto, etc.).

Muestra, como resultado el coste, el coste de los peajes, el tiempo, la distancia y la ruta representada en el plano.

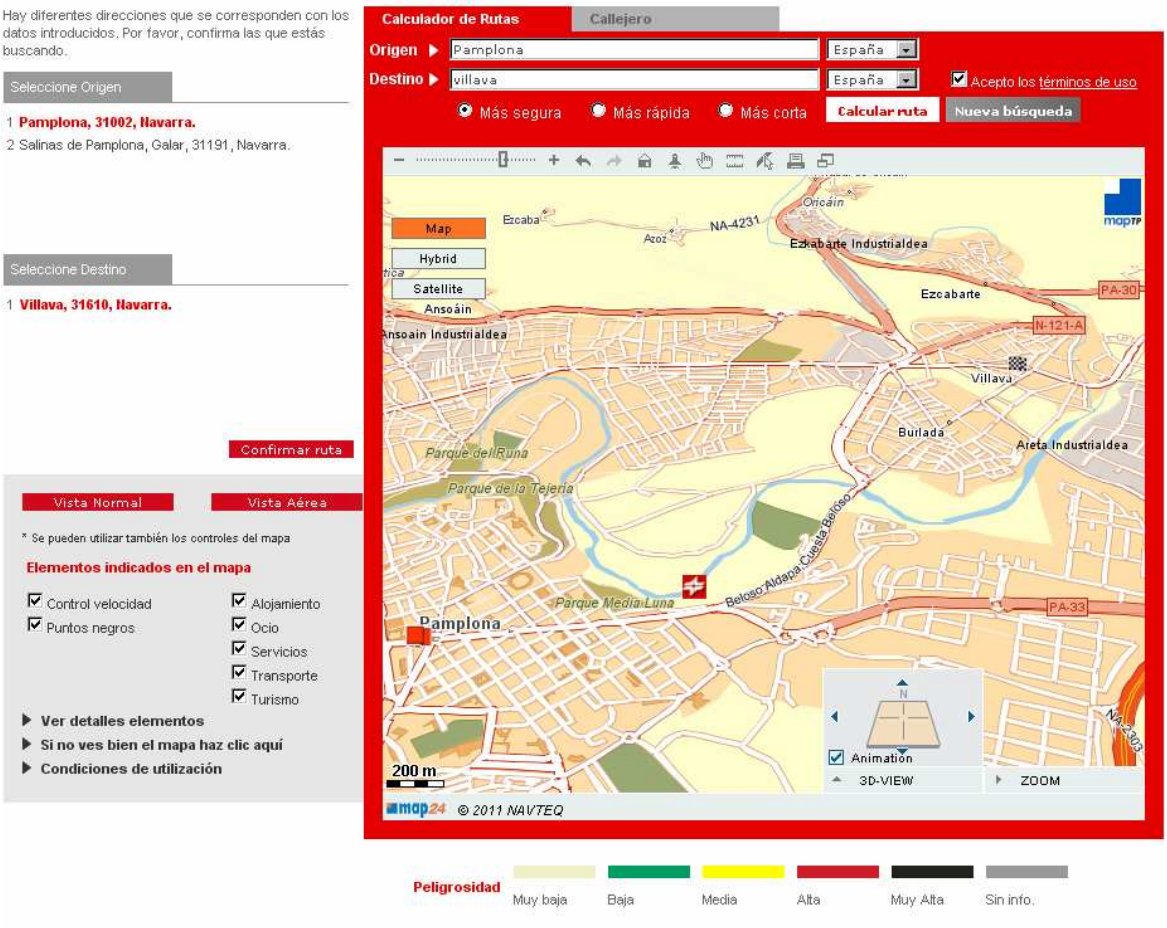
Al hacer clic sobre el mapa, la Vía Michelin ofrece también la opción de añadir una etapa al trayecto y calcular el gasto de carburante en función del precio de la gasolina.



También ofrece la posibilidad de visualizar en el mapa radares, servicios de la ciudad (restaurantes, hospitales, bares, etc.) así como una explicación detallada de la ruta.

Lo que no ofrece la Vía Michelin es la posibilidad de ver rutas alternativas para realizar ese proyecto ni una simulación del trayecto para ver gráficamente un icono que siga la ruta.

- GUIA CEPSA: www.buenviajecepsa.com



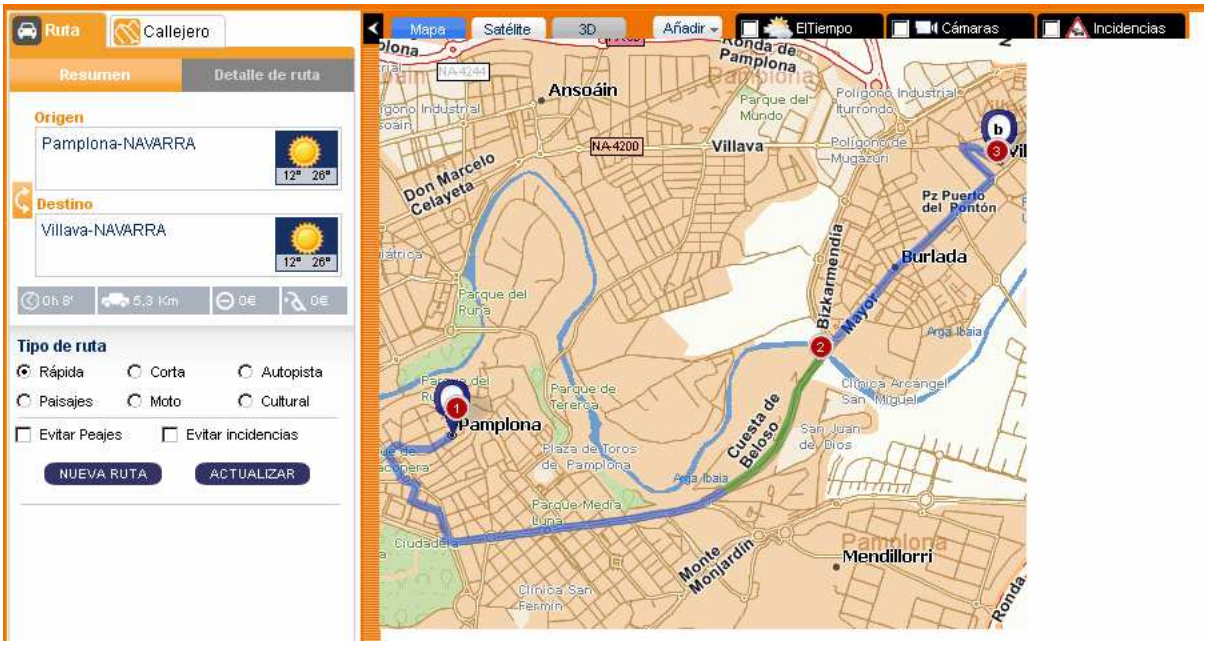
El sistema de navegación de Cepsa es bastante rudimentario ya que ni siquiera ofrece una representación gráfica de la ruta a seguir ni información de la ruta.
Sólo muestra una marca en el punto de inicio y otro en el final.

Permite ver servicios de la ciudad y escoger entre el tipo de ruta (rápido, seguro o corto).

Lo que no permite es introducir etapas en el camino o ver el coste económico ni la duración. Tampoco ofrece la opción de ver rutas alternativas.

Lo que sí ofrece este sistema y no el de la Vía Michelin es la opción de ver la peligrosidad de las rutas (muy baja, baja, media, alta, muy alta o sin información).

- GUÍA REPSOL: http://www.guiarepsol.com/es_es/home/



La Guía Repsol es más completa y útil. Ofrece la posibilidad de introducir una dirección de origen y un destino y muestra los resultados en un mapa de manera gráfica y como texto mostrando indicaciones de la ruta a seguir.

Además, permite seleccionar tipos de ruta y muestra el tiempo, el coste de carburante y de peajes y la duración del trayecto.

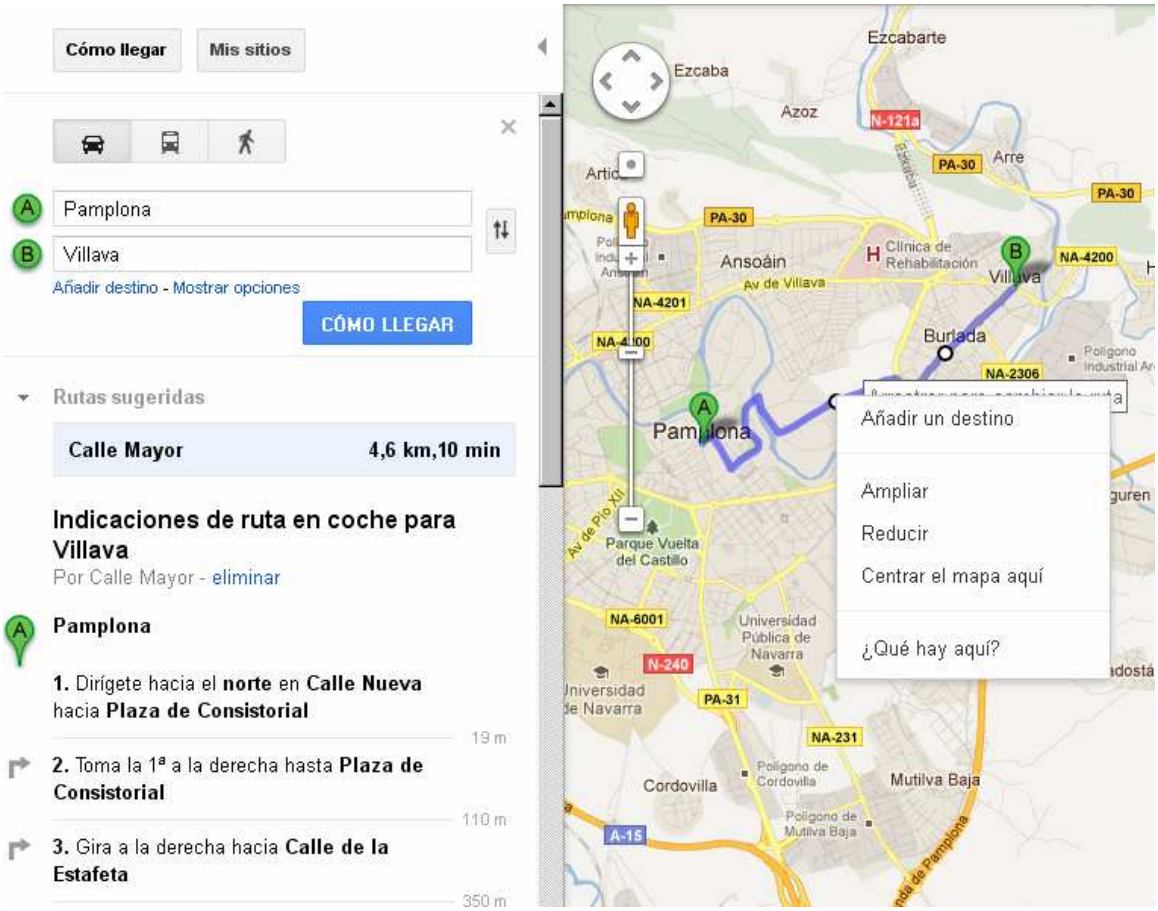
La guía Repsol muestra los radares existentes y proporciona la opción de evitar peajes e incidencias.

Lo que no ofrece es la posibilidad de hacer una simulación gráfica del trayecto ni muestra los servicios disponibles en la ruta. Tampoco permite añadir puntos de paso en el trayecto.

A diferencia de los sistemas de navegación anteriores, la Guía Repsol sí ofrece la posibilidad de ver rutas alternativas y muestra su nueva duración.

- GOOGLE MAPS: http://maps.google.es/maps

Google Maps es el sistema de navegación más utilizado en la actualidad.



Google Maps muestra en un mapa el resultado de manera gráfica. Además, ofrece la posibilidad de arrastrar tanto la ruta como los marcadores para recalcular la ruta de manera más rápida.

Sí ofrece la posibilidad de añadir etapas en el trayecto y además, también muestra la información de la ruta con texto en la ventana de la izquierda explicando las indicaciones que deben seguirse y los metros de cada explicación. Muestra también el tiempo y la distancia total para completar el trayecto.

Lo que no ofrece es la posibilidad de elegir entre rutas alternativas ni el coste económico de la ruta. Así como tampoco muestra servicios de la ruta, ni radares ni el gasto del carburante. Tampoco se puede realizar una simulación de la ruta en tiempo real.

Lo que si permite, a diferencia de todos los sistemas de navegación existentes, es desarrollar nuestro propio sistema de navegación gracias al API de Google Maps que ha desarrollado Google y el cual ofrece muchas funcionalidades para añadir.

A continuación mostramos una tabla comparativa de todos los sistemas de navegación analizados:

Propiedades	Michelin	Google Maps	Cepsa	Repsol
Punto de salida y de llegada	Sí	Sí	Sí	Sí
Diferentes modos de viaje	Sí	Sí	No	No
Introducir etapas	Sí	Sí	No	No
Elegir económico, corto, etc.	Sí	Sí	Sí	Sí
Simulación de la ruta	No	No	No	No
Gasto carburante	Sí	No	No	Sí
Radares	Sí	No	No	Sí
Mostrar servicios ciudad	Sí	No	Sí	No
Información detallada de la ruta	Sí	Sí	No	Sí
Coste económico	Sí	No	No	Sí
Tiempo	Sí	Sí	No	Sí
Distancia	Sí	Sí	No	Sí
Rutas alternativas	No	No	No	Sí
Peligrosidad de la ruta	No	No	Sí	No
Programable	No	Sí	No	No
Ruta “draggable”	No	Sí	No	No

Ahora nosotros tenemos que ver cuál de todos ellos es el que mejor se adapta a nuestras necesidades en función de las características que cumplen o no.

2.2- OBJETIVOS DEL PROYECTO

Lo que se pretende conseguir con este proyecto es crear una pequeña aplicación de localización a partir del análisis de Google Maps.

En una primera fase, se va a realizar un análisis en profundidad tanto del funcionamiento actual de dicha aplicación como del API existente para el conocimiento de todas las posibilidades utilizadas y aquellas que podrían interesar para implementar nuevas utilidades que no se han incorporado.

Se parte de un sistema creado y en funcionamiento, por lo que una de las partes más extensas tratará de realizar un análisis minucioso de dicho funcionamiento. Para ello se debe trabajar desde el punto de vista de un usuario para ver que posibilidades no ofrece y conocer en gran medida que es lo que posteriormente vamos a tener que investigar a nivel de desarrollador.

Este trabajo se desarrolla directamente con la aplicación sobre la plataforma de Google. Se realizarán simulaciones de las distintas acciones que se pueden realizar partiendo de algo tan sencillo como insertar una dirección y que se localice sobre un mapa, hasta algo más complicado como puede ser crear una ruta entre dos puntos a la que se le añadan puntos intermedios o de paso para ver cuál es la respuesta y sus componentes.

Además de ver qué posibilidades nos ofrece en el estado actual, también se observarán posibles mejoras para un futuro. Partiendo de un conocimiento de sistemas similares ya existentes y del estudio de este se deberá tener una idea del resultado buscado.

Una vez que ya se tenga un conocimiento amplio sobre las posibilidades de trabajo que nos da esta aplicación se realizará una investigación en el API para conocer su funcionamiento interno. Este análisis interno se llevará a cabo teniendo en cuenta que se puede trabajar tanto en Flash como en JavaScript.

Antes de comenzar con este proyecto tenemos que ver que vamos a necesitar para su desarrollo. Teniendo en cuenta lo que se quiere hacer, en primer lugar deberemos centrarnos en ver con cuál de las implementaciones de esta aplicación nos quedaremos. Nos ofrece las posibilidades de trabajar con JavaScript o Flash, de las cuales nos quedaremos con la primera.

Partiendo de que ya tenemos un API con el que poder estudiar el funcionamiento de estos sistemas, y un lenguaje de programación que nos ayude a poder crear una pequeña aplicación para comprobar el funcionamiento de lo estudiado, deberemos adentrarnos en un estudio de este lenguaje de programación.

Uno de los motivos de elegir JavaScript para el estudio es que es un lenguaje muy utilizado en el diseño Web, siendo de gran ayuda por ejemplo cuando se quieren agregar diferentes funcionalidades como menús dinámicos, en validación de formularios o comunicación con servidores.

Una de las ventajas que nos aporta este lenguaje es que podemos crear funciones que aprovechen las que ya existen en el API en un script que se integre en el archivo en el cual se tiene también el diseño Web. Además, se deberá establecer comunicación con los servidores de Google Maps que se encargan de procesar las rutas y realizar muchas acciones que no son realmente transparentes para el usuario. También vamos a tener una interfaz de interacción en la que se deben captar datos y mostrar resultados, creyendo que puede ser más conveniente su uso.

3- ARQUITECTURA DEL SISTEMA

Este proyecto tendrá una parte de análisis de la aplicación desde el punto de vista de un usuario cualquiera, que será común para las dos partes en las que posteriormente se dividirá.

Es decir, una vez que se conozca debidamente la aplicación de Google Maps, dividiremos en proyecto en dos partes que consistirán en primer lugar, en analizar el tratamiento de los datos en la interfaz que se muestra a los usuarios y su posterior paso a la parte ejecutora. También incluirá la forma en la que se deben recibir posteriormente para representarlos de nuevo al usuario. La segunda de las partes tendrá lo referente al tratamiento de dichos datos de forma interna por Google.

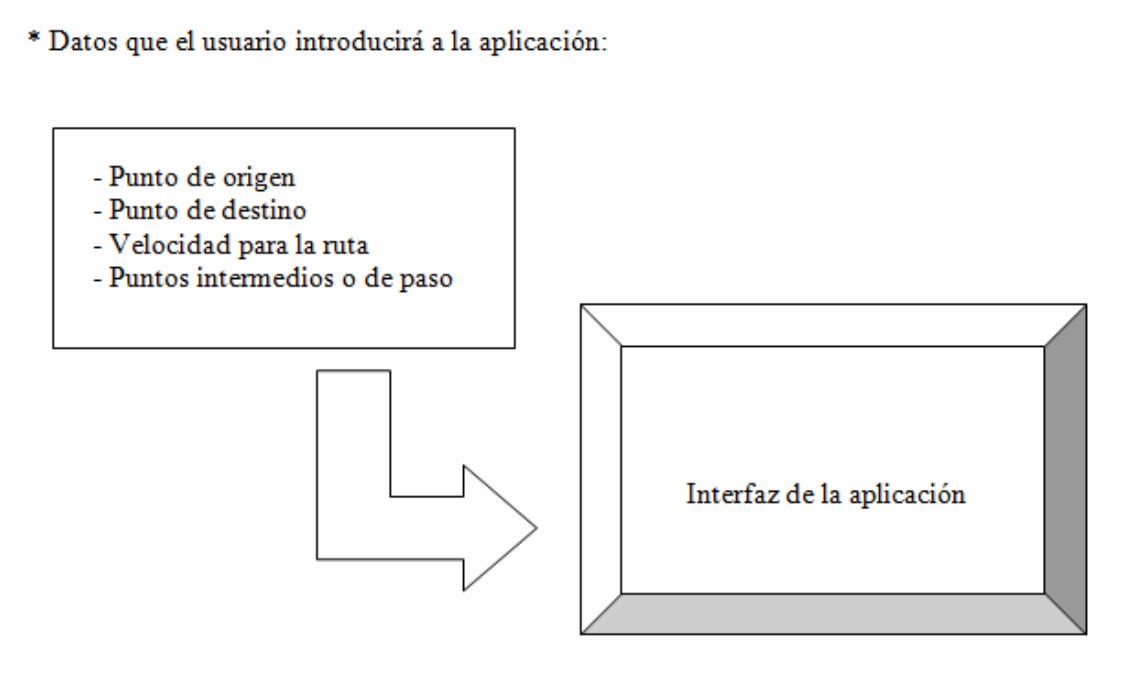
En este proyecto nos centraremos en la primera de las partes. Por lo tanto, veremos el proceso desde que el usuario nos pasa una serie de datos, cómo vamos a almacenarlo para poder hacer después uso de las funciones implementadas en el API para procesar dicha información. Una vez que los datos han pasado por los motores de trabajo de Google nos los devolverá y tendremos que comprobar en qué condiciones lo hace para poder dar una respuesta coherente a la solicitud realizada por los usuarios.

Para este desarrollo crearemos una sencilla interfaz que nos permita trabajar con una serie de datos de entrada de forma similar a la que lo haríamos con Google Maps, y que al mismo tiempo sea nuestra interfaz de respuesta.

La otra parte del proyecto se encarga de recoger esos datos y estructurarlos para poder realizar la consulta a los servidores y una vez realizada la consulta recoger la información que devuelve y pasarnos los datos necesarios para mostrar los resultados.

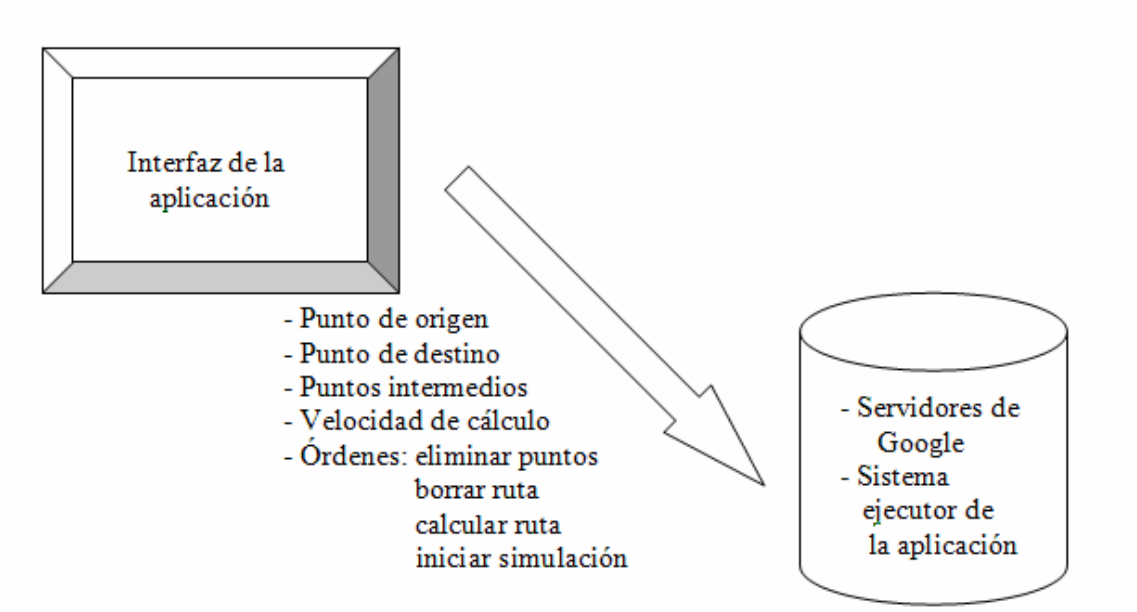
Otra de las funciones del sistema ejecutor será la de pasar a la interfaz el punto por el que realmente pasa la ruta. Esto se debe a que el usuario puede seleccionar un punto que no existe realmente por lo que se debe representar en la ruta real lo más cerca posible de donde ha seleccionado el usuario.

A continuación veremos un esquema en el que se diferencian ambas partes y se puede ver la interacción que se produce entre ellas. La parte más importante que tenemos es la transferencia de datos, ya que de ello dependerá que ambas partes puedan interrelacionarse de forma correcta. Es decir, tenemos que trabajar con una serie de estructuras de datos comunes e interpretables por el API, o mejor dicho por sus funciones para que ambas partes funcionen como si de una única aplicación se tratase.

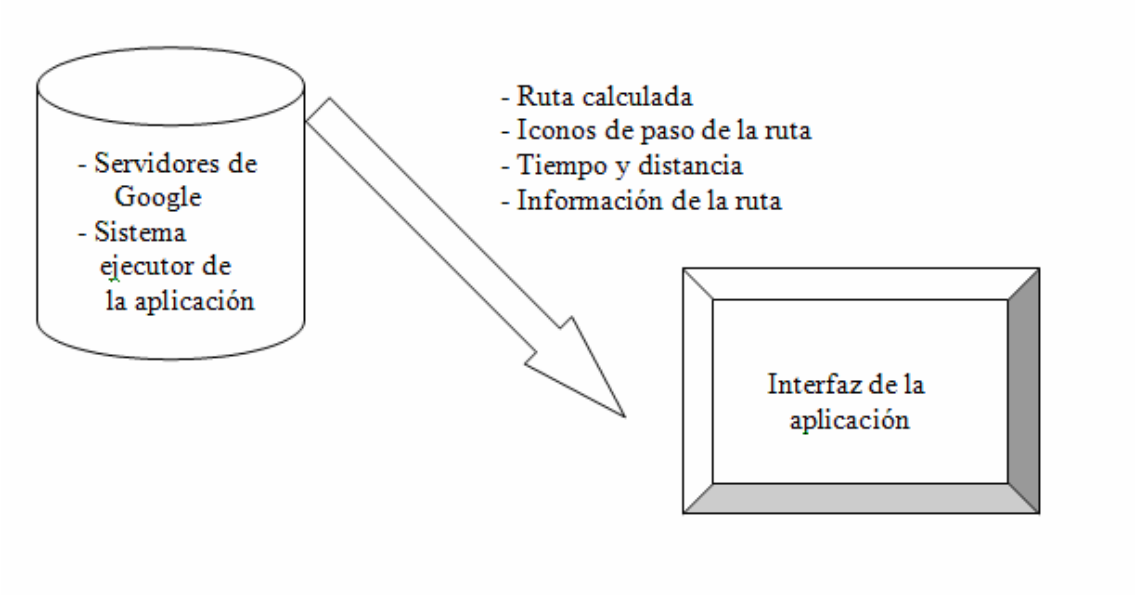


Como vemos en el gráfico anterior, los datos suministrados por el usuario son escasos pero nos van a permitir realizar un análisis de la aplicación en el que veremos muchas de sus funciones detalladamente. Más adelante veremos cómo el usuario hará para introducir dicho valores teniendo en cuenta el análisis previo de la aplicación y del API, para que sea de la forma que más se adecue al funcionamiento existente.

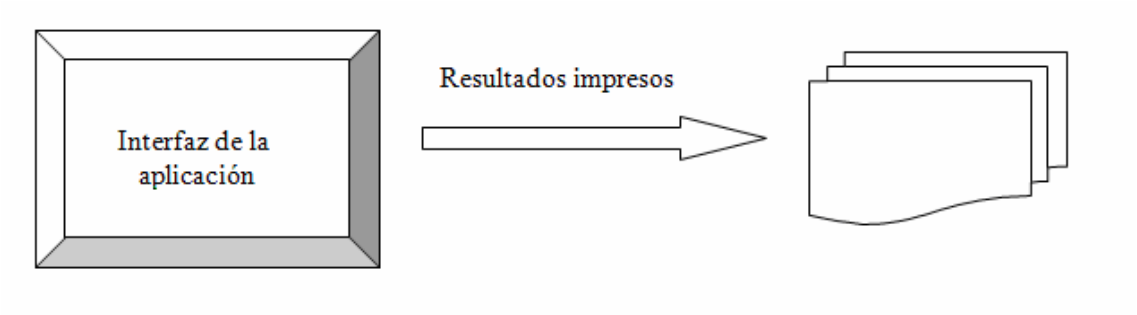
Una vez recibidos dichos datos tendremos que pasárselos a la parte encargada de procesarlos y trabajar con ellos para obtener los resultados esperados por el usuario. Ahora veremos que es lo que vamos a pasar a dicha parte para que realice su trabajo.



Además de los datos que se pasarán a la parte ejecutora de los datos debemos conocer cuál va a ser la respuesta que nos va a dar dicho sistema.



Una vez que ya recibamos toda la información necesaria para mostrar convenientemente los resultados a los usuarios, les facilitaremos que puedan almacenar en ficheros las rutas calculadas de forma que puedan utilizarlas cómo guía de viaje.



3.1- INTERACCIÓN DE LAS PARTES

Teniendo en cuenta que para la realización de este proyecto común se debe interactuar entre los dos proyectos que lo componen, vamos a tener que mantener una consistencia en los datos y elementos que sean comunes para ambas partes.

La primera parte, encargada de la captación de los datos que aporte el usuario y de la representación de los posteriores resultados, deberá iniciar el almacenamiento de algunos datos que después utilizará el sistema ejecutor. Para ello, se establecerán una serie de estructuras de almacenamiento que sean adecuadas para el posterior uso en las funcionalidades que desarrolle el segundo proyecto.

Principalmente vamos a utilizar puntos para la mayoría de actuaciones, ya sea de forma individual o un grupo de los mismos. Por ello, tendremos que tener un criterio común, que cree el sistema de captación y que utilice el sistema ejecutor, y que además le permita a este último trabajar sin problemas con las diferentes funcionalidades que proporciona Google Maps en su API.

Para esta interacción vamos a utilizar los puntos como elementos de la clase 'LatLng' en los que limitaremos el número de decimales para tener la misma estructura en todos ellos. Además, cuando necesitemos guardar más de un punto, lo haremos en un Array. Diferenciaremos dos puntos que emplearemos de diferente forma como son el inicial y el final, que, aunque tendrán la misma estructura, tendrán usos algo diferentes del resto. Es por ello, que los tendremos almacenados en variables independientes.

El sistema de captación de datos pasará los puntos al ejecutor cuando el usuario decida calcular la ruta. Una vez ya esté calculada, también podrá ordenarle que realice diversas acciones sobre la misma, como simular el recorrido o recibir rutas alternativas a la principal.

Si se decide borrar un punto de paso, o todos, las estructuras mencionadas se deberán actualizar para que en futuras solicitudes se realicen de acuerdo a lo especificado por el usuario.

4- HERRAMIENTAS: API DE GOOGLE MAPS [1,2,3]

En este apartado vamos a ver un profundo análisis sobre el API de Google Maps. En él iremos desgranando cada una de las partes analizadas para ver que es necesario en el desarrollo de un sistema de localización.

Dentro del API y teniendo en cuenta la orientación del proyecto, veremos una serie de elementos necesarios y fundamentales para la captación de datos y la posterior visualización de los resultados devueltos.

Analizaremos cada uno de ellos y veremos cómo debemos crearlos y utilizarlos para que el funcionamiento sea correcto teniendo en cuenta que debemos mantener una serie de directrices que permitan que posteriormente trabajemos con las funciones que Google Maps ya tiene creadas para el procesamiento de la información.

Los elementos que son necesarios en esta parte de captación y visualización son los siguientes:

- Mapas de Google Maps
- Puntos
- Marcadores

4.1- MAPA DE GOOGLE MAPS

El mapa es el elemento principal de estos sistemas y por lo tanto debemos conocer cómo es creado y cómo podemos interactuar con él.

```
////////////////////////////////////
//Funcion que crea el mapa inicial//
////////////////////////////////////
function load() {
    directionsDisplay = new google.maps.DirectionsRenderer({suppressMarkers: true});
    //Inicializamos las opciones del mapa
    var centro = new google.maps.LatLng(42.820336190453304, -1.6458892822265625);
    var myOptions = {
        zoom: 12,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        center: centro
    }
    map = new google.maps.Map(document.getElementById("map"), myOptions);
    directionsDisplay.setMap(map);
    //Listener para anadir un punto cuando se hace click sobre el mapa
    google.maps.event.addListener(map, 'click', function (event){
        //Anado los puntos al array de puntos intermedios que enviamos a calcRoute()
        listaPuntos.push({
            location:event.latLng,
            stopover:true
        });
        calcRoute();
    });
    calcRoute();
    //Almacenamos los puntos iniciales y finales que mete el usuario en los text box
    puntoInicial = document.form_ruta.desde.value;
    puntoFinal = document.form_ruta.hasta.value;
}
```

En el código anterior podemos ver la función que se emplea para cargar inicialmente la ventana de Google Maps.

Si comenzamos a analizarlo vemos que necesitamos una variable 'directionsDisplay' en la que se crea un elemento de tipo 'DirectionsRenderer'. Con ella vamos a crear un procesador con la opción de que no nos realice un procesamiento de los marcadores. (Veremos que son más adelante)

```
directionsDisplay = new google.maps.DirectionsRenderer({suppressMarkers: true});
```

Las indicaciones las vamos a poder procesar sobre un mapa, que es lo que en este proyecto nos interesa, o sobre un panel como instrucciones en forma de texto.

Unas tras crear el mapa lo utilizaremos para establecerlo y poder trabajar con él posteriormente. Por ejemplo, una vez que tengamos una solicitud, lo volveremos a utilizar para añadir en el mapa los resultados que esta genere.

A continuación, veremos una serie de opciones que necesitaremos definir antes de crear el mapa. La primera de estas opciones es el 'centro' o el punto en el que se va a centrar el mapa en el momento en el que este se cree. Debemos pasarle los argumentos de latitud y longitud para que se cree correctamente como un elemento de la clase LatLng. Vamos a trabajar con elementos de esta clase ya que nos permiten que en un momento dado, podamos obtener los valores de latitud o longitud de un punto, realizar comparaciones entre ellos o pasarlos a una cadena de texto que nos indique la dirección a la que corresponden dichas coordenadas geográficas.

```
var centro = new google.maps.LatLng(42.820336190453304, -1.6458892822265625);
```

Veamos que es lo que el API nos ofrece al utilizar esta clase.

Constructor	Descripción
LatLng(lat:number, lng:number, noWrap?:boolean)	Fíjate en el orden de la latitud y de la longitud. Si la etiqueta noWrap está establecida en "true", los números se utilizarán según se transmitan, de lo contrario la latitud quedará entre -90 grados y +90 grados, y la longitud quedará envuelta para situarse entre -180 grados y +180 grados.

Métodos	Valor de retorno	Descripción
equals(other:LatLng)	boolean	Función de comparación
lat()	number	Devuelve la latitud en grados.
lng()	number	Devuelve la longitud en grados.
toString()	string	Se convierte en representación de cadena.
toUrlValue(precision?:number)	string	Devuelve una cadena de tipo "lat,lng" para esta clase LatLng. Redondeamos los valores lat/lng en un máximo de seis decimales de manera predeterminada.

Además del punto en el que se debe centrar el mapa, debemos pasarle varios parámetros más al constructor. Uno de ellos es el nivel de zoom que vamos a realizar sobre el mapa para tener una visión general de una zona concreta o un país, o si por el contrario queremos centrarnos en una ciudad o barrio más concreto. Otro es el tipo de mapa que queremos que nos cree. Para las aplicaciones de localización lo más adecuado son los mapas de carreteras, similares a un callejero, aunque también lo podríamos crear con mapas de contienen imágenes de satélite, híbridos (satelite y callejero) y de terreno (rasgos físicos como terrenos y vegetación).

```
var myOptions = {
    zoom: 12,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    center: centro
}
```

En este momento ya tenemos preparados todos los requisitos necesarios para poder crear un mapa. Para crearlo vamos a utilizar un espacio creado en nuestra interfaz, que será su ubicación en el momento de visualizarlo.

```
map = new google.maps.Map(document.getElementById("map"), myOptions);
directionsDisplay.setMap(map);
```

Vamos a ver cómo se define en el API el constructor que hemos explicado

Constructor	Descripción
`Map(mapDiv:Node, opts?:MapOptions)`	Crea un mapa nuevo dentro del contenedor HTML en cuestión, que generalmente suele ser un elemento `DIV`.

Con las instrucciones anteriores, en primer lugar se creará el mapa como anteriormente hemos descrito y utilizando las opciones declaradas. Después de crearlo debemos establecerlo con el elemento directionsDisplay de tipo 'DirectionsRenderer' para poder operar con él.

El resultado final al cargar un mapa de Google Maps será el siguiente, de acuerdo a los parámetros que se han declarado en las explicaciones anteriores.

Parámetros del mapa:

- Zoom: 12
- Tipo: ROADMAP o mapa de carreteras
- Centro: (42.820336190453304, -1.6458892822265625)
  Coordenadas geográficas de Pamplona

4.2- PUNTOS

Los puntos son uno de los elementos más importantes dentro de la aplicación, ya que van a ser los indicadores que vamos a utilizar para indicarle a las funciones qué deben hacer y con qué.

Según la definición existente de la función que va a calcular la ruta, se deben diferenciar los puntos de origen y destino de aquellos que sean puntos intermedios o de paso. Partiendo de este punto, utilizaremos dos variables que nos permitan hacer esta diferenciación y a su vez, modificarlos independientemente en cualquier momento sin alterar la ruta ya existente. Es decir, vamos a tenerlos aislados para que en el momento en el que se produzca alguna modificación sobre ellos, solamente se modifique y se cree una nueva ruta de acuerdo a esos nuevos parámetros.

Vamos a ver lo explicado en el párrafo anterior con un ejemplo que clarifique la explicación. Para recoger el valor de estos dos puntos nos tenemos que referir a los elementos que los contienen en la interfaz.

```
puntoInicial = document.form_ruta.desde.value;
puntoFinal = document.form_ruta.hasta.value;
```

Esta acción la realizaremos en el momento en el que se cargue la ventana inicial de Google Maps. (Código que podemos ver en la imagen del punto 4.1)

Para poder modificar los valores necesitamos una función que se active en el momento en el que se cambie el valor del elemento de la interfaz que lo contenga.

```
<tr>
    <td align="left" >
        <strong>Desde:</strong>
        <input type="text" size="40" id="desde" name="desde" value="" onchange="anadirPunto(this.id)"/>
    </td>
</tr>
<tr>
    <td align="left">
        <strong>Hasta:</strong>
         <input type="text" size="40" id="hasta" name="hasta" value="" onchange="anadirPunto(this.id)"/>
    </td>
</tr>
```

Los elementos que se van a alterar serán dos <input> en los cuales el usuario deberá introducir las direcciones de origen y destino de las rutas. Esto es prácticamente una forma de captación de datos que se emplea en la mayoría de sistemas de localización y que tanto para usuarios como para desarrolladores clarifica mucho el trabajo. En el momento en el que el usuario decida cambiar los valores de estos elementos disponemos de una función que será invocada automáticamente y nos realizará los cambios oportunos.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//Funcion que se activa al cambiar el punto inicial y final en las casillas de estos valores//
////////////////////////////////////////////////////////////////////////////////////////////////
function anadirPunto(origen){
    if (origen=='desde'){
        //Si se ha cambiado el origen tengo que borrar el marcador anterior
        for(var x=0;x<markersArray.length;x++){
            if (markersArray[x].getPosition().equals(anteriorinicio)){
                markersArray[x].setVisible(false);
            }
        }
        puntoInicial=document.getElementById(origen).value;
    }
    else{
        //Si se ha cambiado el destino tengo que borrar el marcador anterior
        for(var x=0;x<markersArray.length;x++){
            if (markersArray[x].getPosition().equals(anteriorfin)){
                markersArray[x].setVisible(false);
            }
        }
        puntoFinal=document.getElementById(origen).value;
    }
    if ((puntoInicial!=" ") && (puntoFinal!=" ")){
        rutaCalculada=true;
    }
    if (rutaCalculada){
        primeravez=true;        //Para que vuelva a pintar los marcadores en verde
        calcRoute();
    }
}
```

En la imagen superior podemos ver cómo vamos a trabajar con las modificaciones sobre los puntos de origen y destino. Lo primero que debemos saber es cuál de los valores se ha modificado, y para ello vamos a utilizar un parámetro de entrada que nos indicará el identificador del <input> que contiene el valor modificado. Esto es suficiente para tener conocimiento de diferenciarlos.

Al modificarse, además de su valor se deben cambiar otros elementos que son los marcadores, de los que hablaremos más adelante ya que es necesario conocer en primer lugar bien el funcionamiento de los puntos para hablar de ellos. Y lo principal que es el cambio de valor lo realizaremos invocando de nuevo a obtener el valor del elemento que responda al identificador que se ha pasado cómo parámetro en la función.

En el caso de que sea el punto inicial haremos:

```
puntoInicial=document.getElementById(origen).value;
```

Y en el caso de que sea el final:

```
puntoFinal=document.getElementById(origen).value;
```

Posteriormente debemos comprobar si tenemos un valor para cada uno de los puntos y si es así, hacer una llamada a la función que realiza el cálculo de la ruta. Aquí también marcamos que es la ‘primeravez’ para que en dicha función nos realice una serie de acciones que posteriormente veremos y que solamente deben suceder en la primera representación de cada ruta.

```
if ((puntoInicial!=" ") && (puntoFinal!=" ")){
    rutaCalculada=true;
}
if (rutaCalculada){
    primeravez=true;        //Para que vuelva a pintar los marcadores en verde
    calcRoute();
}
```

Llegados a este punto, ya hemos visto el tratamiento especial que se realiza sobre los puntos de inicio y final de ruta. Ahora veremos cómo vamos a tratar al resto de puntos de la ruta que serán indicadores de puntos de paso de la misma. El primero momento en el que hacemos referencia a ellos es cuando creamos el mapa. Si nos fijamos, una vez creado, se inicializa un listener para controlar cuando se produzca un clic sobre el mapa. Es en este momento donde comenzamos a trabajar con ellos.

```
//Listener para anadir un punto cuando se hace click sobre el mapa
google.maps.event.addListener(map, 'click', function (event){
    //Anado los puntos al array de puntos intermedios que enviamos a calcRoute()
    listaPuntos.push({
        location:event.latLng,
        stopover:true
    });
    calcRoute();
});
```

Este listener lo que va a hacer es recibir las coordenadas en las cuales se ha producido el clic y almacenarlas en un Array, en el cual tendremos guardados todos los puntos de paso que el usuario vaya introduciendo. Debemos fijarnos también en la forma en la que los almacenamos, ya que se debe mantener la estructura de 'LatLng' para tener una coherencia que nos permita trabajar sin problemas con las funciones de API. Posteriormente, se procede a recalcular la ruta para que se incluya este nuevo punto.

No es cierto que las coordenadas que se almacenan al realizar el clic sean las que se utilicen para representar el punto sobre el mapa. Esto se debe a que el usuario puede pinchar sobre un punto inaccesible como puede ser un edificio o un campo por el cual no transcurra una carretera y no se pueda realizar la ruta. Por lo tanto estas coordenadas sirven para que la ruta se calcule por el lugar más próximo posible al marcado. Una vez que ya se ha calculado la ruta, almacenaremos las coordenadas reales del marcador que se cree, ya que nos permitirá realizar las acciones pertinentes con dicho marcador y que veremos más adelante.

Para guardar estas coordenadas válidas nos vamos a tener que valer de dos Arrays con los que podamos ir guardando las coordenadas pinchadas y posteriormente las reales según la ruta creada sobre el mapa.

```
var listaPuntos = new Array();
```

```
var coorde = new Array();
```

Vamos a ver a continuación cómo vamos utilizando estas dos estructuras simultáneamente a lo largo de la aplicación para poder mantener una coherencia que permita que se pueda trabajar con los puntos que necesitamos. En primer lugar, vamos a ver cómo inicialmente guardamos las coordenadas que se reciben al hacer un clic sobre el mapa.

```
//Listener para anadir un punto cuando se hace click sobre el mapa
google.maps.event.addListener(map, 'click', function (event){
    //Añado los puntos al array de puntos intermedios que enviamos a calcRoute()
    listaPuntos.push({
        location:event.latLng,
        stopover:true
    });
    calcRoute();
});
```

Aquí vemos que en ‘listaPuntos’ vamos añadiendo las coordenadas recibidas del evento que se produce cuando se pincha en el mapa. Además, marcamos la opción ‘stopover’ que indica que este hito será una parada entre el origen y el destino. Esto provoca que la ruta se divida en dos. Hacer estas divisiones nos ayudarán posteriormente mucho para poder analizar otras facetas de los sistemas.

Ya habíamos visto también que no tendremos la misma actuación si se trata de la primera vez que se calcula la ruta sobre el mapa que si ya está calculada y se añaden puntos de paso a la misma. En la siguiente imagen veremos que almacenamos en otro Array las coordenadas de los puntos inicial y final para tener correctamente guardados sus valores en caso de que después tengamos que modificar sus coordenadas por un cambio de origen o destino. Además, sabiendo que las coordenadas son correctas, las utilizaremos para invocar a la función que representa esos puntos en el mapa con los marcadores. (En el siguiente apartado ya analizaremos tanto los marcadores como las funciones que trabajan con ellos).

```
//Si es la primera vez que entramos a calcRoute() nos pinta el punto inicial y el final
if(primeravez){
    //Reducimos los decimales de las coordenadas que devuelve la ruta
    coorde[0]=pintarOrigenDestino(route.legs[i].start_location);
    coorde[1]=pintarOrigenDestino(route.legs[route.legs.length-1].end_location);
    //Alamaceno lo anterior por si se modifica el inicio o el fin y asi poder borrar sus marcadores
    anteriorinicio=coorde[0];
    anteriorfin=coorde[1];
    //Creamos los marcadores
    crearLimites(coorde[0],icons.inicio);
    crearLimites(coorde[1],icons.fin);
    primeravez=false;
}
```

Cuando ya tengamos almacenados esos dos puntos que queremos diferenciar, tendremos que trabajar con el resto de puntos de paso de la ruta. Para saber si se trata de uno de ellos lo primero que se debe hacer es comprobar que las coordenadas del nuevo punto que se quiere guardar no coinciden ni con las del punto de inicio, ni con las del punto de finalización de la ruta, ni con las de los puntos intermedios si es que ya se ha definido alguno. Vamos a ver cómo se realizan esas acciones y cuál es el tratamiento que se hace de las nuevas coordenadas.

```
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].start_location).equals(coorde[cont])){
        existeInicio=true;
        break;
    }
}
if (!existeInicio){
    makeMarker(pintarOrigenDestino(route.legs[i].start_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].start_location);
}
else if (existeInicio){
    existeInicio=false;
}
```

Vemos que se va a buscar en el Array en el que están guardadas las coordenadas que ya son correctas según la ruta que se tiene calculada, si alguna de las posiciones contiene un valor igual al que se quiere guardar. En el caso de no encontrar ningún resultado en la comparación, se procede a crear un nuevo marcador con las nuevas coordenadas y a guardar esos valores en el Array de coordenadas 'coorde'. Este proceso se realiza tanto para el punto inicial como para el final.

```
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].end_location).equals(coorde[cont])){
        existeFin=true;
        break;
    }
}
if (!existeFin){
    makeMarker(pintarOrigenDestino(route.legs[i].end_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].end_location);
}
else if (existeFin){
    existeFin=false;
}
```

Un pequeño detalle que no podemos dejar de ver es que cada vez que se reciben unas nuevas coordenadas, se va a trabajar con ellas limitándoles el número de decimales para que no existan problemas posteriores y se tenga una consistencia y coherencia global de datos. Vamos a ver qué es lo que se hace con dicha función.

```
////////////////////////////////////////////////////////////////////////////////
//Funcion que limita el numero de decimales de una coordenada y crea un nuevo latlng//
////////////////////////////////////////////////////////////////////////////////
function pintarOrigenDestino(a){
    var latitud = a.lat().toFixed(12);
    var longitud = a.lng().toFixed(12);
    return new google.maps.LatLng(latitud,longitud,{noWrap:true});
}
```

Lo único que se hace es limitar a 12 el número de decimales obteniendo por separado los componentes de la coordenada, (latitud,longitud), y creando un nuevo elemento de tipo 'LatLng' con esos nuevos valores.

4.3- MARCADORES

Anteriormente hemos hablado de los puntos que van a contener las coordenadas geográficas de los puntos tanto de inicio y final de la ruta, como de los de paso. Y hablando de ellos también habían surgido los marcadores, los cuales debemos conocer una vez que ya hemos visto cómo vamos a trabajar con las coordenadas seleccionadas y las estructuras que los contengan.

Antes de comenzar a explicarlos, vamos a mostrar la información disponible en el API sobre los marcadores para tener una visión general de todo lo que se puede hacer con ellos, y después veremos más detalladamente ciertas partes.

Constructor	Descripción
`Marker(opts?:MarkerOptions)`	Crea un marcador con las opciones especificadas. Si se especifica un mapa, el marcador se añade al construir el mapa. Ten en cuenta que se debe establecer la posición del marcador que se va a mostrar.

Métodos	Valor de retorno	Descripción
`getClickable()`	`boolean`	
`getCursor()`	`string`	
`getDraggable()`	`boolean`	
`getFlat()`	`boolean`	
`getIcon()`	`string|MarkerImage`	
`getMap()`	`Map|StreetViewPanorama`	
`getPosition()`	`LatLng`	
`getShadow()`	`string|MarkerImage`	
`getShape()`	`MarkerShape`	
`getTitle()`	`string`	
`getVisible()`	`boolean`	
`getZIndex()`	`number`	
`setClickable(flag:boolean)`	None	
`setCursor(cursor:string)`	None	
`setDraggable(flag:boolean)`	None	
`setFlat(flag:boolean)`	None	
`setIcon(icon:string|MarkerImage)`	None	
`setMap(map:Map|StreetViewPanorama)`	None	Procesa el marcador en la panorámica o en el mapa especificado. Si el mapa se establece en "null", se eliminará el marcador.
`setOptions(options:MarkerOptions)`	None	
`setPosition(latlng:LatLng)`	None	
`setShadow(shadow:string|MarkerImage)`	None	
`setShape(shape:MarkerShape)`	None	
`setTitle(title:string)`	None	
`setVisible(visible:boolean)`	None	
`setZIndex(zIndex:number)`	None	

Eventos	Argumentos	Descripción
click	Event	Este evento se activa cuando se hace clic en el icono del marcador.
clickable_changed	None	Este evento se activa cuando la propiedad del marcador en la que se puede hacer clic cambia.
cursor_changed	None	Este evento se activa cuando la propiedad de cursor del marcador cambia.
dblclick	Event	Este evento se activa cuando se hace doble clic en el icono del marcador.
drag	*MouseEvent*	Este evento se activa repetidamente mientras el usuario arrastra el marcador.
dragend	*MouseEvent*	Este evento se activa cuando el usuario termina de arrastrar el marcador.
draggable_changed	None	Este evento se activa cuando la propiedad arrastrable del marcador cambia.
dragstart	*MouseEvent*	Este evento se activa cuando el usuario empieza a arrastrar el marcador.
flat_changed	None	Este evento se activa cuando la propiedad plana del marcador cambia.
icon_changed	None	Este evento se activa cuando la propiedad de icono del marcador cambia.
mousedown	Event	Este evento se activa cuando se activa el evento de ratón DOM MouseDown en el icono de marcador.
mouseout	Event	Este evento se activa cuando el ratón sale del área del icono de marcador.
mouseover	Event	Este evento se activa cuando el ratón entra en el área del icono de marcador.
mouseup	Event	Este evento se activa para el evento DOM MouseUp en el marcador.
position_changed	None	Este evento se activa cuando la propiedad de posición del marcador cambia.
rightclick	Event	Este evento se activa cuando se hace clic con el botón derecho en el marcador.
shadow_changed	None	Este evento se activa cuando la propiedad de sombra del marcador cambia.
shape_changed	None	Este evento se activa cuando la propiedad de forma del marcador cambia.
title_changed	None	Este evento se activa cuando la propiedad de título del marcador cambia.
visible_changed	None	Este evento se activa cuando la propiedad visible del marcador cambia.
zindex_changed	None	Este evento se activa cuando la propiedad zIndex del marcador cambia.

En primer lugar, diremos que un marcador va a ser una representación gráfica sobre el mapa de los puntos seleccionados por los usuarios para que formen la ruta. Nosotros vamos a diferenciar entre origen y destino, y el resto de puntos de paso. Se verá gráficamente incluso con distintos iconos. Para poder tener esta diferenciación es necesario que tengamos definidos diferentes iconos para cada tipo de punto.

```
var icons = {
    marcador: new google.maps.MarkerImage(
        // URL
        'marcador.png'
    ),
    fin: new google.maps.MarkerImage(
        // URL
        'fin.png'
    ),
    inicio: new google.maps.MarkerImage(
        // URL
        'inicio.png'
    )
};
```


Marcador de paso

Marcador de destino

Marcador de inicio

Con la estructura anterior tenemos definidos ya los 3 tipos de iconos que se van a utilizar en la aplicación, inicio, final y marcador intermedio. Para crear los iconos inicial y final tenemos una función a la que invocaremos de la siguiente manera.

```
//Creamos los marcadores
crearLimites(coorde[0],icons.inicio);
crearLimites(coorde[1],icons.fin);
```

Ahora vamos a ver cómo en la función se crea el marcador y se le aplica un listener. Además de crearlo, también lo vamos a almacenar en otro Array, ya que necesitaremos tenerlos guardados para posteriores acciones que los usuarios puedan realizar sobre la ruta, como puedan ser eliminar un punto intermedio o incluso todos los puntos.

```
////////////////////////////////////////////
//Funcion que crea los puntos inicial y final//
////////////////////////////////////////////
function crearLimites(coordenadas,icono){
    marker=new google.maps.Marker({
        position: coordenadas,
        map: map,
        icon: icono
    });
    google.maps.event.addListener(marker, "click", function(event){
        alert("Este icono solamente se puede modificar desde el recuadro de direcciones");
    });
    markersArray.push(marker);
}
```

Para construir un marcador son necesarios 3 valores: las coordenadas sobre las que se debe crear, el lugar en el que lo debe situar (mapa) y el icono que debe mostrarse al representarlo. El constructor directamente lo crea y lo sitúa dentro del mapa en el lugar que lo hemos indicado. Además de crearlo, vamos a añadirle un listener para que no se pueda mover. Así obligaremos a que solamente se cambien sus valores desde los cuadros de texto, a los que les habíamos asociado ya unas funciones que nos controlarán si se modifican sus valores y de forma automática, los actualizarán. Tenemos también el Array ‘markersArray’ en el que los guardaremos para después modificarlos si es necesario.

Para los marcadores intermedios tenemos otra función con la que los crearemos y les daremos las propiedades que se adecuen al comportamiento que queremos que tengan.

```
////////////////////////////////////
//Funcion que pinta los marcadores//
////////////////////////////////////
function makeMarker( position, icon) {
    marker=new google.maps.Marker({
            position: position,
            map: map,
            icon: icon,
            draggable: true
    });
    //Listener para guardar el numero de marcador que se comienza a mover
    google.maps.event.addListener(marker, "dragstart", function(event){
        coordenadas1 = pintarOrigenDestino(this.position);
        for (i=0;i<listaPuntos.length;i++){
            if (coordenadas1.equals(listaPuntos[i].location)){
                numMarcador=i;
                //Borro el contenido de coorde de la posicion numMarcador para borrar ese punto
                var aux = coorde.slice(numMarcador+3);
                coorde = coorde.slice(0,numMarcador+2);
                coorde = coorde.concat(aux);
                break;
            }
        }
        return numMarcador;
    });

    //Listener para cambiar las coordenadas del marcador que se ha movido
    google.maps.event.addListener(marker, "dragend", function(event){
        this.setVisible(false);
        listaPuntos[numMarcador].location=this.position;
        calcRoute();
    });
    markersArray.push(marker);
}
```

El constructor es similar al que hemos visto en los marcadores inicial y final, salvo que le vamos a añadir la propiedad de que sea ‘draggable’, es decir que podamos pinchar sobre él y arrastrarlo por el mapa para cambiar su posición de una forma más sencilla que los otros marcadores.

```
marker=new google.maps.Marker({
        position: position,
        map: map,
        icon: icon,
        draggable: true
});
```

Al añadirle esta propiedad, tenemos que hacer también que el marcador responda correctamente cuando se arrastre y para ellos tenemos dos listeners, uno para cuando comenzamos a arrastrarlo y otro para cuando se posiciona definitivamente en su nueva ubicación.

Vamos a analizarlos detenida e independientemente, para comprender su funcionamiento. En el primero, el evento que se controla es el inicio del arrastre del marcador. Lo que se va a conseguir es almacenar cual es el índice del marcador alterado en el Array en el que los habíamos almacenado para después recolocarlo. Para ello, se deben comparar las coordenadas del punto en el momento justo en el que se inicia el arrastre con las de todas las posiciones del Array en el que tenemos guardados los marcadores. Al encontrarla, se guardas la posición y se elimina el contenido, ya que se reposicionará sobre otras coordenadas diferentes.

```
//Listener para guardar el numero de marcador que se comienza a mover
google.maps.event.addListener(marker, "dragstart", function(event){
    coordenadas1 = pintarOrigenDestino(this.position);
    for (i=0;i<listaPuntos.length;i++){
        if (coordenadas1.equals(listaPuntos[i].location)){
            numMarcador=i;
            //Borro el contenido de coorde de la posicion numMarcador para borrar ese punto
            var aux = coorde.slice(numMarcador+3);
            coorde = coorde.slice(0,numMarcador+2);
            coorde = coorde.concat(aux);
            break;
        }
    }
    return numMarcador;
});
```

En el segundo listener, se controla el evento de situar el marcador sobre unas coordenadas nuevas. Simplemente se oculta el marcador arrastrado, se guardan las nuevas coordenadas obtenidas y se invoca a la función que calcula la nueva ruta y la representa en el mapa de forma gráfica.

```
//Listener para cambiar las coordenadas del marcador que se ha movido
google.maps.event.addListener(marker, "dragend", function(event){
    this.setVisible(false);
    listaPuntos[numMarcador].location=this.position;
    calcRoute();
});
```

Cuando ya se han creado el marcador y se le han añadido los listeners para controlar en todo momento se comportamiento, se almacena el marcador en el Array de marcadores ‘markersArray’ del que ya hemos hablado antes.

```
markersArray.push(marker);
```

Ahora que ya conocemos qué son los puntos y los marcadores de los mapas, veremos cómo se puede hacer para eliminarlos, es decir, que eliminemos un marcador y no quede ningún rastro de él. Tenemos que tener en cuenta que para hacer esta eliminación, se deben tratar todas las estructuras de almacenamiento que se han utilizado hasta el momento, que son ‘listaPuntos’, ‘coorde’ y ‘markersArray’. Veamos la función que realiza esta acción:

```
//////////////////////////////////////////////////////////
//Funcion que elimina el ultimo punto intermedio a?adido//
//////////////////////////////////////////////////////////
function eliminarPunto() {
    listaPuntos.pop();
    coorde.pop();
    markersArray[markersArray.length-1].setMap(null);
    markersArray.pop();
    calcRoute();
}
```

En las listas de coordenadas, únicamente se debe eliminar la última posición utilizando el método 'pop()', que en JavaScript elimina dicha posición del Array. Cuando tenemos que eliminar el marcador, es diferente porque en primer lugar debemos hacer que la última posición del Array se anule en el mapa, ya que necesitamos tener el marcador para poder aplicarle dicha opción, y después eliminarla de la misma manera que en los otros Arrays.

De esta forma conseguimos que se puedan eliminar los puntos intermedios de la ruta secuencialmente, comenzando desde el último que se haya añadido hasta el primero. Esta funcionalidad puede facilitar que el usuario corrija la ruta en caso de tener algún punto intermedio innecesario.

Otra opción que se le da al usuario es poder eliminar rápidamente todos los puntos intermedios para dejar la ruta únicamente con los puntos de inicio y fin. En este caso, marcamos la opción de que es la primera vez que se calcula la ruta, y debemos borrar completamente todas las estructuras. Las listas de puntos de eliminan y se vuelven a crear. Para el caso de los marcadores, se deben eliminar todos ellos del mapa. Tras dejar las estructuras vacías, se recalculará la ruta para los puntos límites de la misma.

```
//////////////////////////////////////////////////////
//Funcion que elimina todos los puntos intermedios//
//////////////////////////////////////////////////////
function limpiarRuta(){
    primeravez=true;
    listaPuntos=null;
    listaPuntos=new Array();
    coorde=null;
    coorde=new Array();
    if (markersArray) {
        for (var k=0;k<markersArray.length;k++){
            markersArray[k].setMap(null);
        }
    }
    calcRoute();
}
```

5- REQUISITOS

Fundamentalmente, el requisito principal del proyecto común es encontrar nuevos usos y funcionalidades para los sistemas de navegación. Estudiar lo existente para poder ofrecer mejoras útiles a los usuarios.

Es importante encontrar utilidades para los usuarios, que sean fáciles de lanzar y de entender su funcionamiento. Los resultados obtenidos podrán ser útiles tanto como para usuarios comunes como para desarrolladores de software que buscan mejoras en los sistemas de navegación.

En nuestra aplicación tenemos 3 tipos de actores:

- Analista de la aplicación
- Usuario
- Usuarios expertos

El rol del analista es el principal en este proyecto ya que a pesar de que vamos a desarrollar una aplicación, la actividad principal va a ser la investigación de su funcionamiento. Serán los encargados de tener una visión completa del sistema, conociendo todas sus funciones y sobre todo, cómo deben implementarse y qué es necesario para ello.

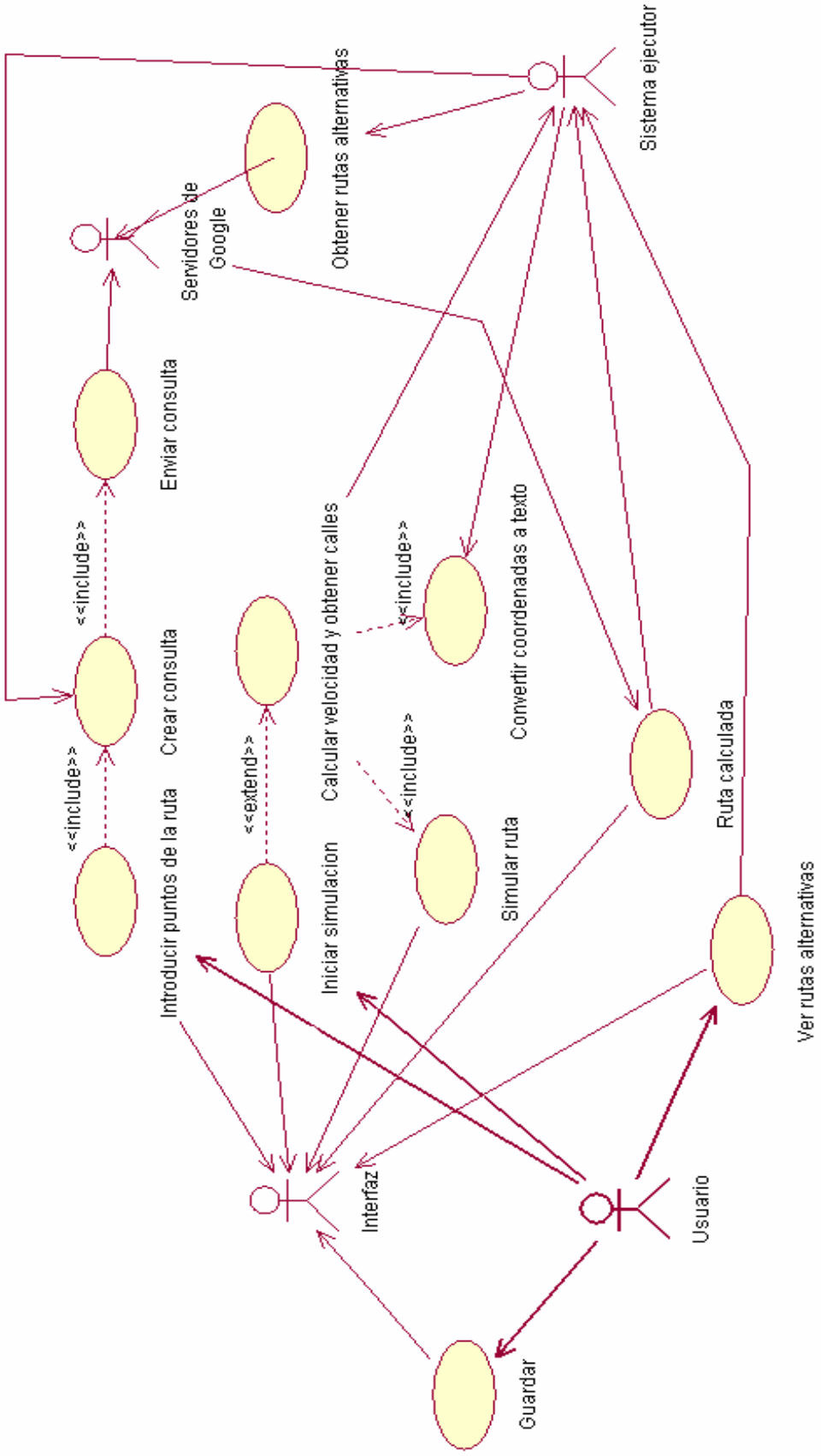
El rol de usuario puede ser desempeñado por cualquier persona dispuesta a probar la aplicación que se ha creado para comprobar el funcionamiento de toda la investigación realizada por el analista. No es necesario que tenga ningún tipo de conocimiento acerca de la aplicación ni qué se pretende conseguir con ella. De hecho, el propio analista puede desempeñar esta función ya que además, podrá comprobar con mayor detalle todo lo que ha investigado anteriormente.

Por último, el rol de usuarios expertos puede ser cualquier desarrollador que quiera introducir diferentes funcionalidades a un servicio de navegación que trabaje con el API de Google Maps y que crea que le puedan ser útiles las funcionalidades que nosotros hemos añadido.

Es un requisito el desarrollar una interfaz clara y sencilla desde la que cualquier usuario pueda realizar sus rutas de manera personalizada sin que se sienta confuso con las nuevas opciones que ofrecemos. Del mismo modo, el software que desarrollamos también debe ser claro para que cualquier desarrollador sepa rápidamente cuál es el apartado que le interesa reutilizar de todos los que creamos.

A continuación exponemos los diagramas de caso de uso y de secuencia para facilitar la comprensión de qué debemos hacer y quiénes van a interactuar en cada acción.

5.1- DIAGAMAS DE CASOS DE USO



En el diagrama de casos de uso vemos que el usuario puede realizar cuatro funciones básicas:

- Introducir puntos de la ruta
- Iniciar simulación
- Ver rutas alternativas
- Guardar

El usuario introduce los puntos de la ruta en la interfaz, la cual pasa los puntos al sistema ejecutor quien a su vez reorganiza estos puntos y crea la estructura adecuada para enviar estos puntos a los servidores de Google Maps. Los servidores realizan el cálculo de la ruta y obtienen un resultado que envían al sistema ejecutor. Éste recoge los puntos y se los envía, de manera ordenada, a la interfaz, quien realiza la organización de los puntos para que sean entendibles por el usuario.

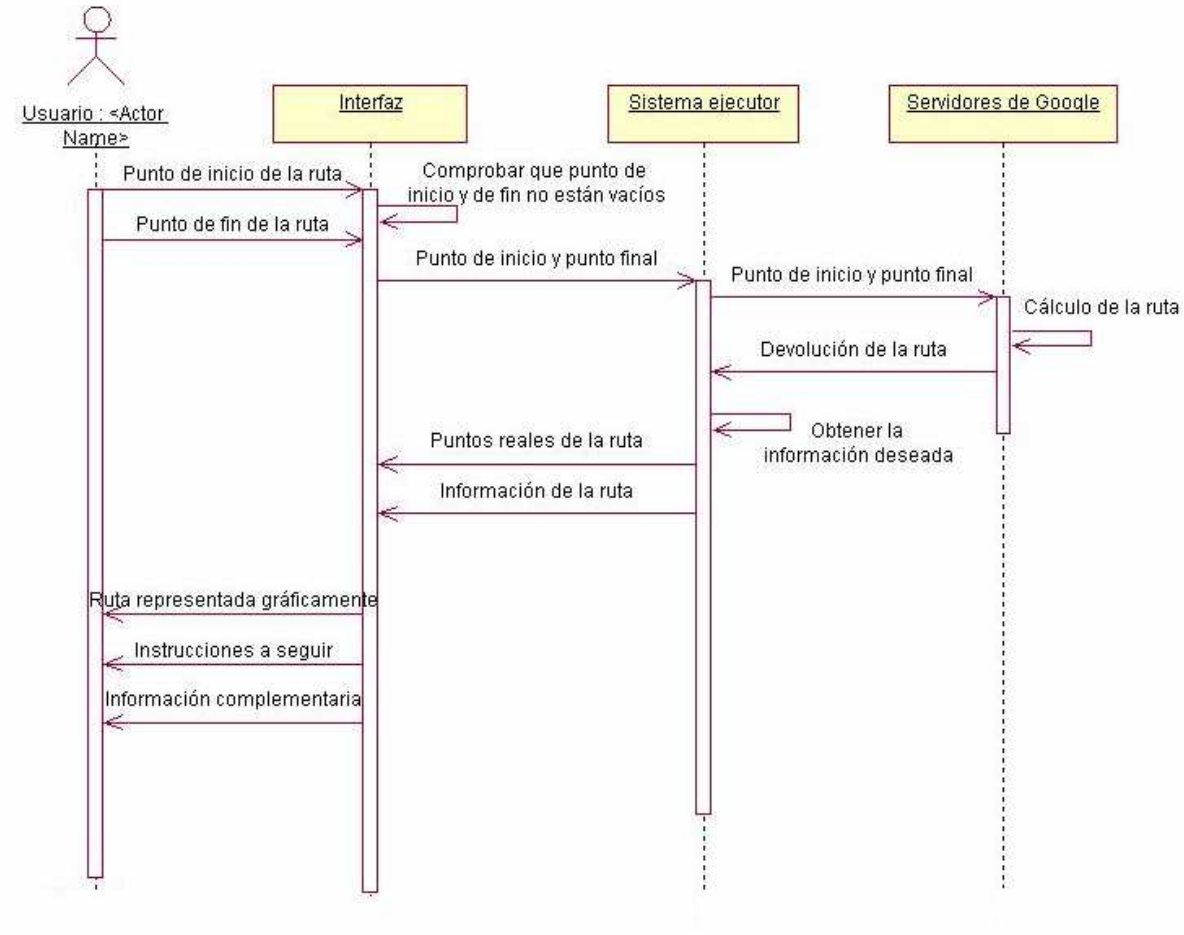
Cuando el usuario decide iniciar la simulación, la interfaz notifica la acción al sistema ejecutor quien calcula la velocidad en la que se va a ir mostrando la simulación y la información que se tiene que mostrar en cada punto. En este evento no actúan los servidores de Google Maps.

Si se selecciona ver rutas alternativas, la interfaz notifica al sistema ejecutor de esta acción y el sistema ejecutor ordena a los servidores que calculen las rutas alternativas para esos puntos. Cuando ya han calculado todas las rutas, el sistema ejecutor recoge las rutas alternativas y las pasa a la interfaz para que las muestre en el mapa y en el panel de informaciones de cada ruta.

Por último, si el usuario decide guardar la ruta, la interfaz creará una nueva ventana en la que mostrará el mapa con la ruta representada gráficamente y las instrucciones de cómo realizar el trayecto.

5.2- DIAGRAMAS DE SECUENCIA

- Diagrama de cómo calcular la ruta básica:

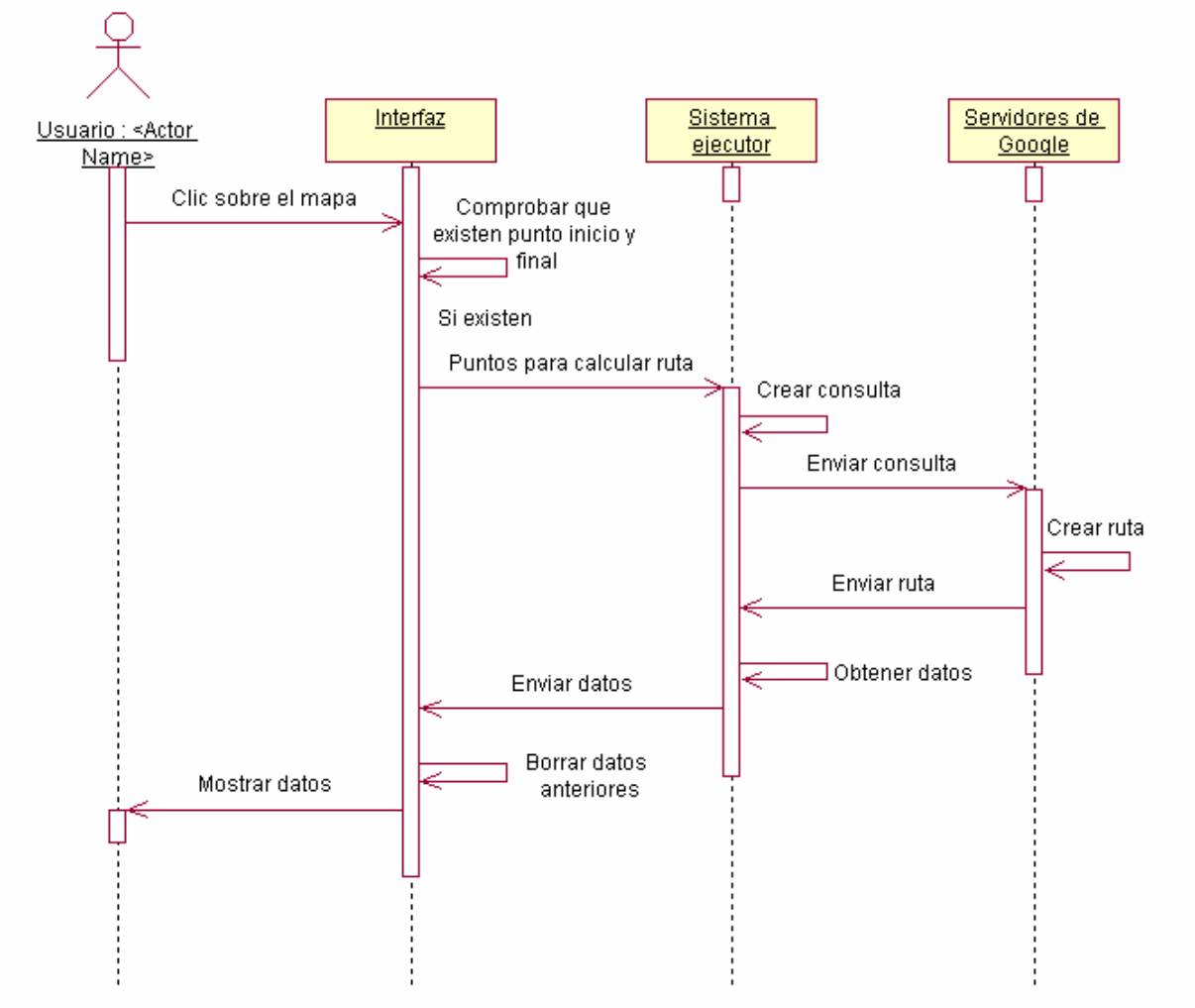


Cuando el usuario inicia nuestro sistema de navegación, el mapa se centra sobre la ciudad en la que se encuentra y el usuario debe introducir, en los dos espacios dedicados para ello, la cadena de texto con la dirección desde la que quiere salir y otra con la dirección a la que quiere llegar.

Es el usuario quien inicia la acción al introducir el punto inicial y final ya que la interfaz recibe la petición y la pasa al sistema ejecutor. El sistema ejecutor debe crear la consulta de acuerdo a las clases del API para que los servidores de Google puedan comprender la consulta y la envía a los servidores.

Cuando los servidores reciben la consulta, calculan la ruta óptima que une esos dos puntos y la devuelve al sistema ejecutor. Éste, se encarga de pasarle a la interfaz los datos que se quieren mostrar y la interfaz devuelve al usuario tanto una representación gráfica en el mapa como un texto explicativo con las indicaciones a seguir.

- Diagrama de cómo introducir puntos de paso en la ruta:

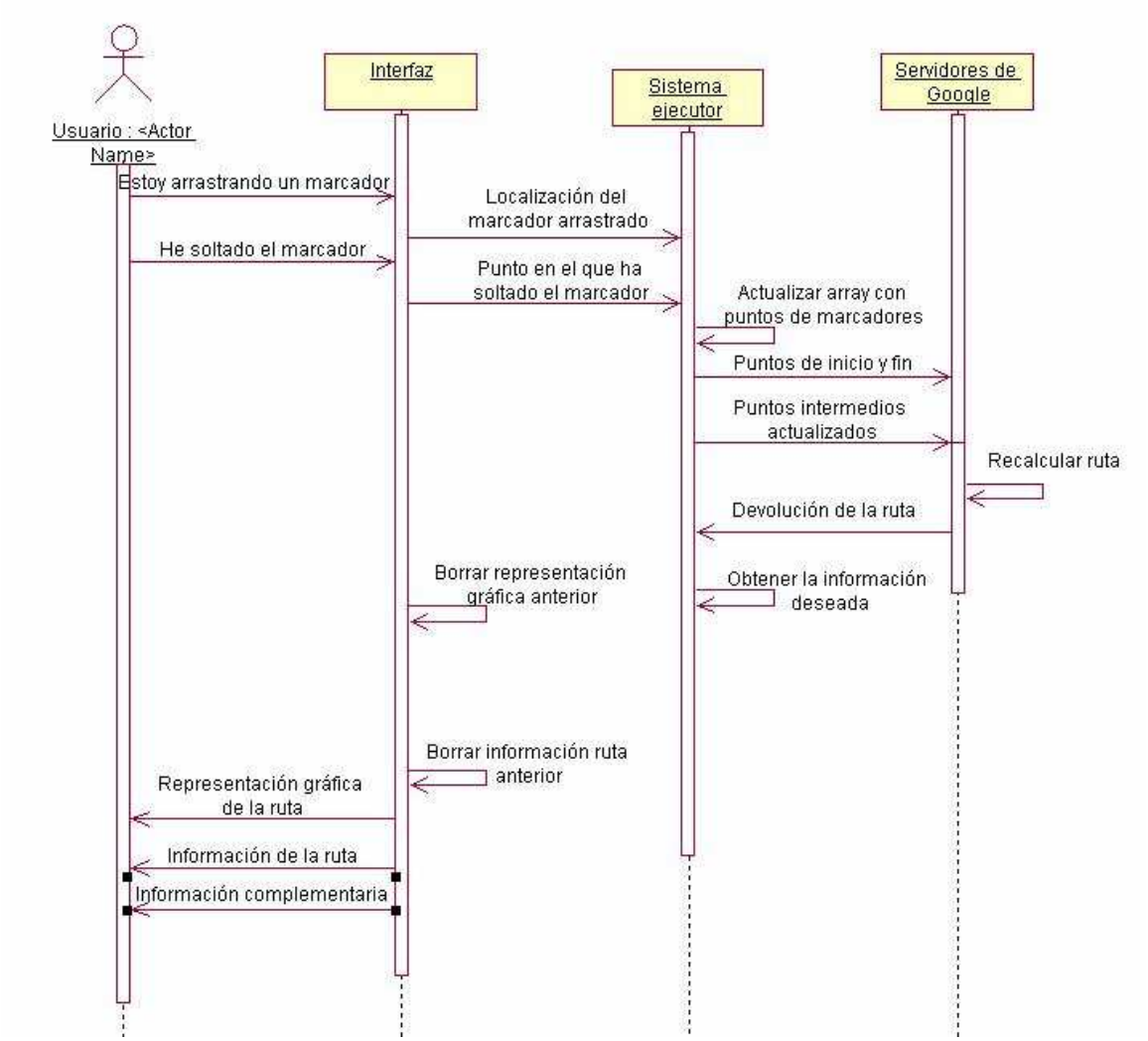


Cuando el usuario desea introducir etapas en el recorrido óptimo obtenido nada más introducir el punto de inicio y final, debe hacer clic sobre el mapa sobre el punto por el que desea pasar. Al centrar el sistema de navegación en ciudades, es fácil que el usuario quiera desviar su camino para hacer algún tipo de parada en el camino.

Cuando el usuario hace clic, la interfaz recoge el punto sobre el que ha clicado, y se lo envía al sistema ejecutor junto con los puntos iniciales y finales. El sistema ejecutor, a su vez, envía a los servidores de Google la nueva consulta que esta vez incorpora el punto de paso seleccionado.

Una vez que los servidores hayan recalculado la ruta, el sistema ejecutor recoge la ruta y envía la información a la interfaz para que la haga visible al usuario.

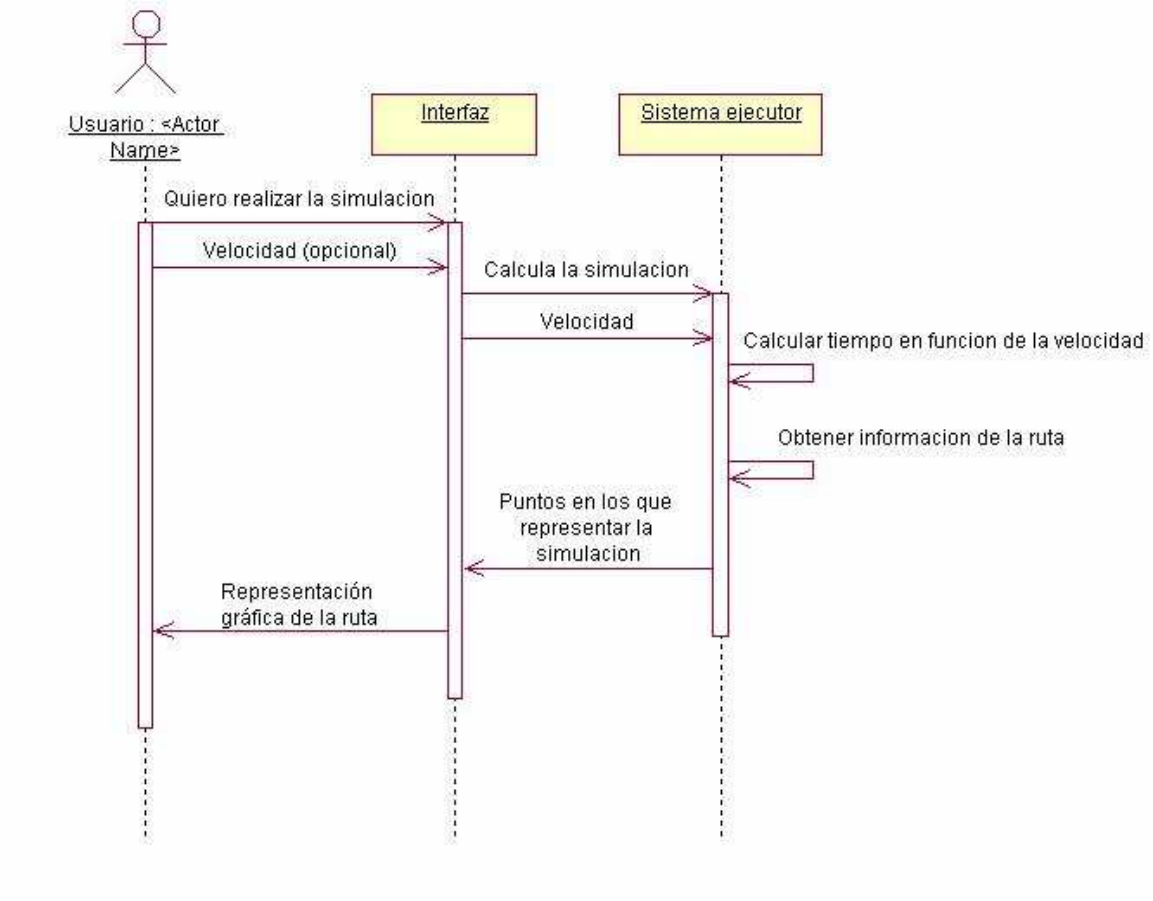
- Diagrama de cuando el usuario arrastra un marcador del mapa para recalcular la ruta:



Cuando el usuario hace clic sobre el mapa, está indicando que quiere añadir etapas, o puntos de paso, en la ruta que se ha creado. Esto lo puede hacer bien para evitar pasar por una calle que ha mostrado la ruta o también para indicar que quiere pasar por un punto concreto que no está en el resultado. Una vez creada la etapa, se señalizan con marcadores que se pueden arrastrar. Estos marcadores señalizan un punto que se ha pasado a los servidores para que recalculen la ruta. Si el usuario se ha equivocado o simplemente no quiere pasar por ese punto sino por otro, puede arrastrar el marcador y la ruta ya no pasará por el punto anterior sino que ahora pasará por el punto en el que se ha soltado el marcador que se ha arrastrado.

Este nuevo punto es recogido por la interfaz mediante "*listeners*" y lo pasa al sistema ejecutor. Éste debe ser capaz de eliminar el punto en el que se encontraba inicialmente el marcador e introducir el nuevo en la consulta que envía a los servidores de Google. Cuando se crea la nueva consulta, la interfaz borra el marcador anterior y crea uno nuevo en el nuevo punto. También borra la ruta anterior pintada sobre el mapa y pinta la nueva.

- Diagrama de cómo simular la ruta:

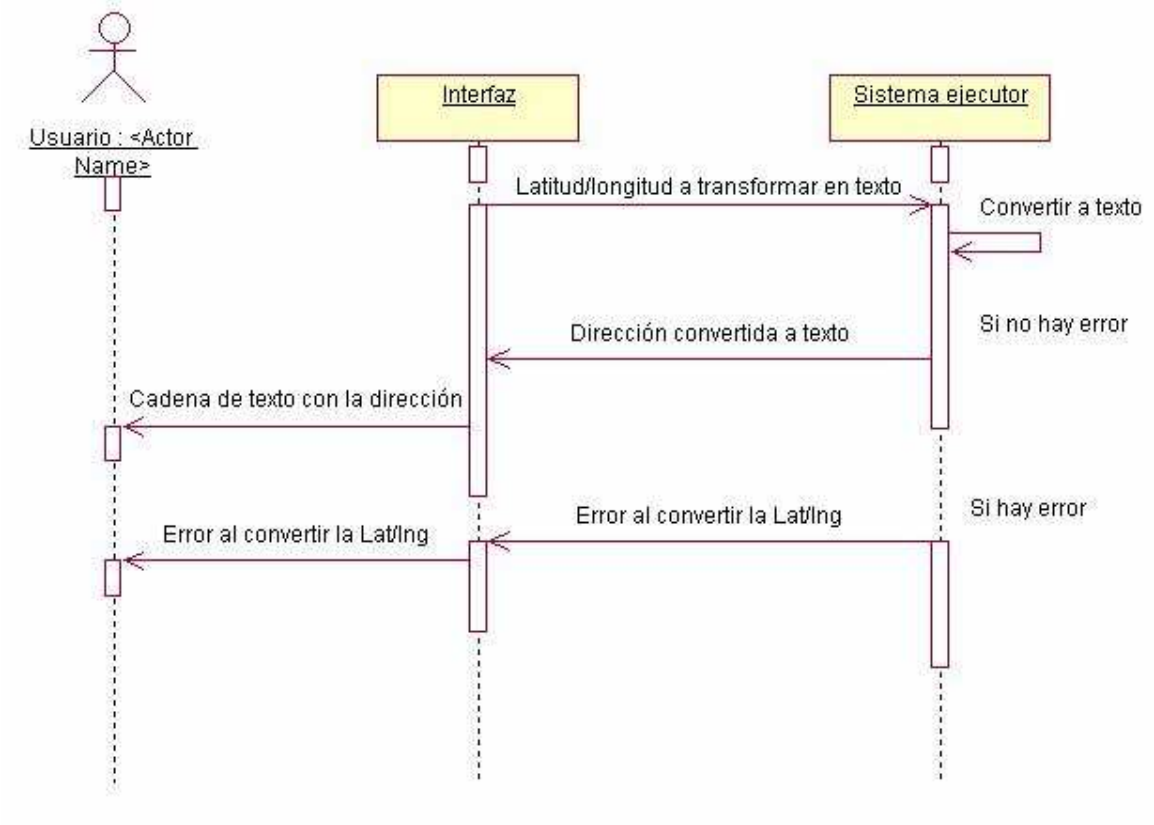


Hemos introducido la posibilidad de ver una representación gráfica del trayecto a seguir en tiempo real. La interfaz notifica al sistema ejecutor el deseo de llevar a cabo una representación y, si el usuario ha introducido una velocidad a la que realizarla, también le pasa la velocidad.

El sistema ejecutor analiza todos los puntos que tiene para esa ruta, son puntos internos para los cuales existe una posición, en formato Latitud/Longitud, y para algunos de ellos también existen unas instrucciones (por ejemplo gire a la derecha, o salga por la tercera salida). Con toda esta información, se crea una serie de tablas para que la interfaz pueda mostrar una imagen en el punto real con su indicación concreta para ese punto.

Con respecto a la velocidad, si el usuario introduce una velocidad en km/h el sistema ejecutor divide la distancia total entre el número de puntos y lo relaciona con la velocidad introducida, para así calcular el nuevo tiempo necesario para realizar la ruta. En caso de no introducir ninguna velocidad, el sistema ejecutor realizará las mismas acciones pero con la velocidad utilizada por defecto por los servidores.

- Diagrama de cómo convertir una dirección en formato Latitud/Longitud a cadena de texto para que sea comprensible por los usuarios:

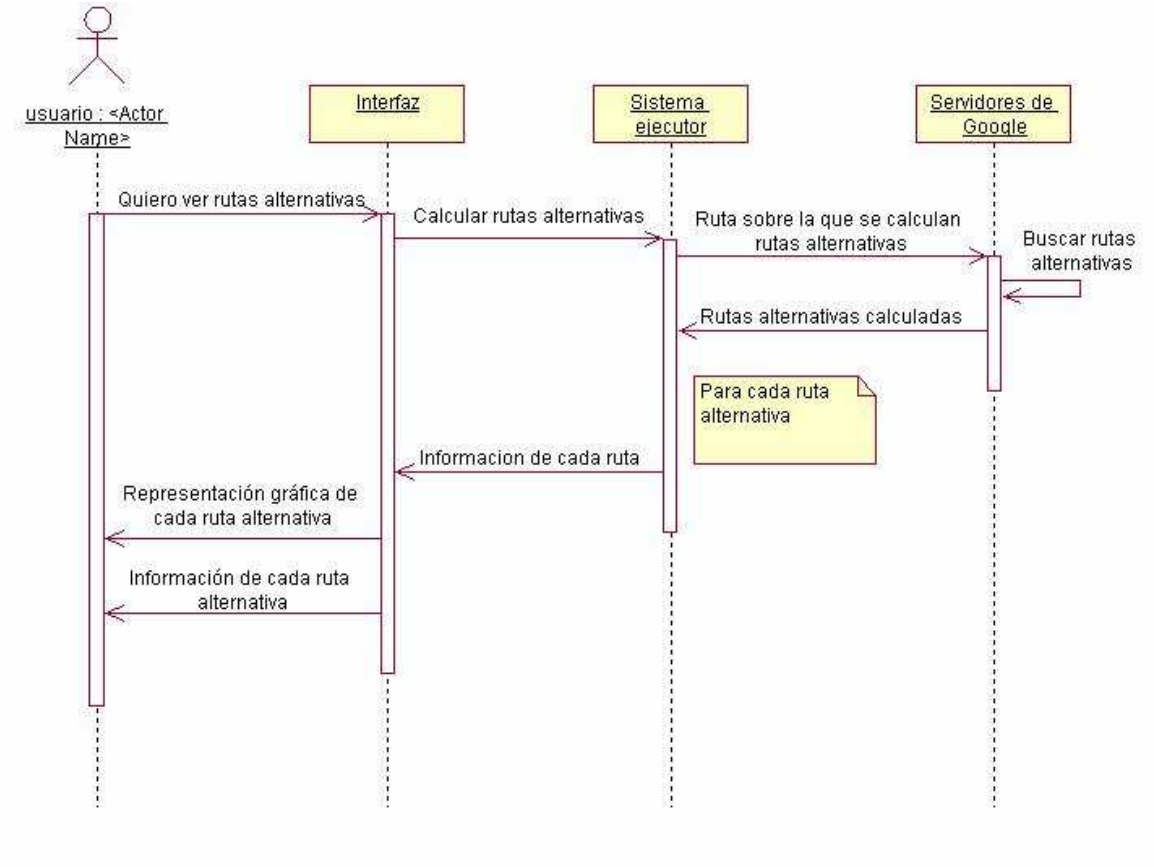


Hay ocasiones en las que el usuario le pasa a la interfaz una posición en formato Latitud/Longitud. Esto es por ejemplo cuando hace clic sobre el mapa y se crea un marcador. Después, si el usuario quiere ver cuál es la dirección sobre la que ha hecho clic, es necesario que el sistema convierta la dirección a formato cadena de texto.

Para realizar esta acción no es necesaria la ayuda de los servidores sino que la realizamos gracias a una función que se encarga de convertir a cadenas de texto posiciones geodésicas.

Es posible que no existe una dirección exacta para el punto sobre el que se ha hecho clic así que o bien muestra un mensaje de error o bien muestra una aproximación mostrando únicamente el código postal de la zona y la ciudad.

- Diagrama de alternativas



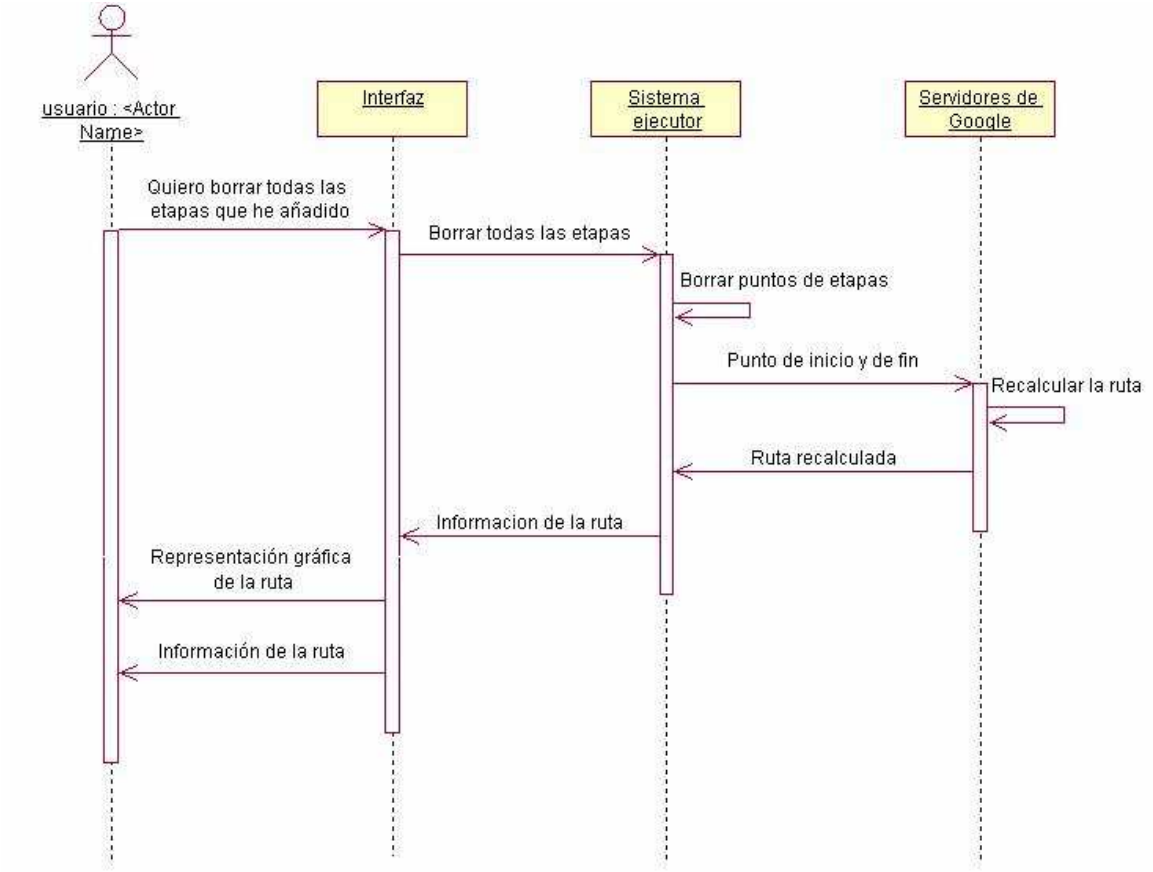
Es posible que el usuario quiera ver rutas alternativas a la ruta óptima que se ha obtenido. Bien porque sabe que esa ruta a la hora que va a pasar hay mucho tráfico o bien porque simplemente quiere ver nuevas rutas.

La interfaz notifica al sistema ejecutor que se quieren ver las alternativas a la ruta y el sistema ejecutor ordena a los servidores de Google que calculen estas rutas.

Cuando el sistema ejecutor recibe las rutas alternativas debe recorrer una a una cada ruta para sacar la información de cada una e ir pasándoselas a la interfaz.

La interfaz a su vez, mostrará de manera gráfica en el mapa las nuevas rutas. Lo hemos creado para que las muestre en un color diferente a la óptima para que el usuario pueda diferenciarlas de manera sencilla. Además, creará una nueva ventana con información sobre las rutas alternativas para indicar cómo y por dónde se debe ir así como el tiempo y la distancia de las nuevas rutas.

- Diagrama de cómo limpiar todos los puntos intermedios de la ruta:

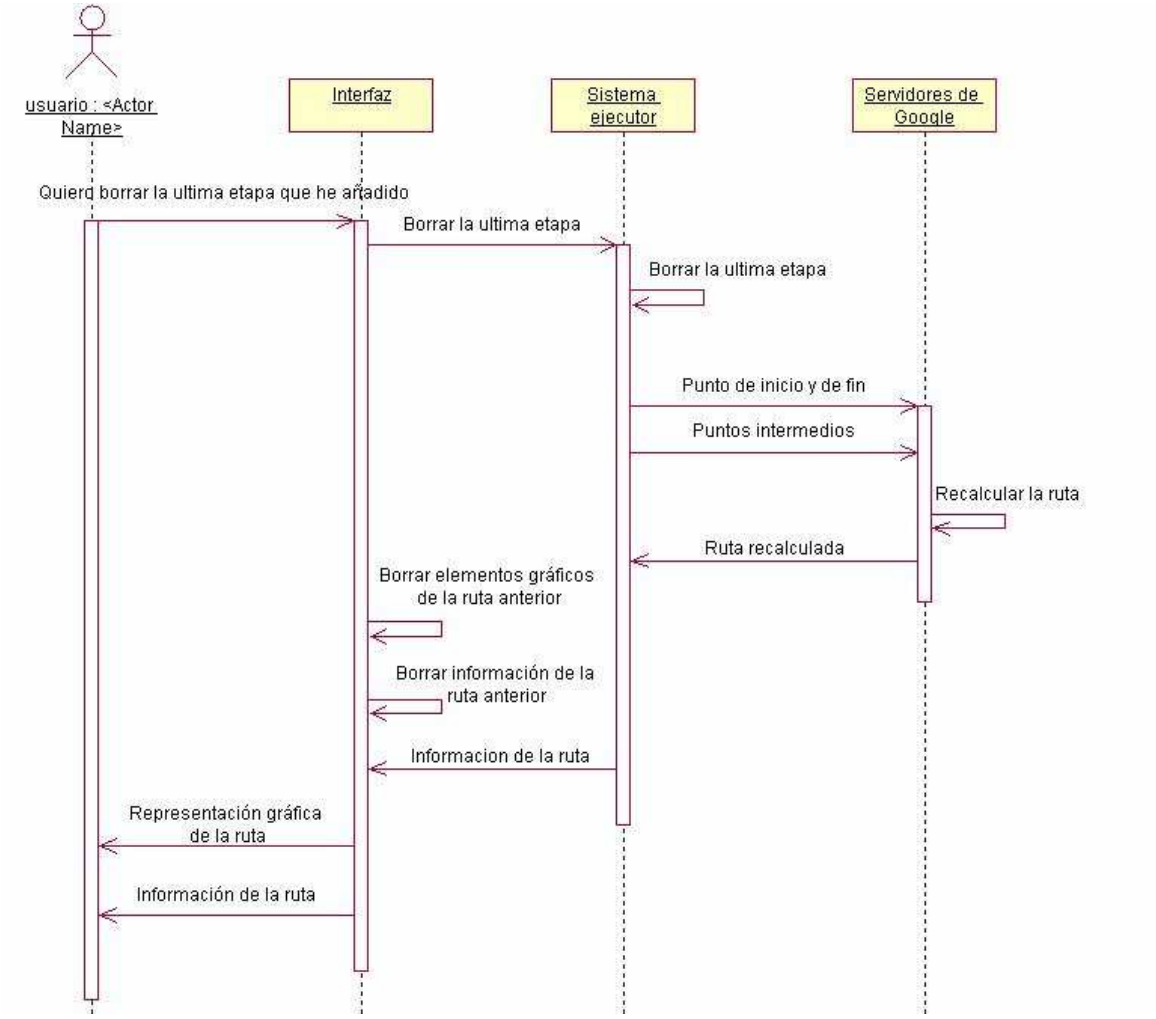


Es posible que el usuario haya introducido muchos puntos intermedios entre el origen y destino de una ruta. La introducción de los puntos intermedios la hemos explicado en uno de los diagramas anteriores, por lo que ya sabemos que a medida que ha hecho clic sobre el mapa se han ido añadiendo marcadores y con ello la ruta se ha ido modificando.

Puede darse el caso en el que el usuario haya introducido demasiados puntos o que al final decida pasar únicamente por la ruta óptima, por lo que quiere borrar todos los puntos que ha introducido. Esto se puede hacer de uno en uno o bien con el botón "*Limpiar ruta*" que borra todos los marcadores de vez y vuelve a calcular la ruta óptima.

Para ello, el sistema ejecutor vuelve a solicitar a los servidores de Google la ruta óptima pasándole únicamente el punto de inicio y de fin. Los servidores recalculan la ruta y la interfaz borra todos los resultados anteriores para mostrar los nuevos.

- Diagrama explicativo de cómo eliminar un único punto intermedio

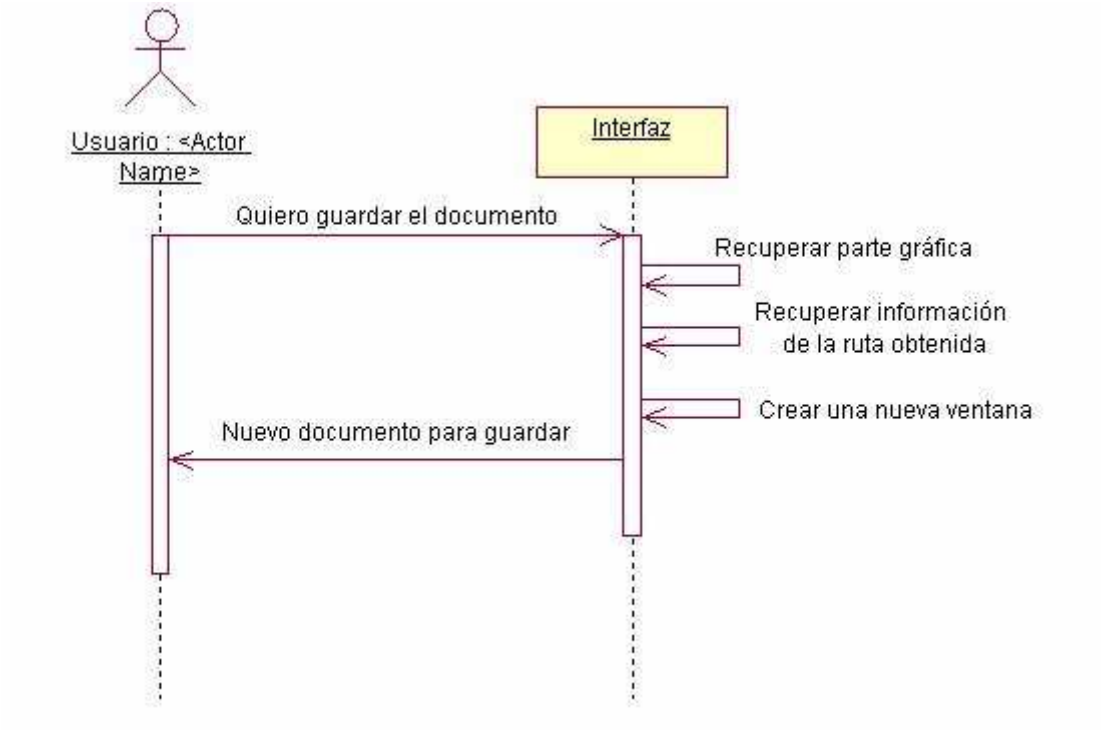


A medida que el usuario ha ido haciendo clic sobre el mapa, se han creado marcadores y se ha recalculado la ruta. Estos puntos intermedios los hemos tenido que almacenar en un array que el sistema ejecutor le pasa a los servidores junto con el punto de inicio y de fin para que calculen la nueva ruta con las etapas.

Si el usuario se ha equivocado y ha introducido un punto que no quería introducir, puede eliminarlo pulsando el botón "*Eliminar punto*". La interfaz se encarga de eliminar el punto del mapa y del array para enviárselo de nuevo al sistema ejecutor para que éste ordene a los servidores que recalculen la ruta.

Una vez recalculada la ruta, el sistema ejecutor envía la información a la interfaz. La interfaz se encarga de borrar los datos anteriores relativos al punto que se ha eliminado y repinta la ruta otra vez sobre el mapa.

- Diagrama para guardar la ruta calculada en un archivo



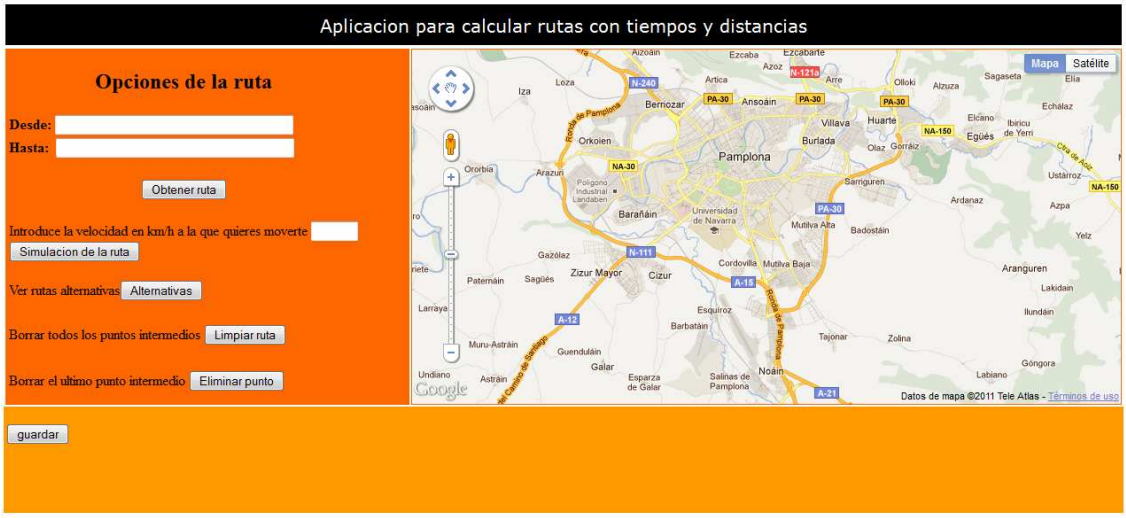
Es posible que el usuario desee guardar los resultados bien para imprimirlos o para tenerlos almacenados. En esta acción sólo interactúan usuario e interfaz, pues ya está la ruta calculada y es la interfaz la que realiza esta acción.

Cuando el usuario pulsa el botón de "*Guardar*" la interfaz debe ser capaz de crear una nueva ventana con el mapa y la información de cómo seguir el trayecto.

Captar el mapa no es muy sencillo, ya que Google todavía está intentando mejorar esta utilidad en la versión 3. El problema es que se puede exportar el mapa pero no exporta ni la ruta dibujada ni los marcadores. A base de investigar en JavaScript conseguí introducir el mapa, con la ruta y los marcadores en la nueva ventana que creo. Además, añado las indicaciones existentes para esa ruta y facilito la opción de imprimir.

6- ANÁLISIS Y DISEÑO

El interfaz del sistema desarrollado se muestra en la siguiente figura.

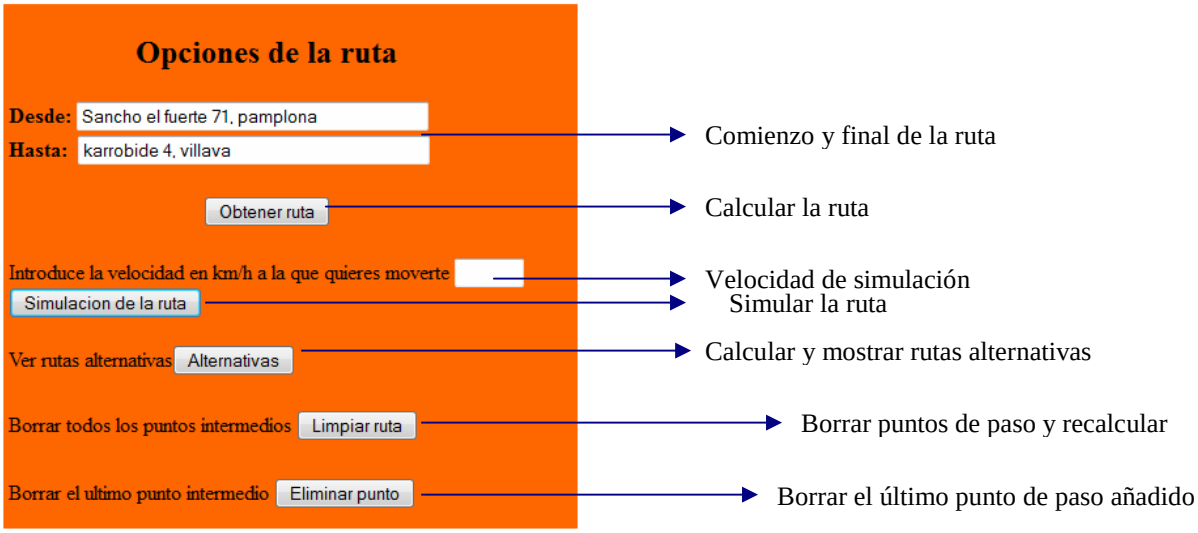


Sobre dicha imagen se marcan las funcionalidades que se implementan para el sistema: mapa, lugar donde muestran los marcadores de inicio (1), fin (2) y puntos de paso (3); rutas alternativas (4).

En la siguiente imagen se ve la simulación de la ruta mediante un icono que representa el movimiento a lo largo de la misma.

En la parte izquierda de la imagen se muestra los botones y opciones que permiten interactuar al usuario con el sistema. En algunos casos servirán para realizar una acción directamente y en otras cómo parámetros para que se realicen esas acciones.

En el primer mapa mostrado podemos ver cómo los puntos 'desde' y 'hasta' se representan con dos iconos de color verde (1 y 2). La velocidad no se muestra representada sino que se emplea para el movimiento del vehículo, que podemos ver en el segundo mapa.



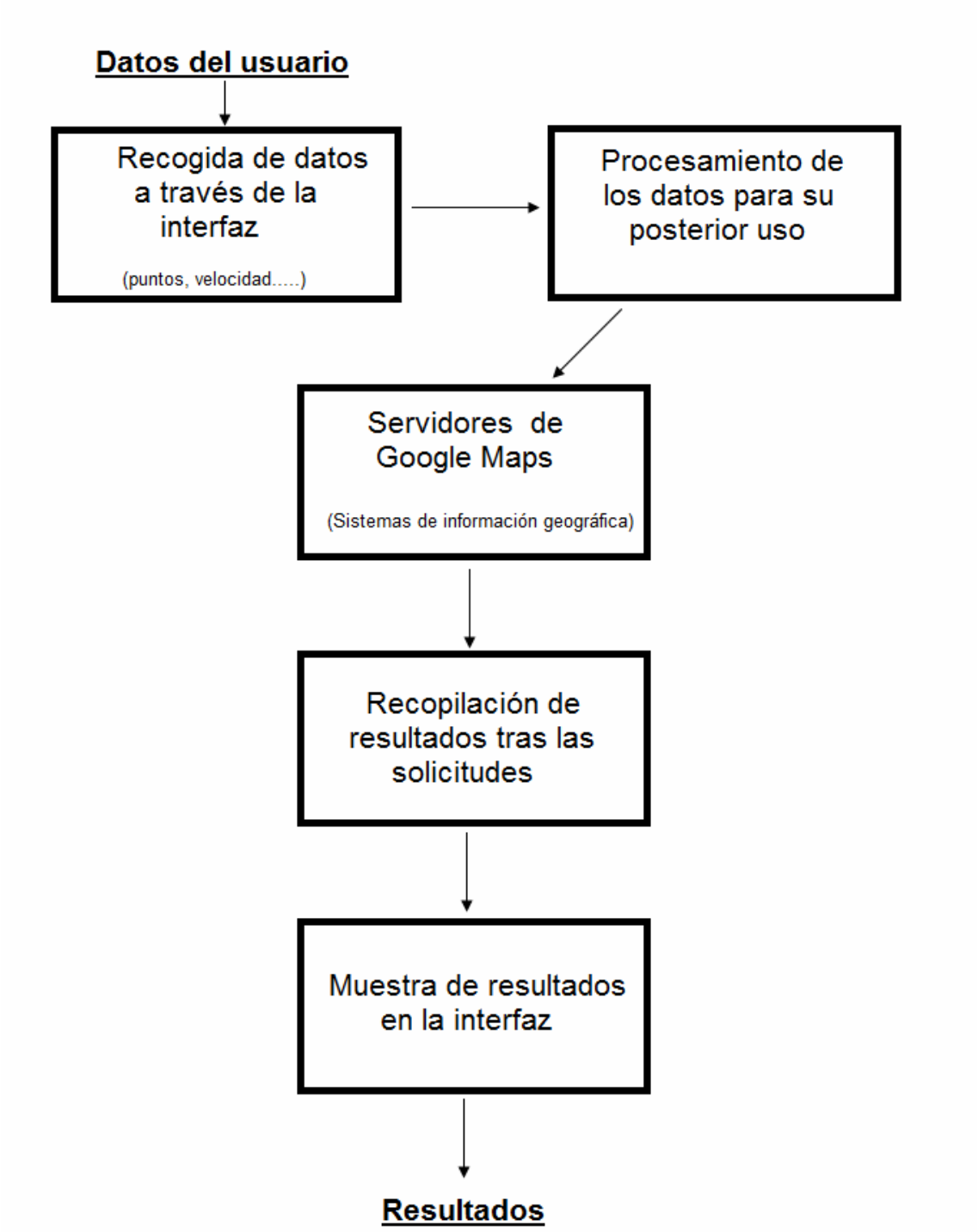
Y en la parte inferior se muestra el espacio donde se escriben los mensajes para el usuario.

Después de ver las diferentes partes de la interfaz, explicaremos cómo se realizan las acciones correspondientes en una serie de diagramas. En cada uno se mostrará si son necesarios los datos que se emplearán y también los pasos a seguir por los diferentes bloques de trabajo.

El sistema desarrollado está orientado a módulos y las funcionalidades se organizan en diferentes módulos o bloques funcionales que agrupan los procedimientos para realizar diferentes tareas. Estos bloques pueden representar una acción directa, llevada a cabo por una serie de funciones, o un sistema de trabajo, cómo son los servidores de Google Maps.

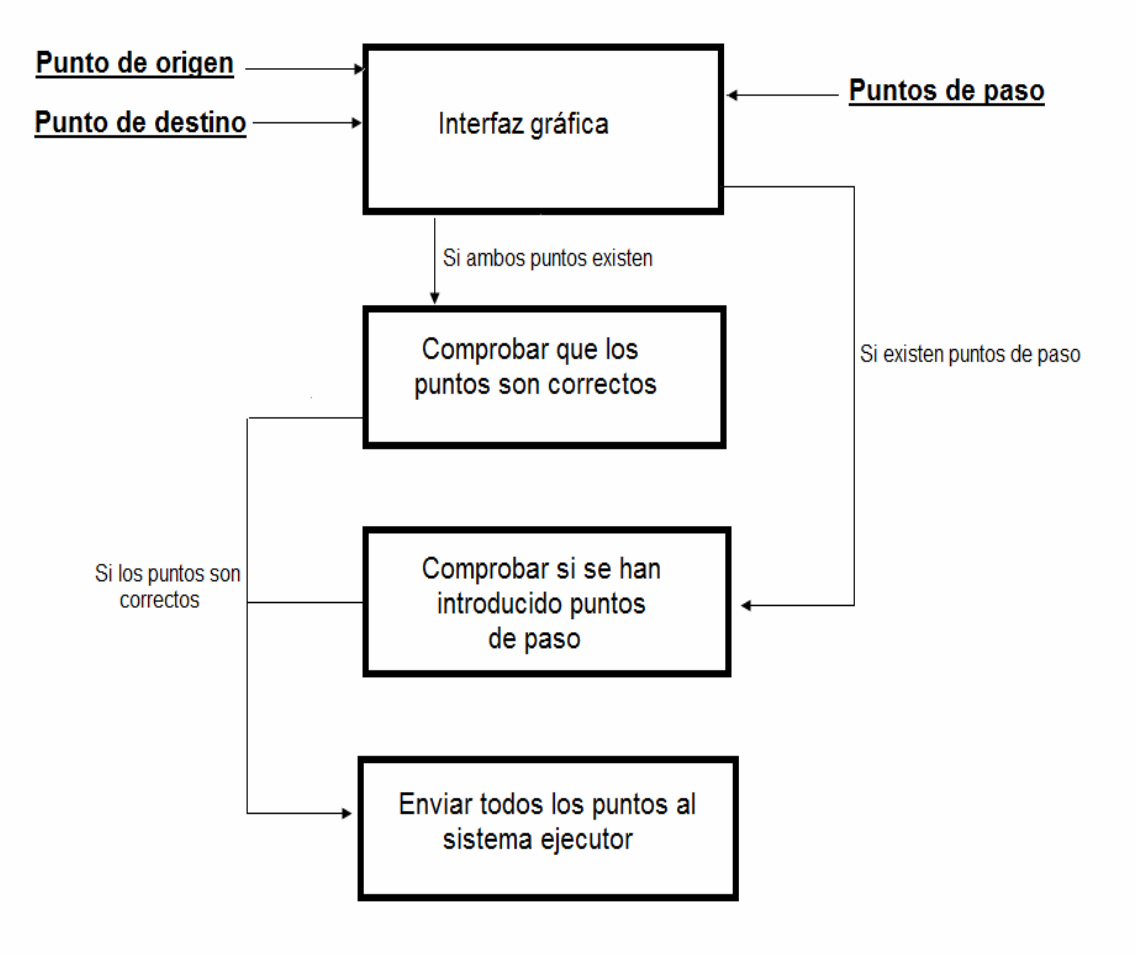
En la siguiente sección se van a describir los diferentes bloques funcionales y la funcionalidad que proporciona cada uno de ellos.

6.1- DIAGRAMAS DE BLOQUE



Este diagrama de bloques muestra, de manera general, el funcionamiento del proyecto, desde la interacción del usuario, pasando por la interfaz, el sistema ejecutor hasta los servidores de Google para realizar la consulta y el proceso inverso para devolver resultados al usuario.

- Diagrama para ver la captación de los datos necesarios para calcular una ruta

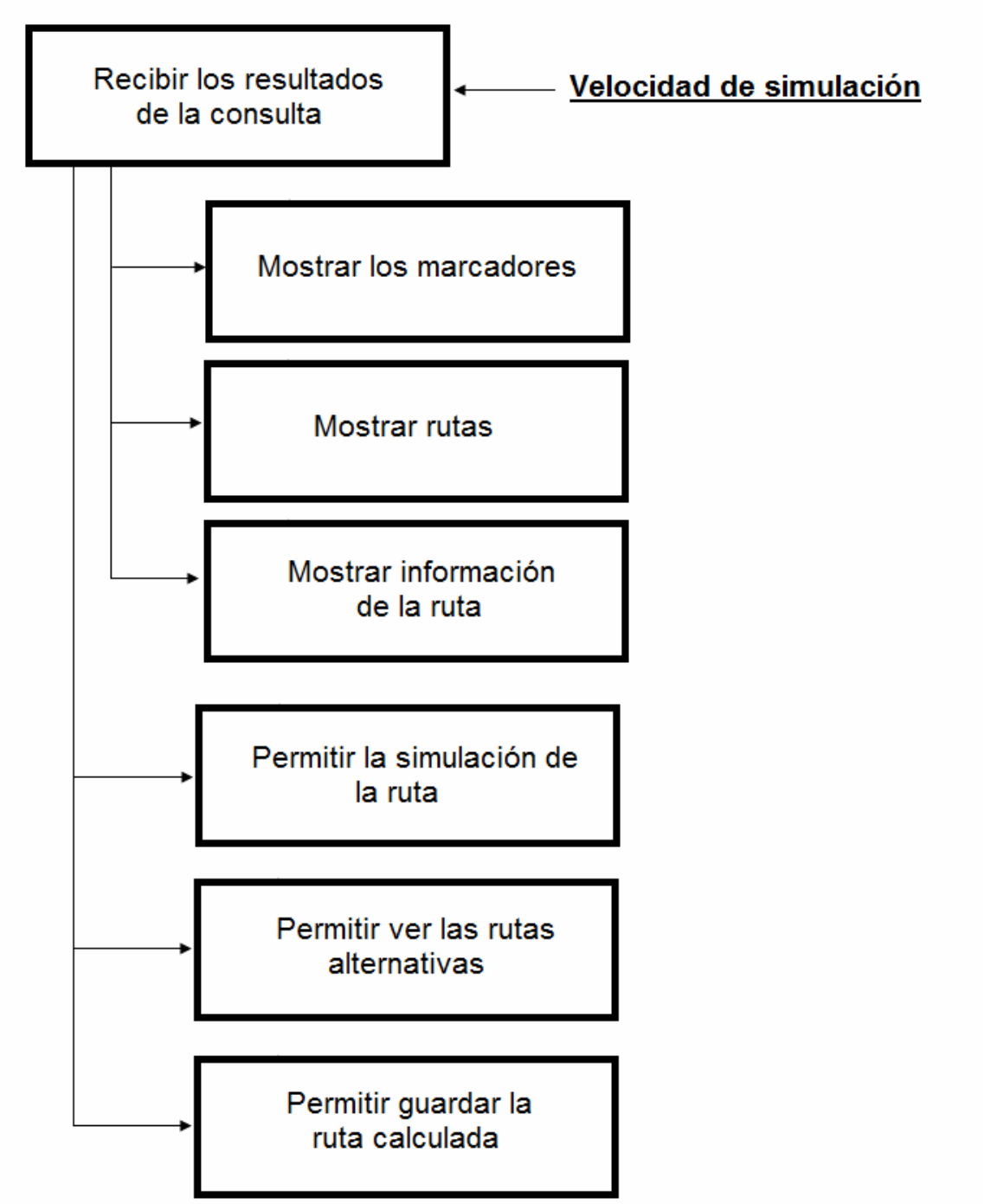


En este diagrama vemos cómo se recopilan los datos necesarios para que se cree una ruta. Cuando se dispone de ellos y se ha comprobado que son correctos, serán enviados al sistema ejecutor para que trabaje con ellos. En este momento, él se encargará de procesar la consulta y de devolvernos los resultados para poder mostrarlo en la interfaz.

Si alguno de los datos que se reciben por la interfaz no es correcto, no se podrá enviar ninguno de ellos al otro sistema, ya que será imposible que la consulta se realice sin alguno de ellos.

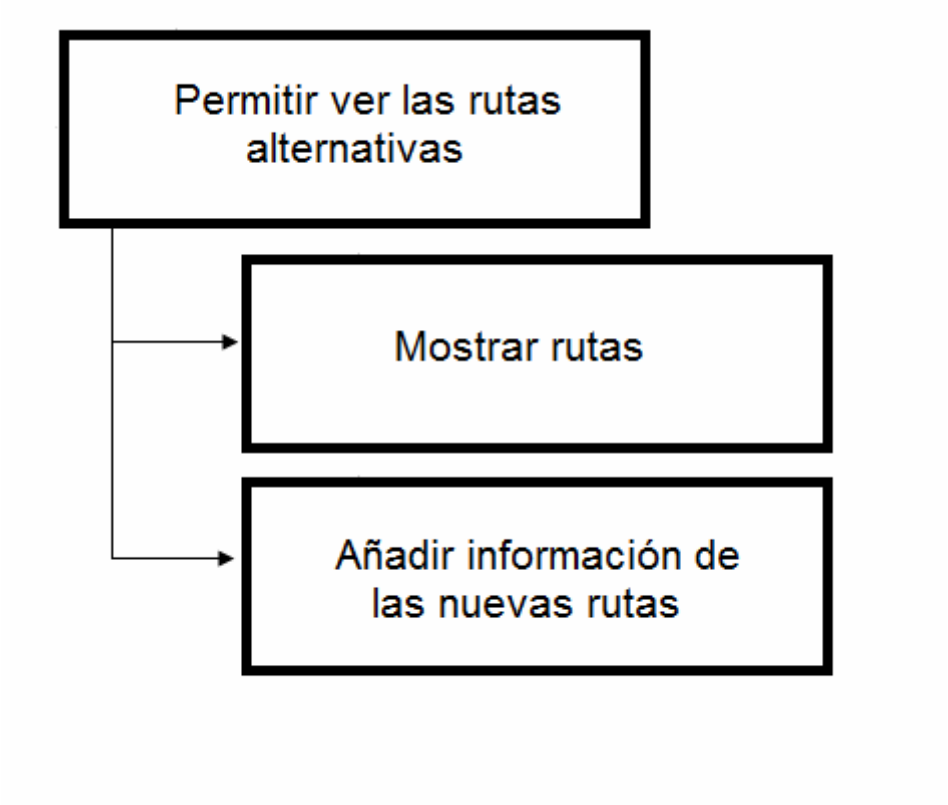
Para enviar los puntos de paso se emplea una estructura de tipo Array, ya que de este modo se facilita su posterior uso al sistema ejecutor, que deberá enviarlos en su consulta de esta misma manera.

- Diagrama para ver la representación de una ruta



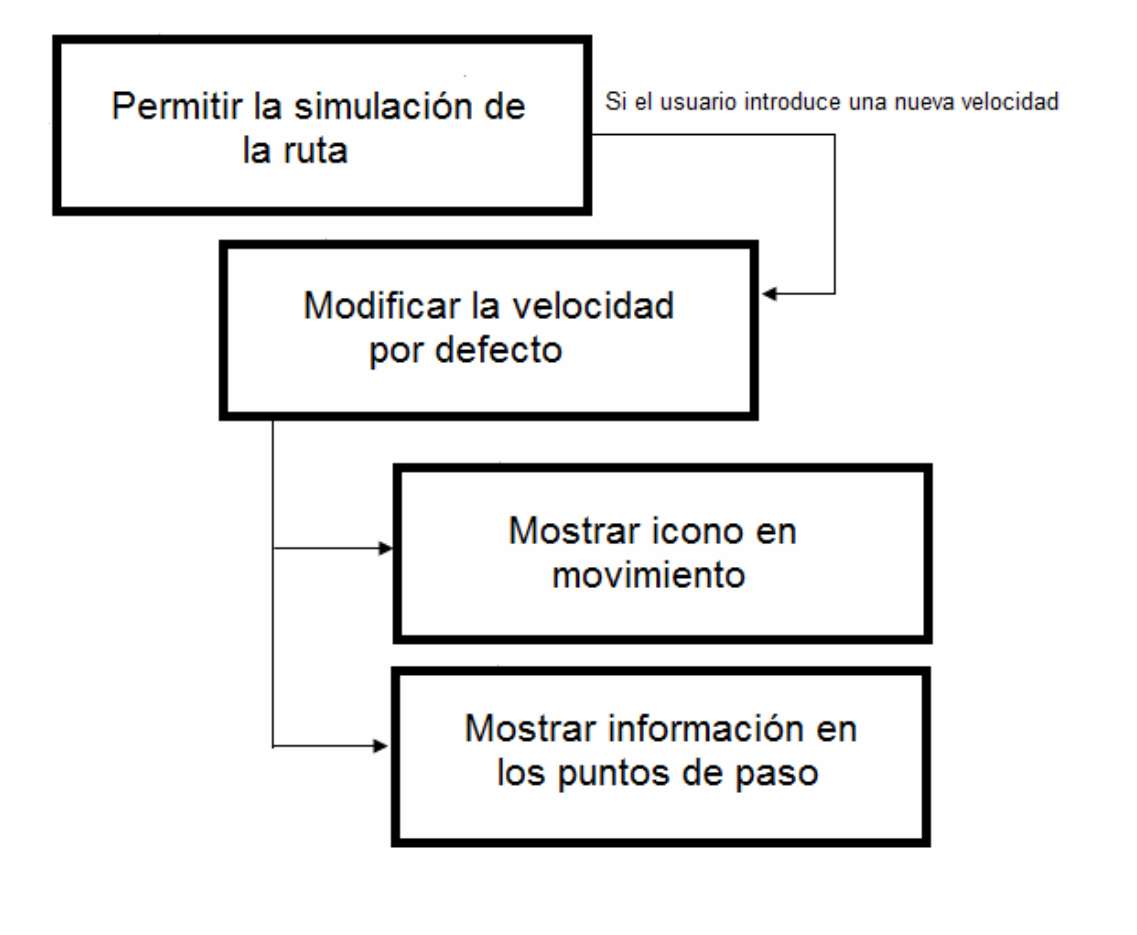
Es este segundo diagrama podemos ver que se realiza una vez que se reciben los resultados de la consulta del sistema ejecutor para calcular la ruta. Además de la representación de la ruta, a partir de ese momento se permitirá que se realicen varias acciones más sobre la ruta calculada.

- Diagrama para ver la representación de las rutas alternativas



Es este tercer diagrama podemos ver que se realiza si el usuario desea ver las rutas alternativas a la calculada como ruta óptima. Sobre el mismo mapa, aunque con distinto color, se mostrarán el resto de rutas y se añadirá su información a la que ya estaba disponible anteriormente.

- Diagrama para ver la simulación de la ruta



Aquí vemos cómo se va a realizar la simulación de la ruta calculada sobre el mapa. La principal variante es si se realizará el movimiento según la velocidad que por defecto estima Google Maps o si se toma la que el usuario decide que llevará en ese recorrido. Después se trata de mostrar el movimiento por dicha ruta y de mostrar información concreta sobre los diferentes puntos de paso e instrucciones a cumplir en el recorrido.

7- IMPLEMENTACIÓN [4,5]

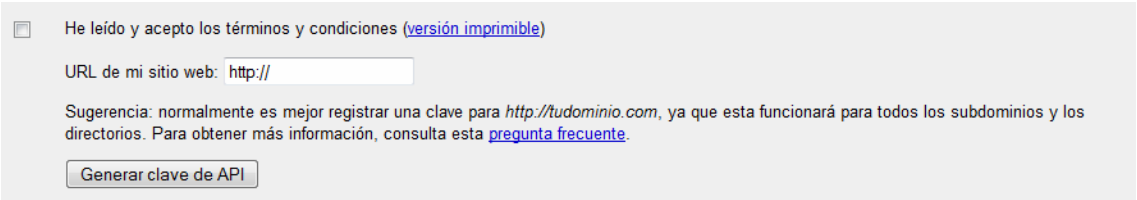
En este apartado se van a destacar las partes más relevantes de la programación realizada en la aplicación. A partir del API de Google Maps, se ha podido crear un sistema de localización del cual veremos qué es lo que se ha necesitado modificar para conseguir un funcionamiento más adecuado a las necesidades de la aplicación.

1. REGISTRO EN GOOGLE MAPS

El API de Google Maps permite insertar Google Maps en páginas Web. La clave única de Google Maps API es válida para un "directorio" o para un dominio único. Para obtener una clave de API de Google Maps, es necesario disponer de una cuenta de Google, ya que la clave de API estará conectada con la cuenta de Google.

A continuación veremos algunas condiciones que impone Google Maps por utilizarlo.

- No existe limitación en el número de visitas diarias a la página que se pueden generar mediante Google Maps API.
- El número de solicitudes de codificación geográfica que se pueden enviar diariamente es limitado.
- El API de Google Maps no incluye publicidad.
- Si se utilizan otras API junto con el API de Google Maps, se debería consultar también sus condiciones del servicio. En concreto, tener en cuenta que el control GoogleBar del API de JavaScript de Google Maps utiliza el API AJAX para búsquedas y que esa API dispone de sus propias condiciones del servicio.
- Los usuarios finales deberán poder acceder de forma gratuita a tu servicio.
- No se puede modificar ni ocultar los logotipos ni la atribución del mapa.
- Se deberá indicar si la aplicación utiliza un sensor (por ejemplo, un localizador GPS) para identificar la ubicación del usuario.
- Se puede utilizar el API (salvo en el caso de Static Maps API) en sitios Web o en aplicaciones de software. En el caso de sitios Web, se debe con la URL en la que se encuentre tu implementación. En el caso de otras aplicaciones de software, regístrate con la dirección de la página en la que se puede descargar tu aplicación.
- Google actualizará las API periódicamente.
- Google se reserva el derecho de suspender o finalizar el uso de este servicio en cualquier momento.



2. AUTOLOCALIZACIÓN DEL MAPA

Un tema importante es poder situar el mapa en el lugar concreto en el que se encuentra el usuario. De este modo, automáticamente la aplicación se redirigirá a la posición geográfica y facilitará la visualización del mapa para interactuar con él.

Para ello se creará el mapa en una localización elegida por el programador, que puede relacionarse con el lugar en el que se tenga pensado utilizar la aplicación. Y una vez que ya se tenga el mapa creado se localizará sobre la posición del usuario.

Esta es la función que se encarga de cargar inicialmente el mapa. Dentro de ella, y una vez creado dicho mapa, llamaremos a una nueva función encargada de la localización.

```
////////////////////////////////////
//Funcion que crea el mapa inicial//
////////////////////////////////////
function load() {
    directionsDisplay = new google.maps.DirectionsRenderer({suppressMarkers: true});
    //Inicializamos las opciones del mapa
    centro = new google.maps.LatLng(36.820336190453304, -1.6458892822265625);
    var myOptions = {
        zoom: 12,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        center: centro
    }
    geocoder = new google.maps.Geocoder();      //Variable para transformas un elemento Latlng a cadena de texto
    map = new google.maps.Map(document.getElementById("map"), myOptions);
    directionsDisplay.setMap(map);
    geolocalizame();
    //Listener para anadir un punto cuando se hace click sobre el mapa
    google.maps.event.addListener(map, 'click', function (event){
        //Anado los puntos al array de puntos intermedios que enviamos a calcRoute()
        listaPuntos.push({
            location:event.latLng,
            stopover:true
        });
        calcRoute();
    });
    calcRoute();
    //Almacenamos los puntos iniciales y finales que mete el usuario en los text box
    puntoInicial = document.form_ruta.desde.value;
    puntoFinal = document.form_ruta.hasta.value;
```

Vemos a continuación cómo se realiza esa localización.

```
function pedirPosicion(pos) {
    centro = new google.maps.LatLng(pos.coords.latitude,pos.coords.longitude);
    map.setCenter(centro);
    map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
}

function geolocalizame(){
    navigator.geolocation.getCurrentPosition(pedirPosicion);
}
```

Se recalcula el centro, o posición central del mapa, y se sitúa el mapa sobre dichas coordenadas. También es necesario volver a indicar con que tipo de mapa se va a trabajar.

3. PUNTOS INTERMEDIOS

Los puntos de paso son elementos importantes en esta aplicación y tienen un comportamiento diferente a los de inicio y fin. Ahora vamos a ver lo que los diferencia y que acciones se producen como resultado.

Como diferencia principal, se van a poder arrastrar en el mapa. Es decir, podremos modificar su localización en el mapa de una forma muy sencilla.

```
////////////////////////////////////
//Funcion que pinta los marcadores//
////////////////////////////////////
function makeMarker( position, icon) {
    marker=new google.maps.Marker({
            position: position,
            map: map,
            icon: icon,
            draggable: true
    });
    //Listener para guardar el numero de marcador que se comienza a mover
    google.maps.event.addListener(marker, "dragstart", function(event){
        coordenadas1 = pintarOrigenDestino(this.position);
        for (i=0;i<listaPuntos.length;i++){
            if (coordenadas1.equals(listaPuntos[i].location)){
                numMarcador=i;
                //Borro el contenido de coorde de la posicion numMarcador para borrar ese punto
                var aux = coorde.slice(numMarcador+3);
                coorde = coorde.slice(0,numMarcador+2);
                coorde = coorde.concat(aux);
                break;
            }
        }
        return numMarcador;
    });

    //Listener para cambiar las coordenadas del marcador que se ha movido
    google.maps.event.addListener(marker, "dragend", function(event){
        this.setVisible(false);
        listaPuntos[numMarcador].location=this.position;
        calcRoute();
    });
```

Aquí vemos cómo se crean y se les añaden dos listeners para controlar su movimiento en el mapa. Lo que les diferencia de los otros puntos es que tienen la opción ‘draggable’, la cual permite que se arrastren.

En el primer listener, se controla que el marcador comience a arrastrarse. En primer lugar se almacenan las coordenadas actuales, y a partir de ellas se buscará la posición del Array que la contiene. Se deberá eliminar el contenido correspondiente a esta posición.

En el segundo listener, se controla el reposicionamiento del marcador. Por lo tanto, se oculta en marcador actual y se almacenan las nuevas coordenadas obtenidas al colocarlo en una nueva posición del mapa. Después se manda calcular la ruta para representarlo en ese nuevo cálculo.

Un tema muy importante a tener en cuenta con estos nuevos puntos es que, aunque ya se tengan unas nuevas coordenadas para insertar el marcador, no tienen porque ser las definitivas. Es decir, puede ser que el marcador se sitúe en un lugar imposible de transitar y que cuando se calcule la nueva ruta no se puedan utilizar las

coordenadas almacenadas. Por ello, cuando se recalcule la ruta con ese nuevo punto, se tendrá que comprobar si es posible utilizarlo, o si no, almacenar el punto que la aplicación haya utilizado para el cálculo. Este nuevo punto estará localizado lo más cerca posible de la situación que el usuario haya seleccionado. Para entenderlo mejor, veremos el código encargado de ello.

```
//Recorremos nuestro array de puntos para ver si ya tenemos esa coordenada y asi no volvemos a pintar el marcador
//Primero comparamos la coordenada inicial
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].start_location).equals(coorde[cont])){
        existeInicio=true;
        break;
    }
}
if (!existeInicio){
    makeMarker(pintarOrigenDestino(route.legs[i].start_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].start_location);
}
else if (existeInicio){
    existeInicio=false;
}
//Ahora comparamos la coordenada final
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].end_location).equals(coorde[cont])){
        existeFin=true;
        break;
    }
}
if (!existeFin){
    makeMarker(pintarOrigenDestino(route.legs[i].end_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].end_location);
}
else if (existeFin){
    existeFin=false;
}
```

En el código anterior se compraban las coordenadas existentes y en caso de que no coincida con ninguna de las existentes se dibuja el marcador. Y en la función siguiente, se almacenan las coordenadas correctas en un array que se emplea para el cálculo de las rutas.

```
//Sobreescribimos listaPuntos para poder tener almacenados en ese arrray los puntos reales por los que pasa la ruta
listaPuntos=null;
listaPuntos=new Array();
for (var j=2;j<coorde.length;j++){
    listaPuntos.push({
        location:coorde[j],
        stopover:true
    })
}
```

4. RUTAS ALTERNATIVAS

En el momento de calcular la ruta solicitada por el usuario, además de la óptima que será la que se represente de forma más destacada, también se van a calcular una serie de rutas alternativas para otorgar varias posibilidades. Así, el usuario podrá ver cual prefiere, o seleccionarlas según sea la densidad de tráfico de cada una de ellas, posibles obras en el recorrido, etc.

Para obtener estas rutas alternativas se debe añadir una opción en la solicitud de la ruta al servidor de Google Maps, para que también las calcule. Aquí vemos cómo se añade la opción de 'provideRouteAlternatives'.

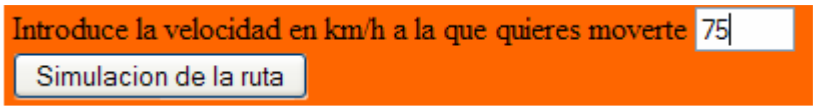
```
//Creamos la consulta que queremos calcular
var request = {
    origin: puntoInicial,
    destination: puntoFinal,
    waypoints: listaPuntos,
    optimizeWaypoints: true,
    provideRouteAlternatives: true,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
};
```

El resto del cálculo de la ruta no varía en nada respecto a calcularla sin las alternativas. Para mostrar sobre el mapa dichas rutas, se emplea una función independiente que será invocada cuando el usuario quiera ver estas rutas. En ella se especifica una línea de distinto color para hacer una diferenciación entre la ruta óptima y las alternativas.

```
////////////////////////////////////////////////////////////////////////////////
// Funcion que muestra en una nueva ventana la informacion de las rutas alternativas //
////////////////////////////////////////////////////////////////////////////////
function prueba(){
    var testwindow2 = window.open("","Rutas alternativas","width=1000,height=1000,status=no,scrollbars=yes");
    testwindow2.moveTo(0, 0);
    var rutas=respuesta.routes.length-1;
    testwindow2.document.write("<body style='background-color:#ff6600;'><h1>Existen "+rutas+" rutas altenativas </h1>");
    for (var f = 1; f<respuesta.routes.length; f++){
        var route = respuesta.routes[f];
        testwindow2.document.write("<h3><u>Ruta alternativa "+f+" </u></h3>");
        var directionsDisplay2=new google.maps.DirectionsRenderer;
        directionsDisplay2.setOptions({
            suppressMarkers: true,
            polylineOptions: {
                strokeColor: '#ED1C24',
                strokeWeight: 3,
                strokeOpacity:0.6
            }
        });
        testwindow2.document.write("<b>Desde:</b>" +route.legs[0].start_address + "<br />");
        testwindow2.document.write("<b>Hasta:</b> " +route.legs[0].end_address + "<br />");
        testwindow2.document.write("<b>Distancia:</b> " +route.legs[0].distance.text + "<br />");
        testwindow2.document.write("<b>Duracion:</b> " +route.legs[0].duration.text + "<br />");
        testwindow2.document.write("<b>Instrucciones:</b> <br />");
        for(var n=0;n<route.legs[0].steps.length;n++){
            testwindow2.document.write(route.legs[0].steps[n].instructions+ "<br />");
        }
        directionsDisplay2.setMap(map);
        directionsDisplay2.setDirections(respuesta);
        directionsDisplay2.setRouteIndex(f);
    }
}
```

5. SIMULACIÓN DE LA RUTA

Cuando el usuario decide ver una representación de la ruta que ha solicitado, nuestro sistema le va a devolver un icono en el mapa, que se irá moviendo por la ruta. A medida que se mueve, mostrará indicaciones a seguir. Esta simulación la hacemos en tiempo real, es decir, que si la duración del trayecto son 6 minutos, la duración de la simulación será de seis minutos también.



El usuario puede introducir una velocidad a la que creo que, aproximadamente, irá por la ruta o bien no introducir ninguna y dejar que se calcule con la velocidad por defecto.

Google Maps determina por defecto una velocidad de:

- 26 km/h en las ciudades
- 93 km/h en trayectos que unen ciudades.

El sistema ejecutor, al recibir la orden de simular, lo primero que debe hacer es comprobar si el usuario ha introducido o no una velocidad. Con esta velocidad, calculará cada cuánto tiempo debe cambiar la posición del icono gráfico del mapa para representar su movimiento.

En caso de no haber introducido una ruta, simplemente calcula el tiempo de movimiento del gráfico dividiendo la distancia entre el número de puntos intermedios de la ruta.

Si ha introducido velocidad el proceso es más complejo. Solicitamos al usuario que introduzca la velocidad en km/h. En primer lugar hace una regla de tres de modo que:

Si por ejemplo la velocidad introducida son 70 km/h:

70_____60 minutos
Distancia_ruta_____ x minutos

De aquí sabemos cuál va a ser el nuevo tiempo para realizar la ruta a la velocidad indicada. Una vez que tenemos la duración total de la ruta a la nueva velocidad, podemos calcular el tiempo de simulación del mismo modo que cuando no introducíamos velocidad.

Una vez que sabemos el tiempo tenemos que centrarnos en los puntos en los que se va a ir dibujando el gráfico. Google Maps tiene almacenados puntos en todas las rutas. Puntos internos que, o bien no tienen información o bien son puntos en los que indican una serie de instrucciones para que el conductor no tenga problema en seguir el camino.

Lo que hemos hecho ha sido crear un único array uniendo estos dos tipos de puntos, de modo que tenemos muchos puntos intermedios de una ruta y además, de algunos de ellos, tenemos también indicaciones para facilitar la conducción al usuario.

Por lo tanto, a medida que nuestro icono vaya cambiando de posición según los puntos, también irá creando ventanas de información en las que muestre las instrucciones a realizar en ese punto.

Por último, para conseguir que el coche se vaya parando el tiempo calculado anteriormente en cada ruta y después cambie de posición, hemos utilizado un "*setTimeout*" para conseguir que para un tiempo determinado.

6. CODIFICACIÓN GEOGRÁFICA

La codificación geográfica es el proceso de transformar direcciones (como "Carlos III 3, Pamplona, España") en coordenadas geográficas, del tipo Latitud/Longitud (como 37.423021, -122.083739), que se pueden utilizar para colocar marcadores o situar el mapa.

El API de Google Maps proporciona una clase “geocoder” que permite codificar de forma geográfica las direcciones de forma dinámica a partir de los datos introducidos por el usuario.

Esto es útil cuando el usuario introduce una dirección y deseamos obtener más información de ese punto. En nuestro proyecto el usuario introduce dos direcciones, la de origen y la de destino. Al realizar la consulta a los servidores diferenciamos entre punto de origen, puntos de paso y punto final. Los puntos de origen y final pueden ser cadenas de texto, por lo que no necesitamos la codificación.

Sin embargo, cuando los servidores devuelven una Latitud/Longitud y queremos ver qué dirección es sí necesitamos realizar la codificación geográfica inversa. Cuando realizamos la simulación y queremos mostrar por pantalla en qué posición se encuentra el icono, necesitamos realizar la codificación geográfica inversa.

Para poder realizarlo, existe una clase en el API llamada “Geocoder” y que tiene el siguiente método:

En el campo de “GeocoderRequest” ponemos la dirección del tipo Latitud/Longitud y a continuación obtenemos la conversión.

7. INDICACIONES DE LA RUTA

Para mostrar al usuario todas las indicaciones que se disponen de una ruta lo primero que se debe hacer es configurar el idioma para que éstas se escriban en el idioma que el usuario va a utilizar, en nuestro caso el castellano. Para ello se debe introducir en el script en el que se invoca al API de Google Maps, al comienzo de nuestro código, dicha orden.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no"/>
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<title>Sistema de Navegación Personalizado</title>
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?sensor=false&language=es"></script>
```

Una vez configurado el idioma, vamos a ver cómo se van a mostrar las indicaciones en la interfaz de la aplicación. Para no saturarla de información, se facilitará un enlace que despliegue dichas indicaciones en el momento en el que el usuario lo crea conveniente.

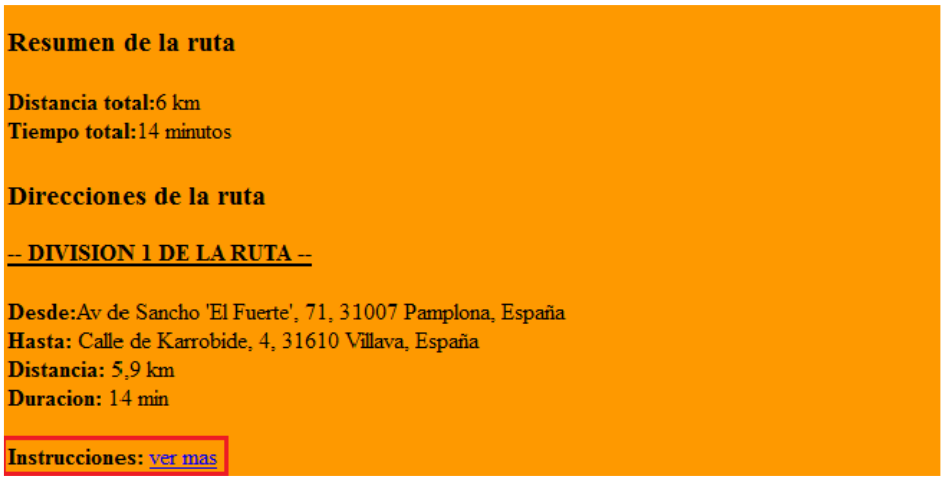
```
////////////////////////////////////////////////////////////////////////////
//Funcion que crea "ver mas" en las instrucciones completas de la ruta//
////////////////////////////////////////////////////////////////////////////
function vermas(o){
    if(o==1) panelIns.innerHTML="<b>Instrucciones: </b>"+mas;
    if(o==0) panelIns.innerHTML=texto+" "+men;
}
```

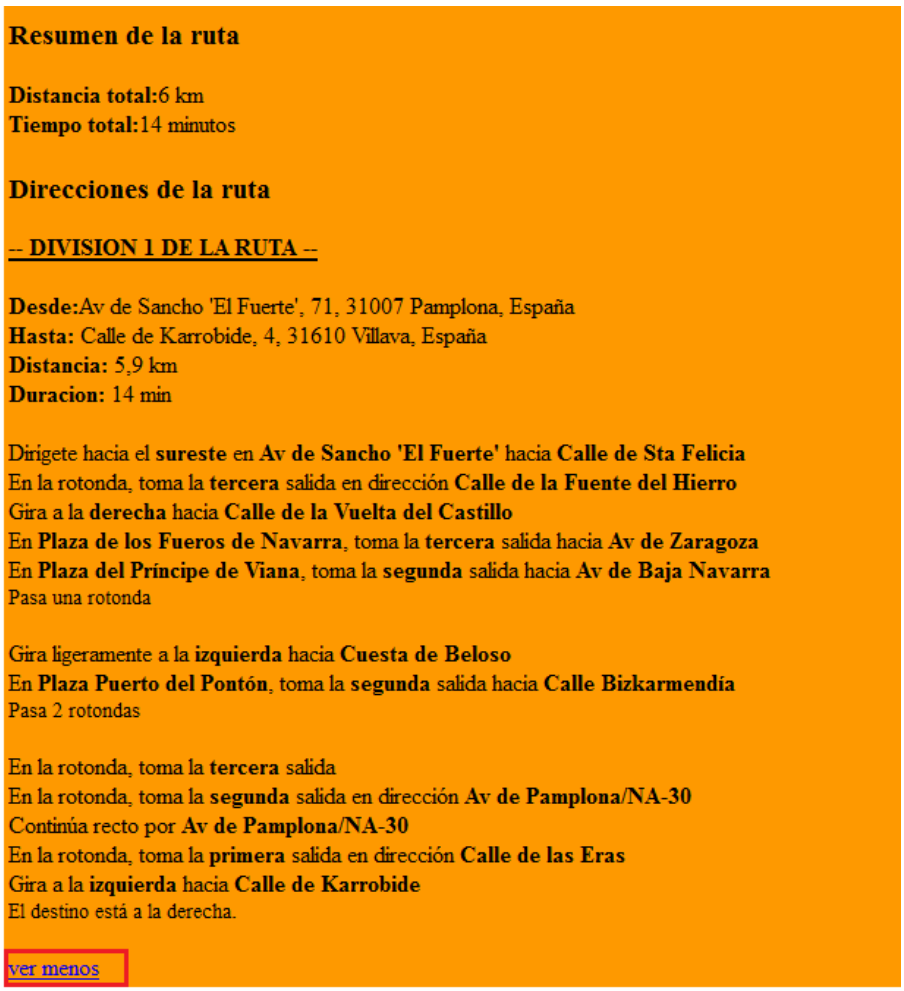
Con esta opción, el usuario podrá mostrar u ocultar la información en la ventana principal según le convenga.

```
for(var n=0;n<route.legs[i].steps.length;n++){
    texto += route.legs[i].steps[n].instructions + "<br />";
        simulaPuntos.push({
            location:route.legs[i].steps[n].instructions,
            stopover:true
        })
        var aux=pintarOrigenDestino(route.legs[i].steps[n].start_location);
        simulaCoorde.push({
            location:aux,
            stopover:true
        })
}
//Si es la primera vez que entramos a calcRoute() nos pinta el punto inicial y el final
if(primeravez){
    //Reducimos los decimales de las coordenadas que devuelve la ruta
    coorde[0]=pintarOrigenDestino(route.legs[i].start_location);
    coorde[1]=pintarOrigenDestino(route.legs[route.legs.length-1].end_location);
    //Alamaceno lo anterior por si se modifica el inicio o el fin y asi poder borrar sus marcadores
    anteriorinicio=coorde[0];
    anteriorfin=coorde[1];
    //Creamos los marcadores
    crearLimites(coorde[0],icons.inicio);
    crearLimites(coorde[1],icons.fin);
    primeravez=false;
    mas="<a href='javascript:vermas(0)'>ver mas</a>";
    men="<a href='javascript:vermas(1)'>ver menos</a>";
    panelIns.innerHTML = "<b>Instrucciones:</b> "+ mas +"<br /> ";
}
```

En el momento en el que se calcula por primera vez la ruta es donde se colocan los enlaces en la interfaz, ya que es el momento en el cual se ha generado la información para mostrar.

En esa misma función de calcular la ruta es donde se van a guardar todas las instrucciones obtenidas recorriendo todas las divisiones de la ruta.





8. IMPRIMIR DOCUMENTO CON EL RECORRIDO

En el momento en el que el usuario haya obtenido la ruta solicitada y tenga toda la información referente a ella, se le proporciona la posibilidad de que obtenga un documento con el mapa y los marcadores de la ruta, las divisiones de ésta y la información del recorrido.

Esta funcionalidad se crea para que los usuarios no estén limitados a tener la información solamente cuando hagan uso de la aplicación, sino que puedan disponer de ella cuando realicen el recorrido que han marcado en ella.

Para crear este documento existe una función que sobre una nueva ventana muestra toda la información y da la posibilidad de imprimir dicho documento.

```
////////////////////////////////////////
//Funcion que permite imprimir la ruta//
////////////////////////////////////////
function imprimir(){
    var testwindow = window.open("","Resultado final","width=1000,height=1000,status=no,scrollbars=yes");
    testwindow.moveTo(0, 0);
    //Creo un div en la ventana nueva para poder incrustar el mapa
    testwindow.document.write('</br> <div id="map2" style="width: 800px; height: 400px"></div> </br>');

    var DocumentContainer = document.getElementById('map').cloneNode(true);
    testwindow.document.getElementById("map2").appendChild(DocumentContainer);

    var childNodeArray = document.getElementById('map').childNodes;
    var i=0;
    while (i<childNodeArray.length){
        testwindow.document.getElementById("map2").appendChild(childNodeArray[i]);
        i++;
    }
    testwindow.document.write('</br>');
    var texto=document.getElementById("resumen");
    testwindow.document.write(texto.innerHTML+'</br></br>');
    texto=document.getElementById("direcciones");
    testwindow.document.write(texto.innerHTML);
    texto=document.getElementById("instrucciones");
    testwindow.document.write(texto.innerHTML);
    testwindow.document.write('</br> <form><input type="button" name="imprimir" value="Imprimir" onclick="window.print();"> </form></br>');
    testwindow.document.close();
    testwindow.focus();
    testwindow.print();
    primeravez=true;
    reload();
}
```

8- CONCLUSIONES

A través de este proyecto he conseguido descubrir cómo funcionan los sistemas de navegación existentes. Además, he podido estudiar cómo acceder a los sistemas de información geográficos y cómo obtener la información deseada para poder ser mostrada a los usuarios.

Tras la realización del proyecto he llegado a la conclusión de que un sistema de navegación está dividido en tres partes. Por un lado está la parte gráfica, por otro la recepción y el envío de la información y por último los sistemas de información geográficos.

La parte gráfica se encarga de recoger los datos que el usuario introduce gracias a la interfaz. Estos datos van a ser obligatoriamente el punto de inicio y el final. Además de poder añadir etapas en la ruta.

Cuando la parte gráfica envía estos datos al sistema receptor, éste crea la consulta y la realiza al sistema de información, quien calcula runa ruta y la devuelve. En función de qué quiere visualizar el resultado, el sistema receptor determina qué información pasarle a la parte gráfica y cuál no.

Lo que hemos conseguido con este proyecto ha sido diferenciarlo de un sistema de navegación normal. Esto se debe a que hemos ofrecido al usuario funcionalidades inexistentes en otros sistemas de navegación. Estas funcionalidades son por ejemplo ver ruta alternativas, ver una simulación de por dónde pasa la ruta con las indicaciones a realizar en cada punto e incluso la opción de introducir una velocidad deseada y realizar la simulación conforme a esa velocidad.

En resumen, es un sistema de navegación más pero que permite al usuario nuevas funcionalidades hasta ahora inexistentes.

9- LINEAS FUTURAS

Tras finalizar el desarrollo del proyecto, podemos ver también hacia dónde se puede enfocar este trabajo. La investigación nos ha permitido conocer tanto el funcionamiento de los sistemas de localización, cómo ver qué más se les puede añadir atendiendo a las necesidades que se tengan.

Con ellos nos hemos dado cuenta de que podemos crear uno de estos sistemas y adaptarlo totalmente al tipo de usuario al que vaya destinado. Será cuestión de realizar un análisis de lo que el usuario desea y cuáles son los fines, ya que éstos pueden ser desde didácticos hasta lo que actualmente se conoce en el mercado de GPS.

Una nueva idea a partir de lo que nosotros hemos desarrollado, podría ser crear una aplicación similar para turistas que estén visitando una ciudad y deseen realizar un recorrido por la misma. De este modo, podrían realizar la visita de la forma óptima, minimizando sus recorridos y teniendo la posibilidad de ser ellos los que introduzcan los puntos de paso deseados.

Esta es una idea, pero su uso podría extenderse para aplicaciones de diversos tipos.

10- BIBLIOGAFÍA

[1] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/services.html

[2] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/reference.html

[3] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/examples/index.html

[4] http://www.fundacionjosepons.com/estudios/multimedia/1117035859Javascript.pdf

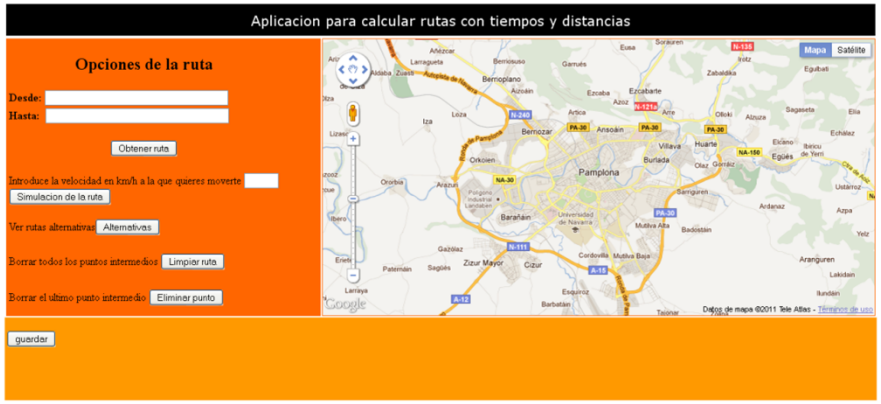
[5] http://www.desarrolloweb.com/javascript/

‘Captura y visualización de datos en un navegador GPS’

Andrés Goñi Gallego

5 de octubre de 2011

Sistema general

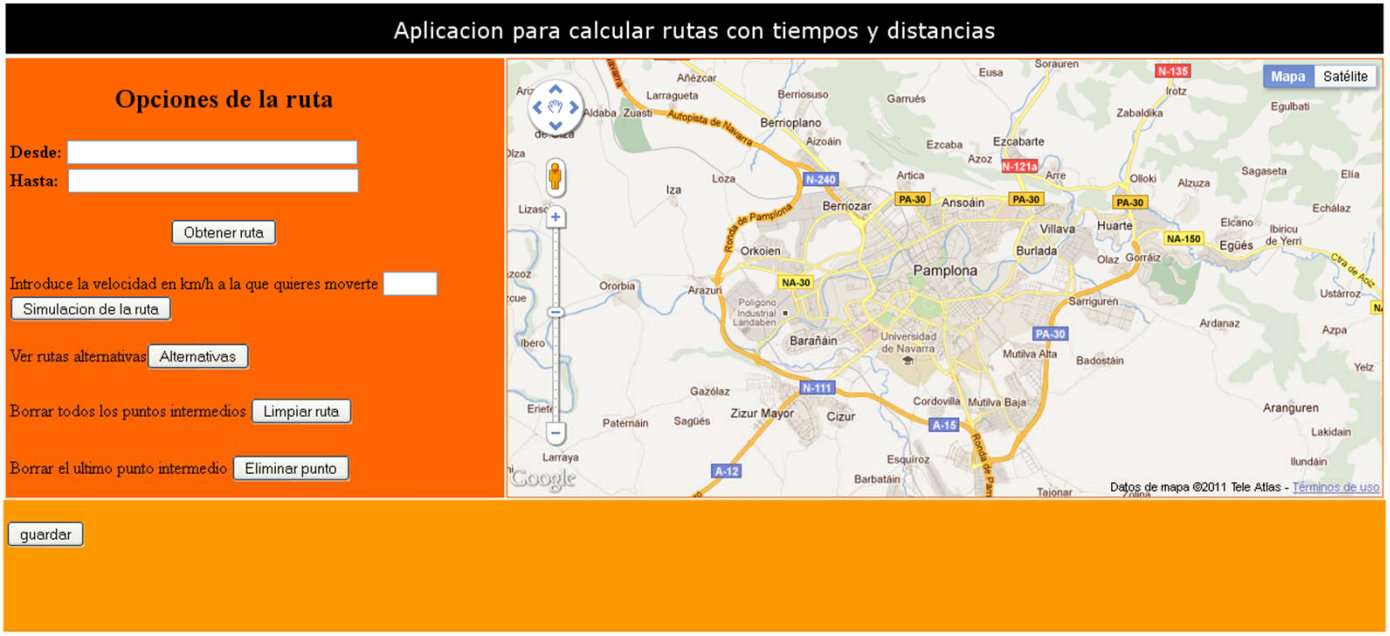


Servidores de Google Maps

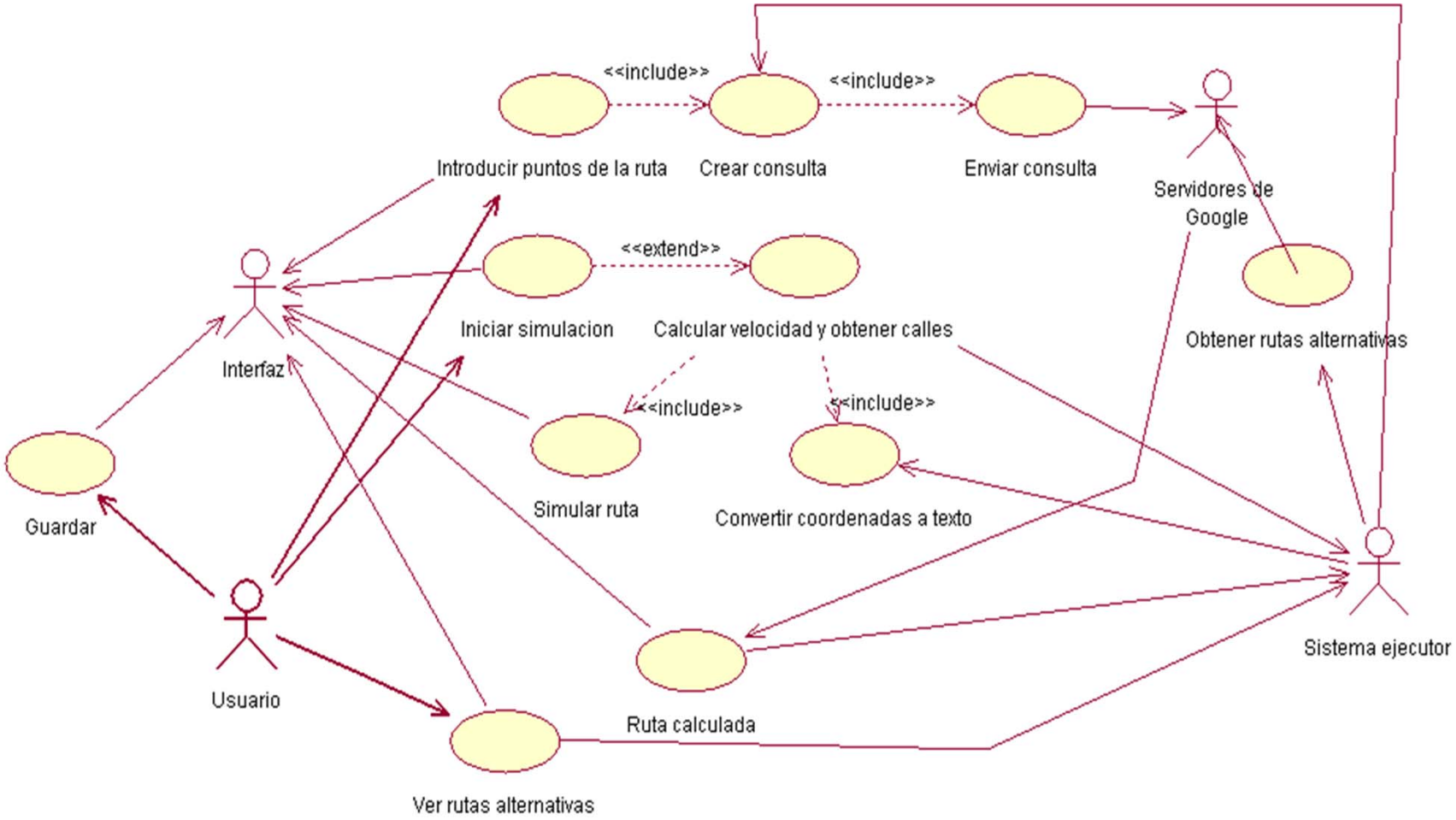
Sistema ejecutor

Interfaz del usuario

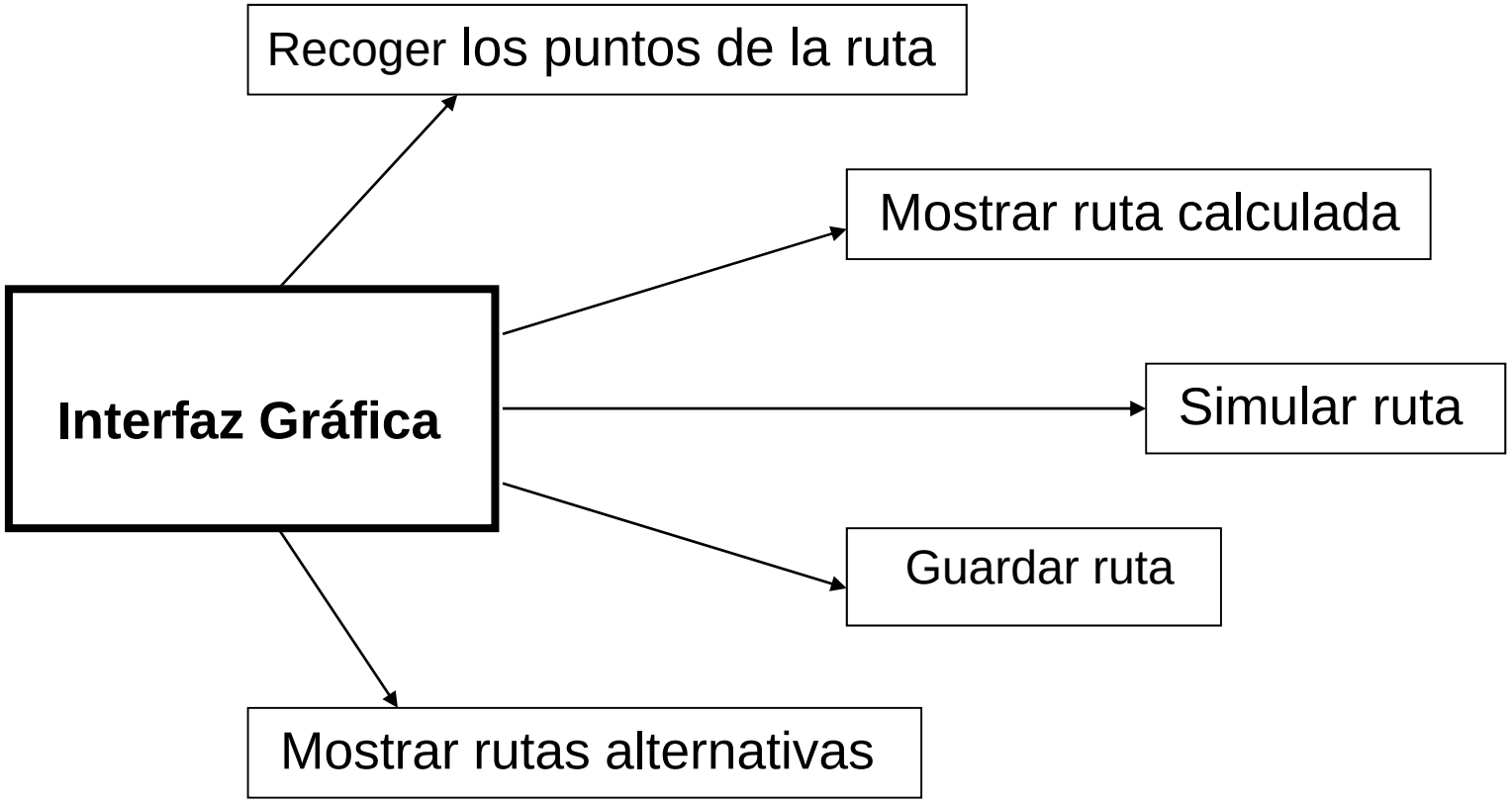
- Aspecto inicial de la interfaz de la aplicación



Casos de uso



Bloques de trabajo



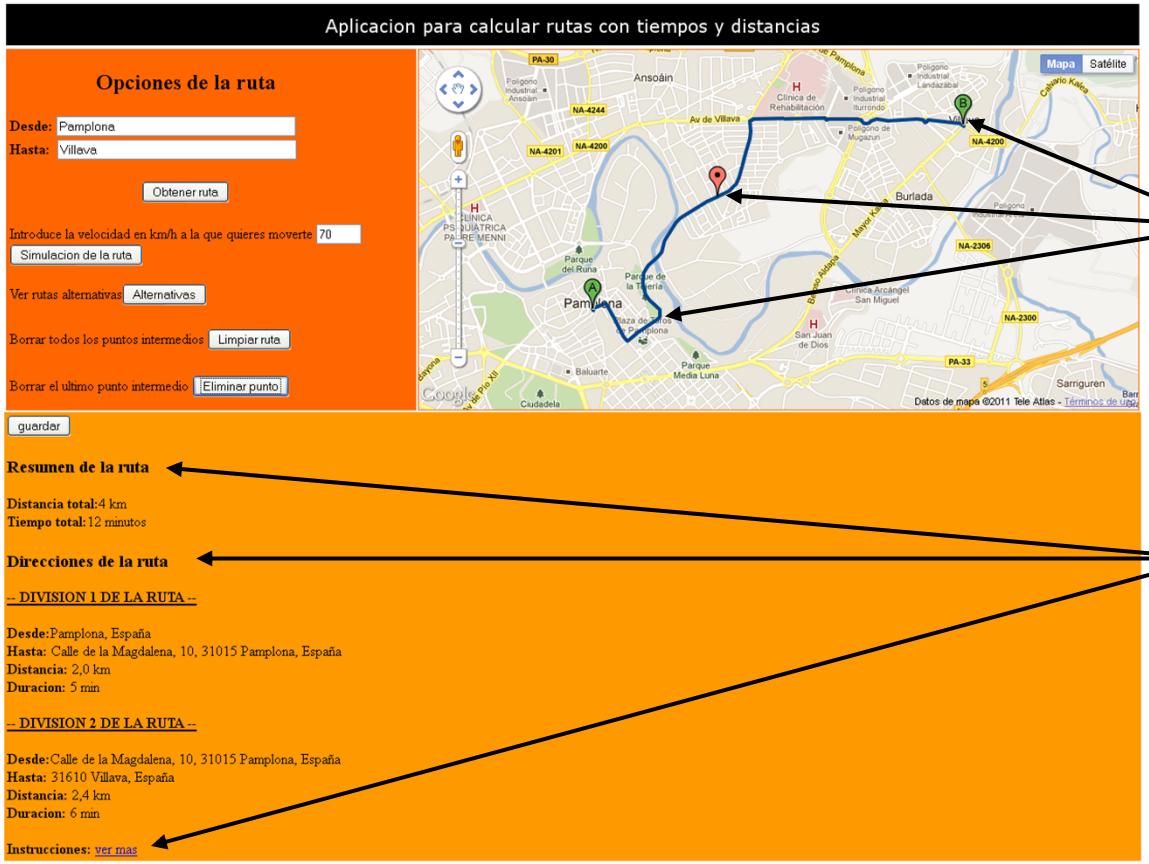
Entrada de datos

- Captura de datos

¿Qué datos nos interesan?

¿Cómo interactuamos con ellos?

Muestra de datos

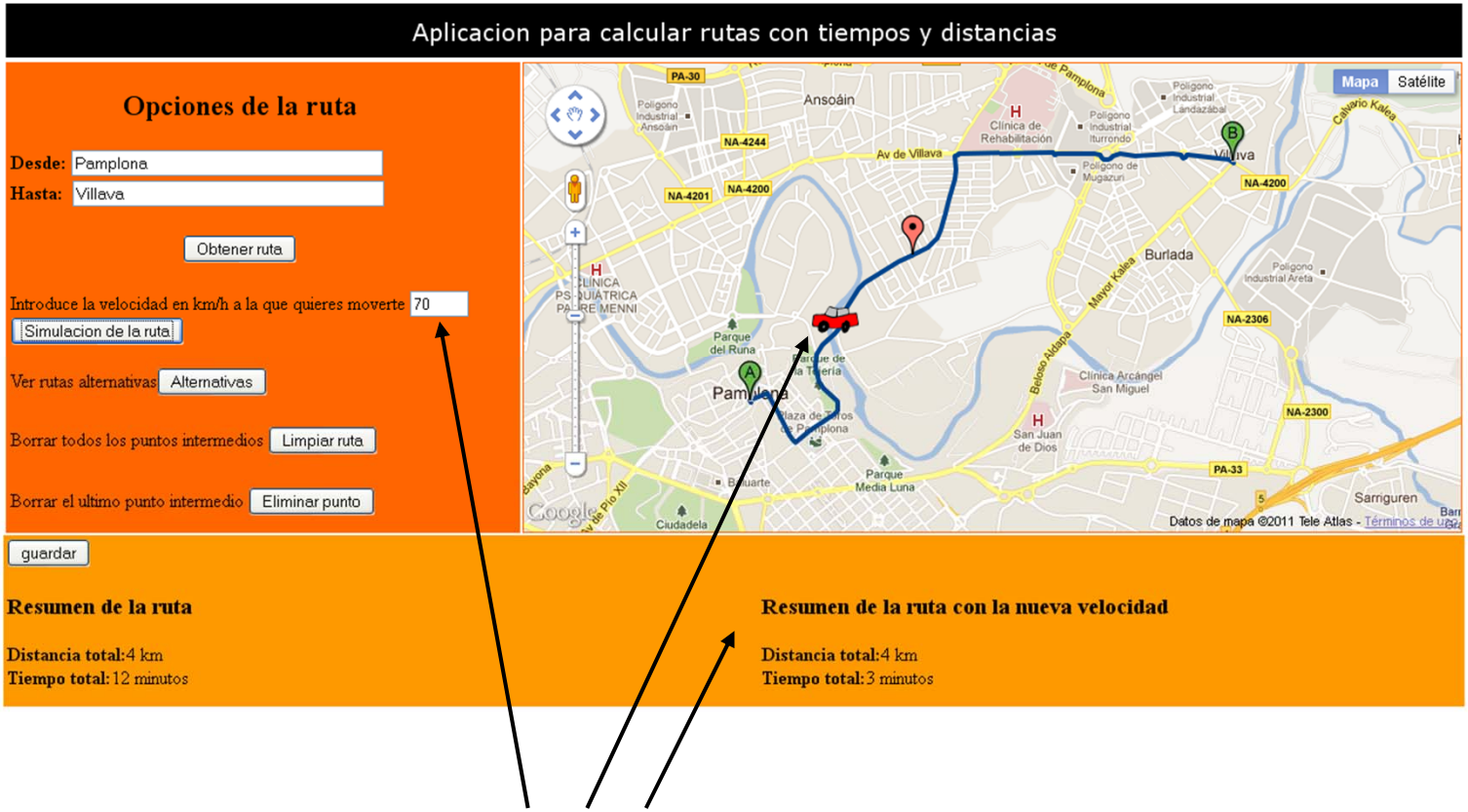


Datos sobre el mapa

Datos sobre la ruta

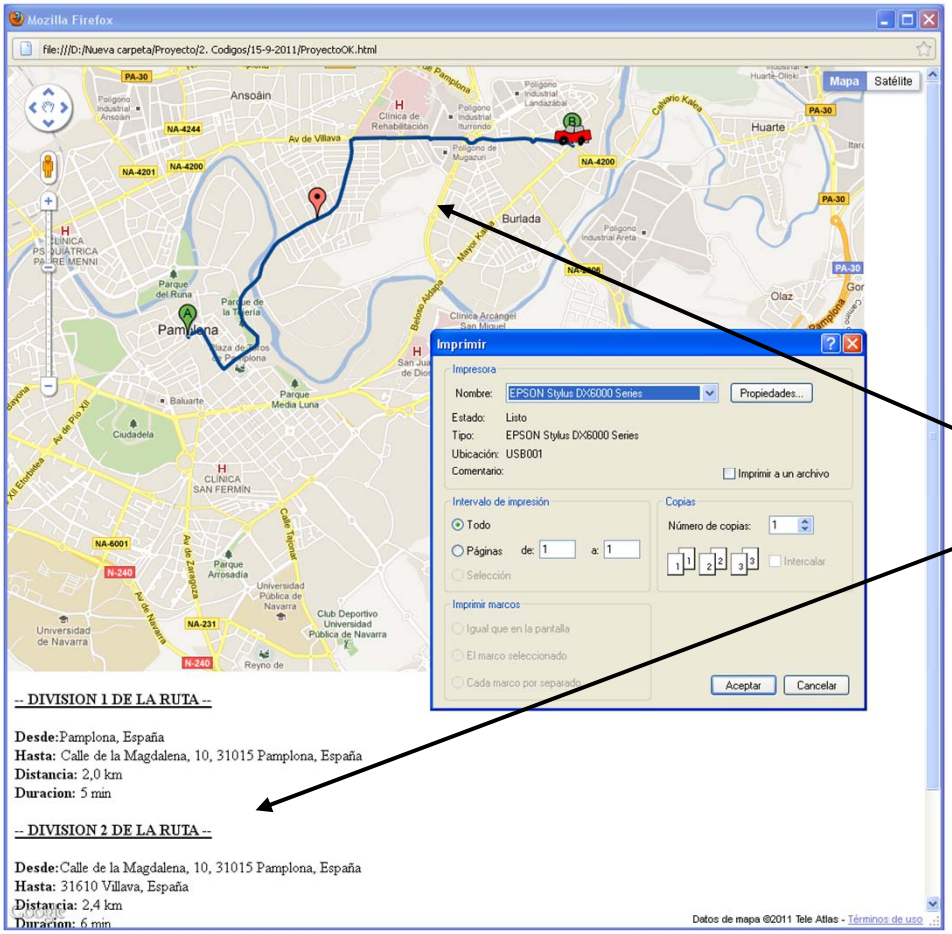
¿Se puede modificar la ruta ya creada?

Simulación de las rutas



Elementos de la simulación

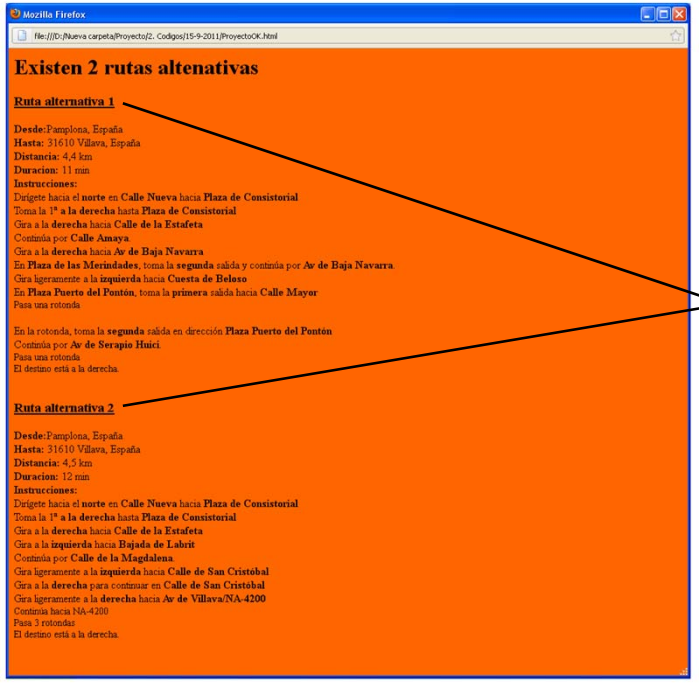
Guardar la información obtenida



¿Qué se almacena?

¿Cómo se almacena?

Rutas alternativas



¿Cómo y para qué las obtenemos?

Conclusiones

- Conocer la división en: interfaz, sistema ejecutor y motor de datos (Google Maps)
- Lo más importante es conocer qué se quiere desarrollar y para quién.
- Grandes posibilidades de adaptabilidad.
- Diferentes posibilidades de interacción.

Líneas futuras

- Podemos crear un sistema de localización para diferentes aplicaciones:
  - GPS
  - Didácticos
  - Turismo en ciudades
- La aplicación a crear marcará el desarrollo del sistema como herramienta a utilizar.