UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA GEODÉSICA CARTOGRÁFICA Y TOPOGRÁFICA

# Trabajo
# Final de máster


## Desarrollo de una app móvil para metabúsqueda de vuelos

**Máster en Ingeniería Geomática y Geoinformación**

**Alumno:** **Javier Carrascosa Basterra**

**Fecha:** **Septiembre 2020**

# Índice

# Índice de figuras

# Introducción

El fin de proyecto se basa en buscar el precio más barato para un trayecto en avión. Normalmente, este trayecto se busca una sola vez en un metabuscador, cometiendo el común error de pensar que en ese metabuscador se encuentran todas las posibles combinaciones, aerolíneas y precios. Pero la realidad es que, aunque algunos metabuscadores pueden compartir una parte de su espectro, por lo general los resultados difieren.

# Objetivo

Con el fin de obtener el precio más barato posible se desarrolló la aplicación CheapTravel, la cual busca vuelos en diferentes metabuscadores para ofrecer al usuario el precio que realmente sí es el más barato. CheapTravel es una aplicación desarrollada con HTML + CSS y Javascript, y posteriormente compilada con Cordova.

# Información disponible

Si el objetivo es buscar vuelos de manera automatizada, existen al menos dos maneras: Web Scapping, donde una vez el usuario introduce los datos de vuelo, se autorrellenan formularios en diferentes páginas web para extraer esos datos, o la segunda opción: realizar peticiones a una o varias API.

# Metabuscadores

Si buscamos un vuelo Paris, Charles de Gaulle, Francia – Nueva York, John F Kennedy, Estados Unidos para el sábado, 5 de septiembre en diferentes metabuscadores obtenemos los siguientes resultados.

- Skyscanner

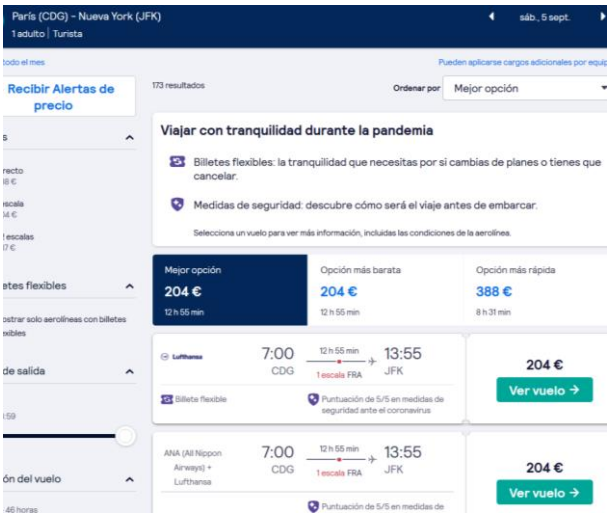



*Figura 1. Búsqueda de un vuelo en Skyscanner*

- Tripadvisor

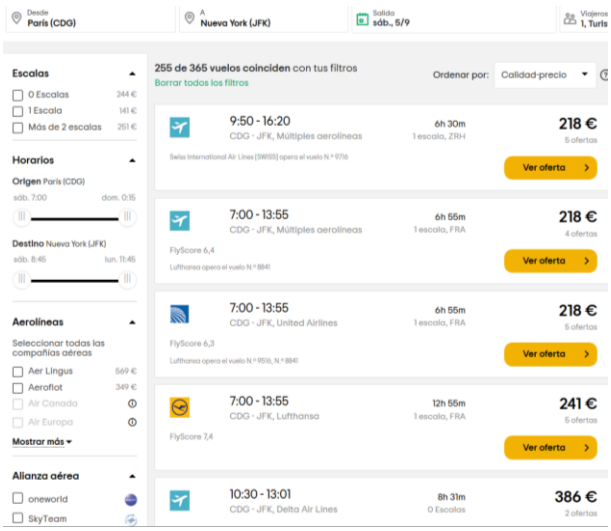



*Figura 2. Búsqueda de un vuelo en Tripadvisor*

- Cheapflights

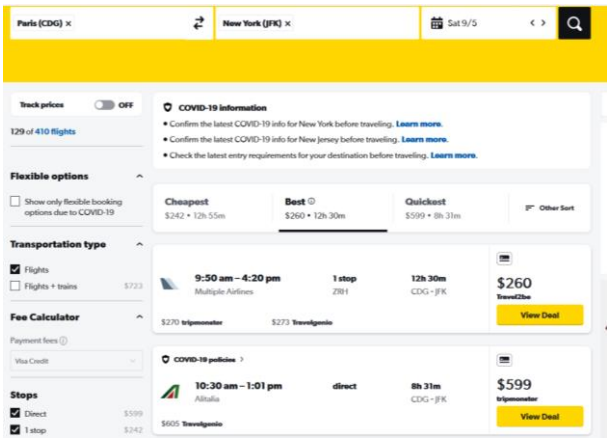



*Figura 3. Búsqueda de un vuelo en CheapFlights*

- Expedia

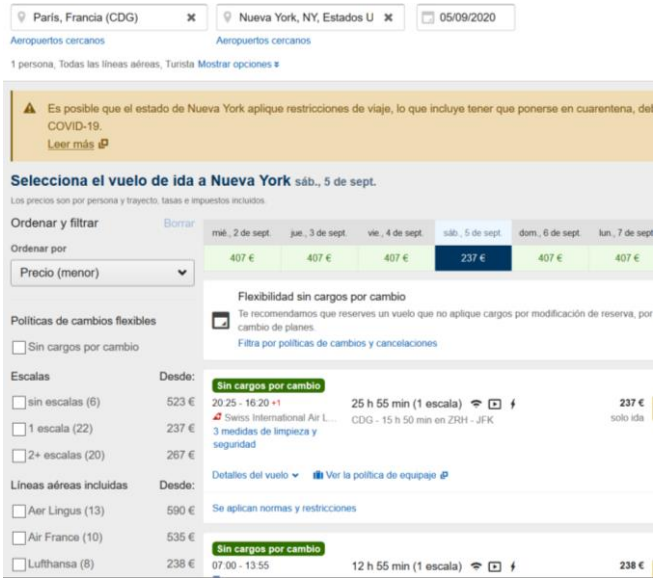



*Figura 4. Búsqueda de un vuelo en Expedia*

- Edreams

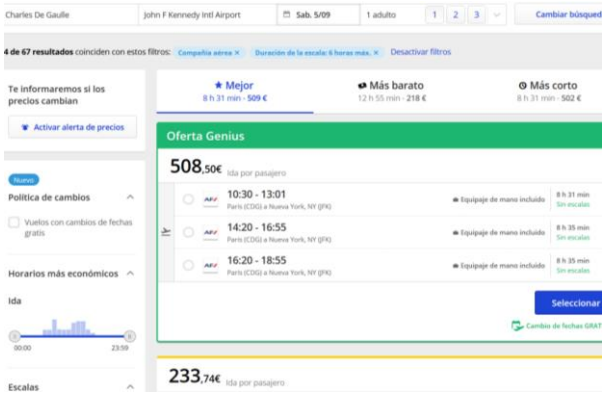



*Figura 5. Búsqueda de un vuelo en Edreams*

O en un vuelo Madrid, Adolfo Suárez Madrid–Barajas, España – Dubai, Aeropuerto Internacional, EAU para el domingo 6 de septiembre.

- Skyscanner

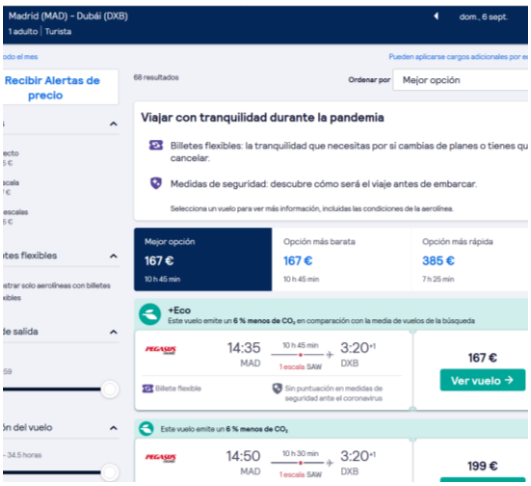



*Figura 6. Búsqueda de un vuelo en Skyscanner.*

- Tripadvisor

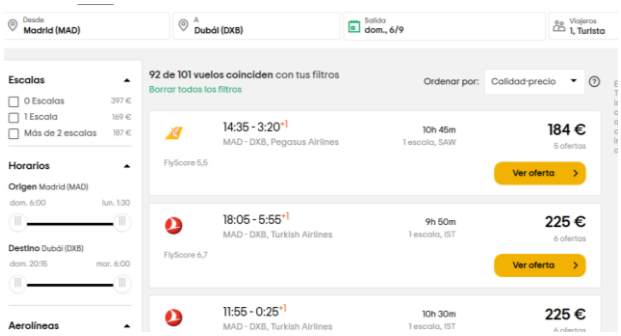



*Figura 7. Búsqueda de un vuelo en TripAdvisor*

- Cheapflights


*Figura 8. Búsqueda de un vuelo en CheapFlights*

- Edreams

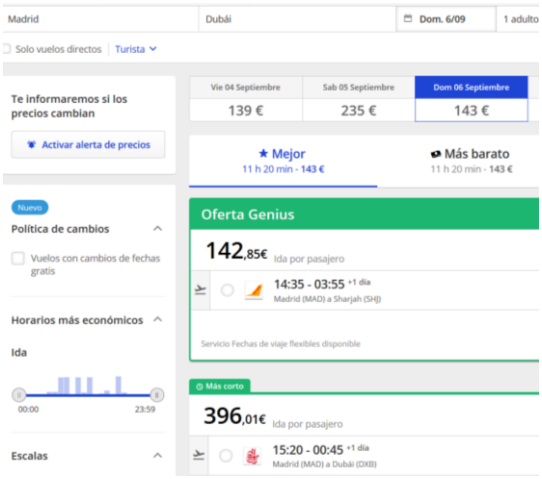



*Figura 9. Búsqueda de un vuelo en Edreams.*

- Expedia

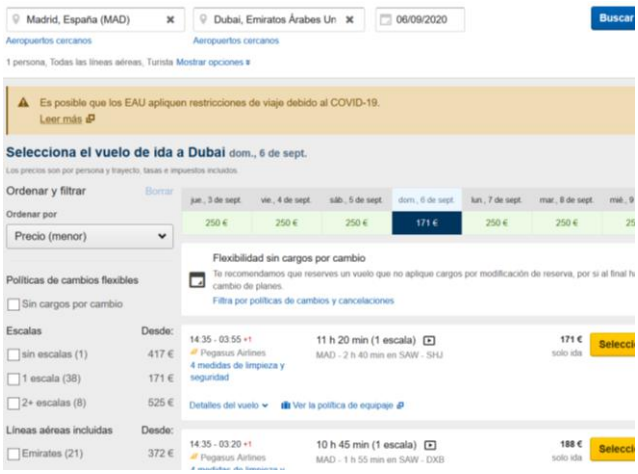



*Figura 10. Búsqueda de un vuelo en Expedia.*

Dada la variedad de precios, es necesario realizar una aplicación móvil que con una única búsqueda compare precios en diferentes metabuscadores para pagar el precio mínimo.

## ¿Qué es una API?

*"Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.*

*Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales)."*

De una forma más sencilla, una API es una aplicación a la que se le envía una petición HTTP con parámetros, y cuando la procesa, te devuelve un objeto JSON con la información que el usuario ha solicitado. Existen varios tipos de peticiones.

*"HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser safe, idempotent, o cacheable.*

- *GET. El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.*
- *HEAD. El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.*
- *POST. El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.*
- *PUT. El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.*
- *DELETE. El método DELETE borra un recurso en específico.*
- *CONNECT. El método CONNECT establece un túnel hacia el servidor identificado por el recurso.*
- *OPTIONS. El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.*
- *TRACE. El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.*
- *PATCH. El método PATCH es utilizado para aplicar modificaciones parciales a un recurso."*

En el caso de CheapTravel, se emplean peticiones de tipo GET y HEAD.

## Aeropuertos

Un aspecto destacado de la búsqueda automatizada de vuelos e información relevante respecto a ellos es que todo está en inglés. Es por ello que la aplicación también lo está.

También destaca que todas las peticiones a cualquier API de vuelos requieren un código para identificar al aeropuerto. Este código se llama IATA.

*"El código de aeropuertos de IATA es un código de tres letras que designa a cada aeropuerto en el mundo. Estos códigos son decididos por la Asociación Internacional de Transporte Aéreo (International Air Transport Association) IATA. Las letras mostradas claramente en las etiquetas de equipaje usadas en las mesas de embarque de los aeropuertos son una muestra del uso de estos códigos. BOG=Bogotá."*

En CheapTravel, el usuario no conoce estos códigos. Una posible solución es realizar una petición a una API con el nombre del aeropuerto y que te devuelva el código. El problema es que estas API son de pago. La alternativa elegida fue realizar un fichero que contenía todos los aeropuertos del mundo el formato texto.

Con un código de Python se procesó el fichero para convertirlo a JSON y posteriormente se almacenó como una variable global de Javascript. Esta variable contiene 7697 aeropuertos de todo el mundo en el siguiente formato:

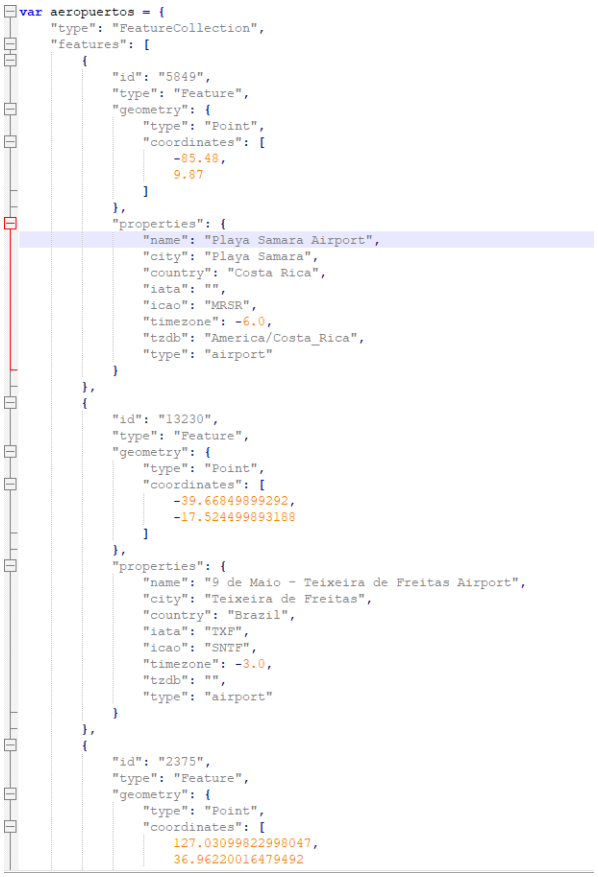



*Figura 11. Variable js que almacena una lista con todos los objetos JSON de los aeropuertos.*

Cada objeto de la lista se corresponde con un aeropuerto, y posee un identificador, sus coordenadas geográficas, la ciudad y el país donde se encuentra, su código IATA, su zona horaria y el tipo, que puede ser aeropuerto o aeródromo.

## RapidAPI

RapidAPI es un portal con decenas de enlaces a APIs de multitud de categorías.

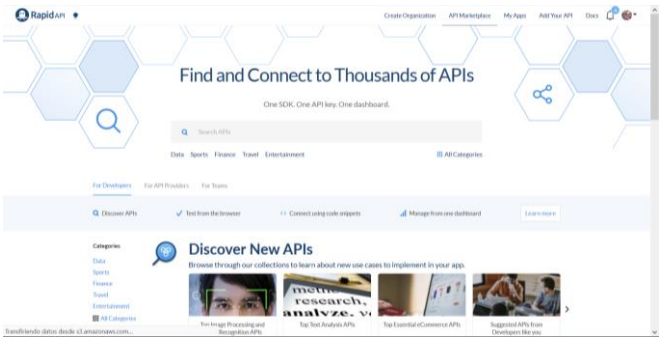



*Figura 12. Portal de RapidAPI.*

Pero la gran ventaja de este portal es que te proporciona un extracto de código, en el lenguaje que se desee, de la petición que se debe hacer a la API. En mi caso, realizo peticiones XMLHtppRequest en Javascript.

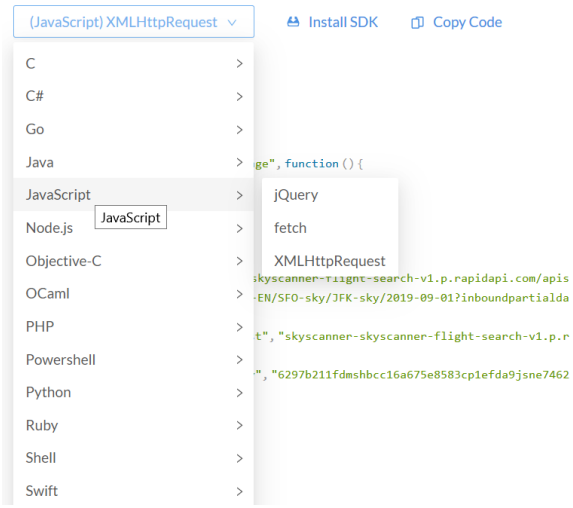



*Figura 13. Ejemplo de petición GET en el portal RapidAPI.*

Además, también te permite testear la API desde la propia página web, introduciendo los parámetros de búsqueda que se deseen, como el código IATA del origen y el destino del vuelo, la divisa o la fecha del viaje. Clickando "Test Endpoint", se envía la petición y devuelve la respuesta en el recuadro de abajo a la derecha.

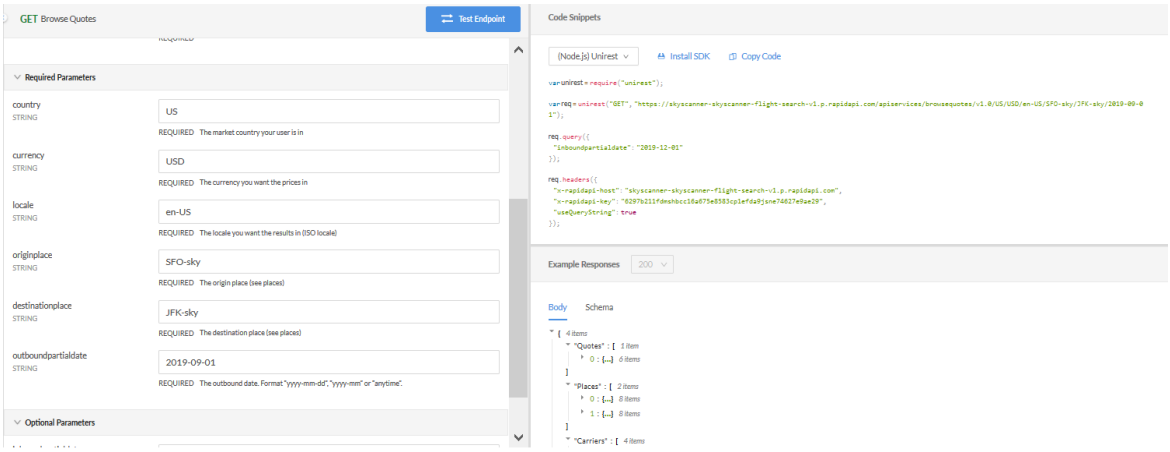



*Figura 14. Ejemplo de petición GET en el portal RapidAPI.*

Por ejemplo, un vuelo desde Madrid (MAD) a París, Charles de Gaulle (CDG), sin fecha (anytime).

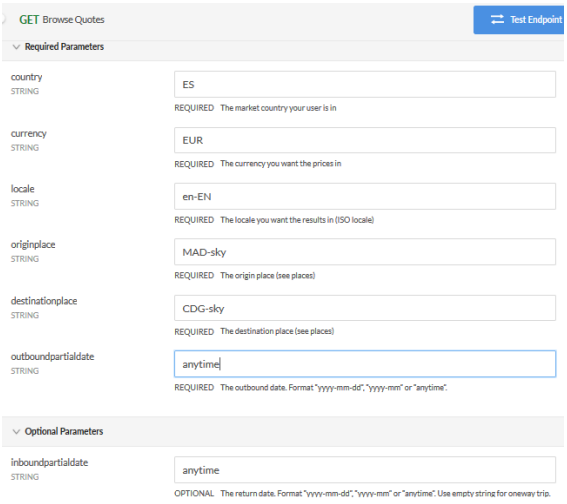



*Figura 15. Ejemplo de petición GET en RapidAPI.*

Se traduce en la siguiente petición:

(JavaScript) XMLHttpRequest  Install SDK  Copy Code

```
var data = null;

var xhr = new XMLHttpRequest();
xhr.withCredentials = true;

xhr.addEventListener("readystatechange", function () {
  if (this.readyState === this.DONE) {
    console.log(this.responseText);
  }
});

xhr.open("GET", "https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/b
rowsequotes/v1.0/ES/EUR/en-EN/MAD-sky/CDG-sky/anytime?inboundpartialdate=anytime");
xhr.setRequestHeader("x-rapidapi-host", "skyscanner-skyscanner-flight-search-v1.p.rapidapi.c
om");
xhr.setRequestHeader("x-rapidapi-key", "6297b211fdmshbcc16a675e8583cp1efda9jsne74627e9ae2
9");

xhr.send(data);
```

*Figura 16. Ejemplo de petición GET en RapidAPI.*

Donde es posible ver que los parámetros de la petición se introducen después de la URL de Skyscanner, y devuelve un objeto JSON como el siguiente:


*Figura 17. Respuesta JSON de RapidAPI.*

Todas las API de RapidApi requieren de uno o varios token, que se introducen en la cabecera de la petición. Normalmente estos tokens son gratuitos, aunque a veces es necesaria una suscripción.

## Skyscanner API

Skyscanner verifica precios con 1.200 agencias de viajes.

La API de Skyscanner es una API *freemium* que permite buscar vuelos y obtener precios de vuelos de la base de datos de precios de Skyscanner, así como obtener cotizaciones en vivo directamente de las agencias de venta de billetes. Esta API requiere una suscripción, aunque en este caso es gratuita y se pueden realizar peticiones ilimitadas.

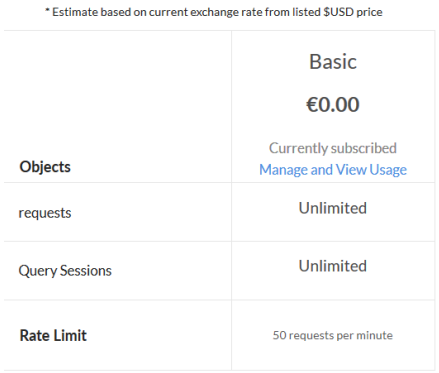



*Figura 18. Suscripción a la API de Skyscanner.*

Los endpoints de una API son todos los tipos de peticiones que se le pueden hacer a una API de RapidAPI. En el caso de Skyscanner vemos que tenemos 3 categorías de peticiones, todas de tipo GET: buscar lugares, vuelos o mercados.

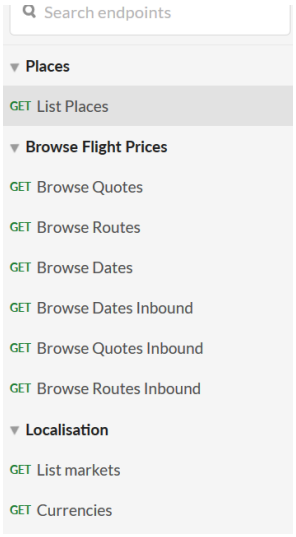



*Figura 19. Endpoints de la API de Skyscanner.*

## Tripadvisor API

*"Tripadvisor, la plataforma de viajes más grande del mundo, ayuda a 463 millones de viajeros cada mes a hacer de su viaje el mejor. Los viajeros de todo el mundo utilizan la web y la aplicación de Tripadvisor para consultar más de 859 millones de comentarios y opiniones de 8,6 millones de alojamientos, restaurantes, experiencias, vuelos y cruceros."*

La API freemium de TripAdvisor es un servicio web que consulta precios de vuelos en tiempo real, reservas de hoteles, restaurantes, atracciones y más, como se ve en su sitio web. En mi caso, estoy suscrito al plan gratuito.

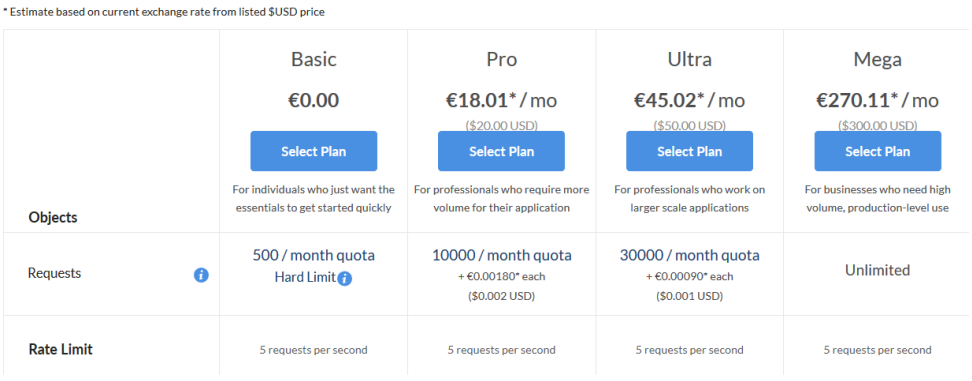



*Figura 20. Suscripción a la API de TripAdvisor.*

TripAdvisor tiene una categoría de peticiones compartidas para el resto de categorías: restaurantes, vuelos, hoteles y atracciones.

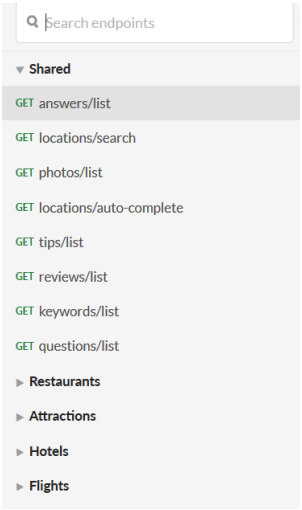



*Figura 21. Endpoints de la API de TripAdvisor.*

El proceso de las peticiones compartidas consiste en enviar una petición al servidor con, por ejemplo, el origen, destino y fecha del vuelo, y la API nos contesta con un código. Para conocer los posibles precios para esa ruta y fecha, es necesario realizar una nueva petición con el código devuelto previamente.

## FlightData API

Se trata de una API rusa donde los datos se transfieren desde el caché, que se forma sobre la base de búsquedas de usuarios de los sitios Aviasales.ru y Jetradar.com durante las últimas 48 horas. Los precios se dan en rublos a partir del momento en que el boleto se coloca en los resultados de búsqueda.

Flight Data tiene diversos endpoints, aunque en mi caso solo empleo uno, el cual sirve para buscar vuelos introduciendo solo el origen, pudiendo servir así como fuente de inspiración al usuario.

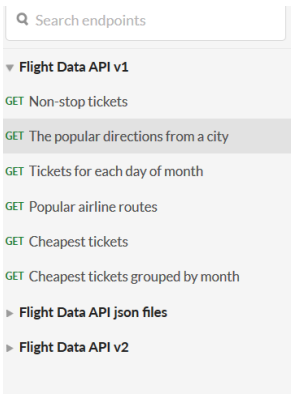



*Figura 22. Endpoints de la API de Flight Data.*

En este caso, solo se introduciría el origen y la moneda que se desea, y se obtendría una respuesta JSON con los posibles destinos junto con su precio.

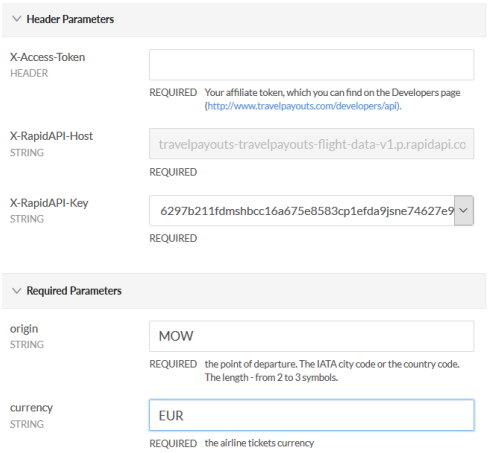



*Figura 23. Ejemplo de petición GET en RapidAPI.*

# Funcionamiento de la aplicación

## Formulario

### Origen y destino

Lo primero que necesita prácticamente toda aplicación es un formulario. En mi caso, necesito que el usuario especifique un origen y un destino. Uno de los problemas es que muchas veces el usuario desconoce el nombre del aeropuerto, y si únicamente introduce el nombre de la ciudad puede haber errores, por ejemplo, con Valencia, España y Valencia, Venezuela.

La solución adaptada ha sido una lista desplegable con el siguiente formato: 'Ciudad, aeropuerto, país', habiendo tomado estos datos de la lista de aeropuertos de la que dispongo.

En cuanto a HTML, esta lista desplegable se consigue con un <form> que tenga dos <input> dentro, uno para el origen y otro para el destino, que son myInput y myInput2.

```
<form autocomplete="off" action="/action_page.php">
    <div class="autocomplete" style="width:300px;">
        <input id="myInput" type="text" name="myCountry" placeholder="Origin" >
        <input id="myInput2" type="text" name="myCountry" placeholder="Destination" >
        <input size="12" placeholder="YYYY-MM-DD" type="text" id="fecha" name="fecha" data-toggle="tooltip" title="Date">
    </div>
</form>
```

*Figura 24. Extracto de código HTML de la aplicación.*

Una posibilidad era utilizar el elemento <datalist>, muy común en Javascript, pero resultó que Cordova no lo puede compilar.

Este elemento necesita de una función en Javascript que la autocomplete. Esta función es muy larga y compleja, por lo que mostraré sólo un fragmento, aunque el código completo se incluirá en el Anexo.

```
function autocomplete(inp, arr) {
  /*the autocomplete function takes two arguments,
  the text field element and an array of possible autocompleted values:*/
  var currentFocus;
  /*execute a function when someone writes in the text field:*/
  inp.addEventListener("input", function(e) {
      var a, b, i, val = this.value;
      /*close any already open lists of autocompleted values*/
      closeAllLists();
      if (!val) { return false;}
      currentFocus = -1;
      /*create a DIV element that will contain the items (values):*/
      a = document.createElement("DIV");
      a.setAttribute("id", this.id + "autocomplete-list");
      a.setAttribute("class", "autocomplete-items");
      /*append the DIV element as a child of the autocomplete container:*/
      this.parentNode.appendChild(a);
      /*for each item in the array...*/
      for (i = 0; i < arr.length; i++) {
        /*check if the item starts with the same letters as the text field value:*/
        if (arr[i].substr(0, val.length).toUpperCase() == val.toUpperCase()) {
          /*create a DIV element for each matching element:*/
          b = document.createElement("DIV");
          /*make the matching letters bold:*/
          b.innerHTML = "<strong>" + arr[i].substr(0, val.length) + "</strong>";
          b.innerHTML += arr[i].substr(val.length);
          /*insert a input field that will hold the current array item's value:*/
          b.innerHTML += "<input type='hidden' value='" + arr[i] + "'>";
          /*execute a function when someone clicks on the item value (DIV element):*/
          b.addEventListener("click", function(e) {
              /*insert the value for the autocomplete text field:*/
              inp.value = this.getElementsByTagName("input")[0].value;
              /*close the list of autocompleted values,
              (or any other open lists of autocompleted values:*/
              closeAllLists();
          });
          a.appendChild(b);
        }
      }
  });
  /*execute a function presses a key on the keyboard:*/
  inp.addEventListener("keydown", function(e) {
      var x = document.getElementById(this.id + "autocomplete-list");
      if (x) x = x.getElementsByTagName("div");
      if (e.keyCode == 40) {
        /*If the arrow DOWN key is pressed,
        increase the currentFocus variable:*/
        currentFocus++;
        /*and and make the current item more visible:*/
        addActive(x);
      } else if (e.keyCode == 38) { //up
        /*If the arrow UP key is pressed,
        decrease the currentFocus variable:*/
```

*Figura 25. Función JS de autocompletado del formulario.*

Además de Javascript y HTML, son necesarios los estilos CSS para obtener el comportamiento esperado.

```
* {
  box-sizing: border-box;
}

body {
  font: 16px Arial;
}

/*the container must be positioned relative:*/
.autocomplete {
  position: relative;
  display: inline-block;
  z-index: 90000;
}


.autocomplete-items {
  position: absolute;
  border: 1px solid #d4d4d4;
  border-bottom: none;
  border-top: none;
  z-index: 99;
  /*position the autocomplete items to be the same width as the container:*/
  top: 100%;
  left: 0;
  right: 0;
}

.autocomplete-items div {
  padding: 10px;
  cursor: pointer;
  background-color: #fff;
  border-bottom: 1px solid #d4d4d4;
}

/*when hovering an item:*/
.autocomplete-items div:hover {
  background-color: #e9e9e9;
}

/*when navigating through the items using the arrow keys:*/
.autocomplete-active {
  background-color: DodgerBlue !important;
  color: #ffffff;
}
```

*Figura 26. Estilo CSS de la función de autocompletado del formulario.*

De esta manera, a medida que el usuario va escribiendo, la lista muestra los resultados coincidentes.

Además, esta lista desplegable tiene que estar siempre disponible, porque su llamada se realiza en una función asíncrona que siempre está activa, la cual se llama onDeviceReady.

```
    lista2 = [];
    for (var i=0; i<aeropuertos.features.length; i++) {

        var nombre = aeropuertos.features[i].properties.name;
        var ciudad = aeropuertos.features[i].properties.city;
        var pais   = aeropuertos.features[i].properties.country;

        lista2.push(ciudad+','+nombre+','+pais);

    }

    autocomplete(document.getElementById("myInput"), lista2);
    autocomplete(document.getElementById("myInput2"), lista2);

}
```

*Figura 27. Extracto de código JS.*

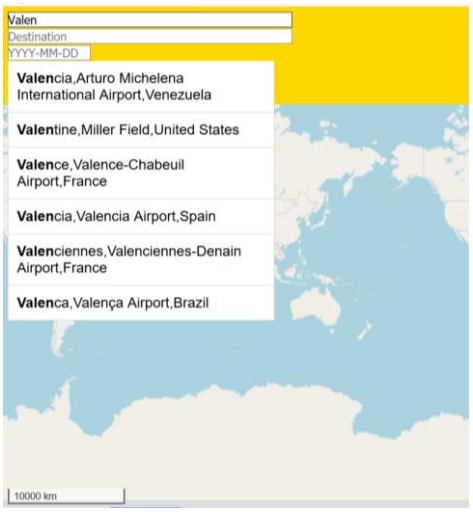



*Figura 28. Autocompletado del formulario.*

## Fecha

El <input> de la fecha, como se aprecia en la imagen superior, requiere un formato de fecha específico, el cual es año-mes-día. Cada vez que se introduce una fecha, hay una función que revisa si el formato es adecuado y devuelve True o False, y salta una alerta si es False.

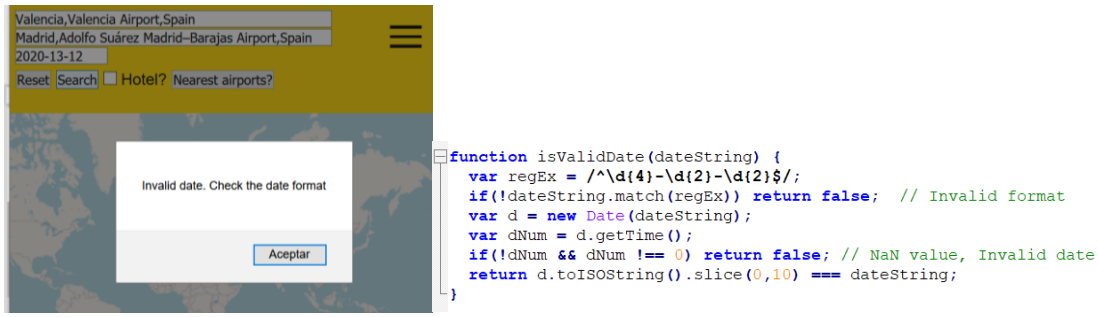



*Figura 29. Función de comprobación de la fecha.*

```
if(isValidDate(fecha) == false && fecha.length > 0) {
    alert('Invalid date. Check the date format');
    return;
}
```

¿Por qué se revisa también si hay una fecha escrita? Porque la API de Skyscanner permite buscar un vuelo para una trayectoria sin una fecha fija. Así, si la fecha se queda vacía, se mostrarán vuelos sólo para Skyscanner con fecha=anytime, ya que TripAdvisor sí necesita una fecha.

```
if (origenValido == true && destinoValido == true && fecha == '') {
    //pintar polilinea
    polilinea(coord1[1],coord1[0],coord2[1],coord2[0]);

    skyscanner("https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/browsequotes/v1.0/ES/EUR/en-ES/"+iata1+"/"+iata2+"/anytime");
    document.getElementById("show_results").style.visibility="visible";
}
```

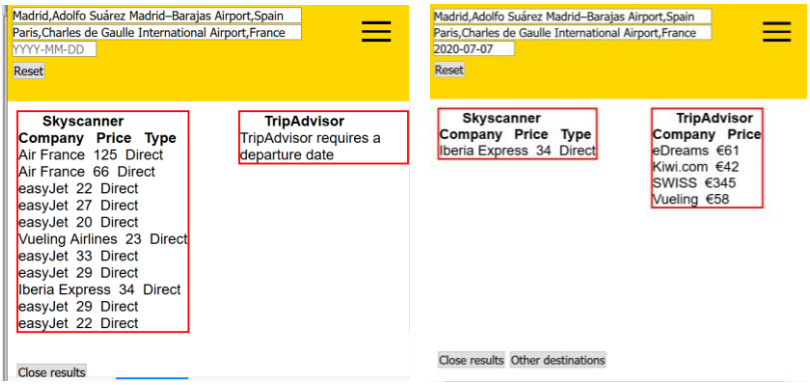



*Figura 30. Comparación de búsqueda con y sin fecha.*

## Búsqueda

Una vez rellenado completa o parcialmente el formulario, el usuario debe pulsar el botón para buscar (Search). Al clickarlo, se llama a la función getInfo(). Lo primero que ocurrirá es que se tomarán los valores del formulario y se comprobará que el origen y el destino son correctos, es decir, que se han elegido de la lista de desplegables.

```
function getInfo() {

    origen = document.getElementById("myInput").value;
    destino = document.getElementById("myInput2").value;
    fecha = document.getElementById("fecha").value;
    hotel = document.getElementById("hotel");

    var string1 = origen.split(',');
    var string2 = destino.split(',');

    var origenValido = false;
    var destinoValido = false;

    for (var i=0; i<aeropuertos.features.length; i++) {

        if (string1[1] == aeropuertos.features[i].properties.name && string1[2] == aeropuertos.features[i].properties.country) {

            origenValido = true;

            coord1 = aeropuertos.features[i].geometry.coordinates;
            iata1 = aeropuertos.features[i].properties.iata;
            country1 = aeropuertos.features[i].properties.country;
            city1 = aeropuertos.features[i].properties.city;

            //Fijar vista
            map.setView(coord1,2.5);

            //representar punto
            represent_points(coord1[0],coord1[1],'icons/origen.png',country1,city1);

            break;
        }
    }

    for (var i=0; i<aeropuertos.features.length; i++) {
        if (string2[1] == aeropuertos.features[i].properties.name && string2[2] == aeropuertos.features[i].properties.country) {

            destinoValido = true;

            coord2 = aeropuertos.features[i].geometry.coordinates;
            iata2 = aeropuertos.features[i].properties.iata;
            country2 = aeropuertos.features[i].properties.country;
            city2 = aeropuertos.features[i].properties.city;


            //Fijar vista
            map.setView(coord2,2.5);

            //representar punto
            represent_points(coord2[0],coord2[1],'icons/destino.png',country2,city2);
            break;
        }
    }
```

*Figura 31. Función JS para encontrar y representar los aeropuertos.*

Si el origen es correcto, se representará en el mapa con un icono . Lo mismo ocurrirá con el destino .

Si por el contrario, el origen y/o el destino son inválidos, se mostrará una alerta.

```
if (origenValido == false) {
    alert("Invalid origin. Choose the airport's city name from the list");
    map.setView([0.0,0.0],2);
}

if (destinoValido == false && destino != '') {
    alert("Invalid destination. Choose the airport's name from the list");
    map.setView([0.0,0.0],2);
}
```

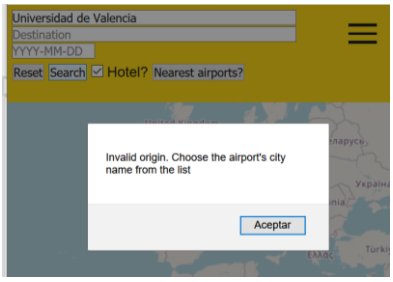



*Figura 32. Comprobación de origen y destino.*

## Búsqueda de vuelos sin destino

Si el usuario no tiene claro el destino o simplemente busca inspiración, puede introducir solo el aeropuerto de origen.

Cada destino incluirá en la etiqueta el precio del billete según la API Flight Data.

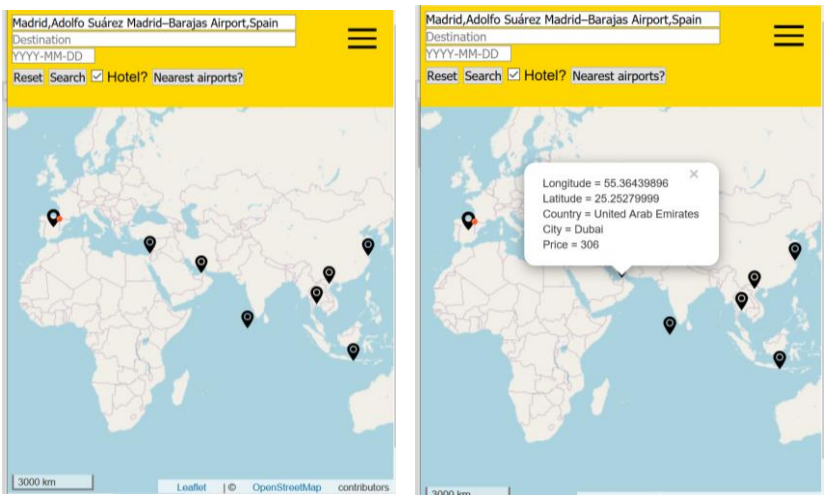



*Figura 33. Búsqueda de vuelos sin destino y fecha.*

```
function sin_destino(url){
    //Peticion HTTP a la API
    var data = null;

    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;

    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === this.DONE) {
            //console.log(this.responseText);
        }
    });

    xhr.open("GET", url);
    xhr.setRequestHeader("x-rapidapi-host", "travelpayouts-travelpayouts-flight-data-v1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "dc7559557dmsh3c6b6a2286baac9p1ccb38jsnac71766520ea");
    xhr.setRequestHeader("x-access-token", "b5377824b1fbdc4fa7b4d6d33db46c01");

    xhr.send(data);

    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                var jsonDataAPI_sin = JSON.parse(jsonObject);
                lista = Object.values(jsonDataAPI_sin.data);
                console.log(lista);
                if (lista.length<2){
                    map.setView([0.0,0.0],1.5);
                    alert('No routes available for this airport');
                }

                for (j=0;j<lista.length;j++) {
                    destinoA = lista[j].destination;
                    precio = lista[j].price;

                    for(k=0;k<aeropuertos.features.length;k++) {
                        if(destinoA == aeropuertos.features[k].properties.iata){
                            coord3 = aeropuertos.features[k].geometry.coordinates;
                            city3 = aeropuertos.features[k].properties.city;
                            country3 = aeropuertos.features[k].properties.country;

                            //representar punto
                            represent_points2(coord3[0],coord3[1],'icons/destino.png',country3,city3,precio);


                        }
                    }
                }
                map.setView([0.0,0.0],1.5);

        }
    }
}
```

*Figura 34. Petición GET a la API Flight Data.*

La función, una vez recibe el objeto JSON de respuesta, lo procesa. Para ello, compara el código IATA recibido con todos los códigos almacenados en la variable aeropuertos y así obtener el nombre, la ciudad, el país y las coordenadas del destino para así poder representarlo.

```
if (destino == '' && origenValido == true) {
    //peticion HTTP a la API estableciendo solo el origen del vuelo
    sin_destino("https://travelpayouts-travelpayouts-flight-data-v1.p.rapidapi.com/v1/city-directions?origin="+iata1+"&currency=EUR");
}
```

*Figura 35. Petición GET a la API Flight Data.*

## Búsqueda de vuelos con destino

### *Skyscanner*

Como ya se ha introducido antes, tenemos dos posibles escenarios: si se introduce una fecha o si no.

```
if (origenValido == true && destinoValido == true && fecha == '') {
    //pintar polilinea
    polilinea(coord1[1],coord1[0],coord2[1],coord2[0]);

    skyscanner("https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/browsequotes/v1.0/ES/EUR/en-ES/"+iata1+"/"+iata2+"/anytime");
    document.getElementById("show_results").style.visibility="visible";
}

if (origenValido == true && destinoValido == true && isValidDate(fecha)==true) {

    //pintar polilinea
    polilinea(coord1[1],coord1[0],coord2[1],coord2[0]);

    //Skyscanner
    skyscanner("https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/browsequotes/v1.0/ES/EUR/en-ES/"+iata1+"/"+iata2+"/"+fecha);

    //Tripadvisor
    tripadvisor1("https://tripadvisor1.p.rapidapi.com/flights/create-session?currency=EUR&ta=1&tc%252C5&c=0&d1="+iata2+"&o1="+iata1+"&dd1="+fecha)


}
```

*Figura 36. Filtrado de peticiones GET.*

La función que realiza la petición a la API de Skyscanner es la siguiente:

```
function skyscanner(url) {
    var data = null;
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;
    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === this.DONE) {
            //console.log(this.responseText);
        }
    });

    xhr.open("GET", url);
    xhr.setRequestHeader("x-rapidapi-host", "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "6297b211fdmshbcc16a675e8583cp1efda9jsne74627e9ae29");

    xhr.send(data);
    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                jsonDataAPI_sky = JSON.parse(jsonObject);
                console.log(jsonDataAPI_sky);

                lista = Object.values(jsonDataAPI_sky.Quotes);
                lista2 = Object.values(jsonDataAPI_sky.Carriers);
                dicc = [];
                dicc2 = [];

                //Generar un diccionario con el nombre de la aerolinea y su id
                for (i=0;i<lista2.length;i++) {
                    aerolinea_name = lista2[i].Name;
                    aerolinea_id = lista2[i].CarrierId;
                    dicc.push({
                        "Name": aerolinea_name,
                        "Id": aerolinea_id });
                }

                for (i=0;i<lista.length;i++) {
                    precio = lista[i].MinPrice;
                    aerolinea_id2 = lista[i].OutboundLeg.CarrierIds;

                    if (lista[i].Direct == true) {
                        direct = 'Direct';
                    }

                    if (lista[i].Direct == false) {
                        direct = 'Transfer(s)';
                    }
```

*Figura 37. Petición GET a la API de Skyscanner.*

```
                    //Comparar el id de cada vuelo con el diccionario para saber el nombre de la aerolinea
                    for (j=0;j<dicc.length;j++) {
                        if (aerolinea_id2 == dicc[j].Id) {
                            var name = dicc[j].Name;

                            dicc2.push({
                                "Name": name,
                                "Type": direct,
                                "Price": precio });
                        }
                    }

                    if (i==12) {
                        break;
                    }

                }
            }
        }
}
```

*Figura 38. Petición GET a la API de Skyscanner.*

La respuesta JSON está dividida en dos partes.

- La primera parte informa de las aerolíneas que cubren la ruta y de su identificador.
- En la segunda parte se obtiene el identificador de la aerolínea con su precio.

El objetivo es mostrar el nombre de la aerolínea junto con el precio del billete, por ello hay que almacenar en un diccionario ambas partes para posteriormente comparar el identificador que va junto al precio con el identificador de las aerolíneas. Esta comparación y representación se realiza cuando el usuario aprieta el botón Show Flights, que sirve para mostrar los vuelos disponibles.

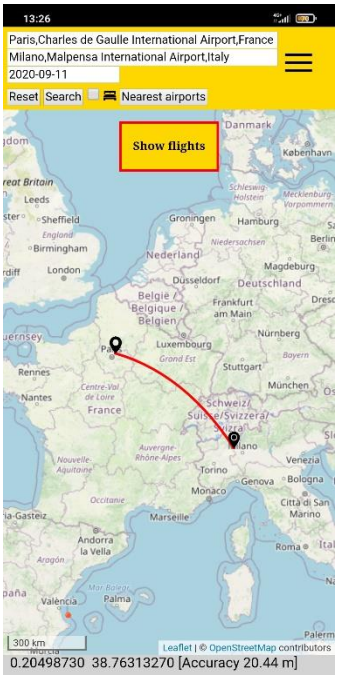



*Figura 39. Búsqueda de un vuelo con fecha.*

## *Tripadvisor*

La API de TripAdvisor funciona de una manera diferente. En primer lugar, necesita hacer una petición genérica para obtener, con los códigos IATA del origen y el destino y la fecha, un

identificador SID, que es un código interno de TripAdvisor para realizar la petición a la API de vuelos.

```
function tripadvisor1(url) {
    var data = null;

    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;

    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === this.DONE) {
            //console.log(this.responseText);
        }
    });

    xhr.open("GET", url);
    xhr.setRequestHeader("x-rapidapi-host", "tripadvisor1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "6297b211fdmshbcc16a675e8583cp1efda9jsne74627e9ae29");

    xhr.send(data);
    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                jsonDataAPI_trip = JSON.parse(jsonObject);

                //Necesitamos el sid de la ruta
                sid = jsonDataAPI_trip.search_params.sid;

                tripadvisor_vuelo(sid);

        }
    }
}
```

*Figura 40. Código del primer tipo de petición a la API de TripAdvisor.*

Una vez tenemos ese código SID, realizamos la segunda petición.

```
function tripadvisor_vuelo(sid) {

    var data = null;

    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;

    xhr.addEventListener("readystatechange", function () {
        if (this.readyState === this.DONE) {
            //console.log(this.responseText);
        }
    });

    xhr.open("GET", "https://tripadvisor1.p.rapidapi.com/flights/poll?currency=EUR&n=15&ns=NON_STOP%252CONE_STOP&so=PRICE&o=0&sid="+sid);
    xhr.setRequestHeader("x-rapidapi-host", "tripadvisor1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "dc7559557dmsh3c6b6a2286baac9p1ccb38jsnac71766520ea");

    xhr.send(data);

    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                jsonDataAPI_vuelo = JSON.parse(jsonObject);
                console.log(jsonDataAPI_vuelo);

                //precios
                lista_precios = Object.values(jsonDataAPI_vuelo.summary.pp);
                dicc_trip = [];

                for (i=0;i<lista_precios.length;i++) {
                    precio = lista_precios[i].p;
                    aerolinea = lista_precios[i].t;
                    dicc_trip.push({
                        "Aerolinea": aerolinea,
                        "Precio": precio });
                }


                //otros destinos--------------------
                lista_destinos = Object.values(jsonDataAPI_vuelo.summary.ap);
                dicc_destinos = [];
```

*Figura 41. Código del segundo tipo de petición a la API de TripAdvisor.*

```
            for (j=0;j<lista_destinos.length;j++) {
                code = lista_destinos[j].k;
                precio = lista_destinos[j].p;

                for(k=0;k<aeropuertos.features.length;k++) {
                    if(code == aeropuertos.features[k].properties.iata){
                        coord3 = aeropuertos.features[k].geometry.coordinates;
                        city3 = aeropuertos.features[k].properties.city;
                        country3 = aeropuertos.features[k].properties.country;
                        //console.log(coord3);
                        dicc_destinos.push({
                            "Precio": precio,
                            "City": city3,
                            "Country": country3,
                            "Coord": coord3 });

                    }
                }
            }

            if(dicc_trip.length > 0 || dicc2.length > 0) {
                document.getElementById("show_results").style.visibility="visible";
            }

            if(dicc_trip.length == 0 && dicc2.length == 0) {
                alert('No flights available for this route and/or date');
            }

        }
    }
}
```

*Figura 42. Almacenaje de otros posibles destinos de la respuesta JSON de TripAdvisor.*

Una vez obtenemos la segunda respuesta, generamos dos diccionarios.

- El primero almacena la aerolínea y el precio para la ruta establecida por el usuario.
- El segundo almacena una lista de posibles destinos, junto con el precio, para el aeropuerto de origen, en caso de que el usuario desee verlo. Estos posibles destinos vienen por defecto en la respuesta JSON de la API.

Una vez el diccionario de TripAdvisor y el de Skyscanner se han generado, es posible ver el botón de Show Flights. En caso de que ambos diccionarios estuvieran vacíos después de la búsqueda, querría decir que no hay vuelos para la ruta seleccionada y en ese caso se mostraría una alerta.

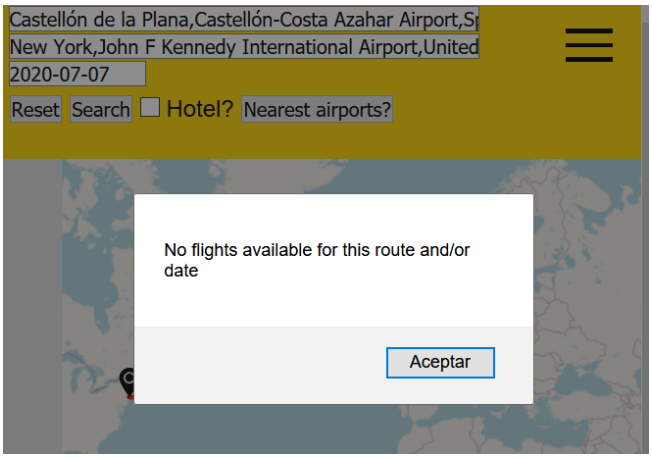



*Figura 43. Comprobación de vuelos para la ruta seleccionada.*

## Mostrar resultados

El o los resultados de la búsqueda se muestran cuando la aplicación ha recibido una o más respuestas de las API, que es cuando se muestra el botón Show Flights.

```
function show_results1() {
    results_sky.innerHTML =  "      "+"Skyscanner".bold()+"<br />";

    results_trip.innerHTML =  "      "+"TripAdvisor".bold()+"<br />";

    if (dicc2.length == 0) {
        results_sky.innerHTML += ("No flights available for this date"+"<br />"+"in Skyscanner"+ "<br/>");
    }

    if (dicc2.length > 0) {
        results_sky.innerHTML +=  "Company".bold()+"   "+"Price".bold()+"   "+"Type".bold()+"<br />";
        for(i=0;i<dicc2.length;i++) {
            name = dicc2[i].Name;
            price = dicc2[i].Price;
            direct = dicc2[i].Type;

            results_sky.innerHTML += (name+ "  " +price+ "  " +direct+ "<br />");
        }
    }



    document.getElementById("map").style.visibility="hidden";
    document.getElementById("km").style.visibility="hidden";
    document.getElementById("container").style.visibility="visible";
    document.getElementById("show_results").style.visibility="hidden";
    document.getElementById("send").style.visibility="hidden";
    document.getElementById("nearest").style.visibility="hidden";
    document.getElementById("hotel").style.visibility="hidden";
    document.getElementById("hotel1").style.visibility="hidden";


    if(dicc_trip == undefined){
        results_trip.innerHTML += ("TripAdvisor requires a departure date"+ "<br/>");
        return;
    }

    if (dicc_trip.length == 0) {
        results_trip.innerHTML += ("No flights available for this date"+"<br />"+"in TripAdvisor"+ "<br/>");
    }


    if (dicc_trip.length == 0) {
        results_trip.innerHTML += ("No flights available for this date"+"<br />"+"in TripAdvisor"+ "<br/>");
    }
    if (dicc_trip.length > 0) {
        document.getElementById("otros_destinos").style.visibility="visible";
        results_trip.innerHTML +=  "Company".bold()+"   "+"Price".bold()+"<br />";
        for (k=0;k<dicc_trip.length;k++) {
            linea = dicc_trip[k].Aerolinea;
            precio = dicc_trip[k].Precio;
            results_trip.innerHTML += (linea+ "  " +precio+ "<br />");
        }
    }
}
```

*Figura 44. Código JS del botón que muestra los resultados.*

Cuando el usuario aprieta el botón para ver los resultados, los diccionarios con los precios generados anteriormente se cargan, además de los rótulos de Skyscanner y TripAdvisor. Estos resultados se imprimen en dos 'cajas', estando ambas dentro de un contenedor junto a dos botones.

```
<div class="container" id="container">
    <div id="results_sky">
    </div>

    <div id="results_trip">
    </div>

    <div class="botones">
        <button class="btn-primary" id="close_results" type="submit" onclick="close_results()" data-toggle="tooltip" title="Close results">Close results</button>
        <button class="btn-primary" id="otros_destinos" type="submit" onclick="otros_destinos()" data-toggle="tooltip" title="Other destinations">Other destinations</button>
    </div>
</div>
```

*Figura 45. Código HTML del contenedor de resultados.*

Si alguno de los dos diccionarios no tiene resultados, se imprime en su respectiva 'caja' que no hay resultados para los parámetros introducidos por el usuario en ese metabuscador.

Cuando el usuario muestra los resultados, se hacen invisibles multitud de elementos como el mapa, botones de buscar o del aeropuerto más cercano y se visualizan otros como el de cerrar los resultados o el de mostrar otros posibles destinos.

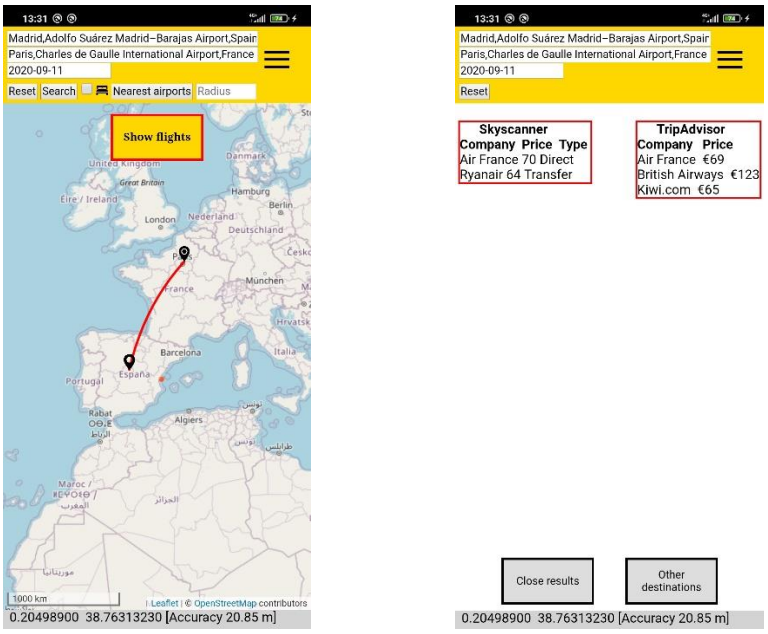



*Figura 46. Búsqueda de un vuelo con fecha.*

## Búsqueda de hoteles

Si el usuario activa el checkbox de hotel, la aplicación buscará hoteles en la ciudad de destino.

```
if (hotel.checked == true && destinoValido == true) {

    tripadvisor_loc(string2[0]);

}
```

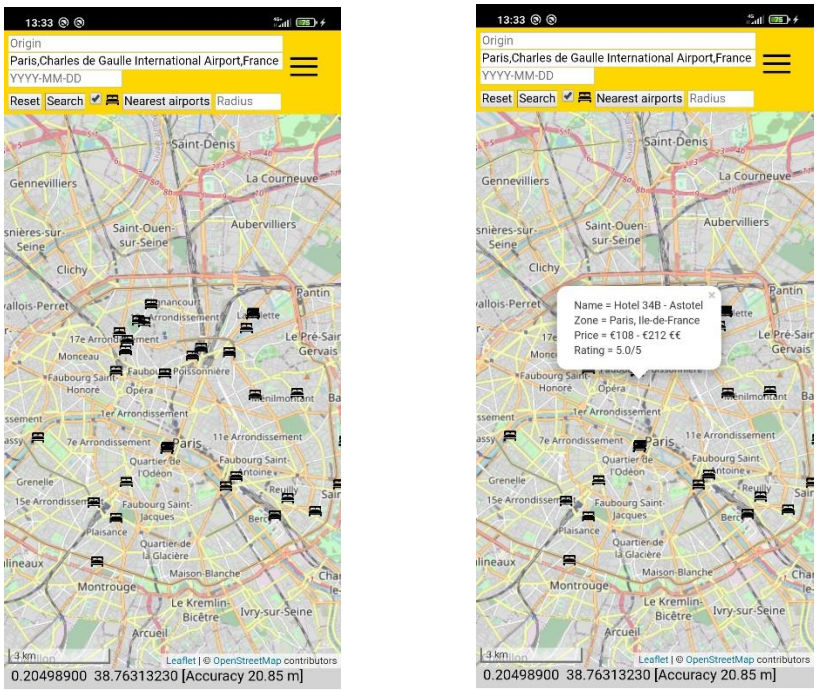



*Figura 47. Búsqueda de hoteles.*

Cada hotel incluye su nombre, la zona de la ciudad en la que está ubicado, el rango de precio por noche acompañado de entre uno y cinco símbolos de € - €€€€€ en función del rango de precios en el que lo ubique TripAdvisor, y por último la valoración de los clientes.

Para obtener esta información, primero hay que realizar una petición a la API de TripAdvisor solicitando el código de la ciudad donde queremos buscar el hotel.

```
function tripadvisor_loc(ciudad) {
    var data = null;

    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;

    xhr.addEventListener("readystatechange", function () {
    if (this.readyState === this.DONE) {
        //console.log(this.responseText);
    }
    });

    xhr.open("GET", "https://tripadvisor1.p.rapidapi.com/locations/search?location_id=1&limit=15&sort=relevance&offset=0&lang=en_EN&currency=EUR&units=km&query="+ciudad);
    xhr.setRequestHeader("x-rapidapi-host", "tripadvisor1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "6297b211fdmshbcc16a675e8583cp1efda9jsne74627e9ae29");

    xhr.send(data);
    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                jsonDataAPI_loc = JSON.parse(jsonObject);
                //console.log(jsonDataAPI_loc.data);

                lista = Object.values(jsonDataAPI_loc.data);

                //Obtenemos el id de la ciudad
                var id3 = lista[0].result_object.location_id;
                //Buscamos hoteles con el id de la ciudad
                tripadvisor_hotel(id3);

            }
        }
}
```

*Figura 48. Primera petición a la API de hoteles de TripAdvisor.*

Una vez tenemos el código, solicitamos los hoteles en una nueva petición.

```
function tripadvisor_hotel(id) {

    var data = null;

    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;

    xhr.addEventListener("readystatechange", function () {
    if (this.readyState === this.DONE) {
        //console.log(this.responseText);
    }
    });

    xhr.open("GET", "https://tripadvisor1.p.rapidapi.com/hotels/list?offset=0&currency=EUR&limit&order=asc&lang=en_EN&sort=recommended&nights&location_id="+id+"&adults=1&rooms=1");
    xhr.setRequestHeader("x-rapidapi-host", "tripadvisor1.p.rapidapi.com");
    xhr.setRequestHeader("x-rapidapi-key", "6297b211fdmshbcc16a675e8583cp1efda9jsne74627e9ae29");

    xhr.send(data);
    xhr.onreadystatechange = function() {
        if (xhr.readyState ==4 && xhr.status==200){
                jsonObject = xhr.responseText; //text string
                jsonDataAPI_hot = JSON.parse(jsonObject);
                console.log(jsonDataAPI_hot);

                lista = Object.values(jsonDataAPI_hot.data);
                for (i=0;i<lista.length;i++) {
                    lat = lista[i].latitude;
                    lon = lista[i].longitude;
                    name = lista[i].name;
                    barrio = lista[i].location_string;
                    price = lista[i].price;
                    price_level = lista[i].price_level;
                    rating = lista[i].rating;

                    represent_points4(lon,lat,'icons/hotel.png',name,barrio,price,price_level,rating)
                }
        }
    }
}
```

*Figura 49. Segunda petición a la API de hoteles de TripAdvisor.*

Es en ésta donde representaremos y añadiremos su etiqueta a cada hotel. Este sistema de doble petición a la API es muy similar al que se realiza con los vuelos en TripAdvisor.

## Otras funciones

### Geolocalización

En una aplicación basada en un mapa, una parte importante es la geolocalización del usuario. Si ésta está activada en el dispositivo y se le da permiso a la aplicación para que la utilice, en el mapa saldrá un icono 📍 indicando la ubicación del usuario, y una vez averiguada la posición, se podrá visualizar el botón 'Nearest airports', además de que la visualización se acercará a la posición del usuario.

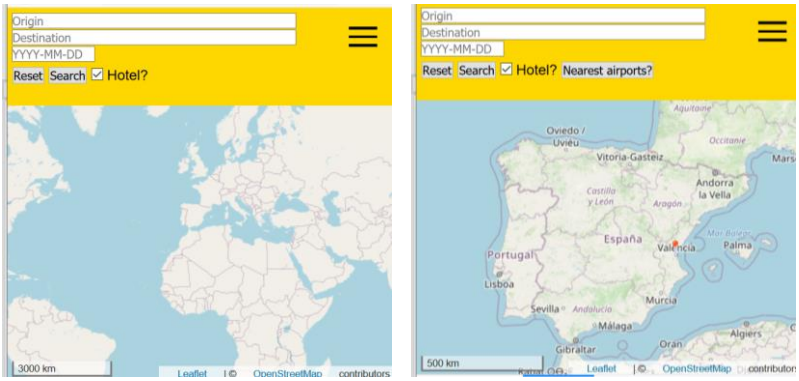



*Figura 50. Geolocalización del usuario.*

¿Para qué puede ser útil conocer esta geolocalización? Para averiguar, por ejemplo, el aeropuerto más cercano al usuario, o para saber qué aeropuertos tiene el usuario en X kilómetros de radio.

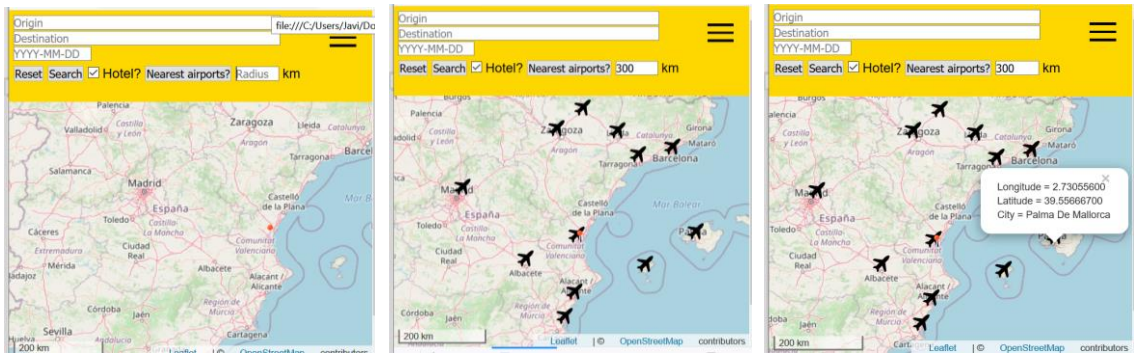



*Figura 51. Búsqueda de aeropuertos cercanos.*

Estos aeropuertos se averiguan gracias a una función desarrollada de tal manera que compara la posición del usuario con todos los aeropuertos del mundo y representa aquellos que estén dentro del radio establecido por el usuario.

```
function nearest_airport(lat,lon) {
    map.setView([lat,lon],6);
    radio1 = document.getElementById("radio").value;
    ubicacion = [lat,lon];

    for (var i=0; i<aeropuertos.features.length; i++) {
        aerop = aeropuertos.features[i].geometry.coordinates;
        city1 = aeropuertos.features[i].properties.city;
        nombre = aeropuertos.features[i].properties.name;
        distancia = getDistance(ubicacion,[aerop[1],aerop[0]]);

        if (distancia/1000 <= radio1) {
            console.log(city1);
            console.log(distancia);
            //representar punto
            represent_points3(aerop[0],aerop[1],'icons/aeropuerto.png',nombre,city1);
        }
    }
}
```

```
function getDistance(origin, destination) {
    // return distance in meters
    var lon1 = toRadian(origin[1]),
        lat1 = toRadian(origin[0]),
        lon2 = toRadian(destination[1]),
        lat2 = toRadian(destination[0]);

    var deltaLat = lat2 - lat1;
    var deltaLon = lon2 - lon1;

    var a = Math.pow(Math.sin(deltaLat/2), 2) + Math.cos(lat1) * Math.cos(lat2) * Math.pow(Math.sin(deltaLon/2), 2);
    var c = 2 * Math.asin(Math.sqrt(a));
    var EARTH_RADIUS = 6371;
    return c * EARTH_RADIUS * 1000;
}
```

*Figura 52. Código JS para encontrar los aeropuertos en un radio especificado.*

El método window.navigator.geolocation es un estándar para obtener la geolocalización en aplicaciones móviles y navegadores web, y éste siempre ha de tener las 3 funciones: getLocation, locationError y currentLocationSucces.

Además, la aplicación te informará de tu posición actual y de la precisión con la que ésta se ha calculado en una cadena de texto en la parte inferior de la aplicación.

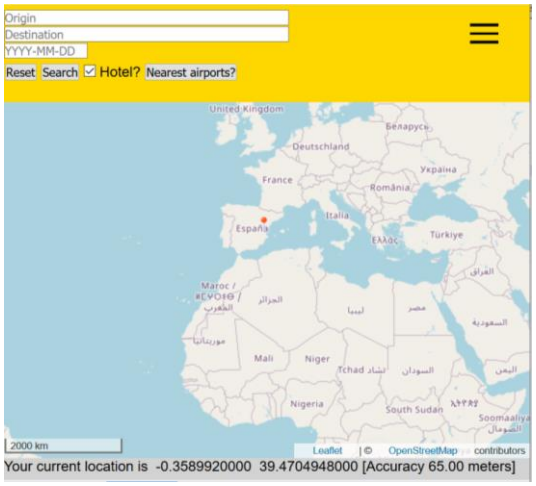



*Figura 53. Geolocalización del usuario junto a la precisión.*

## Marcadores

Cada vez que una posición es representada en el mapa, se realiza una llamada a una función. Tenemos 4 tipos de funciones para representar puntos, ya que no siempre los atributos de las etiquetas van a ser iguales.

La primera función sirve para representar el origen y el destino del vuelo.

```
function represent_points(lon,lat,imagen,country1,city1){
    //Fijar vista
    //map.setView([lat,lon],2);

    //Anadimos el marcador del origen al mapa
    var myIcon = L.icon({
    iconUrl: imagen,
    iconSize: [20, 20],
    iconAnchor: [10,20],
    riseOnHover: true
    });
    Or_marker = L.marker([lat,lon],{icon: myIcon}).addTo(map);

    mapMarkers.push(Or_marker);

    //Atributos marcador
    var markTag = "Longitude = " +lon.toFixed(8)+ "<br/>"+
                  "Latitude = " +lat.toFixed(8)+ "<br/>"+
                  "Country = " +country1+ "<br/>"+
                  "City = " +city1;
    Or_marker.bindPopup(markTag);
}
```

*Figura 54. Primera función para representar marcadores en el mapa.*

Esta función únicamente la posición, el país y la ciudad del marcador.

La segunda función se emplea para representar otros destinos e incluye el precio en la etiqueta.

```
function represent_points2(lon,lat,imagen,country1,city1,precio){
    //Fijar vista
    //map.setView([lat,lon],2);

    //Anadimos el marcador del origen al mapa
    var myIcon = L.icon({
    iconUrl: imagen,
    iconSize: [20, 20],
    iconAnchor: [10,20],
    riseOnHover: true
    });
    Or_marker = L.marker([lat,lon],{icon: myIcon}).addTo(map);

    mapMarkers.push(Or_marker);

    //Atributos marcador
    var markTag = "Longitude = " +lon.toFixed(8)+ "<br/>"+
                  "Latitude = " +lat.toFixed(8)+ "<br/>"+
                  "Country = " +country1+ "<br/>"+
                  "City = " +city1+ "<br/>"+
                  "Price = "+precio;
    Or_marker.bindPopup(markTag);
}
```

*Figura 55. Segunda función para representar marcadores en el mapa.*

La tercera es igual que la primera, pero no incluye el país porque el JSON de respuesta de la API no incluye el país, solo la ciudad.

```
function represent_points3(lon,lat,imagen,city1){
    //Fijar vista
    //map.setView([lat,lon],2);

    //Anadimos el marcador del origen al mapa
    var myIcon = L.icon({
    iconUrl: imagen,
    iconSize: [20, 20],
    iconAnchor: [10,20],
    riseOnHover: true
    });
    Or_marker = L.marker([lat,lon],{icon: myIcon}).addTo(map);

    mapMarkers.push(Or_marker);

    //Atributos marcador
    var markTag = "Longitude = " +lon.toFixed(8)+ "<br/>"+
                  "Latitude = " +lat.toFixed(8)+ "<br/>"+

                  "City = " +city1;

    Or_marker.bindPopup(markTag);
}
```

*Figura 56. Tercera función para representar marcadores en el mapa.*

Y la última función es la que se emplea para representar los hoteles 🛏 e incluir el precio por noche y el rango de precios en el que se encuentra.

```
function represent_points4(lon,lat,imagen,name,barrio,price,price_level,rating){
    //Fijar vista
    map.setView([lat,lon],12);

    //Anadimos el marcador del origen al mapa
    var myIcon = L.icon({
    iconUrl: imagen,
    iconSize: [15, 15],
    iconAnchor: [10,20],
    riseOnHover: true
    });
    Or_marker = L.marker([lat,lon],{icon: myIcon}).addTo(map);

    mapMarkers.push(Or_marker);

    //Atributos marcador
    var markTag = "Name = " +name+ "<br/>"+
                  "Zone = " +barrio+ "<br/>"+

                  "Price = " +price+ ' ' +price_level+ "<br/>"+
                  "Rating = "+rating;
    Or_marker.bindPopup(markTag);
}
```

*Figura 57. Cuarta función para representar marcadores en el mapa.*

Por último, entre el origen y el destino se dibuja una curva en rojo, la cual ha sido buscada por internet y adaptada a las necesidades de la aplicación.

```
function curve(lat1,lon1,lat2,lon2) {
    var latlngs = [];
    var latlng1 = [lat1, lon1],
      latlng2 = [lat2, lon2];

    var offsetX = latlng2[1] - latlng1[1],
      offsetY = latlng2[0] - latlng1[0];

    var r = Math.sqrt(Math.pow(offsetX, 2) + Math.pow(offsetY, 2)),
      theta = Math.atan2(offsetY, offsetX);

    var thetaOffset = (3.14 / 10);
    var r2 = (r / 2) / (Math.cos(thetaOffset)),
      theta2 = theta + thetaOffset;

    var midpointX = (r2 * Math.cos(theta2)) + latlng1[1],
      midpointY = (r2 * Math.sin(theta2)) + latlng1[0];

    var midpointLatLng = [midpointY, midpointX];
    latlngs.push(latlng1, midpointLatLng, latlng2);
    var pathOptions = {
      color: 'red',
      weight: 3
    }

    var curvedPath = L.curve(
      [
        'M', latlng1,
        'Q', midpointLatLng,
        latlng2
      ], pathOptions).addTo(map);

    map.setView([(lat1+lat2)/2,(lon1+lon2)/2],3.75);
}
```

*Figura 58. Función JS de la curva entre origen y destino.*

## Otros destinos

Cuando el usuario realiza una búsqueda en la que introduce una fecha, busca en TripAdvisor además de Skyscanner. El JSON de respuesta de TripAdvisor contiene una lista de objetos con destinos alternativos desde el origen establecido por usuario. Si el usuario lo desea, cuando visualiza los resultados, puede clickar el botón ‘Other destinations’.

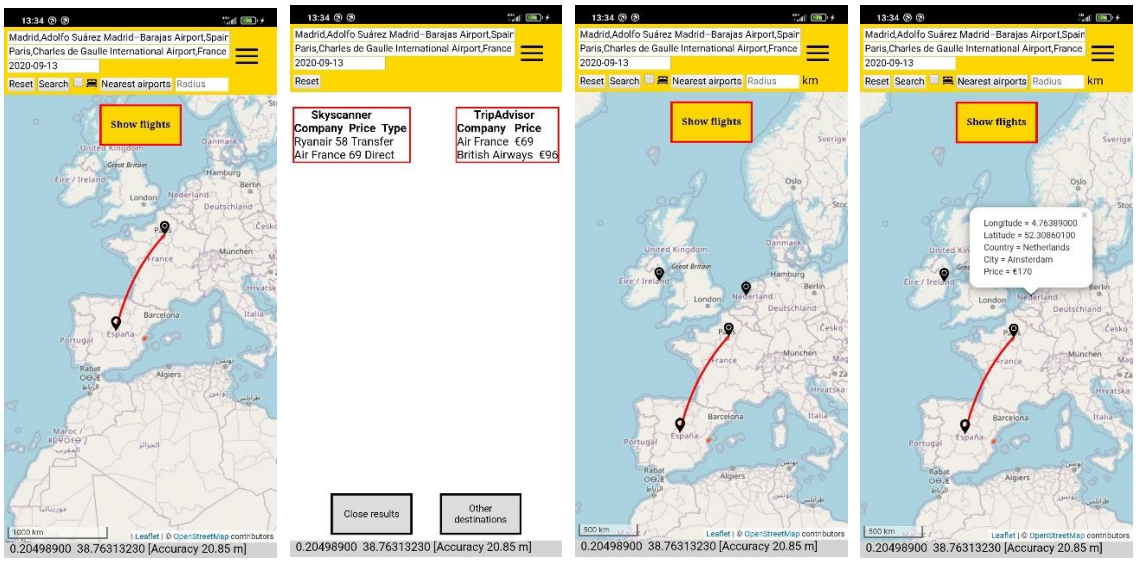



*Figura 59. Otros destinos de TripAdvisor.*

Estos posibles destinos se almacenan en un diccionario generado con la respuesta de la API de TripAdvisor.

```
//otros destinos----------------------
lista_destinos = Object.values(jsonDataAPI_vuelo.summary.ap);
dicc_destinos = [];

for (j=0;j<lista_destinos.length;j++) {
    code = lista_destinos[j].k;
    precio = lista_destinos[j].p;

    for(k=0;k<aeropuertos.features.length;k++) {
        if(code == aeropuertos.features[k].properties.iata){
            coord3 = aeropuertos.features[k].geometry.coordinates;
            city3 = aeropuertos.features[k].properties.city;
            country3 = aeropuertos.features[k].properties.country;
            //console.log(coord3);
            dicc_destinos.push({
                "Precio": precio,
                "City": city3,
                "Country": country3,
                "Coord": coord3 });

        }
    }
}
```

*Figura 60. Código JS del almacenaje de otros destinos.*

El cual se lee y carga cuando el usuario aprieta el botón ‘Other destinations’.

```
function otros_destinos() {


    document.getElementById("container").style.visibility="hidden";
    document.getElementById("map").style.visibility="visible";
    document.getElementById("show_results").style.visibility="visible";
    document.getElementById("send").style.visibility="visible";
    document.getElementById("nearest").style.visibility="visible";
    document.getElementById("hotel").style.visibility="visible";
    document.getElementById("hotel1").style.visibility="visible";
    document.getElementById("otros_destinos").style.visibility="hidden";

    for (i=0;i<dicc_trip.length;i++) {
        precio = dicc_destinos[i].Precio;
        ciudad = dicc_destinos[i].City;
        pais = dicc_destinos[i].Country;
        coord = dicc_destinos[i].Coord;

        //representar punto
        represent_points2(coord[0],coord[1],'icons/destino.png',pais,ciudad,precio);
    }
    map.setView([coord[0],coord[1]],4);
}
```

*Figura 61. Representación del diccionario con otros destinos.*

```
<button class="btn-primary" id="close_results" type="submit" onclick="close_results()" data-toggle="tooltip" title="Close results">Close results</button>
<button class="btn-primary" id="otros_destinos" type="submit" onclick="otros_destinos()" data-toggle="tooltip" title="Other destinations">Other destinations</button>
```

## Botón reset

Muchas aplicaciones necesitan un botón de reseteado para tener un comportamiento más fluido y amigable para el usuario. En el caso de CheapTravel, tenemos 3 formularios y puede ser un poco desagradable para el usuario tener que borrar los campos de los formularios manualmente.

```
function remove() {
    //eliminar marcadores
    for(var i = 0; i < mapMarkers.length; i++){
        map.removeLayer(mapMarkers[i]);
    }

    //eliminar polilinea
    for(i in map._layers) {
        if(map._layers[i]._path != undefined) {
            try {
                map.removeLayer(map._layers[i]);
            }
            catch(e) {
                console.log("problem with " + e + map._layers[i]);
            }
        }
    }
    //restablecer cuadro resultados
    results_sky.innerHTML =  "";
    results_trip.innerHTML =  "";

    document.getElementById('myInput').value = "";
    document.getElementById('myInput2').value = "";
    document.getElementById("radio").style.visibility="hidden"
    document.getElementById("container").style.visibility="hidden";
    document.getElementById("show_results").style.visibility="hidden";
    document.getElementById("map").style.visibility="visible";
    document.getElementById("otros_destinos").style.visibility="hidden";
    document.getElementById("send").style.visibility="visible";
    document.getElementById("hotel").style.visibility="visible";
    document.getElementById("hotel1").style.visibility="visible";

    if(pointLng != undefined) {
        document.getElementById("nearest").style.visibility="visible";
    }
    map.setView([0.0,0.0],2);
}
```

*Figura 62. Código JS para el reseteo de la aplicación.*

Además, esta función elimina los posibles marcadores y curvas que pueda haber presentes en el mapa, y también restablece el contenedor de resultados para que no se junten resultados de búsquedas diferentes. También restablece la visualización del mapa a su posición original, al igual que la visualización de los botones salvo de los aeropuertos más cercanos que, si la geolocalización había tenido éxito, se mantendrá.

## Botón Close results

Este botón permite intercambiar la visualización entre los resultados y el mapa, es decir, realiza la función opuesta a Show Flights. Al pinchar sobre él, se ocultará el contenedor de resultados y se mostrará nuevamente el mapa.

```
function close_results() {
    results_sky.innerHTML = '';
    document.getElementById("container").style.visibility="hidden";
    document.getElementById("map").style.visibility="visible";
    document.getElementById("show_results").style.visibility="visible";
    document.getElementById("send").style.visibility="visible";
    document.getElementById("hotel").style.visibility="visible";
    document.getElementById("hotel1").style.visibility="visible";
    document.getElementById("otros_destinos").style.visibility="hidden";

    if(pointLng != undefined) {
        document.getElementById("nearest").style.visibility="visible";
        document.getElementById("km").style.visibility="hidden";
    }


}
```

*Figura 63. Código JS del botón que cierra el contenedor de los resultados.*

## Otras páginas

Aparte del mapa y los formularios, CheapTravel tiene un menú que contiene dos páginas más que pueden ayudar al usuario a obtener cierta información sobre el funcionamiento de la aplicación.

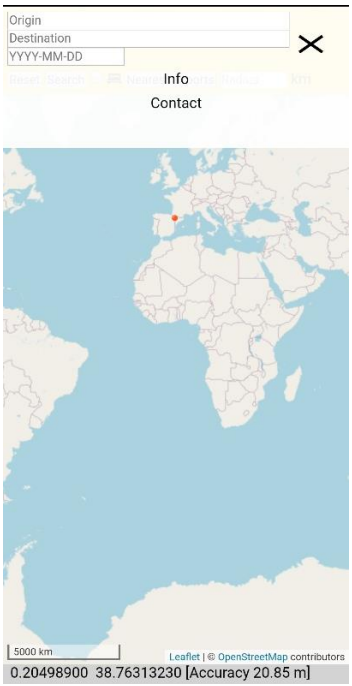



*Figura 64. Menú de la aplicación para elegir otras páginas.*

## Información de CheapTravel

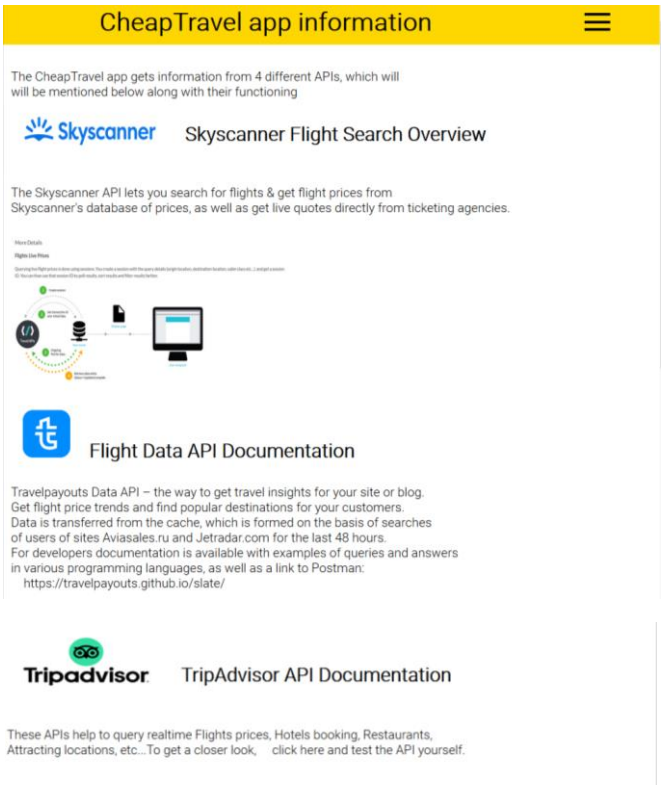



*Figura 65. Página de información de la aplicación.*

Esta página explica el funcionamiento de las APIs de RapidAPI, e incluye enlaces sus páginas web.

## Contacto

Esta página da información sobre el desarrollador y la universidad en la que se ha desarrollado la aplicación, y permite al usuario ponerse en contacto con el mismo si encuentra algún problema o tiene preguntas.

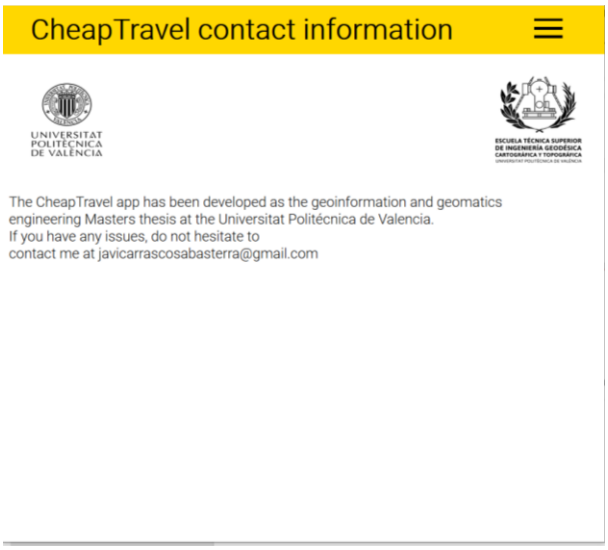



*Figura 66. Página de contacto de la aplicación.*

## Cordova

Apache Cordova es un marco de desarrollo móvil de código abierto. Permite utilizar tecnologías web estándar: HTML5, CSS3 y JavaScript para el desarrollo multiplataforma. Las aplicaciones se ejecutan dentro de contenedores dirigidos a cada plataforma y se basan en enlaces API que cumplen con los estándares para acceder a las capacidades de cada dispositivo, como sensores, datos o el estado de la red.

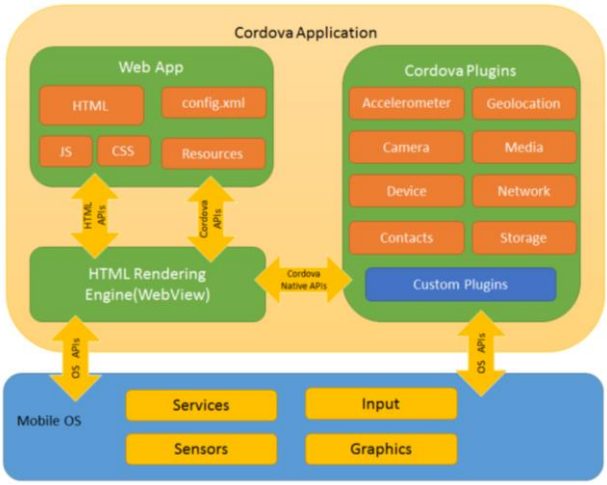
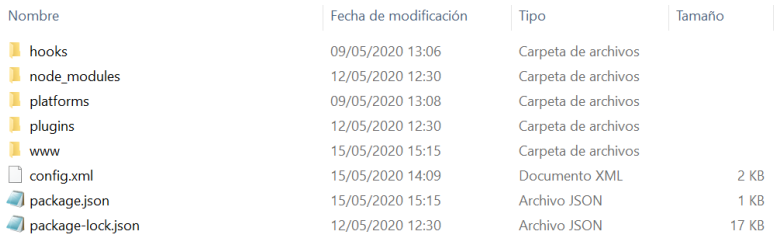



*Figura 67. Esquema de funcionamiento de Cordova.*

Cordova no es un software ni sencillo ni amigable, ya que únicamente funciona a través de la terminal. Es necesario tener instalado Node.js, además de JAVA JDK 1.8 y los SDK de Android que se vayan a emplear, actualmente Android 10.

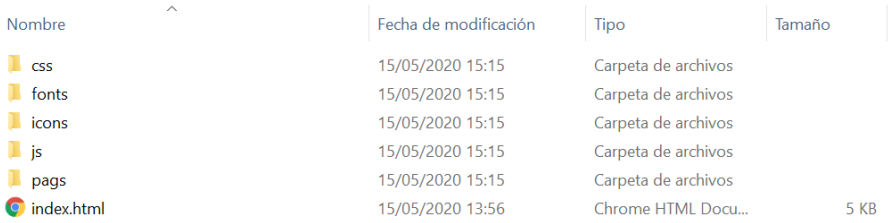
Una vez están listos todos los prerrequisitos, es posible crear un proyecto de Cordova en un directorio con la siguiente estructura.



| Nombre | Fecha de modificación | Tipo | Tamaño |
|---|---|---|---|
| hooks | 09/05/2020 13:06 | Carpeta de archivos | |
| node_modules | 12/05/2020 12:30 | Carpeta de archivos | |
| platforms | 09/05/2020 13:08 | Carpeta de archivos | |
| plugins | 12/05/2020 12:30 | Carpeta de archivos | |
| www | 15/05/2020 15:15 | Carpeta de archivos | |
| config.xml | 15/05/2020 14:09 | Documento XML | 2 KB |
| package.json | 15/05/2020 15:15 | Archivo JSON | 1 KB |
| package-lock.json | 12/05/2020 12:30 | Archivo JSON | 17 KB |

*Figura 68. Aspecto de un proyecto de Cordova.*

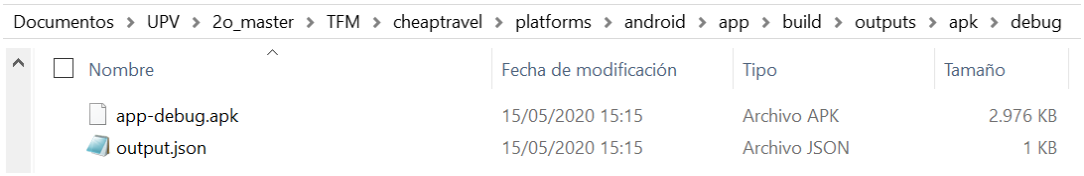
En la carpeta www se depositará el proyecto de HTML + CSS y Javascript.



| Nombre | Fecha de modificación | Tipo | Tamaño |
|---|---|---|---|
| css | 15/05/2020 15:15 | Carpeta de archivos | |
| fonts | 15/05/2020 15:15 | Carpeta de archivos | |
| icons | 15/05/2020 15:15 | Carpeta de archivos | |
| js | 15/05/2020 15:15 | Carpeta de archivos | |
| pags | 15/05/2020 15:15 | Carpeta de archivos | |
| index.html | 15/05/2020 13:56 | Chrome HTML Docu... | 5 KB |

*Figura 69. Códigos HTML, CSS y Javascript.*

Una vez creado el proyecto, es posible añadir plataformas como Android o IOS. Por último, sólo faltaría construir la APK para la plataforma deseada. Una APK es un fichero de instalación de una aplicación en Android, de la misma manera que un .exe es un ejecutable en Windows.



Documentos › UPV › 2o_master › TFM › cheaptravel › platforms › android › app › build › outputs › apk › debug

| Nombre | Fecha de modificación | Tipo | Tamaño |
|---|---|---|---|
| app-debug.apk | 15/05/2020 15:15 | Archivo APK | 2.976 KB |
| output.json | 15/05/2020 15:15 | Archivo JSON | 1 KB |

*Figura 70. Fichero APK generado por Cordova.*

Para instalar la APK es necesario activar en el teléfono las 'Opciones de desarrollador', y más concretamente, la depuración USB.

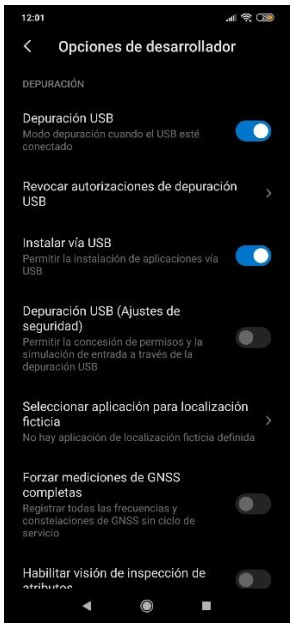



*Figura 71. Activación de las opciones de desarrollador en un teléfono Android.*

El proyecto de Cordova contiene un fichero de configuración config.xml, el cual es clave para permitir a la aplicación realizar cierto tipo de funcionalidades, entre las que destacan:

- Nombre e icono de la aplicación

```
<name>CheapTravel</name>
<icon src="www/icons/aeropuerto.png" />
```

*Figura 72. Nombre e icono de la aplicación en el fichero config.xml*

- Permisos para la geolocalización y visualización del mapa

```
<access origin="*" />

<allow-navigation href="*" />
```

*Figura 73. Geolocalización y visualización del mapa en el fichero config.xml*

# Soporte

## Leaflet

Leaflet es la biblioteca JavaScript de código abierto líder para mapas interactivos aptos para dispositivos móviles. Posee todas las características de mapeo que la mayoría de los desarrolladores necesitan.

Leaflet está diseñado teniendo en cuenta la simplicidad, el rendimiento y la usabilidad. Funciona de manera eficiente en todas las principales plataformas móviles y de escritorio, se puede ampliar con muchos complementos, tiene una API sencilla y bien documentada y un código fuente simple y legible al que es posible contribuir.

Leaflet ha contribuido al desarrollo de CheapTravel en las siguientes facetas:

- El mapa, incluido un control de escala y de visualización (L.tileLayer, L.control.scale)

```
//world map...
map.setView([0.0,0.0],2);
L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
    attribution: "&copy; <a href=\"http://osm.org/copyright\">OpenStreetMap</a> contributors",
    //pane: 'labels'
}).addTo(map)
L.control.scale({imperial: false, maxWidth:150}).addTo(map);
```

*Figura 74. Carga del mapa de Leaflet en la aplicación.*

- Los marcadores junto con sus etiquetas (L.marker, L.icon)

```
function represent_points(lon,lat,imagen,country1,city1){
    //Fijar vista
    //map.setView([lat,lon],2);

    //Anadimos el marcador del origen al mapa
    var myIcon = L.icon({
    iconUrl: imagen,
    iconSize: [20, 20],
    iconAnchor: [10,20],
    riseOnHover: true
    });
    Or_marker = L.marker([lat,lon],{icon: myIcon}).addTo(map);

    mapMarkers.push(Or_marker);

    //Atributos marcador
    var markTag = "Longitude = " +lon.toFixed(8)+ "<br/>"+
                  "Latitude = " +lat.toFixed(8)+ "<br/>"+
                  "Country = " +country1+ "<br/>"+
                  "City = " +city1;
    Or_marker.bindPopup(markTag);
}
```

*Figura 75. Función de Leaflet para representar puntos en el mapa.*

- Las polilíneas curvas (L.curve)

```
function curve(lat1,lon1,lat2,lon2) {
    var latlngs = [];
    var latlng1 = [lat1, lon1],
      latlng2 = [lat2, lon2];

    var offsetX = latlng2[1] - latlng1[1],
      offsetY = latlng2[0] - latlng1[0];

    var r = Math.sqrt(Math.pow(offsetX, 2) + Math.pow(offsetY, 2)),
      theta = Math.atan2(offsetY, offsetX);

    var thetaOffset = (3.14 / 10);
    var r2 = (r / 2) / (Math.cos(thetaOffset)),
      theta2 = theta + thetaOffset;

    var midpointX = (r2 * Math.cos(theta2)) + latlng1[1],
      midpointY = (r2 * Math.sin(theta2)) + latlng1[0];

    var midpointLatLng = [midpointY, midpointX];
    latlngs.push(latlng1, midpointLatLng, latlng2);
    var pathOptions = {
      color: 'red',
      weight: 3
    }

    var curvedPath = L.curve(
      [
        'M', latlng1,
        'Q', midpointLatLng,
        latlng2
      ], pathOptions).addTo(map);

    map.setView([(lat1+lat2)/2,(lon1+lon2)/2],3.75);
}
```

*Figura 76. Función para representar curvas en el mapa.*

## Open Street Map

Es el proveedor de las teselas de imagenes que carga Leaflet. Se trata de mapas con una licencia abierta con un alojamiento apoyado por UCL, Bytemark Hosting, y otros socios.

# Bibliografía


Red Hat. (2012). Qué son las API y para qué sirven. [Consulta 30 marzo 2020]

<https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>

Wikipedia. Asociación Internacional de Transporte Aéreo (IATA). [Consulta 18 jun 2020]

<https://es.wikipedia.org/wiki/Asociaci%C3%B3n_Internacional_de_Transporte_A%C3%A9reo#C%C3%B3digo_IATA>

MDN Web Docs. Métodos de petición HTTP. [Consulta 30 marzo 2020]

<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

TripadAvisor. Información sobre Tripadvisor. [Consulta 10 abril 2020]

<https://tripadvisor.mediaroom.com/es-about-us>

RapidAPI. Skyscanner API. [Consulta 10 abril 2020]

<https://rapidapi.com/skyscanner/api/skyscanner-flight-search>

RapidAPI. TripAdvisor API. [Consulta 5 abril 2020]

<https://rapidapi.com/apidojo/api/tripadvisor1>

RapidAPI. Flight Data API. [Consulta 10 abril 2020]

<https://rapidapi.com/Travelpayouts/api/flight-data>

Open Street Maps. [Consulta 23 mayo 2020]

<https://www.openstreetmap.org/#map=4/26.75/1.89>

Leaflet. Documentación. [Consulta 17 marzo 2020]

<https://leafletjs.com/>

Cordova. Documentation.

<https://cordova.apache.org/>

Material de apoyo de la asignatura ‘Desarrollo de aplicaciones geoespaciales en dispositivos móviles’.

# Anexo

## Index.html

```
<!DOCTYPE HTML>
<html lang="es">
<head>
    <title>CheapTravel</title>
    <meta charset="UTF-8"/>
    <!-- Tag for mobile devices -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no"/>

    <link rel="stylesheet" type="text/css" href="css/proyecto.css">
    <link rel="stylesheet" type="text/css" href="css/all.css">
    <link rel="stylesheet" type="text/css" href="css/leaflet.css">
    <link rel="stylesheet" type="text/css" href="css/lista.css">

    <script type="text/javascript" src="js/geolocation.js"> </script>
    <script type="text/javascript" src="js/leaflet.js"></script>
    <script type="text/javascript" src="js/leaflet.curve.js"></script>
    <script type="text/javascript" src="js/lista.js"></script>

    <!--menu-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300">
    <link rel="stylesheet" href="./css/font.css">
    <link rel="stylesheet" href="./css/main.css">
    <!--menu-->
</head>

<body onload="onDeviceReady()">
<div class="app-gui">

    <div class="rotulo" id="rotulo">
    <!-- FORMULARIO ---------------------------------------- -->
    <div id="d.formulario">

        <!--menu-->
        <input type="checkbox" id="btn-nav" class="checkbox">
        <header>
            <div class="header-container">
                <label for="btn-nav" class="btn-label">
                    <div class="header-button"></div>
                </label>
            </div>
        </header>

        <nav class="menu">
        <ul>
            <li>
```

```
            <li>
                <a href="pags/Info.html">Info</a></li>
            <li>
                <a href="pags/contacto.html">Contact</a></li>
        </ul>
        </nav>
        <!--menu-->

        <form id="f.formulario" class="form-horizontal" onsubmit="event.preventDefault();" autocomplete="off"><!-- The form name is the same that the table name -->
        <!-- Prevent default evita que el formulario se envíe -->
        <form autocomplete="off" action="/action_page.php">
            <div class="autocomplete" style="width:300px;">
                <input id="myInput" type="text" name="myCountry" placeholder="Origin" >
                <input id="myInput2" type="text" name="myCountry" placeholder="Destination" >
                <input size="12" placeholder="YYYY-MM-DD" type="text" id="fecha" name="fecha" data-toggle="tooltip" title="Date">
            </div>
        </form>

        <div id="button_formulario" class="btn-group">

            <!-- Saco los botones fuera para que no se envíe el formulario -->
            <div class="bloque1">
                <button class="btn-primary" id="reset" type="reset" onclick="remove()" data-toggle="tooltip" title="Limpia el formulario">Reset</button>
                <button class="btn-primary" id="send" type="submit" onclick="getInfo()" data-toggle="tooltip" title="Envia los datos">Search</button>
                <input type="checkbox" id="hotel" name="hotel" value="Hotel?">
                <input type="image" src="icons/hotel.png" alt="Submit" width="15" height="15" id='hotel1'>
            </div>

            <button class="btn-primary" id="nearest" type="submit" onclick="nearest_airport(pointLtd,pointLng)" data-toggle="tooltip" title="Search your nearest airport">Nearest air
            <input size="5" placeholder="Radius" value="" type="text" id="radio" name="radio" data-toggle="tooltip" >
            <label id="km" for="km">km</label><br>

        </div>

        </form>
    </div>
    </div>
    <!-- FORMULARIO ---------------------------------------- -->

    <div id="map">
        <button class="btn-primary" id="show_results" type="submit" onclick="show_results1()" data-toggle="tooltip" title="Show results">Show flights</button>
    </div>

    <div class="container" id="container">
        <div id="results_sky">
```

```
        </div>
        <div id="results_trip">
        </div>
        <div class="botones">
            <button class="btn-primary" id="close_results" type="submit" onclick="close_results()" data-toggle="tooltip" title="Close results">Close results</button>
            <button class="btn-primary" id="otros_destinos" type="submit" onclick="otros_destinos()" data-toggle="tooltip" title="Other destinations">Other destinations</button>
        </div>
    </div>

    <div id="bottom-info">
        <span id="locText"></span>
    </div>

    <script type="text/javascript" src="js/aeropuertos.js"></script>
    <script type="text/javascript" src="js/api.js"> </script>
    <script type="text/javascript" src="js/proj4.js"> </script>
    <script type="text/javascript" src="js/proyecto.js"> </script>
    <script type="text/javascript" src="js/marcadores.js"></script>
    <script type="text/javascript" src="js/otros.js"></script>

</div>
</body>
</html>
```