

# InfoHogar



Arquitectura e Integración de Sistemas Software

Grado de Ingeniería del Software

Curso 2016-2017

Joaquín Romero Moreno (joaquinvsll@gmail.com)

Asen Rangelov Baykushev (asen\_ran@mail.bg)

Arturo Pérez Sánchez (rezekyt@gmail.com )

Francisco Ferreras Villén (poli.ferreras@gmail.com)

**Tutor:** Alfonso Eduardo Márquez Chamorro

**Número de grupo:** 3

**Enlace de la aplicación:** <http://infohogar-162116.appspot.com/>

## HISTORIAL DE VERSIONES

Fecha	Versión	Detalles	Participantes
26/03/2017	1.0	- Incluye introducción, prototipos de las interfaces de usuario y diagramas UML de componentes y despliegue.	-Joaquín Romero Moreno -Asen Rangelov Baykushev -Arturo Pérez Sánchez -Francisco Ferreras Villén
30/04/2017	2.0	- Incluye introducción, prototipos de las interfaces de usuario, diagramas UML de componentes, despliegue, clases y secuencia.	-Joaquín Romero Moreno -Asen Rangelov Baykushev -Arturo Pérez Sánchez -Francisco Ferreras Villén
21/05/2017	3.0	Versión final. - Incluye introducción, prototipos de las interfaces de usuario, diagramas UML de componentes, despliegue, clases y secuencia, detalles de implementación, pruebas, y documentación de la API REST.	-Joaquín Romero Moreno -Asen Rangelov Baykushev -Arturo Pérez Sánchez -Francisco Ferreras Villén

# Índice

1	Introducción .....	4
1.1	Aplicaciones integradas .....	4
1.2	Evolución del proyecto .....	5
2	Prototipos de interfaz de usuario .....	6
2.1	Vista de index.....	6
2.2	Vista de búsqueda.....	7
2.3	Vista de viviendas .....	8
2.4	Vista de habitaciones .....	9
2.5	Vista de oficina.....	10
2.6	Vista de locales .....	11
2.7	Vista de garajes.....	12
2.8	Vista de resultados.....	13
2.9	Vista de lugares cercanos .....	14
3	Arquitectura .....	16
3.1	Diagrama de componentes.....	16
3.2	Diagrama de despliegue .....	18
3.3	Diagrama de secuencia de alto nivel .....	19
3.4	Diagrama de clases .....	20
3.5	Diagramas de secuencia .....	21
4	Implementación .....	22
5	Pruebas.....	30
6	Manual de usuario .....	37
6.1	Mashup .....	37
6.2	API REST .....	37
	Referencias .....	39

# 1 Introducción

Idealista es una de las compañías más importantes de España, que ofrece a través de un servicio online contenido de compraventa inmobiliaria. Con esta aplicación, se pretende facilitar al usuario la compra o alquiler de viviendas apoyándonos en el uso de esta plataforma. En la mayoría de ocasiones en las que se pretende encontrar una vivienda ideal, surgen ciertas dudas lo cual nos impide decidirnos entre dos o más posibles compras. Con esta aplicación se pretenden disminuir las dudas y ampliar la información que obtendremos por cada una de ellas, pudiendo observar no sólo las características del domicilio en sí, sino que también se pretende añadir información sobre las cercanías a éste mediante Google Places conociendo así si el lugar está muy aislado o no, si hay lugares de ocio cercanos, etc. Además, también se facilita la comparación entre ellos permitiéndonos guardar nuestros favoritos en GoogleDrive.

Los usuarios podrán especificar qué es lo que buscan (alquiler, venta, límites de precio, locales, garajes...) y una vez hayan localizado su búsqueda, se les ofrecerá la opción de buscar lugares cercanos a cada oferta que vea. El usuario podrá especificar sus preferencias en cuanto a lugares cercanos se refiere.

Un ejemplo sería la búsqueda de una vivienda en Sevilla. Una vez que se ha localizado una vivienda que nos llame, podremos buscar todas las estaciones de autobuses cercanas, todos los restaurantes, gimnasios, etc.

## 1.1 Aplicaciones integradas

Nombre aplicación	URL documentación API
Idealista	No es pública
GooglePlaces	<a href="https://developers.google.com/places/">https://developers.google.com/places/</a>
GoogleDrive	<a href="https://developers.google.com/drive/">https://developers.google.com/drive/</a>
GoogleMaps	<a href="https://developers.google.com/maps/?hl=es-419">https://developers.google.com/maps/?hl=es-419</a>

TABLA 1. APPLICACIÓN INTEGRADAS

**Idealista:** nos permitirá visualizar todo el contenido inmobiliario que deseemos encontrar además de poder filtrarlo según nuestras preferencias.

**GooglePlaces:** nos ofrece toda la información sobre los lugares de interés cercanos a un inmueble determinado.

**GoogleMaps:** será la herramienta que utilizaremos para obtener las coordenadas de un lugar concreto.

**GoogleDrive:** nos proporciona la posibilidad de guardar en “My Drive” información sobre la vivienda en concreto que seleccionemos.

## 1.2 Evolución del proyecto

En un principio, la idea original del proyecto era proporcionar información sobre el tiempo para cada oferta que se visualizara en la aplicación usando AccuWeather. Decidimos abandonar esta idea ya que parecía poco práctica para la compra y tan sólo válida para el alquiler. Además estuvimos a punto de no usar la API de idealista ya que era privada y tuvimos que comunicarnos con ellos para pedírsela por motivos académicos, pero finalmente nos la concedieron. Tras esto, decidimos ponernos en marcha proporcionando información más relevante a la hora de comprar una vivienda como es los lugares cercanos a ésta.

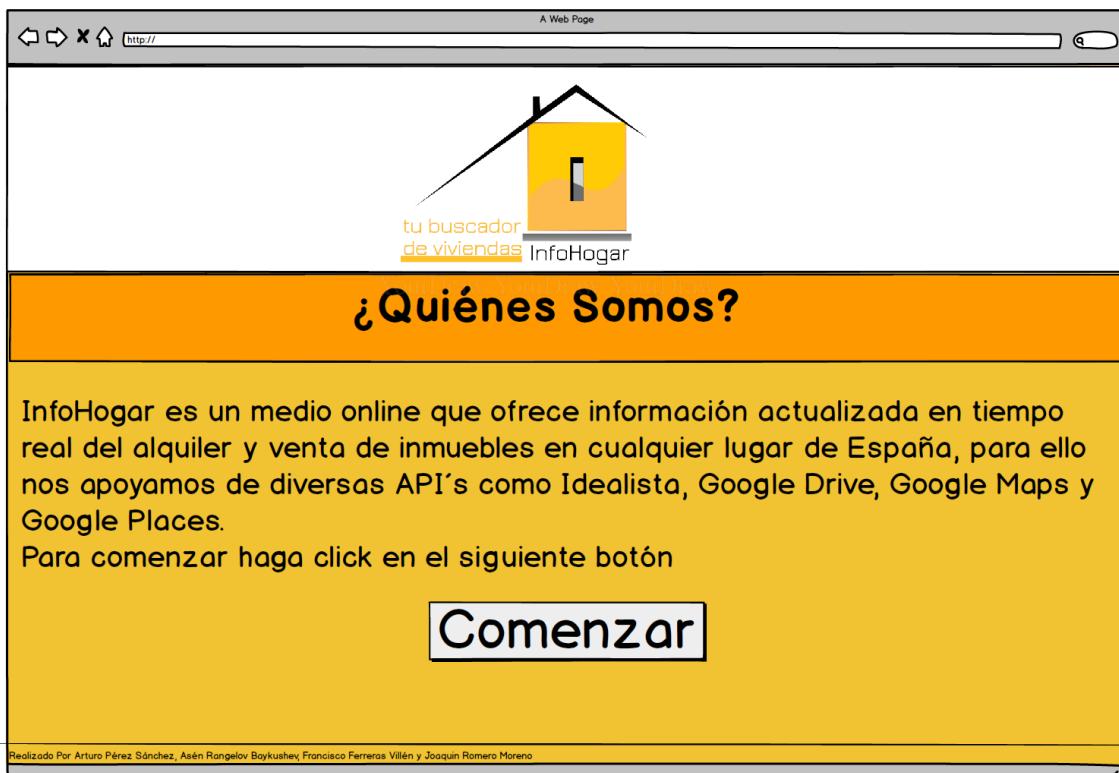
Finalmente, se asignaron partes del proyecto a cada miembro siempre estableciendo comunicación para obtener opinión o ayuda.

En el entregable dos hemos tenido que realizar un cambio de última hora. Hemos sustituido OneNote por Google Drive ante la imposibilidad de implementarlo, además debido a la mayor cantidad de usuarios de Google Drive frente a los de Microsoft. Este cambio ha provocado que tengamos que modificar gran parte del proyecto (Diagramas, descripciones...).

## 2 Prototipos de interfaz de usuario

### 2.1 Vista de index

En esta primera vista se mostrará una breve descripción del proyecto e incluirá un botón "Comenzar" que nos conducirá a la página de búsqueda.





## 2.2 Vista de búsqueda

En la página de búsqueda seleccionaremos la ciudad, el tipo de inmueble que buscamos además de si estamos buscando compra o alquiler, en función del tipo de inmueble que se seleccione nos llevará a una página diferente (Garajes, Habitaciones, Viviendas, Locales, Oficinas).

## 2.3 Vista de viviendas

Esta es la vista de viviendas, aquí seleccionaremos el tipo de vivienda, tamaño, nº de baños, nº de habitaciones y precio que buscamos, al hacer click en el botón “Buscar” nos conducirá a la página de resultados.

The screenshot shows a web browser window for 'InfoHogar' at the URL <http://infohogar-162116.appspot.com/>. The page has a yellow header with a house icon and the text 'tu buscador de viviendas InfoHogar'. Below the header is a large orange banner with the text '¿Cuáles Son Sus Criterios?'. The main content area is yellow and contains the following form fields:

- Tipo De Vivienda:** A dropdown menu currently set to 'Piso', with other options: Chalet, Casa Rústica, Estudio, Duplex, Ático.
- Tamaño (m<sup>2</sup>):** Input fields for 'Min' and 'Max' with up/down arrows.
- Precio (€):** Input fields for 'Min' and 'Max' with up/down arrows.
- Nº Habitaciones**: Radio buttons for 1, 2, 3, 4+.
- Nº Baños**: Radio buttons for 1, 2, 3+.

At the bottom left is a link 'Búsqueda Avanzada' and a large black button labeled 'Buscar'.

## 2.4 Vista de habitaciones

Esta es la vista de habitaciones en la cual deberemos introducir el rango de precio y personas con las que compartir la casa, además de las preferencias (fumadores y mascotas) que buscamos, al hacer click en el botón “Buscar” nos conducirá a la página de resultados.

The screenshot shows a web browser window for 'InfoHogar' at the URL <http://infohogar-162116.appspot.com/>. The page features a yellow header with the text '¿Cuáles Son Sus Criterios?'. Below the header, there are two sets of input fields: 'Personas Para Compartir:' with 'Min' and 'Max' dropdowns, and 'Precio (€)' with 'Min' and 'Max' dropdowns. There are also checkboxes for 'Hombre' and 'Mujer', and dropdown menus for 'Fumadores:' and 'Mascotas:', both set to 'Indiferente'. At the bottom left is a link 'Busqueda Avanzada' and a large central button labeled 'Buscar'.

## 2.5 Vista de oficina

Esta es la vista de oficina, en la que deberemos introducir el tipo de edificio y distribución así como el tamaño y precio en el que estemos interesados, al hacer click en el botón “Buscar” nos conducirá a la página de resultados.

The screenshot shows a web browser window for 'InfoHogar' at the URL <http://infohogar-162116.appspot.com/>. The page features a logo of a yellow building with a black roof and a small chimney. Below the logo, the text 'tu buscador de viviendas' and 'InfoHogar' is displayed. A large orange header bar contains the question '¿Cuáles Son Sus Criterios?'. Below this, there are two rows of search filters. The first row includes a dropdown menu 'Tipo De Edificio' set to 'Indiferente', a 'Tamaño (m<sup>2</sup>)' input field with 'Min' and 'Max' sliders, and another dropdown menu 'Distribución' set to 'Indiferente'. The second row includes a 'Precio (€)' input field with 'Min' and 'Max' sliders. At the bottom of the form is a large blue 'Buscar' button. A link 'Busqueda Avanzada' is located just above the 'Buscar' button. The footer of the page contains the text 'Realizado Por Arturo Pérez Sánchez, Asen Hongelov Boykushhev, Francisco Ferreras Villén y Joaquín Homero Moreno'.

## 2.6 Vista de locales

Esta es la vista de locales en la cual podremos introducir el rango de precio, el tipo, la ubicación y el tamaño, al hacer click en el botón “Buscar” nos conducirá a la página de resultados.

The screenshot shows a web browser window for 'InfoHogar' at the URL <http://infohogar-162116.appspot.com/>. The page features a logo of a house with a yellow door and a blue roof, with the text 'tu buscador de viviendas' and 'InfoHogar'. Below the logo, a large orange header bar contains the question '¿Cuáles Son Sus Criterios?'. The main search form is divided into several sections: 'Tipo:' with checkboxes for 'Local' and 'Nave'; 'Ubicación:' with a dropdown menu showing options like 'En Centro Comercial', 'A pie de calle', 'EntrePlanta', 'Subterraneo', and 'Otros'; 'Tamaño (m<sup>2</sup>)' with 'Min' and 'Max' input fields; and 'Precio (€)' with 'Min' and 'Max' input fields. At the bottom of the form is a large 'Buscar' button. A small link 'Busqueda Avanzada' is visible above the 'Buscar' button. The footer of the page includes a copyright notice: 'Realizado Por Arturo Pérez Sánchez, Asen Rangelov Baykushev, Francisco Ferreras Villén y Joaquín Romero Moreno'.

## 2.7 Vista de garajes

Esta es la vista de garajes en la cual deberemos introducir el rango de precio y las características que buscamos, al hacer click en el botón “Buscar” nos conducirá a la página de resultados.

The screenshot shows a web browser window for 'InfoHogar' at the URL <http://infohogar-162116.appspot.com/>. The page features a logo of a yellow garage door with a black outline. Below the logo, the text 'tu buscador de viviendas' and 'InfoHogar' is displayed. A large orange header bar contains the question '¿Cuáles Son Sus Criterios?'. The main content area has a yellow background and is titled 'Características:' in bold. To the left of the characteristics list is a vertical list of checkboxes: 'Puerta Automática', 'Plaza para motos', and 'Vigilancia'. To the right of the characteristics is a price input field labeled 'Precio (€):' with 'Min' and 'Max' dropdown arrows. At the bottom center is a large white button with the word 'Buscar' in bold black text. Below the 'Buscar' button is a link 'Busqueda Avanzada' underlined in blue. The footer contains small text: 'Realizado Por: Arturo Pérez Sánchez, Asen Horneleyová, Roviusaely, Francisco Ferrero, Vilén V., Rocío Romero Moreno'.

## 2.8 Vista de resultados

Tras realizar la búsqueda aparecerá la página de resultados en la que nos aparecerá una lista con todos los resultados que se ajusten a las características indicadas (si son muchos resultados se dividirán en distintas páginas), además, por cada resultado tendremos 3 botones: el botón "Guardar" que nos guiará a nuestra cuenta de Google Drive, el botón "Ver" que nos llevará a la página de idealista, y el botón “Lugares cercanos” que nos llevará a una página con los diferentes lugares cercanos.

InfoHogar

<http://infohogar-162116.appspot.com/>



**tu buscador de viviendas** InfoHogar

## Resultados De Su Búsqueda

Casa Terrera en 742 Evergreen Terrace, Springfield

 500€/mes Ver [idealista](#) Guardar 

4 Habitaciones Lugares Cercanos 

Chalet en 1600 Pennsylvania Ave NW

 200.000€/mes Ver [idealista](#) Guardar 

412 Habitaciones Lugares Cercanos 

[\*\*<< Anterior\*\*](#) [\*\*Siguiente >>\*\*](#)

Realizado Por Arturo Pérez Sánchez, Asen Rangelov Boykushhev, Francisco Ferreras Villén y Joaquín Romero Moreno

## 2.9 Vista de lugares cercanos

InfoHogar

tu buscador de viviendas InfoHogar

## Seleccione un tipo de lugar

Banco Estación De Autobuses Estación De Trenes Gimnasio  
Farmacia Gasolinera Hospital Policía Peluqueria  
Tienda Restaurante

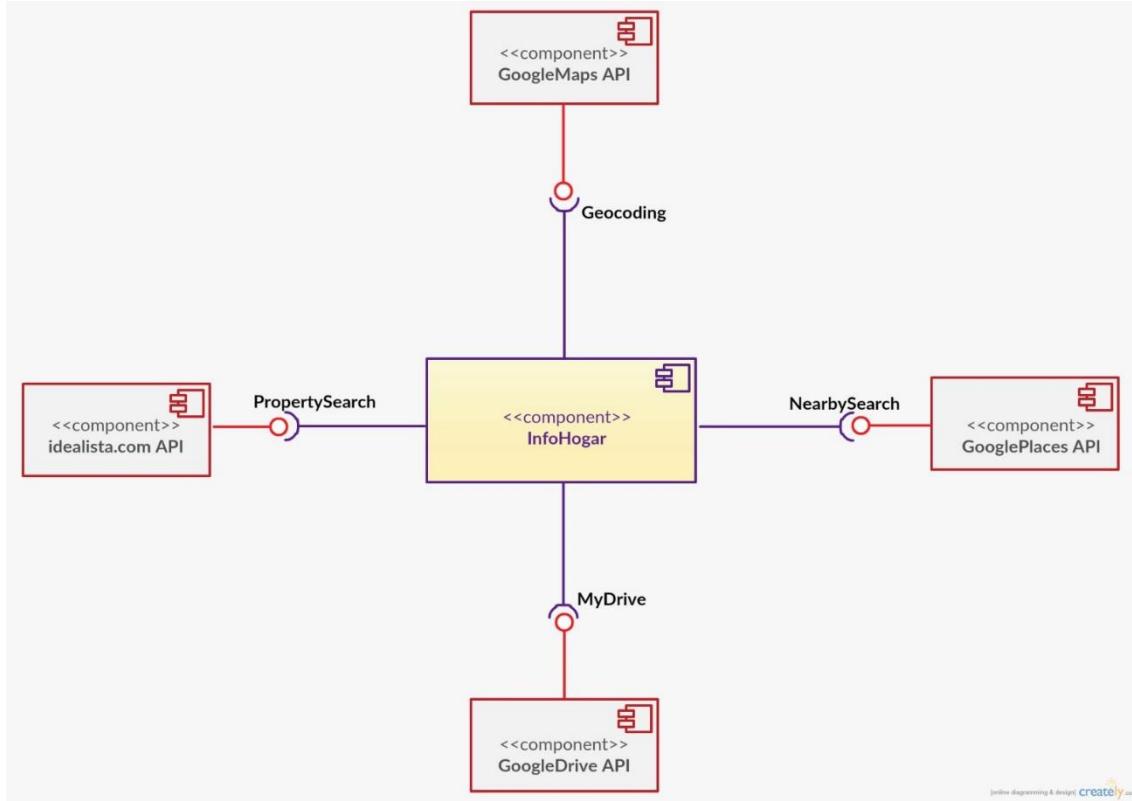
 Restaurante "El Rincón De Beirut"  
Dirección: C/ San Fernando, 21  
Precios: Variados Rating: 4.7

Realizado Por Arturo Pérez Sánchez, Asén Rangelov Baykushev, Francisco Ferreras Villén y Joaquín Romero Moreno

Al hacer click en “Lugares cercanos” nos aparecerá esta vista en la cual podremos seleccionar un tipo de lugar y nos aparecerá una lista con todos los lugares cercanos.

### 3 Arquitectura

#### 3.1 Diagrama de componentes



El objetivo principal de InfoHogar es proporcionarle al usuario un sofisticado buscador de viviendas que le facilite el arduo proceso de encontrar la vivienda perfecta. Para conseguir este objetivo, la aplicación requerirá el uso de varios tipos de servicios y, por lo tanto, se tendrán que integrar varias aplicaciones externas que proporcionen estos servicios.

Dado que InfoHogar no dispondrá de su propia base de datos en la que se almacenen anuncios de viviendas, InfoHogar requerirá que otra aplicación ya existente le suministre los datos. Para ello, hemos decidido integrar en InfoHogar la aplicación Idealista, la cual es considerada uno de los portales líderes inmobiliarios en España. Hemos elegido integrar Idealista ya que sus desarrolladores proporcionan una simple API de búsqueda que permite el acceso a todos los datos de la página web idealista.com. La API de Idealista ofrece la posibilidad de buscar viviendas según una gran variedad de criterios usando llamadas RESTful y devolver el resultado de la búsqueda en formato JSON. En fin, los servicios que Idealista ofrece se ajustan perfectamente a las necesidades de InfoHogar.

No obstante, al integrar la aplicación Idealista en nuestra aplicación, nos encontramos con un problema. Uno de los parámetros requeridos que todas las solicitudes enviadas hacia Idealista deben contener son las coordenadas geográficas del sitio donde se deseé

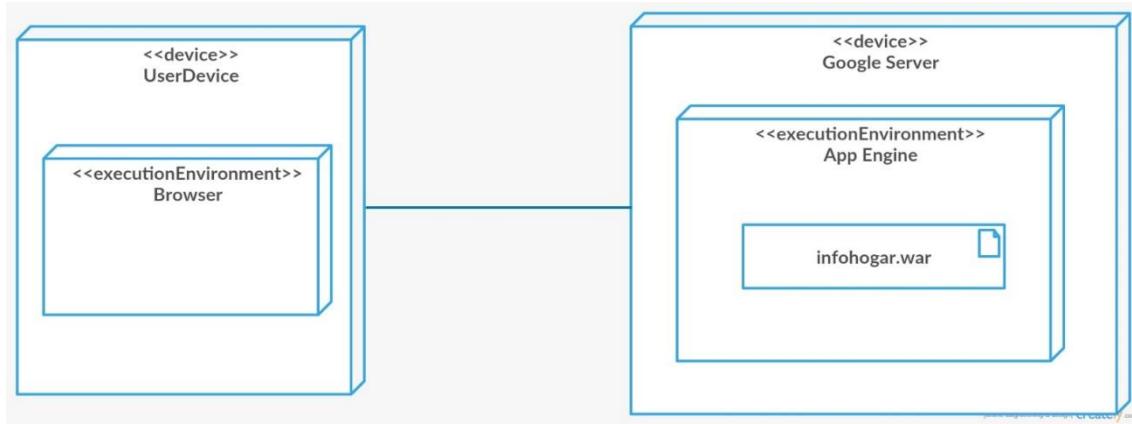
buscar viviendas. Nosotros queremos evitar a toda costa que el usuario de InfoHogar tenga que introducir coordenadas para hacer su búsqueda. Por lo tanto, InfoHogar requerirá también de los servicios que proporciona la API de Google Maps, y, en concreto, los servicios de la Google Maps Geocoding API. Geocoding es el proceso en el que se obtienen las coordenadas geográficas a partir de una dirección dada. Es justo lo que InfoHogar necesita para que el usuario no tenga que introducir coordenadas sino una dirección válida.

En resumen, lo que InfoHogar va a hacer es simplemente usar Idealista para ofrecer a los usuarios la posibilidad de buscar viviendas. No obstante, lo que nosotros pretendemos hacer es que la aplicación que vamos a desarrollar proporcione, además, un valor añadido. Es decir, queremos dotar nuestra aplicación de nuevas funcionalidades que Idealista no ofrezca.

Como ya se ha dicho anteriormente, en el mundo de los inmobiliarios, muchas veces el cliente busca viviendas sin saber nada acerca de la zona en la que se centre su búsqueda. Si el usuario de un portal inmobiliario (como Idealista, por ejemplo) desea saber si cerca de una determinada vivienda hay restaurantes o tiendas, tendrá que comprobarlo abriendo una nueva ventana en su navegador y buscarlo en Internet. Nuestra idea es que InfoHogar sea capaz de mostrar esta información automáticamente para cualquiera de las viviendas que el usuario elija. Para ello, InfoHogar requerirá usar los servicios conocidos como Nearby Search, esto es, buscar lugares cercanos a una dirección dada. La API de Google Places proporciona estos servicios y por eso, hemos decidido integrar Google Places también.

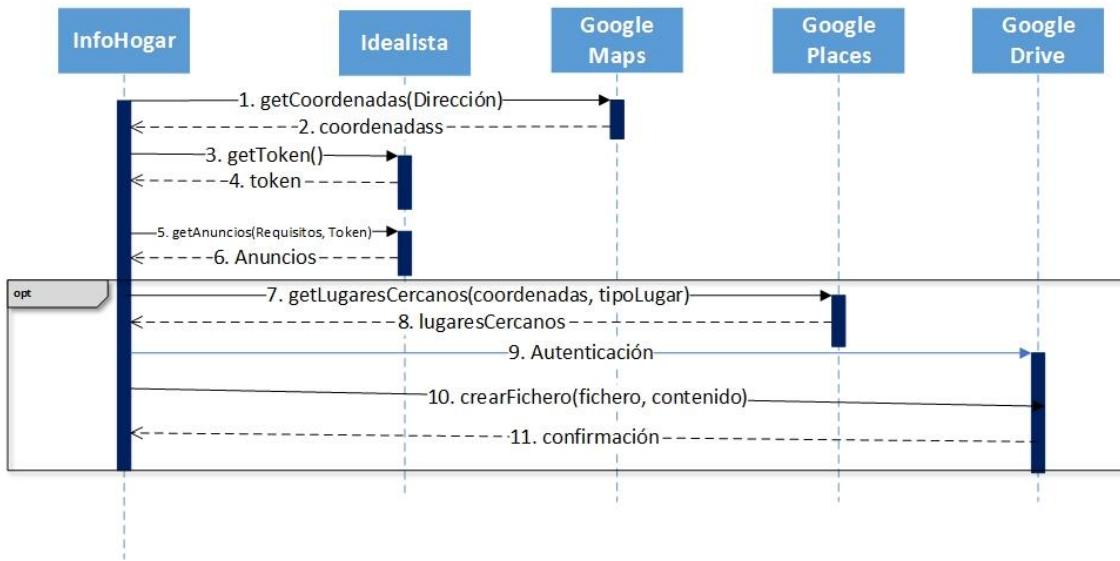
Por último, hemos decidido hacer que el usuario de InfoHogar pueda almacenar en algún lugar la información más relevante sobre los anuncios que más le hayan llamado la atención. Hemos investigado que, hoy en día, muchas personas utilizan la aplicación Google Drive para tomar notas o recopilar información. Por tanto, hemos decidido integrarla, de manera que si el usuario tiene una cuenta en Google, pueda entrar en Google Drive desde InfoHogar y almacenar como un pequeño fichero de texto la información resumida acerca de un anuncio interesante.

### 3.2 Diagrama de despliegue



En este diagrama se ilustran los nodos (dispositivos y entornos de ejecución) y los artefactos que forman parte del despliegue de InfoHogar. La aplicación va a ser desplegada en dos tipos de dispositivos. Al lado del cliente tenemos el dispositivo a través del cual el cliente va a usar la aplicación. Este dispositivo puede ser tanto un ordenador personal como un teléfono móvil o una tablet. Al lado del servidor, el dispositivo que se va a usar será el servidor de Google. Los dos tipos de dispositivos se conectan entre sí a través de una red de comunicaciones (normalmente, una red inalámbrica). Los entornos de ejecución también son de dos tipos. Al lado del cliente, InfoHogar va a utilizar como entorno el navegador web disponible en el dispositivo (por ejemplo, Google Chrome, Mozilla Firefox, etc.) mientras que al lado del servidor el entorno de ejecución será la plataforma AppEngine proporcionada por Google. Es en esta plataforma donde se va a desplegar el artefacto (esto es, la aplicación en sí), que en nuestro caso es un ejecutable que tiene la extensión war (war es el formato que se utiliza para desplegar aplicaciones web Java).

### 3.3 Diagrama de secuencia de alto nivel



En este diagrama de secuencia de alto nivel, se muestra de manera genérica el funcionamiento de nuestra aplicación web InfoHogar.

Para iniciar la aplicación, basta con que el usuario escriba una dirección en el buscador de la misma, la cual obtendrá las coordenadas geográficas de esa dirección que serán proporcionadas por la API de Google Maps, nuestra aplicación le mandará esta dirección y una clave (API KEY) ya que sin estos parámetros no sería posible obtener las coordenadas geográficas.

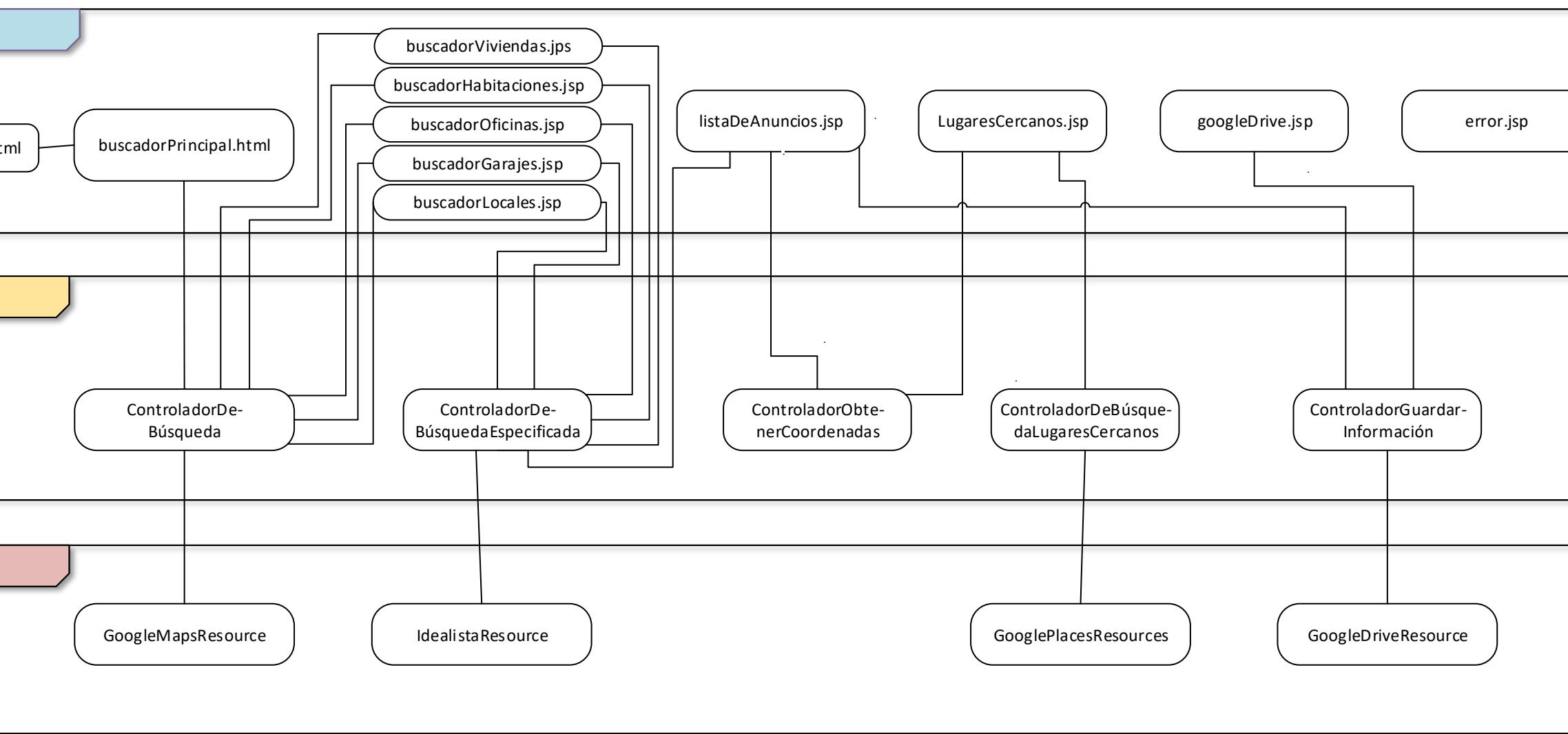
Una vez obtenidas estas coordenadas, InfoHogar pasará a interaccionar con la API de Idealista, la cual dada una ID del cliente y un parámetro secret, nos devolverá un parámetro llamado token que será necesario para el uso de la misma. Al hacer esto, el usuario podrá seleccionar una serie de características que aparecerán en pantalla relacionadas con la/s vivienda/s y al elegirlas nuestra aplicación se encargará de pedir a la API de Idealista y obtener de la misma una serie de propiedades de ella/s(también se le pasarán las coordenadas obtenidas anteriormente).

A continuación, si el usuario desea consultar lugares cercanos, de cualquier tipo, a la/s vivienda/s, tan solo deberá indicar el tipo concreto de los lugares a consultar. Para ello, nuestra aplicación se comunicará con la API de Google Places, enviando un mensaje de petición de lugares cercanos en el que se incluirán las coordenadas de la vivienda, más el tipo de lugares a buscar que haya indicado el usuario.

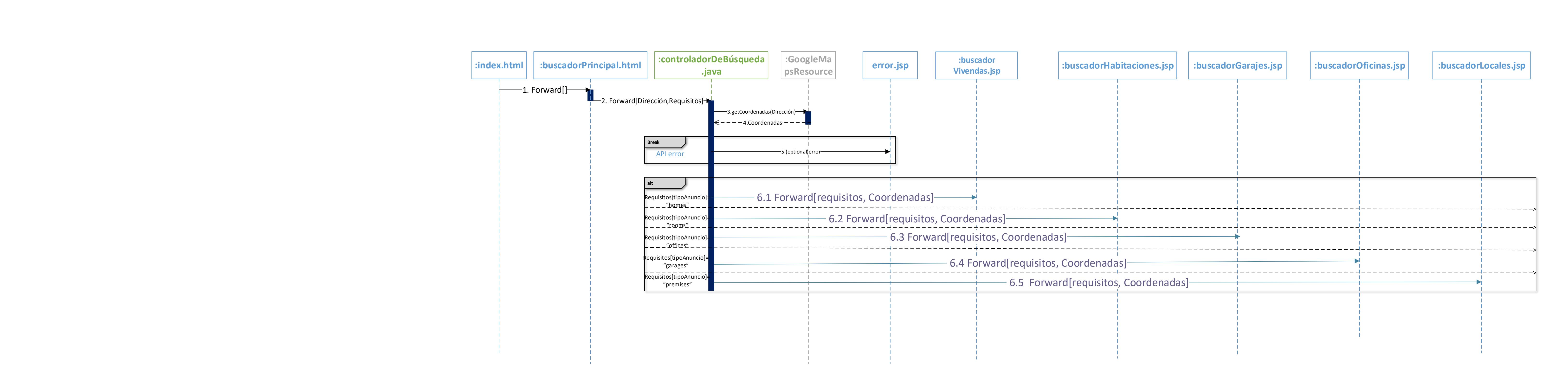
A parte de obtener el/los lugar/es cercano/s, InfoHogar también da la opción al usuario de guardar su búsqueda, es decir, viviendas concretas que él mismo decida. Para hacer esto, el usuario debe disponer de una cuenta en Google Drive e indicar a nuestra aplicación que desea guardar una vivienda en concreto. Cuando el usuario, haya indicado esto, InfoHogar deberá obtener otro parámetro token siguiendo el mismo proceso que con la API de Idealista pero esta vez con la API de Google Drive,

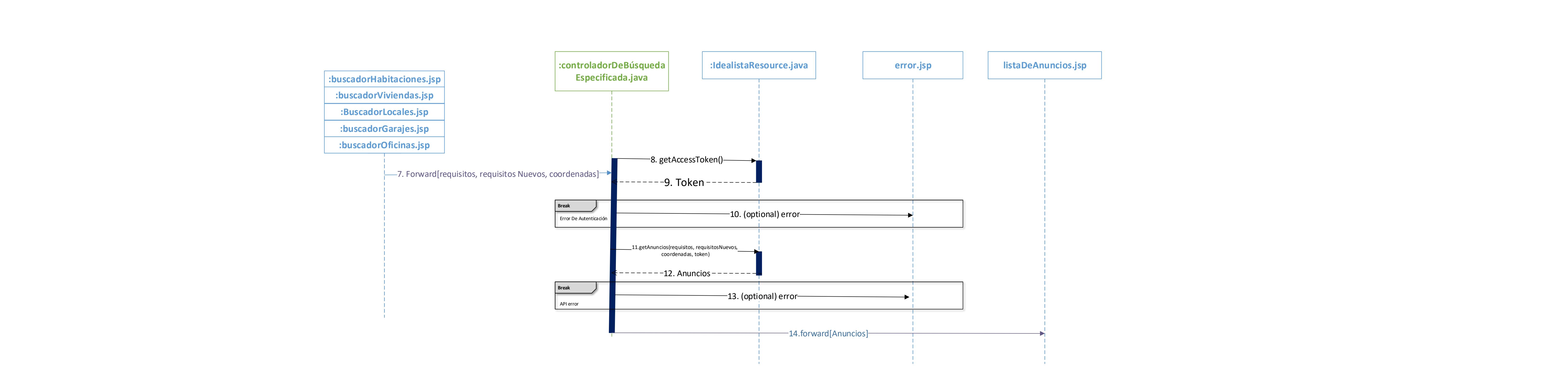
para así posteriormente permitir al usuario guardar su vivienda. Esta se guardará en una nota de Google Drive de la cuenta del usuario en forma de URL, la cual será indicada por él mismo. Cuando el usuario proporcione a nuestra aplicación esta URL, será enviada a la API de Google Drive y nos indicará si se ha guardado correctamente mediante un mensaje mostrado en InfoHogar (se podrán guardar tantas viviendas como el usuario desee).

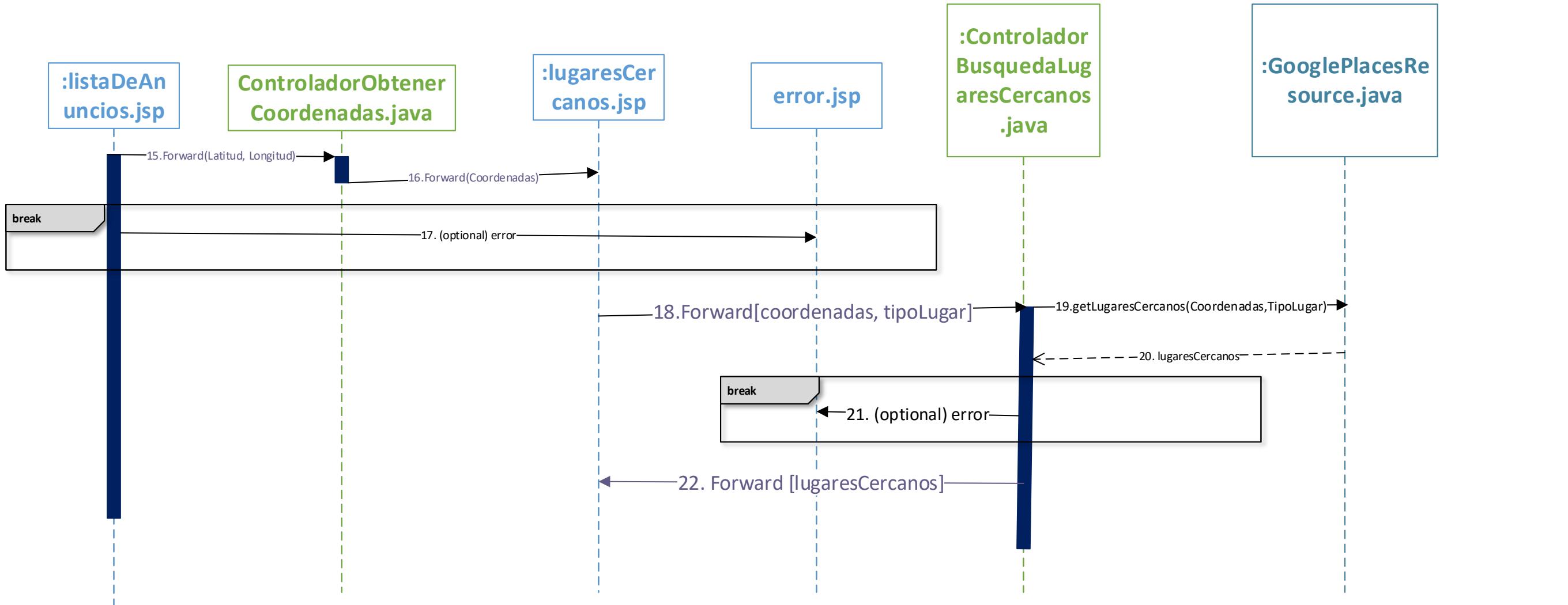
### 3.4 Diagrama de clases

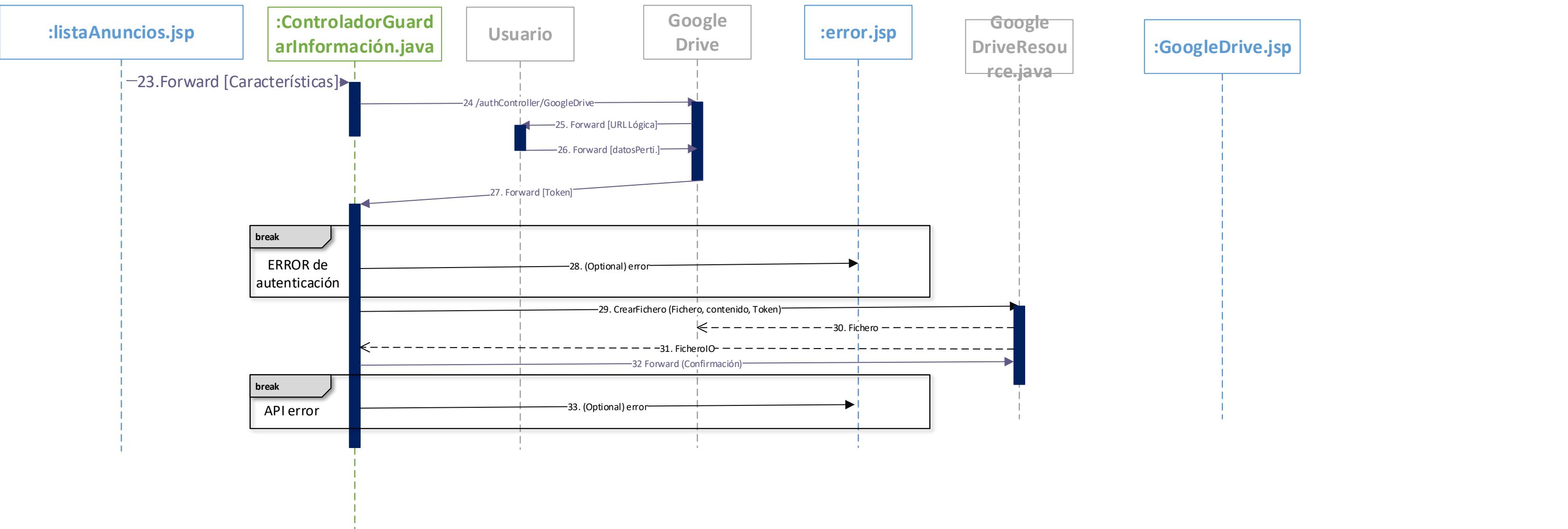


### 3.5 Diagramas de secuencia









## 4 Implementación

Como ya se ha descrito anteriormente, el objetivo de nuestro proyecto es crear una nueva aplicación inmobiliaria integrando un buscador de viviendas externo y proporcionándole a este navegador nuevas funcionalidades que él no tenga. A continuación se describe cómo lo hemos conseguido.

Centrémonos primero en la integración del buscador de viviendas. Ya hemos dicho que el buscador que nosotros usamos es el de idealista.com y hemos explicado por qué hemos decidido integrar este buscador en concreto. Según la API de Idealista, para obtener los resultados de una búsqueda de viviendas, hay que enviar una petición de tipo POST al servicio y en la propia URL que se use para realizar dicha petición hay que insertar una serie de parámetros que especifiquen la búsqueda (por ejemplo, tipo de vivienda, tamaño mínimo, tamaño máximo, precio mínimo, precio máximo, número de habitaciones, etc.). El buscador devuelve los resultados de la búsqueda, es decir, devuelve los inmuebles que tengan todas las características especificadas como parámetros en la URL. Por lo tanto, antes que nada nuestra aplicación tiene que obtener estos parámetros. El cliente de InfoHogar tendrá que especificar sus requisitos y luego la aplicación tendrá que usar los datos introducidos para crear una URL adecuada y enviar la petición al servicio de Idealista. Para hacer esto, hemos decidido usar formularios.

No obstante, a la hora de crear los formularios, nos hemos encontrado con un problema. Algunos de estos parámetros son obligatorios y tienen que estar presentes en la URL de cualquier petición. En concreto, se tienen que especificar las coordenadas geográficas de un punto en España, así como una distancia en metros. De esta forma, la API de Idealista va a devolver sólo los resultados para la región concreta, esto es, todos los inmuebles que se encuentren a una distancia menor o igual a la distancia especificada del punto. Para que el cliente de InfoHogar no tenga que introducir coordenadas geográficas ni distancias, hemos utilizado la siguiente metodología.

Hemos integrado GoogleMaps en la aplicación para que el cliente pueda introducir direcciones en vez de coordenadas. GoogleMaps devuelve dos datos muy importantes, a partir de los cuales se pueden sacar los parámetros requeridos. El primer dato que se obtiene a través de GoogleMaps son las coordenadas geográficas del punto central geográfico de la región que corresponda a la dirección introducida. Estas coordenadas en realidad son uno de los parámetros obligatorios que Idealista exige. El segundo dato que se devuelve son las coordenadas de un punto del extremo de la región correspondiente a la dirección introducida. En concreto, hemos usado el punto más sudoeste. De esta forma tenemos las coordenadas geográficas de dos puntos y para obtener la distancia (que es el segundo parámetro obligatorio) simplemente tenemos que calcular la distancia entre los dos puntos. Para calcular esta distancia, hemos implementado la siguiente función:

```

private double distance(double lat1, double lat2, double lon1, double lon2, double el1, double el2) {

    final int R = 6371; // El radio de la tierra

    Double latDistance = Math.toRadians(lat2 - lat1);
    Double lonDistance = Math.toRadians(lon2 - lon1);
    Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2) + Math.cos(Math.toRadians(lat1))
        * Math.cos(Math.toRadians(lat2)) * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // pasar a metros

    double height = el1 - el2;

    distance = Math.pow(distance, 2) + Math.pow(height, 2);

    return Math.sqrt(distance);
}

```

```

public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    RequestDispatcher rd = null;

    String direccion = req.getParameter("direccion");

    GoogleMapsResource googleMaps = new GoogleMapsResource();
    Geocoding geocoding = googleMaps.getCoordenadas(direccion);

    if (geocoding != null) {
        /* Hemos decidido sacar únicamente el primer resultado de la lista obtenida por el
         * método getResults() visto que en la documentación de la GoogleMaps Geocoding API pone:
         * "Generally, only one entry in the "results" array is returned for address lookups,
         * though the geocoder may return several results when address queries are ambiguous."
        */
        Geometry coordenadas = geocoding.getResults().get(0).getGeometry();
        req.setAttribute("latitud", coordenadas.getLocation().getLat());
        req.setAttribute("longitud", coordenadas.getLocation().getLng());
        // Para realizar una operación de consulta y obtener los anuncios correspondientes a una dirección dada,
        // la API de Idealista requiere que se pasen los parámetros center y distance.
        // Necesitaremos que GoogleMaps devuelva tanto las coordenadas del centro de la zona (el parámetro center)
        // como las coordenadas del punto más sudoeste de la zona. Después tendremos que implementar
        // una función que calcule la distancia entre los dos puntos (el parámetro distance)
        req.setAttribute("latitudPuntoMasSudoeste", coordenadas.getViewport().getSouthwest().getLat());
        req.setAttribute("longitudPuntoMasSudoeste", coordenadas.getViewport().getSouthwest().getLng());
    }
}

```

La API de Idealista proporciona búsqueda tanto de viviendas, como también de oficinas, garajes, habitaciones y locales. Por lo tanto, existen muchos parámetros específicos que son válidos sólo para un concreto tipo de inmueble. Nos hemos dado cuenta de que crear un formulario no es suficiente y hemos decidido crear varios. El primer formulario se corresponde con el buscador principal. En este formulario el usuario especifica los parámetros que son comunes para todos los tipos de inmuebles. Estos parámetros son la dirección a partir de la cual se calculan las coordenadas y la distancia, y la operación que el usuario desee realizar – comprar o alquilar. Además en este formulario el usuario elige el tipo de inmueble a buscar – vivienda, habitación, oficina, garaje o local. En función de lo que el usuario elija, a continuación se abrirá otro formulario que se corresponda con el buscador específico para el tipo de inmueble elegido. Es en este formulario donde el usuario detalla su búsqueda añadiendo otros parámetros. Pero al final, todos los parámetros – tanto los parámetros comunes (los del buscador principal) como los parámetros específicos para un determinado tipo de inmueble (los del buscador secundario) se tienen que pasar a un controlador que se encargue de hacer la petición al servicio de Idealista y obtener los resultados de la búsqueda. Para hacer esto hemos procedido de la siguiente manera:

Los datos introducidos en el primer formulario se mandan a un controlador. En este controlador mediante GoogleMaps se calculan las coordenadas de los dos puntos que determinan la zona de la búsqueda (a partir de la dirección introducida) y los resultados obtenidos se mandan al segundo formulario. Los demás datos se redirigen al segundo formulario tal como son. La dirección donde está ubicado el segundo formulario se determina mediante un switch en función del tipo de inmueble seleccionado en el primer formulario. De esta forma, en el segundo formulario además de los nuevos campos a llenar, también figuran los parámetros introducidos en el primer formulario. Estos parámetros no deben ser transparentes para el usuario, por lo que se colocan en campos de tipo hidden.

```

String tipoAnuncio = req.getParameter("tipoAnuncio");

switch (tipoAnuncio) {
    default:
        rd = req.getRequestDispatcher("/buscadorViviendas.jsp");
        break;
    case "bedrooms":
        rd = req.getRequestDispatcher("/buscadorHabitaciones.jsp");
        break;
    case "offices":
        rd = req.getRequestDispatcher("/buscadorOficinas.jsp");
        break;
    case "garages":
        rd = req.getRequestDispatcher("/buscadorGarajes.jsp");
        break;
    case "premises":
        rd = req.getRequestDispatcher("/buscadorLocales.jsp");
        break;
}

String busqueda = req.getParameter("busqueda");
req.setAttribute("busqueda", busqueda);

```

```

<p>
    <strong> Todos los campos que aparecen en esta página son
    opcionales </strong>
</p>

<div>
    <input type="hidden" name="Latitud" id="Latitud" value="${latitud}">
</div>

<div>
    <input type="hidden" name="Longitud" id="Longitud"
           value="${longitud}">
</div>

<div>
    <input type="hidden" name="LatitudPuntoMasSudoeste"
           id="LatitudPuntoMasSudoeste" value="${latitudPuntoMasSudoeste}">
</div>

<div>
    <input type="hidden" name="LongitudPuntoMasSudoeste"
           id="LongitudPuntoMasSudoeste" value="${longitudPuntoMasSudoeste}">
</div>

<div>
    <input type="hidden" name="copiaTipoAnuncio" id="copiaTipoAnuncio"
           value="homes">
</div>

<div>
    <input type="hidden" name="copiaBusqueda" id="copiaBusqueda"
           value="${busqueda}">

```

Así se consigue que todos los datos (tanto los del primer formulario como los del segundo) se manden al siguiente controlador a la vez. En dicho controlador se calcula la distancia entre los dos puntos geográficos y luego todos los parámetros que se hayan especificado en los dos formularios se van colocando en un Map<String, String>. Las claves de este Map son los nombres de los parámetros, tal y como tienen que aparecer en la URL de la petición que se va a hacer al servicio de la Idealista. El valor para cada clave del Map es el valor que el usuario haya especificado en el campo correspondiente. Por ejemplo, una pareja clave-valor del Map puede ser "bathrooms" = "3" o "newDevelopment" = "true". Al final, todas las parejas clave-valor se van añadiendo a la URL principal de la API de Idealista y de esta manera se va generando la URL de la petición que corresponda a la búsqueda del usuario. Por ejemplo, una posible petición dirigida a la API de Idealista puede tener la siguiente URL:

```
http://api.idealista.com/3.5/es/search?center=40.42938099999995,-3.7097526269835726&country=es&maxItems=50&numPage=1&distance=452&propertyType=bedrooms&operation=rent
```

```
Map<String, String> requisitos = new HashMap<String, String>();
// ----- Parámetros comunes para todas las búsquedas
requisitos.put("operation", req.getParameter("copiaBusqueda"));
requisitos.put("propertyType", req.getParameter("copiaTipoAnuncio"));

String center = req.getParameter("latitud") + "," + req.getParameter("longitud");
requisitos.put("center", center);

Double distance = this.distance(Double.valueOf(req.getParameter("latitud")),
    Double.valueOf(req.getParameter("latitudPuntoMasSudoeste")),
    Double.valueOf(req.getParameter("longitud")),
    Double.valueOf(req.getParameter("longitudPuntoMasSudoeste")), 0., 0.);
Integer distancia = new Integer(distance.intValue());
requisitos.put("distance", distancia.toString());

if (req.getParameterMap().containsKey("precioMin")) {
    requisitos.put("minPrice", req.getParameter("precioMin"));
}

if (req.getParameterMap().containsKey("precioMax")) {
    requisitos.put("maxPrice", req.getParameter("precioMax"));
}
```

```
String uri = "https://api.idealista.com/3.5/es/search?locale=es&maxItems=50&numPage=1&order=price&sort=asc&";
for (String clave : requisitos.keySet()) {
    uri += URLEncoder.encode(clave, "UTF-8");
    uri += "=";
    uri += URLEncoder.encode(requisitos.get(clave), "UTF-8");
    uri += "&";
}
uri = uri.substring(0, uri.length() - 1);
```

```

String uri = "https://api.idealista.com/3.5/es/search?locale=es&maxItems=50&numPage=1&order=price&sort=asc";
for (String clave : requisitos.keySet()) {
    uri += URLEncoder.encode(clave, "UTF-8");
    uri+= "=";
    uri += URLEncoder.encode(requisitos.get(clave),"UTF-8");
    uri += "&";
}
uri = uri.substring(0, uri.length() - 1);

```

No obstante, para realizar búsquedas de inmuebles a través de la API de Idealista, se requiere autenticación mediante OAuth2. Esto quiere decir que en la cabecera de cualquier petición dirigida a la API de Idealista tiene que haber un campo (“Authorization:”) que contenga un token válido. Para obtener un token, se tiene que enviar la siguiente petición POST a la API de Idealista:

```

POST /oauth/token HTTP/1.1
Host: api.idealista.com
Authorization: Basic YWJjOjEyMw==
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&scope=read

```

donde en el campo Authorization se tienen que poner una clave y un código secret codificados a Base64. Como la API de Idealista no es pública, le hemos pedido a los desarrolladores que nos concedan dichos datos. A continuación se muestra cómo hemos procedido para crear la petición.

```

public Token getToken() {
    Token token = null;
    String pAccessToken = IDEALISTA_API_KEY_Y_SECRET_CODIFICADOS_BASE64;
    String uri = "https://api.idealista.com/oauth/token";
    ClientResource cr = this.getClientResource(uri, pAccessToken);

    try {
        Form form = new Form();
        form.set("grant_type", "client_credentials");
        form.set("scope", "read");
        Representation repr = form.getWebRepresentation();

        token = cr.post(repr, Token.class);

    } catch (ResourceException re) {
        System.err.println("Error al obtener el token: " +cr.getResponse().getStatus());
    }

    return token;
}

```

Para instanciar el ClientResource hemos implementado la siguiente función, visto que el campo “Authorization:” es un campo especial y no se puede añadir a la cabecera de la petición directamente como los demás campos. Se tiene que utilizar la clase ChallengeResponse para establecer el esquema de autenticación. En nuestro caso, el esquema es “Basic”.

```

private ClientResource getClientResource(String uri, String pAccessToken)
{
    ClientResource lClientResource = new ClientResource(uri);
    ChallengeResponse lChallengeResponse = new ChallengeResponse(ChallengeScheme.HTTP_BASIC);
    lChallengeResponse.setRawValue(pAccessToken);
    lClientResource.setChallengeResponse(lChallengeResponse);
    return lClientResource;
}

```

Una vez que hemos obtenido el token, podemos enviar la petición correspondiente a la búsqueda. Como hemos dicho anteriormente, en el campo “Authorization:” de la cabecera de la petición tenemos que introducir el token obtenido. El procedimiento que se usa es análogo al que acabamos de ver. La única diferencia es que ahora el esquema de autenticación es “Bearer”.

```

ClientResource cr = this.getClientResource(uri, token);

try {
    StringRepresentation repr = new StringRepresentation("", MediaType.MULTIPART_FORM_DATA);
    anuncios = cr.post(null, PropertySearch.class);
} catch (ResourceException re) {
    System.err.println("Error " + cr.getResponse().getStatus());
}

```

```

private ClientResource getClientResource(String uri, String pAccessToken)
{
    ClientResource lClientResource = new ClientResource(uri);
    ChallengeResponse lChallengeResponse = new ChallengeResponse(ChallengeScheme.HTTP_OAUTH_BEARER);
    lChallengeResponse.setRawValue(pAccessToken);
    lClientResource.setChallengeResponse(lChallengeResponse);
    return lClientResource;
}

```

El resultado de la búsqueda se redirige a la siguiente vista donde los anuncios se muestran por pantalla en una tabla. Para conseguir esto, hemos utilizado forEach de la librería JSTL. Se crea una tabla y para cada uno de los anuncios que formen parte del resultado de la búsqueda se crea una fila de dicha tabla. La fila contiene la información más relevante acerca del anuncio y además aparece una foto del inmueble y un botón a través del cual se puede acceder directamente a la página de Idealista donde se ubique el anuncio.

```

<table id="tablaResultados" class="resultados">

    <c:forEach items="${requestScope.anuncios}" var="anuncio" >
        <tr>
            ...
        </tr>
    </c:forEach>

```

Para ver los lugares cercanos al inmueble hay que enviar una petición a la API de GooglePlaces. En la URL de esta petición tienen que figurar las coordenadas geográficas del punto donde se sitúe el inmueble y el tipo de lugar que el cliente quiera buscar. Para ello, hemos creado un formulario dentro de la fila correspondiente al anuncio. Este formulario contiene las coordenadas geográficas del inmueble (son datos que la API de Idealista devuelve para todos los anuncios) en campos de tipo hidden. El botón que el usuario tiene que pulsar para obtener los lugares cercanos en realidad es el botón con el que se envía el formulario.

```
<td> <form id="${anuncio.propertyCode}" method="post" action="obtenerCoordenadas">
<input type="hidden" name="Latitud" value="${anuncio.latitude}">
<input type="hidden" name="Longitud" value="${anuncio.longitude}">
<input type="submit" name="GooglePlaces" value="Ver Lugares cercanos">
</form></td>
</tr>
```

Los datos del formulario se envían a un controlador cuyo objetivo es simplemente redirigirlos a la siguiente vista. En la siguiente vista aparece un formulario a través del que el usuario tiene que seleccionar el tipo de lugar que desee buscar. Las coordenadas también forman parte de este formulario, pero no son transparentes para el usuario (están dentro de un campo de tipo hidden).

```
><form method="post" id="formulario9" action="controladorVerLugaresCercanos">
<input type="hidden" name="coordenadas" value="${coordenadas}">
<button name="Lugar" type="submit" value="bank">Banco</button> <br> <br>
<button name="Lugar" type="submit" value="bus_station">Estación de autobuses</button> <br> <br>
<button name="Lugar" type="submit" value="train_station">Estación de trenes</button> <br> <br>
<button name="Lugar" type="submit" value="gym">Gimnasio</button> <br> <br>
<button name="Lugar" type="submit" value="pharmacy">Farmacia</button> <br> <br>
<button name="Lugar" type="submit" value="gas_station">Gasolinera</button> <br> <br>
<button name="Lugar" type="submit" value="hospital">Hospital</button> <br> <br>
<button name="Lugar" type="submit" value="police">Policía</button> <br> <br>
<button name="Lugar" type="submit" value="hair_care">Peluquería</button> <br> <br>
<button name="Lugar" type="submit" value="store">Tienda</button> <br> <br>
<button name="Lugar" type="submit" value="restaurant">Restaurante</button> <br> <br>
</form>
</nav>
</c:if>
```

Los datos se mandan al siguiente controlador. El controlador se encarga de enviar una petición a la API de GooglePlaces con los datos del formulario. Si la petición se envía con éxito, se obtiene el resultado de la búsqueda. Los lugares cercanos se recogen y se redirigen a la siguiente vista donde mediante un forEach se muestra por pantalla la información más importante para cada uno de ellos.

Dentro de cada fila de la tabla de los anuncios devueltos como resultados por la API de Idealista también aparece un botón que permite al usuario guardar en un fichero de texto la información más relevante sobre el anuncio y almacenar el fichero en Google Drive. Para implementar esta funcionalidad, hemos creado otro formulario por cada fila que aparezca en la tabla. Los campos de este formulario son de tipo hidden y los datos en estos campos son los datos más relevantes que la API de Idealista devuelve – lo que queremos que aparezca en el fichero de texto. Cuando el usuario pulsa el botón, los datos se mandan al controlador correspondiente. El usuario tiene que iniciar sesión en Google para poder almacenar el fichero. Para ello se utiliza autenticación mediante OAuth2. Cuando se obtiene el token, el controlador crea el fichero de texto y escribe los datos del formulario.

```
<td>
<form id="${anuncio.propertyCode}2" method="post" action="controladorGuardarInformacion" target="_blank">
<input type="hidden" name="direccion" value="${anuncio.address}">
<input type="hidden" name="anuncio" value="${tipoDeAnuncio}">
<input type="hidden" name="precio" value="${anuncio.price}">
<input type="hidden" name="baños" value="${anuncio.bathrooms}">
<input type="hidden" name="provincia" value="${anuncio.province}">
<input type="hidden" name="municipio" value="${anuncio.municipality}">
<input type="hidden" name="comarca" value="${anuncio.district}">
<input type="hidden" name="obraNueva" value="${anuncio.newDevelopment}">
<input type="hidden" name="habitaciones" value="${anuncio.rooms}">
<input type="hidden" name="tamaño" value="${anuncio.size}">
<input type="hidden" name="operacion" value="${anuncio.operation}">
<input type="hidden" name="url" value="${anuncio.url}">

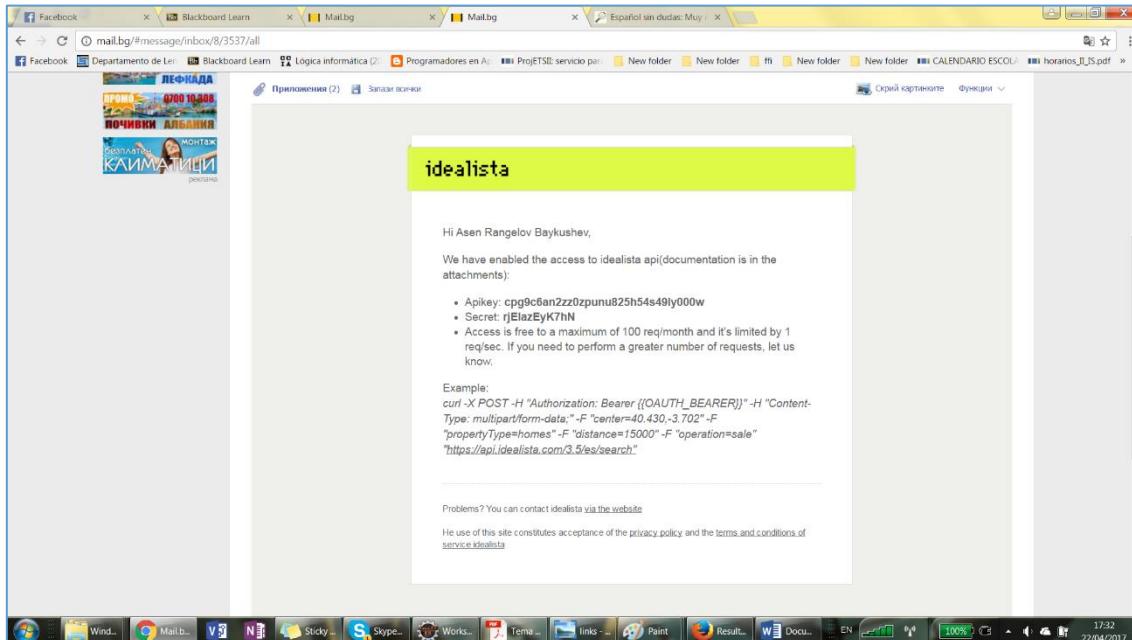
<button type="submit" name="boton" value="GuardarInformacion">
</button>
</form>
</td>
```

## 5 Pruebas

Para disminuir la probabilidad de que el código fuente contenga errores que puedan provocar el mal funcionamiento de la aplicación hemos realizado pruebas de integración. En concreto, hemos optado por hacer pruebas incrementales dado que con este tipo de pruebas es mucho más fácil detectar y aislar posibles errores en el código. Hemos desarrollado la aplicación en pequeños incrementos. La estrategia que hemos usado corresponde con la integración descendente en profundidad. Hemos dividido el proceso de integración en cuatro ramas. Lo que hemos procurado hacer en cada rama es integrar con éxito una de las cuatro componentes, esto es, las interfaces de las aplicaciones externas, en nuestra aplicación híbrida. En cada rama siempre empezamos por las vistas, continuamos con el controlador y terminamos creando el modelo y hacer que toda la interacción entre el modelo, las vistas y el controlador se realice satisfactoriamente. Siguiendo esta estrategia, hemos conseguido desarrollar InfoHogar integrando las aplicaciones GoogleMaps, Idealista, GooglePlaces y GoogleDrive (en este orden). Para tener confianza en que el código no contiene errores y no se produce ningún comportamiento inesperado cuando todavía la integración de la componente no ha finalizado (y no podemos probar en el navegador si la aplicación funciona) hemos realizado dos tipo de pruebas:

- ⊗ Pruebas unitarias. Hemos realizado unas cuantas pruebas unitarias para probar que una componente tiene el comportamiento esperado. Este tipo de pruebas nos han servido mucho porque aumentan nuestra confianza en que hemos implementado correctamente el modelo correspondiente a la componente. Sin embargo, estas pruebas no pueden demostrar que la integración de la componente haya sido exitosa porque en estas pruebas la componente se trata de manera aislada. Además, hemos podido hacer pruebas unitarias sólo para GoogleMaps y GooglePlaces. No nos hemos arriesgado a hacer pruebas unitarias para Idealista porque la API de Idealista no es pública. Sus desarrolladores nos han concedido el acceso a la API pero a condición de que no se realicen más de 100 peticiones por mes desde nuestra aplicación. La realización de pruebas unitarias sobre Idealista conllevaría un alto aumento en el número de peticiones. Por eso, en lugar de hacer pruebas unitarias sobre los distintos métodos que hemos implementado en el modelo de Idealista, hemos decidido integrar la componente por completo y probar que funciona bien en el navegador una vez finalizada la integración. Esto nos ha dificultado mucho el aislamiento y la detección de los errores que se produjeron al principio.

Hemos automatizado todas las pruebas unitarias mediante JUnit.



- ⊗ Pruebas de integración. Para probar que encima de que la componente funciona aislada del resto, interactúa correctamente con las demás componentes hemos realizado pruebas de integración. Hemos ido implementando pruebas de integración mediante el uso de servlets especiales que hemos denominado “servlets de prueba”. En estos servlets, lo que hemos hecho es imprimir en el navegador los valores de los parámetros más importantes (normalmente parámetros que se hayan devuelto por las APIs externas) para comprobar que la aplicación tiene el comportamiento esperado. También sirven para imprimir en el navegador valores de parámetros que se redirijan de las vistas al controlador y sobre las que se realicen pequeñas modificaciones. En el ejemplo que se ilustra a continuación, mediante el uso de un servlet de prueba se verifica que todos los datos de los dos formularios de búsqueda se han procesado correctamente, y además se muestran las coordenadas y la distancia calculada. Ahora podemos tener más confianza en que la URL de la petición dirigida a Idealista se va a construir correctamente.

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
    Double distance = this.distance(Double.valueOf(req.getParameter("latitud")),
        Double.valueOf(req.getParameter("latitudPuntoMasSudoeste")),
        Double.valueOf(req.getParameter("longitud")),
        Double.valueOf(req.getParameter("longitudPuntoMasSudoeste")), 0., 0.);

    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("<!doctype html>");
    out.println("<html>");
    out.println("<head><title>Servlet de Prueba</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet de Prueba 1.</h1><p>");
    out.println("<strong>Operación </strong>" + req.getParameter("copiaBusqueda") + "<br>");
    out.println("<strong>Tipo de inmueble </strong>" + req.getParameter("copiaTipoAnuncio") + "<br>");
    out.println("<strong>Tamaño mínimo </strong>" + req.getParameter("tamMin") + "<br>");
    out.println("<strong>Tamaño máximo </strong>" + req.getParameter("tamMax") + "<br>");
    out.println("<strong>Precio mínimo </strong>" + req.getParameter("precioMin") + "<br>");
    out.println("<strong>Precio máximo</strong>" + req.getParameter("precioMax") + "<br>");
    out.println("<strong>Número de habitaciones </strong>" + req.getParameter("habitaciones") + "<br>");
    out.println("<strong>Número de baños </strong>" + req.getParameter("banos") + "<br>");
    out.println("<strong>Latitud </strong>" + req.getParameter("latitud") + "<br>");
    out.println("<strong>Longitud </strong>" + req.getParameter("longitud") + "<br>");
    out.println("<strong>Distancia </strong>" + distance + "<br>");
    out.println("</body></html>");
}
```

Todos los campos que aparecen en esta página son obligatorios

Dirección:

¿Qué busca?  Viviendas  Habitaciones  Oficinas  Garajes  Locales

Especifique su búsqueda:  Compra  Alquiler

Buscador de viviendas

Todos los campos que aparecen en esta página son opcionales

Tipo de vivienda:

Tamaño (m<sup>2</sup>):

Precio (€):

Número de habitaciones:  0  1  2  3  4+

Número de baños:  0  1  2  3+

Búsqueda avanzada

Filtros:  
 Obra nueva  Con garaje  Con terraza  Con piscina  Con ascensor  Con aire acondicionado  Con trastero  Con armarios empotrados

Servlet de Prueba

Servlet de Prueba 1.

Operación sale

Tipo de inmueble homes

Tamaño mínimo 100

Tamaño máximo 280

Tipo Precio mínimo 120000

Precio máximo 540000

Número de habitaciones 2

Número de baños 2

Latitud 37.3827851

Longitud -5.967682

Distancia 191.58886145952212

Resumen	
Número total de pruebas realizadas	9
Número de pruebas automatizadas	4 (44%)

ID	Prueba 1
Descripción	Prueba para la detección de errores al obtener las coordenadas geográficas a partir de una dirección mediante la Geocoding API de GoogleMaps
Entrada	Se introduce como dirección el campus Reina Mercedes de la Universidad de Sevilla
Salida esperada	Las coordenadas que se han obtenido después de consultar una fuente externa distinta de GoogleMaps.
Resultado	ÉXITO
Automatizada	Sí

ID	Prueba 2
Descripción	Prueba para la detección de errores al obtener la distancia entre dos puntos geográficos mediante la función <i>distance</i>
Entrada	Se introducen como direcciones el campus Reina Mercedes y la Universidad de Sevilla
Salida esperada	La distancia en metros entre el campus Reina Mercedes y la Universidad de Sevilla. Se ha consultado otra fuente para obtenerla.
Resultado	ÉXITO
Automatizada	Sí

ID	Prueba 3
Descripción	Prueba para la detección de errores al utilizar la Geocoding API de GoogleMaps y la función <i>distance</i> a fin de calcular el radio de una región
Entrada	Se introduce como dirección la zona Dos Hermanas (Sevilla)
Salida esperada	La distancia en entre el punto central de Dos Hermanas y el punto más sudoeste. Se ha consultado otra fuente para obtenerla.
Resultado	ÉXITO
Automatizada	Sí

ID	Prueba 4
Descripción	Prueba para la detección de errores al obtener los lugares más cercanos a un punto geográfico mediante NearbySearch (GooglePlaces)
Entrada	Se introducen las coordenadas del campus Reina Mercedes de la Universidad de Sevilla. Se buscan las farmacias más cercanas.
Salida esperada	El nombre de la farmacia "Rosario Núñez Valdes" tiene que ser uno de los primeros resultados devueltos, puesto que la susodicha farmacia está muy cerca del campus.
Resultado	ÉXITO
Automatizada	Sí

ID	Prueba 5
Descripción	Prueba para la detección de errores al pasar <u>todos los datos</u> del buscador principal y el buscador secundario al controlador. También se muestran por pantalla las coordenadas obtenidas de la API de GoogleMaps y la distancia calculada por la función <i>distance</i> .
Entrada	Los datos que el usuario introduce en los dos formularios.
Salida esperada	Los datos imprimidos en el navegador coinciden con las introducidas por el usuario. Además las coordenadas y la distancia se corresponden con la dirección introducida.
Resultado	ÉXITO
Automatizada	No (Se utiliza Servlet de Prueba)

ID	Prueba 6
Descripción	Prueba para la detección de errores al realizar una búsqueda de inmuebles
Entrada	Se realiza una búsqueda de inmuebles mediante nuestra aplicación.
Salida esperada	El resultado que sale es parecido al que devuelve la propia página web de Idealista para la misma búsqueda (Es normal que no coincida del todo puesto que la página web de Idealista utiliza distinta metodología para determinar la zona de búsqueda.)
Resultado	ÉXITO
Automatizada	No (Se utiliza Servlet de Prueba)

ID	Prueba 7
<b>Descripción</b>	Prueba para la detección de errores al pasar al controlador correspondiente las coordenadas de un inmueble devuelto como resultado
<b>Entrada</b>	Las coordenadas geográficas del punto donde se ubica el inmueble seleccionado. Dichas coordenadas son proporcionadas por la API de Idealista.
<b>Salida esperada</b>	Las coordenadas que se han imprimido en el navegador son las mismas que las coordenadas del punto donde se ubica el inmueble seleccionado. Las coordenadas se pueden obtener consultando fuentes externas.
<b>Resultado</b>	<b>ÉXITO</b>
<b>Automatizada</b>	Sí

ID	Prueba 8
<b>Descripción</b>	Prueba para la detección de errores al obtener las coordenadas geográficas a partir de una dirección mediante la Geocoding API de GoogleMaps.
<b>Entrada</b>	Se realiza una búsqueda de lugares cercanos a un determinado inmueble mediante nuestra aplicación pulsando sobre el botón correspondiente.
<b>Salida esperada</b>	El resultado que sale corresponde a la realidad. Es decir, es cierto que los lugares devueltos están cerca del inmueble
<b>Resultado</b>	<b>ÉXITO</b>
<b>Automatizada</b>	No (Se utiliza Servlet de Prueba)

ID	<b>Prueba 9</b>
<b>Descripción</b>	Prueba para la detección de errores al pasar al controlador correspondiente los datos más relevantes de un inmueble devuelto como resultado
<b>Entrada</b>	Los datos más relevantes del inmueble (dirección, precio, tamaño, URL, etc.) Dichos datos son proporcionados por la API de Idealista.
<b>Salida esperada</b>	Los datos que se han imprimido en el navegador se corresponden con los datos del anuncio del inmueble en idealista.com
<b>Resultado</b>	<b>ÉXITO</b>
<b>Automatizada</b>	No (Se utiliza Servlet de Prueba)

## 6 Manual de usuario

### 6.1 Mashup

### 6.2 API REST

HTTP	URI	Descripción
GET	/inmuebles	Muestra todos los inmuebles de la API. Si se han encontrado correctamente aparecerá el código "200". Y si el recurso solicitado no ha sido encontrado, aparecerá el código "404".
GET	/ciudades	Muestra todas las ciudades de la API. Si se han encontrado correctamente aparecerá el código "200". Y si el recurso solicitado no ha sido encontrado, aparecerá el código "404".
POST	/inmuebles	Añade un nuevo inmueble con los datos que se pasan en el cuerpo de la petición en JSON (se genera automáticamente sin pasar ninguna id). Si el nombre del inmueble no es válido, devuelve un "400 Bad Request". Si se ha conseguido añadir, devuelve "201 Created" con la referencia a la URI y el contenido del inmueble.
POST	/ciudades	Añade una nueva ciudad con los datos que se pasan en el cuerpo de la petición en JSON. Los inmuebles de la ciudad no se pueden incluir aquí, para hacerlo se debe usar la operación POST específica para añadir un inmueble a una ciudad (la cual está definida más adelante). Las listas de los inmuebles de las ciudades nuevas están inicialmente vacías al añadirse. Si el nombre de la ciudad no es válido, devuelve un "400 Bad Request". Si se ha conseguido añadir, devuelve "201 Created" con la referencia a la URI y el contenido de la ciudad.

PUT	/inmuebles	Actualiza el inmueble con los datos que se pasan en el cuerpo de la petición en JSON (se debe incluir el id del inmueble). Si el inmueble no existe, devuelve un "404 Not Found". Si se actualiza correctamente, devuelve "204 No Content".
PUT	/ciudades	Actualiza la ciudad con los datos que se pasan en el cuerpo de la petición en JSON (se debe incluir el id de la ciudad). Si la ciudad no existe, devuelve un "404 Not Found". Si se intenta actualizar los inmuebles de la ciudad, devuelve un "400 Bad Request". Para ello se debe usar el recurso Inmueble definido anteriormente. Si se actualiza correctamente, devuelve "204 No Content".
GET	/inmuebles/{inmuebleId}	Muestra el inmueble con id = inmuebleId de la API. Si no existe devuelve un "404 Not Found".
GET	/ciudades/{ciudadId}	Devuelve la ciudad con id = ciudadId. Si no existe, devuelve un "404 Not Found".
DELETE	/inmuebles/{inmuebleId}	Elimina el inmueble con id = inmuebleId de la API. Si no existe, devuelve un "404 Not Found". Si se elimina correctamente, devuelve "204 No Content".
DELETE	/ciudades/{ciudadId}	Elimina la ciudad con id = ciudadId de la API. Si no existe, devuelve un "404 Not Found". Si se elimina correctamente, devuelve "204 No Content".
POST	/ciudades/{ciudadId}/{inmuebleId}	Añade un inmueble a una ciudad de la API. Si el nombre de la ciudad no es válido, devuelve un "400 Bad Request". Si se ha conseguido añadir, devuelve "201 Created".
DELETE	/ciudades/{ciudadId}/{inmuebleId}	Elimina el inmueble con id = inmuebleId de la ciudad con id = ciudadId. Si la ciudad o el inmueble no existe, devuelve un "404 Not Found". Si se elimina correctamente, devuelve "204 No Content".

## Referencias

- [1] *Balsamiq*. <http://balsamiq.com/>. Accedido en Enero 2014.
- [2] J. Webber, S. Parastatidis y I. Robinson. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media. 2010.