UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

MÁSTER EN DESARROLLO DE APLICACIONES SOBRE DISPOSITIVOS MÓVILES.

Título del proyecto:
Desarrollo de un buscador de inmuebles ubicuo y sensible al contexto para dispositivos Android.

Autor:
González Vallés, María Amparo.

Director:
Gironés, Jesús Tomás.

Contenido

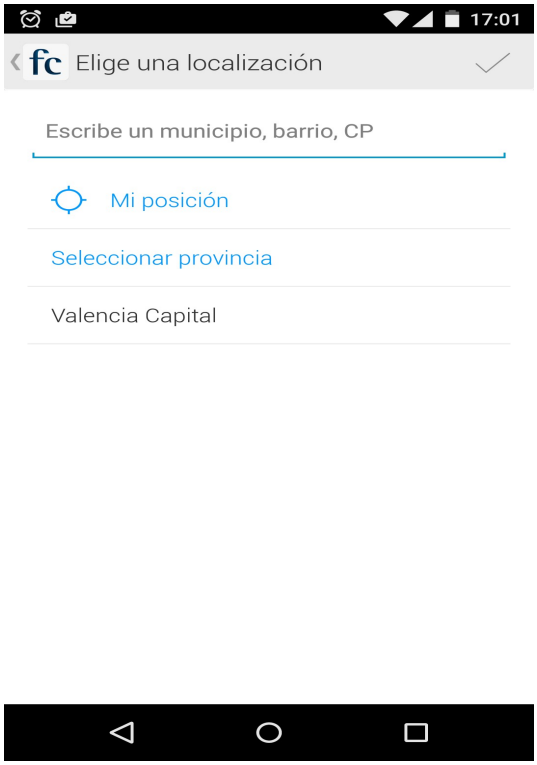
1 INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA

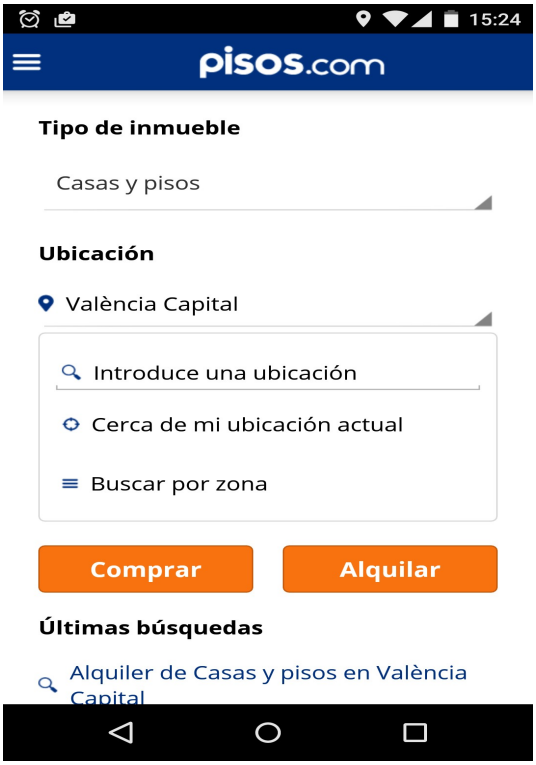
Nos encontramos en un momento en el que la tecnología se enfoca cada vez más hacía sistemas ubicuos que simplifiquen la vida de los usuarios. Sin embargo, tras un análisis del sector inmobiliario no he encontrado ninguna aplicación de búsqueda que aproveche la ubicación del dispositivo para dotar a la aplicación de esta característica.

Algunos buscadores hacen uso de la ubicación, pero ninguno de los analizados ofrece una experiencia de uso ubicua y mucho menos poseen la capacidad de adaptarse dependiendo de su contexto. En estas aplicaciones el usuario tiene que interactuar con el sistema cada vez que desee realizar una búsqueda.

A continuación se muestran dos ejemplos de buscadores que únicamente utilizan la ubicación del dispositivo para ofrecer al usuario la posibilidad de completar el campo de ubicación con la posición actual.



Ejemplo 1 buscador



Ejemplo 2 buscador

1.2 OBJETIVOS

El objetivo de este proyecto es la creación de una aplicación para dispositivos Android que ofrezca un sistema de búsqueda de inmuebles ubicuo con la capacidad de adaptarse dependiendo de su contexto. Para ofrecer esta adaptabilidad, el sistema deberá consultar periódicamente la ubicación del dispositivo y utilizar cada ubicación obtenida para enviar una petición a un servicio web que proporcione la información de los inmuebles que se encuentren disponibles en opción de compra o alquiler cerca de la ubicación especificada.

Los dispositivos para los que será desarrollado el sistema, será tanto dispositivos móviles como wearables. En este segundo las funcionalidades serán limitadas para ofrecer una buena experiencia de uso. La información deberá mantenerse sincronizada entre ambos dispositivos.

1.3 MOTIVACIÓN

La motivación que me ha llevado al desarrollo de este proyecto ha sido personal, promovida por una situación personal que se me planteó en un momento en el que buscaba piso. En ese momento tuve la necesidad de disponer de una herramienta de búsqueda que me permitiera saber si habían pisos disponibles en una zona determinada por la que pasaba en ese momento. Quería saberlo simplemente pasando por allí, sin tener que buscar de que zona se trataba. Un sistema que me ofreciera esa información realizando el mínimo esfuerzo posible.

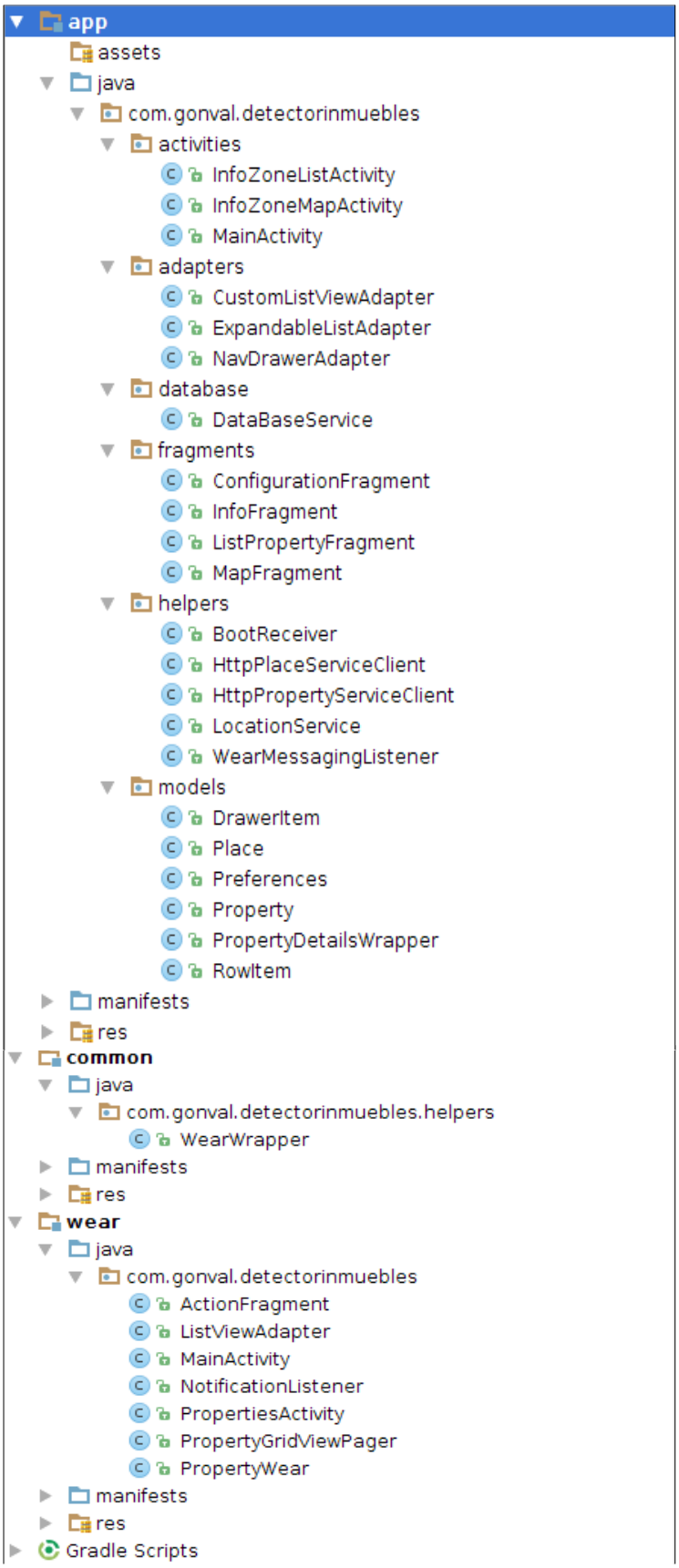
2 ARQUITECTURA

2.1 ESQUEMA DEL DISEÑO

ESTRUCTURA DEL PROYECTO

El sistema creado está compuesto por una aplicación para dispositivos móviles y otra para dispositivos wearables. Se ha optado por desarrollar una aplicación a pantalla completa para wear y descartar la comunicación mediante notificaciones porque se ha considerado interesante dotar a los dispositivos wear de ciertas funcionalidades presentes en la aplicación para móviles. Los datos que se manejan en el sistema son accesibles desde cualquiera de las dos aplicaciones, manteniendo además la información sincronizada en todo momento.

Para su implementación se ha creado un proyecto que sigue la estructura típica de un proyecto Android bajo el IDE de Android Studio. El proyecto está compuesto por tres módulos. El módulo app, que contiene los ficheros de la aplicación para móviles, el módulo wear, que contiene los ficheros de la aplicación para dispositivos wearables y el módulo common, donde se encuentran las clases comunes a ambas aplicaciones.


app
assets
java
com.gonval.detectorinmuebles
activities
InfoZoneListActivity
InfoZoneMapActivity
MainActivity
adapters
CustomListViewAdapter
ExpandableListAdapter
NavDrawerAdapter
database
DataBaseService
fragments
ConfigurationFragment
InfoFragment
ListPropertyFragment
MapFragment
helpers
BootReceiver
HttpPlaceServiceClient
HttpPropertyServiceClient
LocationService
WearMessagingListener
models
DrawerItem
Place
Preferences
Property
PropertyDetailsWrapper
RowItem
manifests
res
common
java
com.gonval.detectorinmuebles.helpers
WearWrapper
manifests
res
wear
java
com.gonval.detectorinmuebles
ActionFragment
ListViewAdapter
MainActivity
NotificationListener
PropertiesActivity
PropertyGridViewPager
PropertyWear
manifests
res
Gradle Scripts

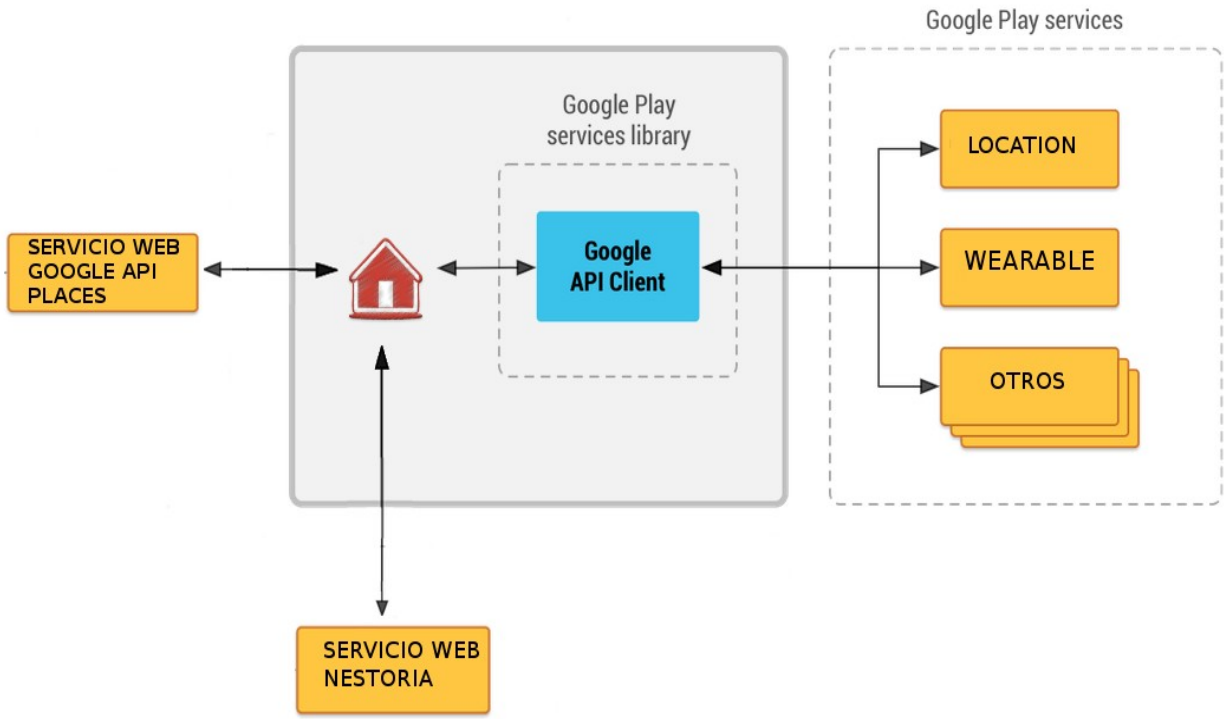
MODELO DEL SISTEMA

El sistema creado sigue una arquitectura cliente/servidor donde la aplicación, como cliente, interactúa con ciertos servicios para obtener la información necesaria para poder llevar a cabo sus funciones.

En primer lugar la aplicación obtiene su posición actual mediante la API de localización de Google, disponible en los servicios de Google Play. Tras obtener la localización del dispositivo, esta se envía junto a otros parámetros en una petición al servicio web ofrecido por el buscador vertical Nestoria, para obtener la información de los inmuebles.

Para obtener información complementaria sobre la zona donde se encuentra cada inmueble se realiza una petición al servicio web de la API de Google Places. Esta petición se realiza bajo demanda del usuario cada vez que desea consultar esta información.

Para la comunicación entre la aplicación para móviles y la aplicación para dispositivos wear también se ha utilizado la API de Capa de Datos Wearable (Wearable Data Layer API). Perteneciente además al API de servicios de Google Play.



MODELO DE SINCRONIZACIÓN

El modelo que sigue la sincronización de inmuebles es el de productor-consumidor. El teléfono actúa como productor generando los *DataMap* y añadiéndolos al Data Layer. Wear actúa como consumidor, ya que solo lee la información de los inmuebles y la representa en la actividad *PropertiesActivity*.

Esta clase *PropertiesActivity* será la encargada de mostrar unos inmuebles u otros dependiendo si el usuario la ha lanzado con el fin de visualizar las alertas recibidas, los inmuebles marcados como favoritos, o los nuevos inmuebles encontrados. En este último caso, tras mostrarlos, la actividad marcará los inmuebles como vistos. Este es el único caso en el que la aplicación de Android Wear modifica los datos sincronizados.

Lectura de inmuebles desde Android Wear

```
if ((mShowMode == SHOW_ALERTS) && property.isFavourite())
      continue;
else if ((mShowMode == SHOW_FAVOURITES) && !property.isFavourite())
      continue;
else if ((mShowMode == SHOW_NOTIFICATION) && !property.isNew())
      continue;

mPropertyAdapter.addProperty(property);

/* Si estamos viendo propiedades nuevas, las marcamos como vistas */
if (property.isNew()) {
  PutDataMapRequest dataMapReq =
      PutDataMapRequest.createFromDataMapItem(dataMapItem);
  dataMapReq.getDataMap().putBoolean("is_new", false);
  mWearWrapper.sendDataItem(dataMapReq.asPutDataRequest());
  property.setNew(false);
}
```

Si desde la aplicación de Wear se necesita realizar alguna operación sobre un inmueble (ya sea eliminarlo o marcarlo como favorito), se envía un mensaje al teléfono para que realice la operación. De esta manera se garantiza la coherencia de los datos entre ambos dispositivos, puesto que el origen de datos siempre es el mismo. Para el envío del mensaje se sigue utilizando la API de sincronización de datos con el fin de evitar añadir complejidad al utilizar una API distinta.

El teléfono recibe estos mensajes de operación en la clase *WearMessagingListener*, la cual extiende del servicio *WearableListenerService*. Cuando esta reciba un mensaje, se ejecutará el método *onDataChanged* el cual inspecciona la URI para averiguar que operación se debe aplicar y sobre que inmueble.

Para simplificar la utilización de la API de sincronización del Data Layer, se ha creado una clase compartida entre los proyectos del teléfono y de Wear llamada *WearWrapper*. Esta clase actúa como wrapper de los métodos más utilizados en ambos proyectos. Entre otras cosas, simplifica la interfaz y abstrae al desarrollador de tareas repetitivas como la obtención del id de nodo del dispositivo, o la comprobación del estado de la conexión a la capa de datos.

2.2 MODELO DE DATOS

La aplicación dispone de una pequeña base de datos SQLite encargada de almacenar la información de los inmuebles localizados y la configuración de búsqueda especificada por el usuario.

Para la manipulación de la base de datos se ha extendido la clase abstracta *SqliteOpenHelper* que nos permite la creación automática de la base de datos y el manejo de futuras versiones.

La base de datos está compuesta por dos tablas, preferences y property. En la tabla preference solamente se almacena un registro, correspondiente a la configuración seleccionada por el usuario. Cuando el usuario modifica los patrones de búsqueda, el registro almacenado se modifica. En la tabla property se almacena todos los inmuebles detectados por el sistema.

preference	
id	integer PK
service	integer
listing_type	text
property_type	text
radio	integer
size_min	integer
size_max	integer
bathroom_min	integer
bathroom_max	integer
bedroom_min	integer
bedroom_max	integer
price_min	integer
price_max	integer
air_conditioning	text
heating	text
storage	text
garage	text
furnished	text
elevator	text
terrace	text

property		
code	text	PK
url	text	
price	integer	
type	text	
size	integer	
bathrooms	integer	
rooms	integer	
thumbnail	text	
image	text	
operation	text	
status	integer	
summary	text	
title	text	
longitude	text	
latitude	text	

2.3 SERVICIOS WEB

LOCATION API

Uno de los puntos fundamentales de la aplicación, es poder obtener periódicamente de ubicación del dispositivo tanto en espacios abiertos como cerrados. Para este fin se ha utilizado la API de localización de los *Servicios de Google Play*. El motivo por el cual se ha optado por este servicio en lugar de la API de localización de Android, es por su buen rendimiento en interiores, factor muy importante para el correcto funcionamiento de la aplicación.

Para utilizar este servicio, se necesita hacer uso de los servicios de Google Play Services, que contiene los servicios de Google. Para ello se ha descargado y añadido al proyecto la librería Google Play Services. También se ha tenido que añadir permisos de localización. Android ofrece dos tipos de permisos de localización, *ACCESS_COARSE_LOCATION* y *ACCESS_FINE_LOCATION*. El uso de uno u otro depende de las necesidades de precisión. Para la implementación de este

proyecto se ha optado por *ACCESS_FINE_LOCATION* debido a que incluye permisos tanto para GPS como para la localización por red.

Para su desarrollo se ha implementado el servicio *LocationService*, que se inicia cuando el usuario activa el servicio de búsqueda. Para asegurar el continuo funcionamiento del servicio, se ha registrado un receptor de anuncios (*BroadcastReceiver*) que recibe y reacciona cuando el sistema lanza un anuncio broadcast *BOOT_COMPLETED.* En ese momento, se hace una consulta a la base de datos, donde se guarda el estado del servicio cada vez que el usuario lo inicia o lo detiene. Si en la base de datos su estado es activo, se vuelve a arrancar. De este modo nos aseguramos de que si el dispositivo se apaga estando el servicio activado, vuelva a iniciarse cuando se arranque de nuevo. Para su uso se ha añadido el permiso *RECEIVE_BOOT_COMPLETED.*

La clase *LocationService* implementa las siguientes interfaces:

- *GoogleApiClient.ConnectionCallbacks.*
- *GoogleApiClient.OnConnectionFailedListener.*
- *LocationListener*.

Para poder conectar con la API, se crea una instancia del cliente de la API de *Google Play Services* y se establece la conexión.

Conexión

```
private GoogleApiClient mGoogleApiClient;
mGoogleApiClient = new GoogleApiClient.Builder(context)
            .addApi(LocationServices.API)
            .addApiIfAvailable(Wearable.API)
            .addConnectionCallbacks(this)
            .build();

 mGoogleApiClient.connect();
```

Una vez establecida la conexión, en el método callback *onConnected()* se realizan las acciones para obtener la localización. Primero se obtiene la última posición conocida haciendo una llamada al método getLastLocation() y después se realiza una llamada al método r*equestLocationUpdates*() para obtener periódicamente la posición actualizada.

Obtener la localización

```
@Override
public void onConnected(Bundle arg0) {

      Location currentLocation =
          LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);

        (…)

      LocationRequest locationRequest = LocationRequest.create()
          .setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY)
          .setInterval(30000);

      LocationServices.FusedLocationApi.requestLocationUpdates(
           mGoogleApiClient, locationRequest, this);
  }
```

En la llamada al método *requestLocationUpdates* se pasa un objeto *LocationRequest* que contiene los parámetros de la petición. Estos parámetros determinan los niveles de precisión requeridos.
Cada actualización de la localización se recibe en el método callback *onLocationChanged()*.

Tanto en el callback *onConnected()*, donde obtenemos la última posición conocida, como en el callback *onLocationChanged()*, donde recibimos periódicamente la localización actualizada, utilizamos el objeto Location obtenido, para realizar una llamada al servicio web que nos proporciona la información de los inmuebles. En esta petición enviaremos la latitud y la longitud extraída del objeto Location junto al resto de parámetros de búsqueda almacenados en la base de datos.

SERVICIO WEB NESTORIA

Para cumplir con las funcionalidades del sistema, se ha necesitado utilizar un servicio web que enviándole la localización del dispositivo, proporcionase la información necesaria de los inmuebles. El servicio web utilizado para este fin, ha sido el que ofrece el buscador vertical Nestoria.

Como se ha comentado en el punto anterior, cada vez que se obtiene la localización del dispositivo, bien sea la última conocida o al recibir la localización actualizada, se hace una petición al servicio web de Nestoria. Para realizar la petición al servicio se ha implementado una clase que extiende la clase abstracta AsyncTask. En esta clase se consultan las preferencias de búsqueda del usuario almacenadas en base de datos y junto a la localización obtenida, se hace una llamada al método sendRequest de la clase implementada *HttpPropertyServiceClient,* donde se realiza una petición http al servicio web.

En la petición realizada se especifican los parámetros de búsqueda, la localización del dispositivo y el radio de búsqueda. Si la petición se realiza con éxito se obtiene una respuesta en formato JSON con la información de los inmuebles disponibles que cumplan con los parámetros enviados. La información obtenida se almacena en un ArrayList de objetos tipo *Property* y se devuelve a la tarea asíncrona.

Para no enviar por duplicado la información al usuario, cuando la tarea asíncrona recibe la lista de los inmuebles, se comprueba que el código que los identifica no se encuentre ya almacenado en la base de datos. Los inmuebles cuyo código se encuentren almacenados son descartados y los que no guardados en base de datos. Si no se descartan todos los inmuebles recibidos, se lanza una notificación para alertar al usuario de la existencia de nuevos inmuebles encontrados.

Conexión Http

```
public static String getJSON(String address){

    StringBuilder builder = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(URI);

    try{
        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
```

```
        int statusCode = statusLine.getStatusCode();
        if(statusCode == 200){
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader =
                new BufferedReader(new InputStreamReader(content));
            String line;
            while((line = reader.readLine()) != null){
                builder.append(line);
            }
        }
    }catch(ClientProtocolException e){
        e.printStackTrace();
    } catch (IOException e){
        e.printStackTrace();
    }
    return builder.toString();
}
```

A continuación se detalla un ejemplo de petición realizada en la aplicación junto a un inmueble extraído de su correspondiente respuesta.

Petición Nestoria

---

http://api.nestoria.es/api

?action	= search_listings
&encoding	= json
&pretty	= 1
&centre_point	= 39.4773833,-0.3848674,1km
&keywords	= terraza,garaje
&bedroom_max	= 4
&listing_type	= rent
&bedroom_min	= 2
&price_min	= 400
&price_max	= 600

Inmueble respuesta JSON

```
{
        "auction_date" : null,
        "bathroom_number" : 2,
        "bedroom_number" : 4,
        "car_spaces" : null,
        "commission" : 0,
        "construction_year" : 0,
        "datasource_name" : "Yaencontre",
        "guid" : "g1-SN40DMxATM0k",
        "img_height" : 120,
        "img_url" :
                "http://3.l.es.nestoria.nestimg.com/lis/e/2/e/4b374180f856b4d899bf
                03a85348c4cf33aa4.2.jpg",
        "img_width" : 160,
        "keywords" : "aire acondicionado, alarma, a estrenar, terraza, portero
                automático, ascensor, conserje, suelo de parquet",
        "lister_name" : "Valencia House",
        "lister_url" : "http://rd.nestoria.es/rd?
                itype=2&l=VbeLaTvdcveTeWPvm&s=&url=2-
```

ePz73aTBU4LVGFByDI474Ner8aIMqOqmkHO6u5ws2qDdmu6ceNje4L
e1MAfBMUMMxpoFo0SuDqCXu_ZIEd28Ci9lpRtbgRHBYlXVUF5-
L_2uj1ayX9zSaUApK-
eBQmYgiQbqpQUMwEXhb28UsjaTDjtq7wyWsOP337nWk-
QuUCjUhitmByfIdWByKE54PlKPAbzeSjKJS-XV6zSG_thvzXexwt-
Dc1ZPJ4rTZOJF8ZCVm6zQaDNJ_f-
poOQrTgACcrv5qlXXXmd7kfHf8QCMGxOcCL8J2V9q-Gyv85Ms-
G7GdmXkN0wyQQFnQPqbWM-al3xf3JMDkTPUIiNZOhbL&v=2",
"**listing_type**" : "let",
"**price**" : 1400,
"**price_currency**" : "EUR",
"**price_formatted**" : "1.400 EUR",
"**price_high**" : 1400,
"**price_low**" : 1400,
"**price_type**" : "monthly",
"**property_type**" : "flat",
"**room_number**" : 4,
"**size**" : 147,
"**size_type**" : "gross",
"**size_unit**" : "m2",
"**summary**" : "Vivienda de lujo, de obra nueva, a estrenar, y en alquiler.
Situada en...",
"**thumb_height**" : 60,
"**thumb_url**" :
"http://2.l.es.nestoria.nestimg.com/lis/e/2/e/4b374180f856b4d899bf
03a85348c4cf33aa4.1.jpg",
"**thumb_width**" : 60,
"**title**" : "Ciutat Vella, Valencia, València",
"**updated_in_days**" : 12.5,
"**updated_in_days_formatted**" : "hace 1 semana"
}

De entre las opciones disponibles para especificar la localización se ha utilizado el parámetro centre_point. Según la documentación estos son todos los argumentos de localización disponibles.

place_name	Cualquier palabra que se introduciría en el campo de búsqueda de Nestoria: código postal, estación de metro, nombre de una ciudade, etc. e.j. "Valencia", "46008"
south_west **north_east**	Un cuadro de búsqueda delimitado por dos localizaciones especificadas mediante latitude,longitude,latitude2,longitude2 e.j. &south_west=51.684183,-.431481 &north_east=51.85415,-3.077859
centre_point	Latitud y Longitud. Con un radio por defecto de 2km. e.j. &centre_point=51.684183,-3.431481
radius	Latitud, longitud y radio especificado en kilometros (km) o millas (mi). El parámetro 'km' o 'mi' debe ser especificado e.j. &centre_point=51.684183,-3.431481,10km

Estos son los parámetros disponibles para especificar los filtros en la petición:

listing_type	'buy' (por defecto), 'rent' o 'share'
price_min **price_max**	Números. En lugar de setear '0' y '999999999' puede especificarse 'min' y 'max'. e.j.&price_min=min&price_max=200000
bedroom_min **bedroom_max**	Números. En lugar de setear '0' y '999999999' puede especificarse 'min' y 'max'. e.j.&bedroom_min=3&bedroom_max=3
room_min **room_max**	Números. En lugar de setear '0' y '999999999' puede especificarse 'min' y 'max'. e.j.&room_min=3&room_max=3
bathroom_min **bathroom_ma**	Números. 0 para 'studio apartment'. En lugar de setear '0' y '999999999' puede especificarse 'min' y 'max'.
size_min **size_max** **size_type**	Números. En lugar de setear '0' y '999999999' puede especificarse 'min' y 'max'.
keywords **keywords_exclude**	Una lista de palabras clave separadas por comas. Puede especificarse el filtro positivo, negativo o ambos. &keywords=jardin&keywords_exclude=terraza, garaje
has_photo	Setear a 1 para obtener solo inmuebles con foto.
updated_min	UNIX timestamp (UTC) e.j.&updated_min=1220155200

Lista de Keywords disponibles para España:

a-estrenar	a-reformar	adosado
aire-acondicionado	alarma	amueblado
apartamento	armarios	ascensor
atico	balcon	barbacoa
buhardilla	bungalow	calefac-central
chalet	chimenea	cocina
cocina-equipada	despensa	doble-cristal
doble-garaje	duplex	en-buen-estado
estudio	ex-vpo	garaje
gimnasio	holiday-home	jacuzzi
jardin	jardin-comunitario	lavadero
lavavajillas	lavavajillas	opcion-a-compra
parcela-de-garaje	piscina	piscina-comunitaria
piscina-individual	piso-compartido	pista-de-deportes
planta-baja	playa-cercana	plaza-de-parking
portero	portero-automatico	renovado
sauna	semi-amueblado	sin-ambueblar
sin-ascensor	spa	suelo-de-madera
terraza	terreno	trastero
video	villa	zona-recreativa

SERVICIO WEB PLACES API

Otra de las funcionalidades disponibles en la aplicación consiste en poder consultar los establecimientos disponibles cerca del inmueble. Para ello se ha utilizado el servicio web de Google de la API Places que nos proporciona toda la información necesaria sobre los establecimientos de la zona solicitada.

A continuación se detalla un ejemplo de petición.

Petición servicio web API Places

https://maps.googleapis.com/maps/api/place/nearbysearch/json
?radius = 500
&key = (clave generada desde la consola de Google)
&location = 39.4773833,-0.3848674
&types = URLEncoder.encode("school|grocery_or_supermarket|hospital|doctor|gym|park", "UTF-8")

URLEncoder es una clase que se utiliza para codificar un String utilizando el formato requerido por *application/x-www-form-urlencoded.* Todos los caracteres excepto ('a'..z', 'A'..'Z') y números ('0'..'9') y caracteres '.', '-', '*', '_' se convierten a un valor hexadecimal antepuesto por '%'. Además los espacios son sustituidos por '+'.

El parámetro types, nos permite filtrar los lugares por su tipo. Se ha hecho una selección de los que se han considerado más interesantes para el usuario.

Estos son todos los tipos disponibles.

accounting	airport	amusement_park	aquarium
art_gallery	atm	bakery	bank
bar	beauty_salon	bicycle_store	book_store
bowling_alley	bus_station	cafe	campground
car_dealer	car_rental	car_repair	car_wash
casino	cemetery	church	city_hall
clothing_store	convenience_store	courthouse	dentist
department_store	doctor	electrician	electronics_store
embassy	establishment	finance	fire_station
florist	food	funeral_home	furniture_store
gas_station	general_contractor	grocery_or_supermarket	gym
hair_care	hardware_store	health	hindu_temple
home_goods_store	hospital	insurance_agency	jewelry_store
laundry	lawyer	library	liquor_store
local_government_office	locksmith	lodging	meal_delivery

meal_takeaway	mosque	movie_rental	movie_theater
moving_company	museum	night_club	painter
park	parking	pet_store	pharmacy
physiotherapist	place_of_worship	plumber	police
post_office	real_estate_agency	restaurant	roofing_contractor
rv_park	school	shoe_store	shopping_mall
spa	stadium	storage	store
subway_station	synagogue	taxi_stand	train_station
travel_agency	university	veterinary_care	zoo

WEAR API

Para la sincronización de información de inmuebles entre el teléfono y el dispositivo Android Wear se ha optado por utilizar la API de sincronización de datos incluido en el Data Layer de Google Play Services. El uso de esta API simplifica el proceso de sincronización, ya que permite el empaquetamiento de varios datos relacionados en un único mensaje.

Cada inmueble sincronizado esta representado en un *DataMap* cuyo URI esta definido siguiendo el siguiente esquema:

wear://<id del nodo>/property/<codigo del inmueble>

A su vez, cada *DataMap* contiene la siguiente información del inmueble:

Clave	Tipo	Descripción
Code	String	Código del inmueble
Features	String	Características clave del inmueble
Summary	String	Resumen del anuncio
Url	String	Url del portal anunciante
Photo	Asset	Foto del anuncio
Is_new	Boolean	True si no ha sido visualizado todavía por el usuario
Is_favourite	Boolean	True si ha sido marcado como favorito por el usuario

Como se especificó anteriormente, un inmueble es insertado en la base de datos si y solo si no ha sido insertado anteriormente. Este mismo criterio es el que se sigue para sincronizar los inmuebles con Android Wear y por tanto, es la misma clase (*LocationService*) la encargada de esta tarea.

Sincronización de inmuebles con Android Wear

```
if(!exists){
  if(property.getLatitude()!= null && property.getLongitude()!= null
  {
    mDbService.insertProperty(property);
    if(mGoogleApiClient.hasConnectedApi(Wearable.API))
      Wearable.DataApi.putDataItem(
                        mGoogleApiClient,
                        property.asPutDataRequest(mContext));
  }
}
```

Con el fin de simplificar la transmisión de inmuebles entre las distintas capas, la clase *Property* implementa un método *asPutDataRequest* que devuelve un objeto *PutDataRequest* contenedor del *DataMap* del inmueble.

Una vez se han sincronizado los inmuebles en el Data Layer, se notifica a Android Wear. Para esto, se ha optado por utilizar un mensaje de la misma API de sincronización en lugar del mecanismo de notificación habitual. Esto posibilita el abrir una actividad concreta con unos extras concretos desde la propia notificación. Android Wear recibirá el mensaje en la clase *NotificationListener* que extiende al servicio *WearableListenerService.* Una vez recibido el mensaje, se generará una notificación local a Wear junto con un Intent asociado a la apertura de la actividad que muestra los inmuebles.

TINYURL

Otra de las funcionalidades disponibles en la aplicación es la posibilidad de compartir la url del inmueble a través de otras aplicaciones.

Debido a que las url proporcionadas por Nestoria son excesivamente largas se ha optado por utilizar el servicio web TinyURL para acortarlas. Este servicio proporciona una url mucho más corta que redirige a la url original.

Para llevar a cabo la solicitud se hecho una petición HTTP de tipo GET con el siguiente formato:

Petición TinyURL

http://tinyurl.com/api-create.php
?url = URLEncoder.encode(longUrl, "utf-8")

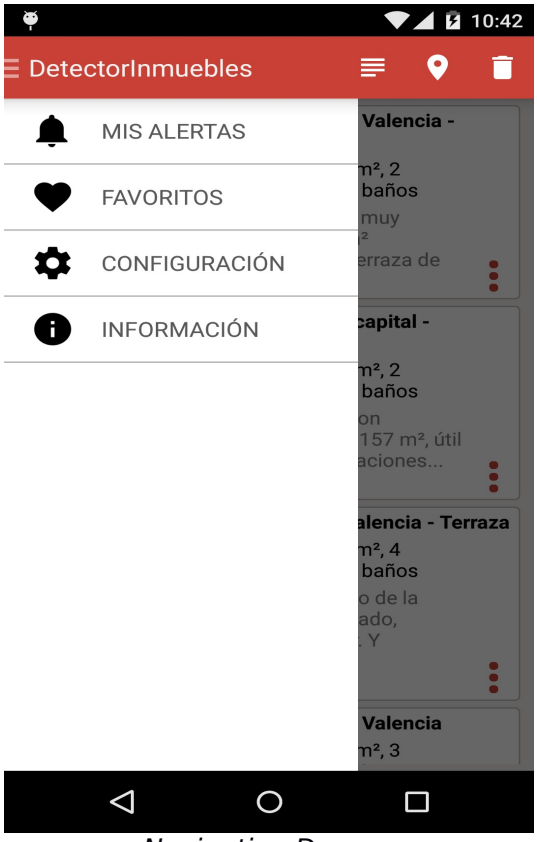
Donde longUrl se corresponde con la url original proporcionada por Nestoria.

Si la solicitud se ha realizado con éxito, se utiliza la nueva url obtenida, sino se utiliza la original.

2.4 VISTAS TELÉFONO

BARRA DE NAVEGACIÓN

Para la navegación entre las diferentes pantallas se ha optado por la utilización de la barra de navegación conocida como *Navigation Drawer*. Esta puede desplegarse de dos formas diferentes, mediante la pulsación del icono de la barra de acciones o desplazándola desde el lado izquierdo de la pantalla.



Navigation Drawer

PANTALLA MIS ALERTAS

Muestra todas los inmuebles detectados. La información puede visualizarse mediante un listado o sobre un mapa. Para cambiar entre ambos modos de visualización se dispone de un icono en la barra de acciones.

En el modo listado, pulsando sobre el inmueble se redirige al usuario a un navegador web donde se carga la url del portal inmobiliario donde se encuentra anunciado el inmueble. La opción de poder ampliar la información consultando el anuncio original se ha considerado importante debido a que la respuesta que se recibe del servicio web no proporciona toda la información del anuncio. Por ejemplo solo proporciona la url de una de las imágenes y se ha considerado importante que el usuario pueda tener acceso al resto.

Cada inmueble dispone de un menú contextual con un listado de opciones.

- **Ver Anuncio:** Abre un navegador web con la url del portal inmobiliario donde se encuentra anunciado el inmueble.

- **Info Zona:** Muestra establecimientos, de posible interés para el usuario, que se encuentren cercanos al inmueble. Con esta opción el usuario puede saber si en la zona del inmueble dispondría de supermercados, colegios, hospitales, etc y a que distancia se encuentran. Estos pueden visualizarse en un mapa o en un listado agrupados por su tipo. En el modo listado se dispone de la distancia en metros al inmueble.

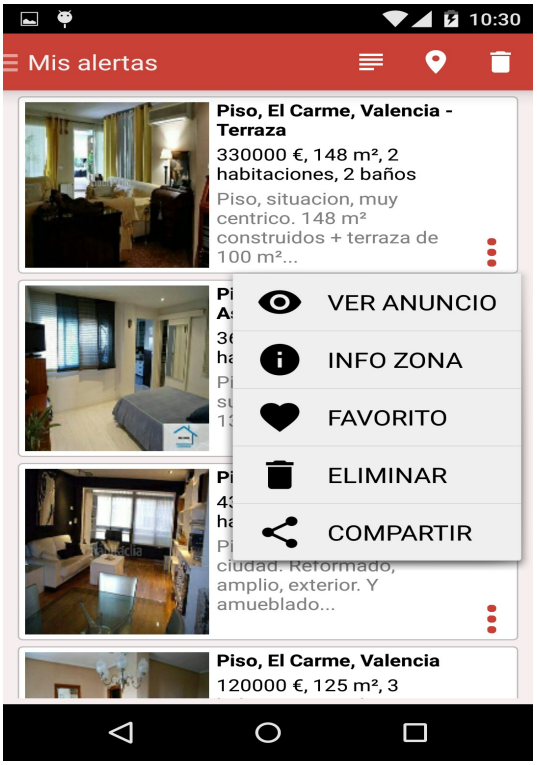
- **Favorito:** Marca el inmueble como Favorito. El inmueble deja de aparecer en este listado y pasa a estar disponible desde la pantalla de Favoritos.

- **Eliminar:** Elimina el inmueble.

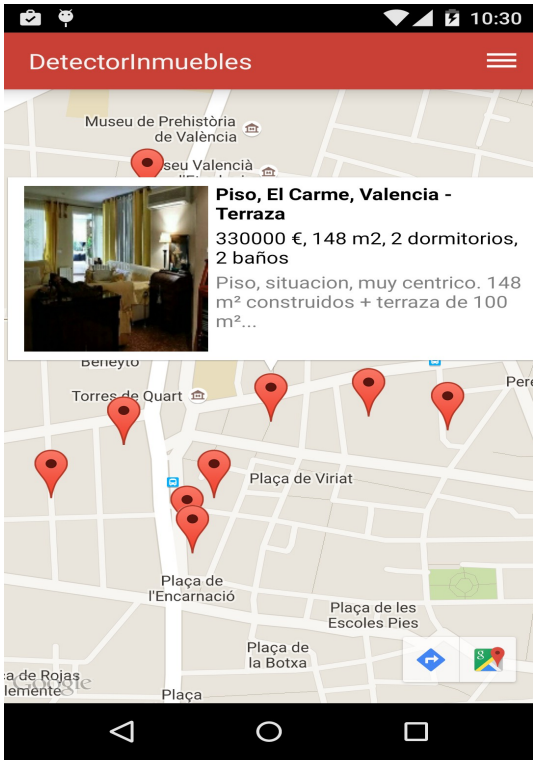
- **Compartir:** Permite compartir la url del anuncio mediante aplicaciones externas.

Además, en este modo de visualización se puede ordenar los inmuebles por su precio. Tanto de menor a mayor como de mayor a menor.
En el modo mapa, pulsando sobre un marcador se muestra la información del inmueble y pulsando sobre la información del inmueble se abre un navegador con la url del anuncio.

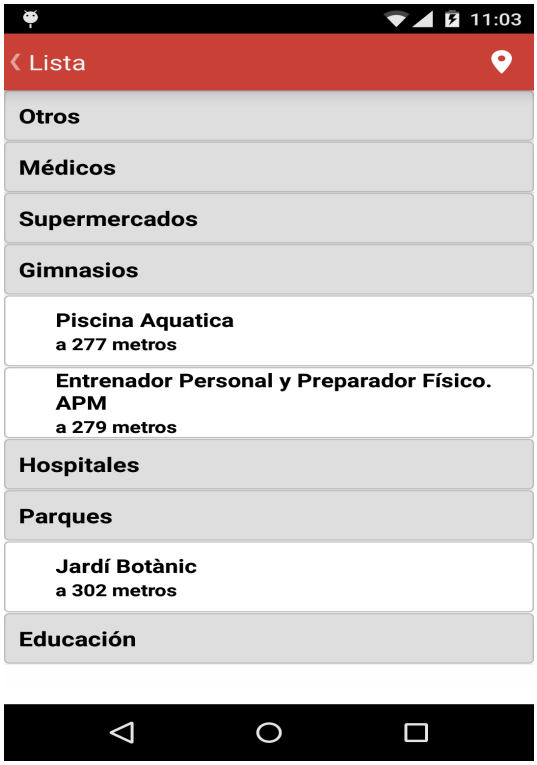
Como se verá más adelante, la estructura de las pantallas *Mis alertas*, *Favoritos* y *Nuevas alertas*, es la misma. Las tres disponen de los modos lista y mapa para visualizar la información de los pisos, disponen de las mismas opciones en la barra de acciones (opción de ordenación del listado por precio, opción para eliminar todos los inmuebles de la lista y la opción de cambiar de modo de visualización) y del mismo menú contextual para cada inmueble. La única diferencia es que a la hora de mostrar los inmuebles favoritos, el menú contextual de estos, no disponen de la opción de favorito.



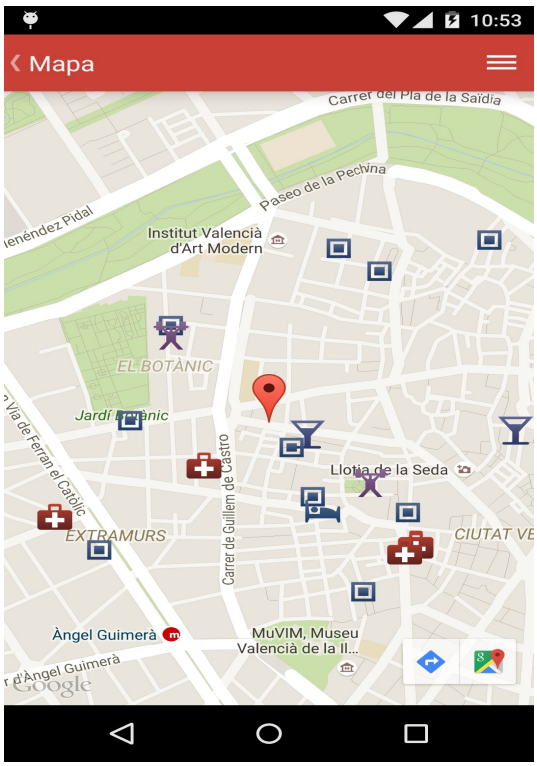
Mis alertas (Lista)



Mis alertas (Mapa)



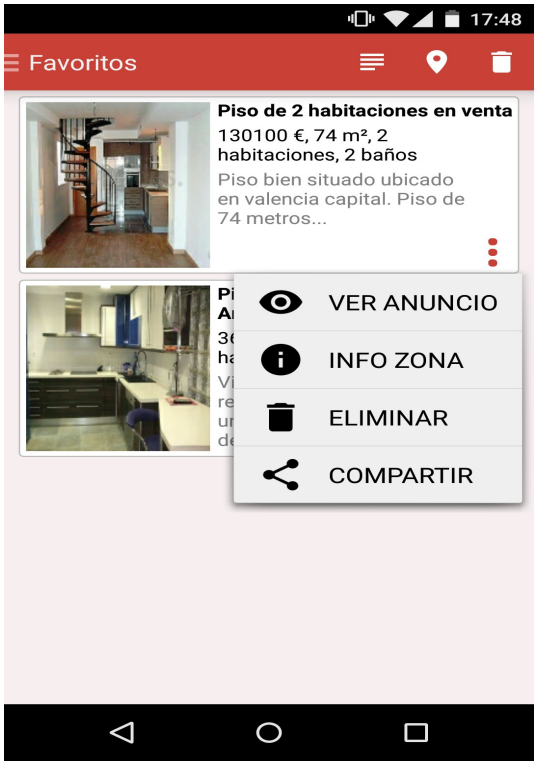
Info Zona (Lista)



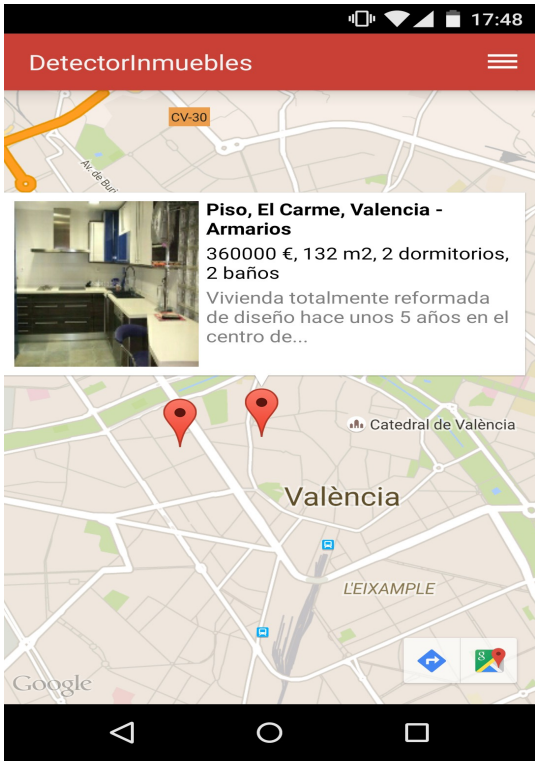
Info Zona (Mapa)

PANTALLA FAVORITOS

Muestra los inmuebles marcados como favoritos. Como ya se ha comentado, de igual modo que en la pantalla *Mis Alertas,* la información puede visualizarse en un listado o en un mapa y cada inmueble dispone de un menú contextual como el que se muestra en la pantalla Mis Alertas, con la excepción de la opción de añadir a Favoritos.



Favoritos (Lista)

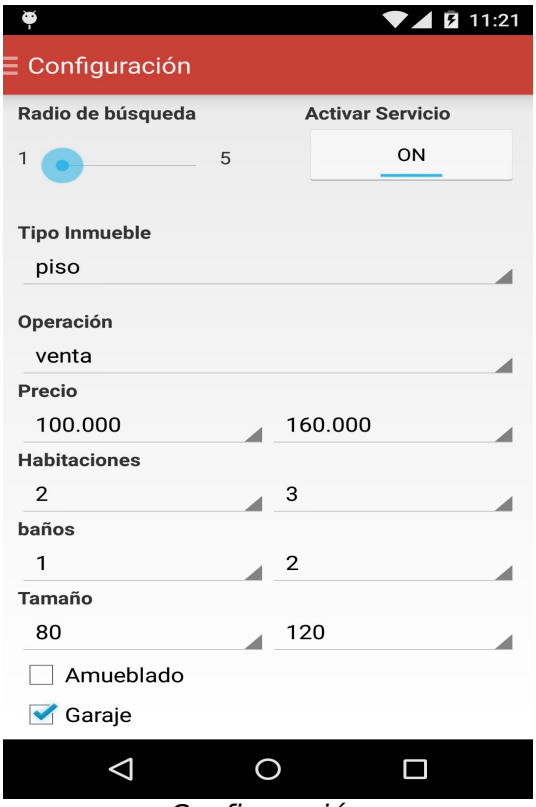


Favoritos (Mapa)

PANTALLA CONFIGURACIÓN

En esta pantalla el usuario pueden especificar las preferencias de búsqueda y activar o desactivar el servicio mediante un toggleButton. Los parámetros de configuración disponibles son los siguientes:

- **Radio de búsqueda:** Radio de búsqueda en kilómetros.
- **Tipo de inmueble:** Piso, ático, chalet, dúplex, estudio.
- **Operación:** Alquiler o venta**.**
- **Precio (mínimo y máximo):** Valor especificado en euros.
- **Habitaciones (mínimo y máximo).**
- **Baños (mínimo y máximo).**
- **Tamaño (mínimo y máximo).** Valor especificado en m².
- **Otras características**: Amueblado, garaje, aire acondicionado, calefacción, ascensor, terraza y trastero.



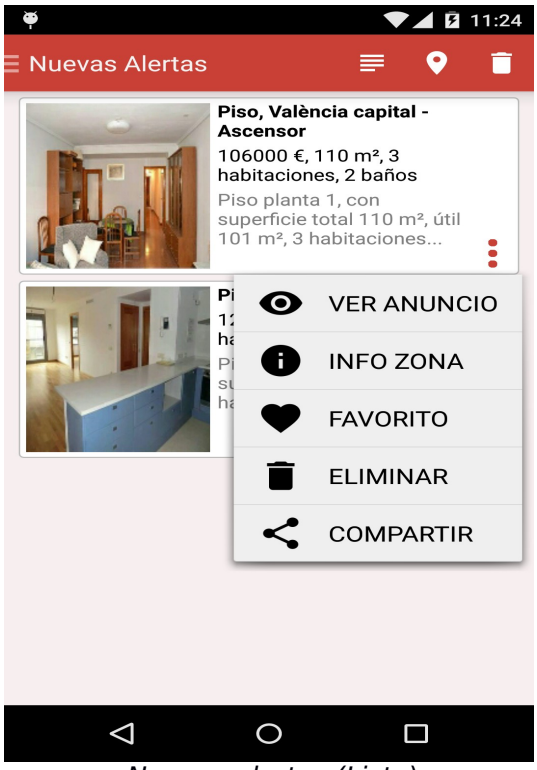
Configuración

NOTIFICACIÓN INMUEBLES

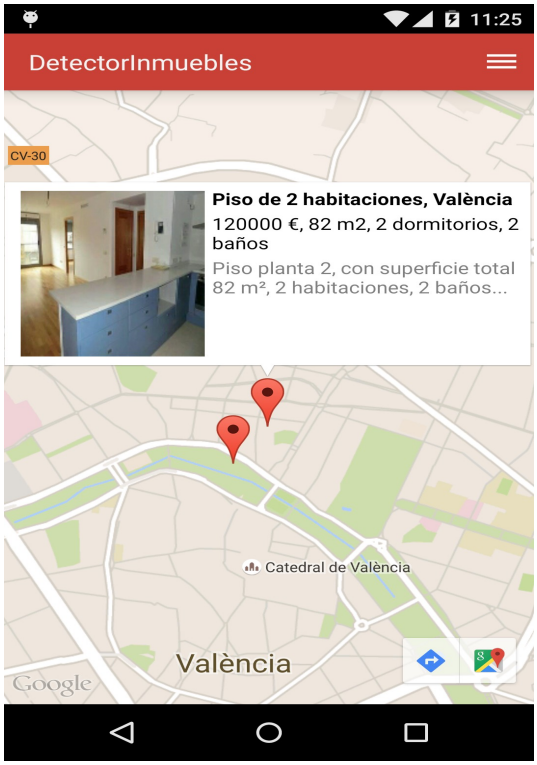
Tras cada petición al servicio web de Nestoria, si en la respuesta se reciben inmuebles, se muestra al usuario una notificación para informarle de las nuevas alertas disponibles. Al pulsar sobre esta notificación, se abre la aplicación en la pantalla *Nuevas alertas,* donde únicamente se muestran las últimas alertas recibidas.

PANTALLA NUEVAS ALERTAS

Muestra los últimos inmuebles localizados. La información también puede visualizarse en modo lista o en modo mapa. Además cada inmueble dispone de un menú contextual como el que se muestra en la pantalla Mis Alertas.



Nuevas alertas (Lista)



Nuevas alertas (Mapa)

2.5 VISTAS WEAR

Para presentar la información de los inmuebles en Android Wear se ha optado por utilizar un selector 2D en el que cada fila esta dedicada a mostrar la información y las operaciones posibles de un inmueble. Para la gestión de las tarjetas se ha implementado la clase *PropertyAdapter*, que extiende de *FragmentGridPagerAdapter*.

Esta clase crea y muestra las tarjetas de información correctas dependiendo de la lista de inmuebles a visualizar (mis alertas, notificaciones o favoritos), así como las tarjetas de operaciones.

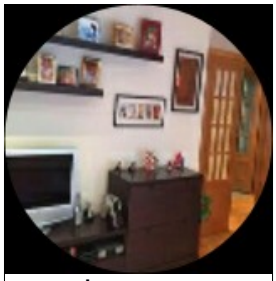
MENÚ PRINCIPAL

Menú principal

El menú principal es la primera vista que se le presenta al usuario. Desde aquí, el usuario puede elegir entre visualizar los inmuebles detectados o los favoritos. Para ambas opciones, las vistas de inmueble disponibles son las mismas, con la diferencia de que el menú de alertas de inmuebles además incluye la operación marcar como favorito.

VISTAS INMUEBLES

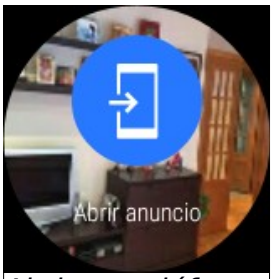
Las vistas de inmueble muestran la información de este así como las operaciones disponibles.



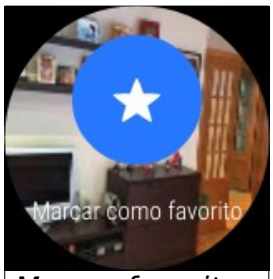
Imagen

Características

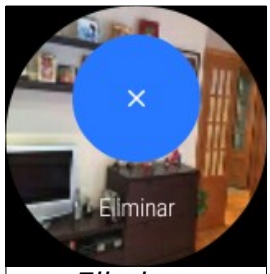
Descripción



Abrir en teléfono



Marcar favorito



Eliminar

- **Imagen**: Muestra la fotografía del inmueble.
- **Descripción**: Muestra una tarjeta con una breve descripción del inmueble.
- **Características**: Muestra una tarjeta con las características principales del inmueble.
- **Abrir en teléfono:** Botón que provoca la apertura de la web del portal anunciante del inmueble en el teléfono.
- **Marcar como favorito:** Botón que marca el inmueble como favorito. A partir de este momento, el inmueble sólo aparecerá en la lista de inmuebles favoritos tanto en Wear como en el teléfono.
- **Eliminar inmueble:** Botón que elimina el inmueble tanto en Wear como en el teléfono.

3 CONCLUSIONES

Tras finalizar el proyecto, puedo decir que los objetivos iniciales han sido cumplidos. Si bien es cierto que durante el desarrollo del proyecto han ido surgiendo ideas que han sido integradas en el mismo, hay otra serie de ideas que se han descartado por no disponer del tiempo suficiente para implementarlas.

Debido al hecho de que el desarrollo de este proyecto sigue una motivación personal, termino esta fase de desarrollo con las ganas e ilusión de seguir trabajando en él y con tiempo poder llevar a cabo todas las ideas que me fueron surgiendo.

A continuación se destacan algunas de ellas:

- Al borrar un inmueble preguntar al usuario si desea no volver a recibir esa alerta.
- Opción de avisar si baja de precio.
- Widget ON/OFF servicio.