Junio 2018

CREACIÓN AUTMÁTICA DE UN MAPA CON INFORMACIÓN CATASTRAL

DESARROLLO DE APLICACIONES SIG

Carlos Bayarri Cebrecos
Raquel Luján García-Muñoz

Índice

1. Introducción

La asignatura de desarrollo de aplicaciones SIG, ubicada en el primer año del máster en Geomática y Topografía, nos proporciona herramientas para programar aplicaciones SIG, dándonos la oportunidad de pasar de ser usuarios de sistemas de información geográfica a desarrolladores de los mismos.

Con este proyecto se pretende aplicar los conocimientos adquiridos a lo largo del curso, mediante el desarrollo de una herramienta SIG para cualquier aplicación.

En nuestro caso, el proyecto que se ha llevado a cabo es el siguiente: desarrollo de un pluging para Qgis que nos permita realizar de manera automática un mapa con información catastral de una parcela del municipio de Valencia. Además, nos permitirá seleccionar puntos de interés para mostrar en el mismo.

1.1. Justificación

El mundo de la programación SIG abre un sinfín de posibilidades, pues son infinitas las posibles aplicaciones.

De las muchas opciones que se podrían haber desarrollado, hemos escogido la realización automática de un mapa con información catastral mediante un plugins de Qgis.

La obtención de mapas con leyenda, escala y todos los elementos necesarios para que sea correcto desde el punto de vista cartográfico es una de las especialidades de nuestra titulación, por tanto, consideramos que poner esto a disposición de distintos usuarios es importante, y por eso el objetivo central de nuestro proyecto es la elaboración de un mapa.

Por otro lado, el Catastro está cobrando cada vez más importancia, ya que se está evolucionando hacia un catastro en el que la información espacial sea fundamental. Además, acceder a la información de catastro no siempre es sencillo, por lo que se ha decidido intentar facilitar esta labor al usuario.

Además, pesando en las posibles necesidades de los usuarios, se ha decidido añadir información sobre puntos de interés cercanos, ya que esta información puede ser de gran utilidad para diversas aplicaciones, por ejemplo, si un usuario quiere obtener información sobre distintas parcelas para ayudarle a decidir cuál comprar.

En cuanto al desarrollo técnico del proyecto, se ha optado por utilizar Qgis. El motivo principal es que se trata de un software libre, por lo que es mucho más sencillo para el usuario poder acceder a él.

Además de esto, se ha decidido desatrollar un pluging para facilitar la interacción con el usuario. De este modo el usuario no tiene por qué tener nociones de programación y puede utilizarse en cualquier ordenador simplemente instalando este pluging.

2. Objetivo

El objetivo de este proyecto es desarrollar una herramienta que nos permita realizar un mapa de manera rápida y sencilla con información catastral de una parcela urbana de Valencia.

Para ello hemos elegido trabajar con Qgis, por tratarse de un software libre, lo que lo hace mucho más accesible. Además, se ha elaborado un pluging para este software, ya que esto nos permite que el usuario que utilice la herramienta no necesite nociones de programación para utilizarla. Por tanto, el desarrollo de este proyecto se ha llevado a cabo buscando simplificar al usuario la interacción con esta y tratando que todos los procesos sean lo más automáticos posibles.

El funcionamiento de la herramienta que se desea obtener es el siguiente: en qgis, mediante un cuadro de diálogo de la herramienta, el usuario carga la capa de parcelas catastrales de Valencia, junto con la ortofoto del PNOA. Tras seleccionar una parcela, el usuario introduce el título que desea para el mapa resultante, su nombre y los puntos de interés que desea seleccionar. Tras esto, se ejecuta la herramienta y al finalizar se abre de manera automática el mapa resultante.

En este documento se detalla cómo se ha llevado a cabo esta herramienta, cómo funciona y los resultados que se han obtenido.

3. Pluging

En este apartado se detallan algunos aspectos sobre la herramienta desarrollada, tanto su funcionamiento como aspectos importantes sobre la misma.

3.1. Aspectos relevantes

Durante el desarrollo de la herramienta, se ha intentado pensar en los detalles para que su manejo sea sencillo para el usuario.

Algunos de los aspectos más importantes de la herramienta implementada que se han establecido para conseguir este objetivo son:

- Pluging válido tanto para Windows como para Linux. El programa realiza la comprobación del sistema operativo que se está utilizando para ajustar el nombre de los directorios de manera automática según el sistema operativo.

- El pluging no necesita que el usuario tenga en su sistema ninguna capa específica, desde la propia herramienta se cargan todas las capas necesarias para realizar el plano, facilitando así su uso.

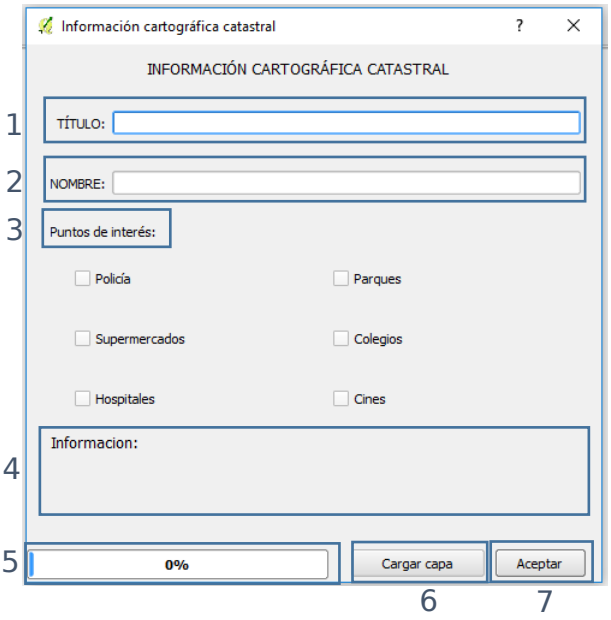
- La interfaz de usuario muestra mensajes y el estado del proceso, para que el usuario sepa en todo momento que el proceso se está ejecutando correctamente. La herramienta permite obtener información catastral de una parcela, gracias a una conexión al servidor de Catastro que, a partir de la referencia catastral que se obtiene automáticamente al seleccionar una de las parcelas, se acceda a información como la calle, el área e, incluso, la imagen de la fachada.

- Además de la información propia de la parcela, la herramienta realiza una consulta espacial y nos muestra los puntos de interés que el usuario a escogido que se encuentran próximos a la parcela.

- No es necesario especificar rutas. Todas las rutas son automáticas, los datos necesarios (capa de parcelas, símbolo del Norte, ect.) se toman de la carpeta del propio plugins y el resultado se guarda automáticamente en la capeta dónde el usuario tenga el proyecto de qgis en el que está trabajando.

3.2. Interfaz gráfica

La interfaz del pluging desarrollado es la que se muestra a continuación:



Las distintas partes de este interfaz son:

1. Título. El usuario debe introducir el título que quiere que aparezca en el cajetín del mapa generado.[1]

2. Nombre. El usuario debe introducir su nombre para que aparezca en el mapa generado.

3. Puntos de interés. Mediante los diferentes check lists se pueden escoger aquellos puntos de interés que queremos que aparezcan, si existen cerca de las parcelas seleccionada, en el mapa resultante.
4. Información. En esta sección aparece la información de los procesos que se van ejecutando, para que el usuario sepa qué proceso se está ejecutando en cada momento.

5. Barra de progreso. En esta barra se muestra el progreso que se va realizando.

6. Botón de cargar capa. Al pulsarlo carga automáticamente la capa de parcelas (.shp) y la ortofoto (mediante wms).

7. Botón de aceptar. Al pulsarlo, con una parcela de la capa “PARCELA” seleccionada, se ejecuta el proceso de creación de mapa.

3.3. Funcionamiento

El funcionamiento de la herramienta en rasgos generales se explica en este apartado.

En primer lugar, es necesario tener un proyecto en Qgis.

[1] Tanto el título como el nombre deben introducirse sin acentos ni caracteres especiales

Una vez instalado el pluging, cuando lo ejecutamos nos aparece la interfaz gráfica que se ha expuesto anteriormente.

Si pulsamos el botón de cargar, automáticamente se cargan las parcelas catastrales urbanas del municipio de Valencia y la ortofoto del PNOA.

Con estas capas ya cargadas, seleccionamos una parcela y, una vez seleccionada, ponemos el título y el nombre que queremos, marcamos los puntos de interés que deseemos y le damos al botón de aceptar. Con esto se ejecuta el proceso de creación automática del mapa. Al finalizar, este se abre automáticamente en el explorador.

4. Código

En esta sección se detalla el código que se ha desarrollado para la herramienta.

4.1. Importar las librerías necesarias

En primer lugar, es necesario cargar las librerías necesarias para trabajar.

```
import sys
import os
import qgis
from qgis.core import *
from qgis.core import (
    QgsProject, QgsComposition, QgsApplication, QgsProviderRegistry)
from PyQt4 import Qt, QtCore, QtGui, uic
from PyQt4.QtGui import *
from qgis.gui import *
from PyQt4.QtGui import QPainter, QColor, QPolygonF, QFont, QPrinter, QMessageBox, QAction, QIcon
from PyQt4.QtCore import QSizeF, QPointF, QRectF, QFileInfo, QSettings, QTranslator, qVersion, QCoreApplication
from qgis.gui import QgsHighlight, QgsMapCanvas, QgsLayerTreeMapCanvasBridge
import time
import datetime
import urllib
from processing import *
from qgis.utils import iface
from PyQt4.QtXml import *
from PyQt4.QtXml import QDomDocument
#import lxml.etree as etree
import webbrowser
import requests
import json
from osgeo import osr
from osgeo import ogr
import os
# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
import os.path
import platform
FORM_CLASS, _ = uic.loadUiType(os.path.join(os.path.dirname(__file__), 'informacionCatastral_dialog_base.ui'))
```

Además de las librerías y módulos propios de Qgis y del diálogo (PyQt4), se han añadidos otras librerías como:

- Sys,os,platform para poder acceder a datos del sistema.
- Webbrowser, para abrir e PDF de manera automática
- Requests, para realizar una petición a una url y obtener un objeto como respuesta.
- Json, para poder trabajar con JSON.
- Time, para poder acceder a la hora y ponerla automáticamente en el PDF.
- Urllib, para acceder al servidor de Catastro.

4.2. Cargar capas

En primer lugar, se deben cargar tanto la capa de parcelas como la ortofoto.

```
def cargar(self):
    self.infoText.setText("Cargando...") # Log message
    proyecto = QgsProject.instance()
    registro = QgsMapLayerRegistry.instance() # Se define el registro para cargar la capa
    mLegend = iface.legendInterface() # Definicion de la leyenda
    # Carga en el proyecto de la capa WMS del PNOA
    iface.addRasterLayer("crs=EPSG:25830&layers=OI.OrthoimageCoverage&styles=&format=image/png&url=http://www.ign.es/wms-inspire/pnoa-ma?","pnoa","wms")
    # Carga de la capa PARCELA de Catastro alojada en el directorio del plugin
    # Nota: para un SO Windows se cambia la barra
    if platform.system() == 'Linux':
        parcelas = os.path.dirname(__file__) + '/PARCELA.SHP'
    else:
        parcelas = os.path.dirname(__file__) + '/PARCELA.SHP'
        parcelas = parcelas.replace('\\','/')
    parcelasLayer = QgsVectorLayer(parcelas,'PARCELA','ogr')
    # define the layer properties as a dict
    symbols = parcelasLayer.rendererV2().symbols()
    symbol = symbols[0]
    symbol.symbolLayer(0).setDataDefinedProperty('color', 'color_rgba(102,46,46,10)')
    symbol.symbolLayer(0).setDataDefinedProperty('color_border', 'color_rgba(0, 0, 0, 50)')
    mLegend.refreshLayerSymbology(parcelasLayer)
    registro.addMapLayer(parcelasLayer)
    self.infoText.setText("Capas cargadas!") # Log message
    QMessageBox.information(None,'Información',"Capa de parcelas catastrales cargada") # Notification message
```

Para cargar estas capas, en primer lugar, accedemos al proyecto y creamos el registro.

El WMS del PNOA se añade directamente con la función addRasterLayer, indicándole la url correspondiente.

Para cargar la capa de parcelas, que está ubicada en la carpeta del plugins es necesario comprobar el sistema operativo, para que, si es Windows, se adapten las barras del directorio. Una vez accedemos a la capa de añade como capa vectorial (QgsVectorLayer) y se establece la simbología.

4.3. Desarrollo del producto

Una vez cargadas las capas se pasa a la elaboración del PDF. Para ello es necesario realizar diferentes pasos que se detallan a continuación.

4.3.1. Pasos previos

Antes de realizar el mapa es necesario realizar algunas acciones.

```
self.infoText.setText("Iniciando...") # Log message
self.progressBar.setValue(5) # Progress bar
aviso = 1
proyecto = QgsProject.instance() # Definicion de variable del proyecto del usuario
pathProject = proyecto.readPath("") # Ruta del proyecto
if pathProject == '':
    QMessageBox.information(None,"Atención".decode('utf-8'),"Necesitas un proyecto QGIS!")
    raise IOError('Necesitas un proyecto QGIS!')
ruta = os.path.join(os.path.dirname(__file__), 'informacionCatastral_dialog_base.ui') # Ruta de la interfaz grafica
# Ruta de la plantilla
if platform.system() == 'Linux':
    myFile = os.path.dirname(__file__) + r'/plantilla.qpt'
else:
    myFile = os.path.dirname(__file__) + r'/plantilla.qpt'
    myFile = myFile.replace('\\','/')
registro = QgsMapLayerRegistry.instance() # Definicion de la variable de Registro
capas = registro.mapLayers() # Definicion de las capas del Registro
campos = ["MUNICIPIO","MASA","HOJA","TIPO","PARCELA","COORX","COORY","VIA","NUMERO","NUMERODUP","NUMSYMBOL","AREA","FECHAAI
"EJERCICIO","NUM_EXP","CONTROL","REFCAT"]
valores = []
mLegend = iface.legendInterface() # Definicion de la leyenda
# Definicion del canvas y parametros
```

En el código anterior, accedemos a la carpeta del proyecto, en caso de o existir (porque el proyecto no está guardado) nos devolverá un error.

Si no se ha producido ningún error, tomamos la ruta del proyecto.

Por otro lado, tomamos la plantilla del directorio del plugin. Es necesario realizar una modificación de la ruta que obtenemos si se está usando Windows como sistema operativo.

Tras esto creamos el registro y accedemos a las capas y definimos la leyenda.

```
# Definicion del canvas y parametros
mCanvas = iface.mapCanvas()
mCanvas.setCanvasColor(QColor("white"))
mCanvas.enableAntiAliasing(True)
mCanvas.setSelectionColor( QColor("red") )
mCanvas.refresh()
mCanvas.refreshAllLayers()
```

Definimos también algunas propiedades del canvas.

```
# Definicion de capa virtual en referencia a la parcela seleccionada
mem_layer = QgsVectorLayer("Polygon?crs=epsg:25830&field=id:integer&index=yes","Parcela seleccionada","memory")
registro.addMapLayer(mem_layer)
global pcat1
global pcat2
self.infoText.setText("Definiendo variables...") # Log message
self.progressBar.setValue(10) # Progres bar
```

Definimos una capa virtual, que va a contener la capa seleccionada y la añadimos.

```
# Proceso para la capa de parcelas catastrales
for clave in capas:
    capa = capas[clave]
    if capa.name() == 'PARCELA':
        capaRenderer = capa.rendererV2() # Renderer de la capa
        if (len(capa.selectedFeatures()) == 0): # Aviso: no se selecciona una entidad
            print "Selecciona una parcela"
            aviso = 1
        elif (len(capa.selectedFeatures()) > 1): # Aviso: se selecciona mas de una entidad
            print "Selecciona una parcela"
            aviso = 2
        else:
            aviso = 0
```

Accedemos a las capas y buscamos la capa “PARCELA”, que es con la que vamos a trabajar. Comprobamos si hay solamente una capa seleccionada (si no hay ninguna o hay más de una se mostrará un error, este error se define más adelante en el código, en función de los valores de “aviso”, tal como se muestra en la siguiente imagen).

```
if aviso == 1:
    QMessageBox.information(None,"Atención".decode('utf-8'),"Selecciona una parcela. Si no tienes la capa de parcelas, cargala con el botón.")
    raise IOError('Selecciona una parcela. Si no tienes la capa de parcelas, cargala con el botón.')
if aviso == 2:
    QMessageBox.information(None,"Atención".decode(utf-8),"Selecciona una sola parcela") # Notification message
    raise IOError('Selecciona una sola parcela')
if aviso == 0:
```

4.3.2. Acceso a la información catastral

Si hay una parcela seleccionada de la capa de parcelas se procede a acceder a su información.

```
self.infoText.setText("Acceso a la entidad seleccionada...") # Log message
self.progressBar.setValue(12) # Progres bar
time.sleep(1)
feature = capa.selectedFeatures()[0] # Acceso a entidad seleccionada
# Extension de la entidad
box = feature.geometry().boundingBox()
xmin = box.xMinimum()
xmax = box.xMaximum()
ymin = box.yMinimum()
ymax = box.yMaximum()
# Extension de la entidad para el overview
xminOver = box.xMinimum()-1000
xmaxOver = box.xMaximum()+1000
yminOver = box.yMinimum()-1000
ymaxOver = box.yMaximum()+1000
```

Accedemos a la parcela seleccionada y almacenamos su extensión (coordenadas máximas y mínimas). Además, también almacenamos una extensión mayor para poder realizar el overview de la parcela.

```
# Extraccion de la referencia catastral (PCAT1 y PCAT2) de la entidad seleccionada
for i in range(len(campos)):
    if (campos[i] == 'PCAT1'):
        pcat1 = feature.attribute(campos[i])
    if (campos[i] == 'PCAT2'):
        pcat2 = feature.attribute(campos[i])
    valores.append(feature.attribute(campos[i]))
# Accediendo al servidor de CATASTRO
self.infoText.setText("Accediendo al servidor de catastro...") # Log message
self.progressBar.setValue(25) # Progress bar
time.sleep(1)
# Definicion de variables iniciales para la recogida de datos y su correspondiente URL
server = 'https://www1.sedecatastro.gob.es/'
url = 'https://www1.sedecatastro.gob.es/CYCBienInmueble/OVCListaBienes.aspx?del=46&muni=900&rc1='+str(pcat1)+'&rc2='+str(pcat2)+''
res = urllib.urlopen(url).read().split("\n") # Obtencion de toda la pagina web correspondiente a la parcela catastral pedida
```

Accedemos a los campos de la capa de parcelas y tomamos los valores de PCAT1 y PCAT2, estos dos campos forman la referencia catastral de la pardela.

Definimos el servidor de Catastro y formamos la url a partir de la parte genérica de esta url y la referencia catastrar de la parcela seleccionada. Realizamos una petición a esta url y obtenemos, en la variable res, toda la página web correspondiente a nuestra parcela.

```
# Se recorre todo el objeto linea a linea
for linea in res:
    if "class='panel-heading amarillo'" in linea:
        for linea2 in linea.split('<'):
            if "title='Tipo de parcela'" in linea2:
                for linea3 in linea2.split('>'):
                    if "(" in linea3:
                        descripcion = linea3 # Obtencion de la descripcion de la parcela
            if "title='Superficie gráfica'" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 65:
                        superficieGrafica = linea3 + 'm2' # Obtencion de la superficie grafica de la parcela
            if "title='Localizaci" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 65:
                        localizacion.append(linea3)
                        calle = linea3 # Obtencion de la calle de la parcela y la direccion de cada vivienda o negocio
            if "title='Uso'" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 55:
                        uso.append(linea3) # Obtencion del uso de cada uno (Urbana o Rustica)
            if "title='Superficie construida'" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 65:
                        superficie.append(linea3) # Obtencion de la superficie de cada uno
            if "title='Año construcción'" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 65:
                        ano.append(linea3) # Obtencion del año de cada uno
            if "title='Coeficiente de participación'" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 65:
                        coeficiente.append(linea3) # Obtenicion del coeficiente de participacion
            if "javascript:CargarBien" in linea2:
                for linea3 in linea2.split('>'):
                    if len(linea3) < 25:
                        refcat.append(linea3) # Obtencion de la referencia catastral
```

Recorremos la respuesta obtenida línea por línea, buscando la información que nos interesa. Por ejemplo, cuando encontremos “title=’Superficie construida’” almacenaremos este cado en la lista “superficie”.

```
# Iteracion independiente para la obtencion de la fotografia de la fachada
for linea in res:
    for linea2 in linea.split('>'):
        if 'captcha' in linea2:
            try:
                fuente = linea2.split('src=')[1].split(' ')[0]
                path =  server+fuente[4:-1]
            except: # Si no tiene fotografia de fachada, se define una imagen por defecto
                if platform.system() == 'Linux':
                    path = os.path.dirname(__file__) + '/no_foto.png'
                else:
                    path = os.path.dirname(__file__) + '/no_foto.png'
                    path = path.replace('\\','/')
```

En el caso de la imagen de la fachada, el proceso es un poco distinto. En primer lugar, tenemos que separar cada línea por el carácter “>”. Recorremos el resultado y buscamos la palabra “captcha”, si la encontramos es que tiene foto, por lo que preparamos la url para poder acceder a ella (el valor del “captcha” varía, por lo que es necesario este proceso para obtener el captcha correspondiente en ese mismo momento). Si no tiene foto, se pone una imagen que está en la carpeta del pluging, de nuevo teniendo en cuenta el sistema operativo.

4.3.3. Petición a la información del OSM

Para los puntos de interés, realizaremos peticiones a la información del OSM.

```
# Transformacion de coordenadas a otro sistema de referencia (4326) para el empleo de la extension en la peticion a OSM
source = osr.SpatialReference()
source.ImportFromEPSG(25830)
target = osr.SpatialReference()
target.ImportFromEPSG(4326)
transform = osr.CoordinateTransformation(source, target)
point1Box = ogr.CreateGeometryFromWkt("POINT ("+str(xminOver)+" "+str(yminOver)+")")
point1Box.Transform(transform)
point2Box = ogr.CreateGeometryFromWkt("POINT ("+str(xmaxOver)+" "+str(ymaxOver)+")")
point2Box.Transform(transform)
xMinOver = point1Box.GetY()
yMinOver = point1Box.GetX()
xMaxOver = point2Box.GetY()
yMaxOver = point2Box.GetX()
params = {'xmin': xMinOver, 'ymin': yMinOver,'xmax': xMaxOver, 'ymax': yMaxOver}
self.infoText.setText("Accediendo al servidor de OSM...") # Log message
self.progressBar.setValue(15) # Progres bar
time.sleep(1)
```

Para ello, en primer lugar, debemos transformas las coordenadas de la extensión del overview, ya que OSM trabaja en 4326.

```
# Definicion de URLs para distintas entidades. API de Overpass Turbo: mineria de datos Open Street Maps.
hospitalOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["amenity"="hospital"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way
)s,%(xmax)s,%(ymax)s);relation["amenity"="hospital"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
policeOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["amenity"="police"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way["
(xmax)s,%(ymax)s);relation["amenity"="police"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
supermarketOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["shop"="supermarket"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way
)s,%(xmax)s,%(ymax)s);relation["shop"="supermarket"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
schoolOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["amenity"="school"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way["
(xmax)s,%(ymax)s);relation["amenity"="school"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
parkOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["leisure"="park"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way["le
x)s,%(ymax)s);relation["leisure"="park"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
cinemaOSM = (
'http://overpass-api.de/api/interpreter?data=[out:json];(node["amenity"="cinema"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s);way["
(xmax)s,%(ymax)s);relation["amenity"="cinema"](%(xmin)s,%(ymin)s,%(xmax)s,%(ymax)s););(._;>;);out body;' % params)
```

Definimos las urls correspondientes a las distintas capas de información, teniendo en cuenta la extensión de la parcela.

```
if checkHospital == True:
    myRequest = requests.get(hospitalOSM) # Obtencion de datos
    data = myRequest.json()
    if myRequest.status_code == 200: # Punto de control de conexion con el servidor
        print 'Response OSM: OK'
    else:
        QMessageBox.information(None,"Atencion","No se obtienen datos del servidor! Respuesta 400") # Notification message
    elementos = data['elements']
    transform = osr.CoordinateTransformation(target,source) # Transformacion de coordenadas 4326 a 25830
    hospitalFeatureLayer = QgsVectorLayer("Point?crs=epsg:25830&field=id:integer&index=yes","Hospitales","memory") # Creacion de capa vectorial
    registro.addMapLayer(hospitalFeatureLayer) # Carga de la capa Hospitales
    providerHospital = hospitalFeatureLayer.dataProvider() # Provider de la capa
    osmHospital = QgsFeature() # Creacion de las entidades de la capa de Hospitales
    for elem in elementos: # Para cada elemento obtenido, crear puntos y cargarlos en la capa
        if 'tags' in elem:
            if 'lat' in elem:
                lat = elem['lat']
                lon = elem['lon']
                featureReprojected = ogr.CreateGeometryFromWkt("POINT ("+str(lon)+" "+str(lat)+")")
                featureReprojected.Transform(transform)
                newPoint = QgsPoint(featureReprojected.GetX(),featureReprojected.GetY())
                osmHospital.setGeometry(QgsGeometry.fromPoint(newPoint))
                providerHospital.addFeatures([osmHospital])
    self.infoText.setText("Hospitales encontrados...")
    self.progressBar.setValue(81) # Progress bar
    mCanvas.refresh()
    mCanvas.refreshAllLayers()
    time.sleep(1)
```

Si el check box correspondiente (en este caso de hospitales) ha sido marcado desde la interfaz, re realiza la petición a OSM. Si a respuesta no es correcta, se lanza un mensaje de error.

Se crea una capa virtual para la capa del OSM.

Se recoge el JSON de respuesta, recorremos los elementos de este JSON y buscamos si en el elemento encontramos “lat” (latitud). Si es así tomamos tanto la latitud como la longitud, le aplicamos la transformación correspondiente para que esté en 25830 que es el sistema de referencia utilizado en la capa de parcelas.

Una vez tenemos las coordenadas transformadas, creamos un punto con la función QgsPoint y a partir de esto creamos la feature con su geometría con setGeometry y la añadimos a la capa virtual de hospitales que habíamos creado.

Este proceso es análogo para todos los distintos puntos de interés que el usuario puede escoger desde la interfaz.

4.3.4. Composición y diseño de mapa

Una vez tenemos toda la información, tanto de catastro como de las distintas capas de OSM, tenemos que modificar la plantilla y añadirle los datos necesarios para obtener nuestra composición de mapa.

```
# Definicion de geometria y simbologia de la entidad selecionada
geometry = feature.geometry()
geometry = geometry.asPolygon()
poly_reproj = geometry
crsSrc = QgsCoordinateReferenceSystem(25830)
crsDest = QgsCoordinateReferenceSystem(25830)
xform = QgsCoordinateTransform(crsSrc, crsDest)
symbols = mem_layer.rendererV2().symbols()
symbol = symbols[0]
symbol.symbolLayer(0).setDataDefinedProperty('color', 'color_rgba(102,46,46,50)')
symbol.symbolLayer(0).setDataDefinedProperty('color_border', 'color_rgb(0, 221, 255)')
mLegend.refreshLayerSymbology(mem_layer)
for i, point in enumerate(geometry[0]):
    pt = xform.transform(point)
    poly_reproj[0][i] = pt
self.infoText.setText("Editando parcela...") # Log message
self.progressBar.setValue(77) # Progress bar
prov = mem_layer.dataProvider()
newFeature = QgsFeature()
newFeature.setAttributes(valores)
newFeature.setGeometry(QgsGeometry.fromPolygon(poly_reproj))
prov.addFeatures([newFeature])
mCanvas.refresh()
mCanvas.refreshAllLayers()
# Accediendo al servidor de OSM
```

En primer lugar, para la entidad seleccionada e establece la simbología que queremos.

```
capas = registro.mapLayers()
lst = capas.keys()
#lst = lst[::-1]
lst2 = []
for i in range(len(lst)):
    capa = lst[i]
    if 'seleccionada' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'Policia' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'Parque' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'Colegio' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'Supermercado' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'PARCELA' in capa:
        lst2.append(capa)
for i in range(len(lst)):
    capa = lst[i]
    if 'pnoa' in capa:
        lst2.append(capa)
```

Para que las capas aparezcan en el orden que queremos, es decir, la ortofoto al fondo, las capas puntuales arriba, etc., es necesario reordenarlas, ya que el orden en el que se añaden no es correcto. Para ello se recorre la lista de capas y, por el orden deseado, se van añadiendo a una nueva lista que es con la que vamos a trabajar.

```
mapRenderer.setLayerSet(lst2) #list of map layers id's
time.sleep(2) # delays for 2 seconds
mCanvas.refresh()
mCanvas.refreshAllLayers()
# Creacion de COMPOSICION
c = QgsComposition(mapRenderer)
# Definicion de PLANTILLA
myTemplateFile = file(myFile, 'rt')
myTemplateContent = myTemplateFile.read()
# Definicion de DOCUMENTO
myDocument = QDomDocument()
myDocument.setContent(myTemplateContent)
c.loadFromTemplate(myDocument)
self.infoText.setText("Ajustando producto final...") # Log message
self.progressBar.setValue(90) # Progress bar
c.setPlotStyle(QgsComposition.Print)
c.refreshItems()
```

Refrescamos todas las capas y el canvas.

A continuación, creamos la composición con QgsComposition y cargamos la plantilla que habíamos cargado previamente. Con estas líneas de código definimos los ítems necesarios para realizar la composición de mapa.

```
# Valores de encuadre para los mapas en el producto final
x1, y1 = 20, 33
x2, y2 = 230.05, 159.85
w1, h1 = 203, 244.25
w2, h2 = 169.5, 90.6
```

A continuación, vamos estableciendo los distintos elementos que componen el mapa. En primer lugar, establecemos las dimensiones y posición de los dos mapas: el principal y el overview.

```
# COMPOSER MAP: Entidad seleccionada
composerMap1 = QgsComposerMap(c, x1 ,y1, w1, h1)
composerMap1.zoomToExtent(rec2)
composerMap1.setFrameEnabled(True)
composerMap1.setFrameOutlineColor(QColor('black'))
composerMap1.setFrameOutlineWidth(0.1)
#composerMap1.setGridEnabled(True)
c.addItem(composerMap1)
```

Para el mapa de la parcela seleccionada, establecemos sus dimensiones. Además, establecemos el zoom con los valores que habíamos definido anteriormente y la línea exterior de la parcela. Tras esto lo añadimos a la composición de mapa.

```
# COMPOSER SCALE BAR: numerica y grafica
scale = QgsComposerScaleBar(c)
scale.setItemPosition(190,265)
scale.setStyle('Numeric') # optionally modify the style
scale.setComposerMap(composerMap1)
scale.applyDefaultSize()
c.addItem(scale)
scale = QgsComposerScaleBar(c)
scale.setItemPosition(27.5,262.5)
scale.setStyle('Line Ticks Up') # optionally modify the style
scale.setComposerMap(composerMap1)
scale.applyDefaultSize()
c.addItem(scale)
```

El siguiente elemento es la escala, tanto numérica como gráfica. Las creamos mediante la opción de Qgis QgsComposerScaleBar, con dos estilos (numérica y gráfica). De nuevo establecemos su posición en la hoja y lo añadimos a la composición de mapa.

```
# COMPOSER PICTURE: Simbolo de norte
northarrowIcon = QgsComposerPicture(c)
northarrowIcon.setSceneRect(QRectF(203,42.5,15,15))
if platform.system() == 'Linux': # La imagen se encuentra en el directorio del plugin
    pathNorth = os.path.dirname(__file__) + '/north.svg'
else:
    pathNorth = os.path.dirname(__file__) + '/north.svg'
    pathNorth = pathNorth.replace('\\','/')
northarrowIcon.setPictureFile(pathNorth)
c.addItem(northarrowIcon)
# COMPOSER PICTURE: Foto fachada
fotoFachada = QgsComposerPicture(c)
fotoFachada.setSceneRect(QRectF(347,55,50,52))
pathFoto = path
fotoFachada.setPictureFile(pathFoto)
c.addItem(fotoFachada)
```

Añadimos también las imágenes, tanto el símbolo del norte como la foto de la fachada. La dinámica es la misma que para los otros elementos: definimos las dimensiones y posiciones, añadimos en este caso las imágenes y las añadimos a la composición del mapa.

```
# COMPOSER MAP: Overview
composerMap2 = QgsComposerMap(c, x2 ,y2, w2, h2)
composerMap2.zoomToExtent(recOver2)
composerMap2.setFrameEnabled(True)
composerMap2.setFrameOutlineColor(QColor('black'))
composerMap2.setFrameOutlineWidth(0.1)
c.addItem(composerMap2)
```

Añadimos también el overview, con las dimensiones y posición que habíamos definido previamente.

```
# Calle
infoparcel = QgsComposerLabel(c)
if calle is not None:
    callex = calle.index("Es:")
    calle2 = calle[:callex]
    infoparcel.setText(' CALLE: '+ str(calle2).decode('utf-8'))
else:
    infoparcel.setText(' CALLE: Desconocida '.decode('utf-8'))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.913,48.753)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
```

Pasamos ahora a la información de la parcela. Para ello creamos etiquetas con QgsComposerLabel. En el caso de la calle, si contiene escalera, nos quedamos únicamente con la parte del string correspondiente a la calle, sin tener en cuenta la escalera. De nuevo establecemos posición, autoajustamos el texto y lo añadimos. Establecemos también el estilo de los textos.

Para poder incluir acento debemos establecer la codificación UTF-8.

```
# Refcat
for j in range(len(valores)):
    if campos[j] == 'REFCAT':
        cadena = ' Referencia catastral: '+str(valores[j])+'   '
infoparcel = QgsComposerLabel(c)
infoparcel.setText(' ' + str(cadena).decode('utf-8'))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.913,77.965)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
# Area
for j in range(len(valores)):
    if campos[j] == 'AREA':
        cadena = ' Area: '+str(valores[j])+' m2'
infoparcel = QgsComposerLabel(c)
infoparcel.setText(' ' + str(cadena).decode('utf-8'))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.913,70.120)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
# Numero portal
for j in range(len(valores)):
    if campos[j] == 'NUMERO':
        cadena = ' Portal: '+str(valores[j])
infoparcel = QgsComposerLabel(c)
infoparcel.setText(' ' + str(cadena).decode('utf-8'))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.607,62.519)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
```

Del mismo modo ponemos la información de la referencia catastral, el área y el número de portal.

```
# Tipo
for j in range(len(valores)):
    if campos[j] == 'TIPO':
        if valores[j] == 'U':
            cadena = ' Tipo : urbana'
        else:
            cadena = ' Tipo : rústica'.decode('utf-8')
infoparcel = QgsComposerLabel(c)
infoparcel.setText(' ' + str(cadena))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.913,55)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
# Fecha
infoparcel = QgsComposerLabel(c)
infoparcel.setText('      ' + str(datetime.date.today()))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(233.607,279.393)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
```

Para el tipo, miramos si el valor es urbano o no para establecer la descripción del tipo.

En el caso de la fecha, la calculamos automáticamente mediante la función de Python “datetime”.

```
# Nombre
if self.txt_nombre.text() == '':
    QMessageBox.information(None,"Atención".decode('utf-8'),"No has introducido un nombre!") # Notification message
    raise IOError('No has introducido un nombre!')
infoparcel = QgsComposerLabel(c)
infoparcel.setText('       ' + str(self.txt_nombre.text()))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(284.317,270.177)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
# Titulo
if self.txt_titulo.text() == '':
    QMessageBox.information(None,"Atención".decode('utf-8'),"No has introducido un titulo") # Notification message
    raise IOError('No has introducido un titulo')
infoparcel = QgsComposerLabel(c)
infoparcel.setText('      ' + str(self.txt_titulo.text()))
infoparcel.setFont(QFont("Cambria",12))
infoparcel.setItemPosition(232.093,259)
infoparcel.adjustSizeToText()
c.addItem(infoparcel)
```

En el caso del título y del nombre, que son datos introducidos por el usuario, en primer lugar, comprobamos que se haya introducido valor, si no, nos mostrará un error. Tomamos el valor introducido pos el usuario y lo ponemos en la posición que queremos.

```
# Elementos de informacion subparcelaria: Pagina 2
# Se cargan como maximo 40 unidades
string = ''
i = 0
subparcel = QgsComposerLabel(c)
subparcel.setText(' INFORMACIÓN SUBPARCELARIA '.decode('utf-8'))
subparcel.setFont(QFont("Cambria",12))
subparcel.setItemPosition(10,320)
subparcel.setFrameEnabled(True)
subparcel.setMarginX(161.25)
subparcel.adjustSizeToText()
c.addItem(subparcel)
subparcel = QgsComposerLabel(c)
subparcel.setText(' Parcela ')
subparcel.setFont(QFont("Cambria",12))
subparcel.setItemPosition(12.5,329.5)
subparcel.setFrameEnabled(True)
subparcel.setMarginX(21)
subparcel.adjustSizeToText()
c.addItem(subparcel)
for i in range(len(refcat)):
    string = str(refcat[i])
    subparcel = QgsComposerLabel(c)
    subparcel.setText(string)
    subparcel.setFont(QFont("Cambria",10))
    subparcel.setItemPosition(15,337.75+5.75*i)
    subparcel.adjustSizeToText()
    c.addItem(subparcel)
    i += 1
    if i == 40:
        break
```

La información de las subparcelas la ponemos de una manera similar, poniendo en los títulos un recuadro alrededor. Mediante un bucle añadimos las siguientes subparcelas.

Este proceso se repetiría para los diferentes ítems se información que tenemos de las subparcelas, con un código análogo al anterior.

```
# PRINTER: Impresion de los resultados a formato PDF
printer = QPrinter()
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName(pathProject + '/out.pdf')
printer.setPaperSize(QSizeF(c.paperWidth(), c.paperHeight()), QPrinter.Millimeter)
printer.setFullPage(True)
printer.setColorMode(QPrinter.Color)
mCanvas.refresh()
mCanvas.refreshAllLayers()
printer.setResolution(c.printResolution())
pdfPainter = QPainter(printer)
#mapRenderer.render(pdfPainter)
c.doPrint(printer, pdfPainter)
pdfPainter.end()
```

Tras colocar todos los elementos en la composición de mapa, imprimimos el resultado en un PDF. Para ello utilizamos QPtrinter. Establecemos el formato PDF y el nombre del fichero de salida (out.pdf) que se almacenará automáticamente en el directorio del proyecto.

Refrescamos el canvas y las capas para asegurarnos de todo lo que hemos establecido e imprimimos el mapa a PDF.

4.3.5. Eliminación de las capas temporales

Tras realizar la composición de mapa, es necesario eliminar las capas temporales que hemos creado.

```
self.infoText.setText("Eliminando capas...") # Log message
self.progressBar.setValue(97) # Progress bar
# Eliminacion de capas utilizadas en el desarrollo del producto
if checkHospital == True:
    registro.removeMapLayer(hospitalFeatureLayer)
if checkPolice == True:
    registro.removeMapLayer(policeFeatureLayer)
if checkSupermarket == True:
    registro.removeMapLayer(supermarketFeatureLayer)
if checkSchool == True:
    registro.removeMapLayer(schoolFeatureLayer)
if checkPark == True:
    registro.removeMapLayer(parkFeatureLayer)
if checkCinema == True:
    registro.removeMapLayer(cinemaFeatureLayer)
registro.removeMapLayer(mem_layer)
self.infoText.setText("Hecho!") # Log message
self.progressBar.setValue(100) # Progress bar
QMessageBox.information(None,"Fin","Realizado correctamente. El PDF se abrirá automáticamente al acpetar".decode('utf-8'))
webbrowser.open_new(pathProject + '/out.pdf') # Apertura automatica del producto
self.progressBar.setValue(0) # Progres bar
del mCanvas
del mapRenderer
del pdfPainter
del printer
del myDocument
del c
```

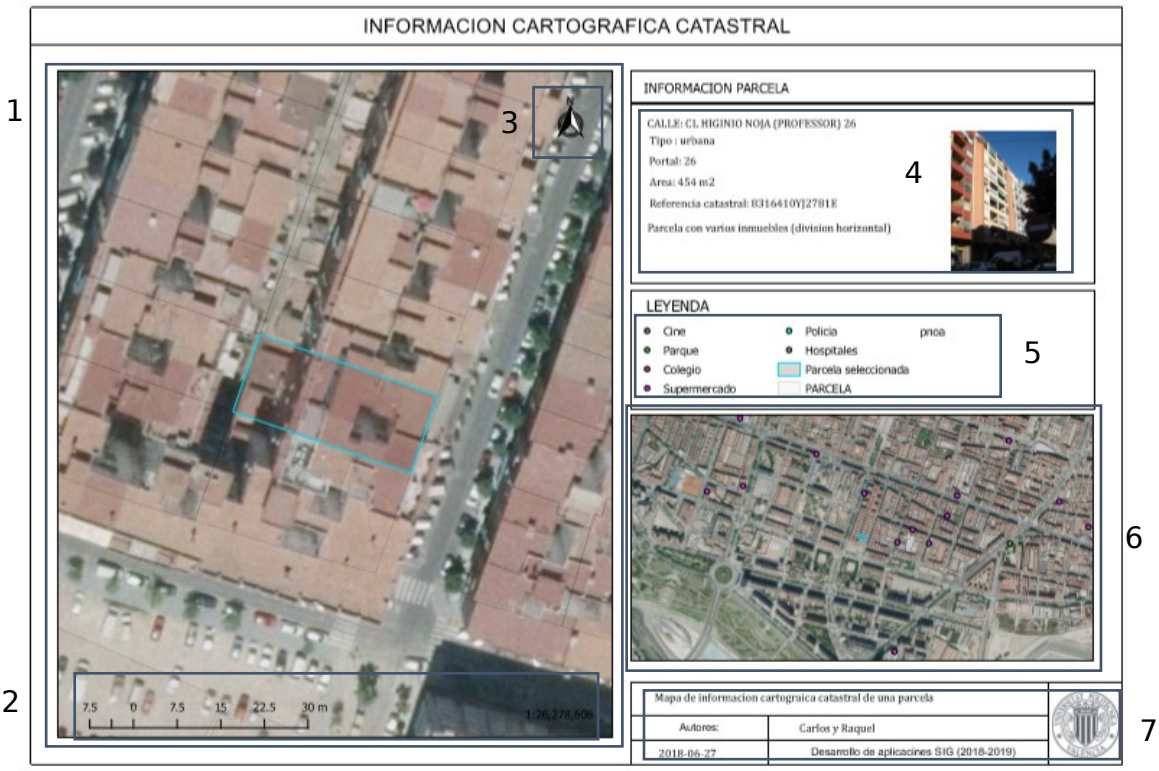
Eliminamos las capas temporales que habíamos cargado, en función de aquellas capas que habían sido seleccionadas por el usuario.

Tras esto, con webbrowser se abre el PDF creado de manera automática y se eliminan los objetos creados.

5. Resultado

Al final del documento se anexa un mapa de ejemplo, resultante de la ejecución del pluging.

A continuación, se muestra una imagen del mismo, para analizar los distintos elementos que lo componen.



Los distintos elementos del mapa son:

1. Mapa de la parcela seleccionada con la ortofoto de fondo.

2. Escala gráfica y numérica.

3. Símbolo del Norte.

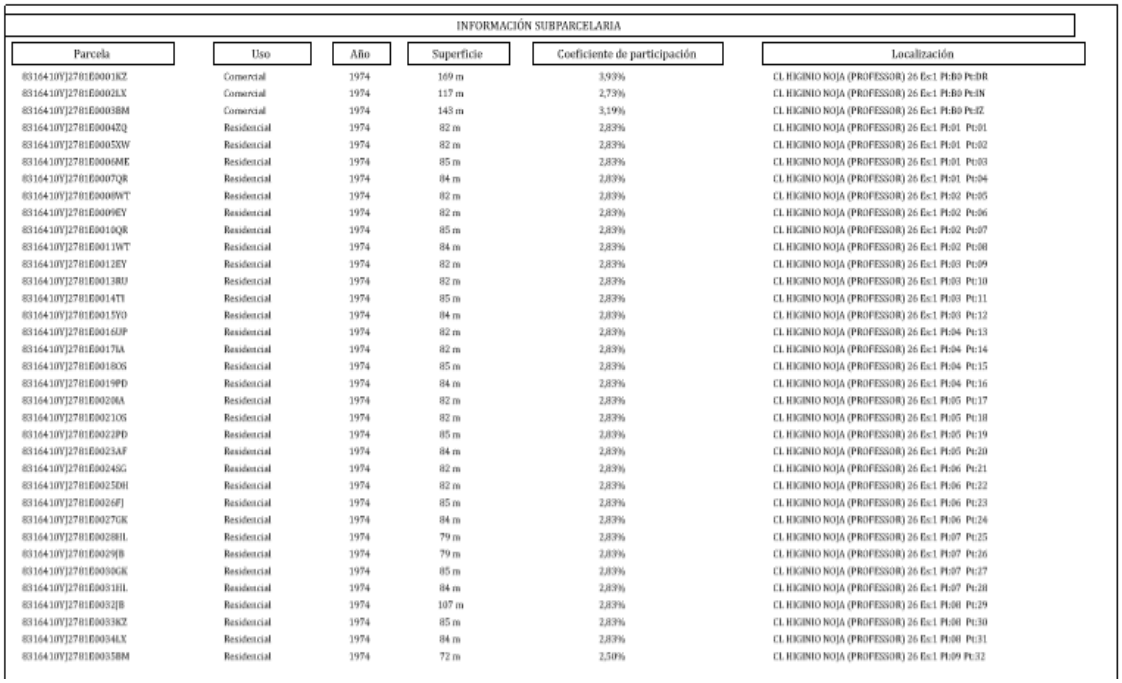
4. Información de la parcela, incluida la imagen si existe, obtenida a partir del servidor de catastro.

5. Leyenda del mapa creada automáticamente.

6. Overview de la parcela seleccionada, nos permite ubicar la parcela en su entorno y ver si hay cerca algún punto de interés.

7. Cajetín. Algunos datos son fijos, otros como a fecha se calculan automáticamente y otros son introducidos por el usuario, como el caso del título y del nombre. Incluye también el logo de la UPV.

Además, en una segunda hoja se muestra información sobre las subparcelas de la parcela seleccionada.



INFORMACIÓN SUBPARCELARIA

Parcela	Uso	Año	Superficie	Coeficiente de participación	Localización
8316410YJ2781E0001KZ	Comercial	1974	169 m	3,93%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:B0 Pt:DR
8316410YJ2781E0002LX	Comercial	1974	117 m	2,73%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:B0 Pt:IN
8316410YJ2781E0003BM	Comercial	1974	143 m	3,19%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:B0 Pt:IZ
8316410YJ2781E0004ZQ	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:01 Pt:01
8316410YJ2781E0005XW	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:01 Pt:02
8316410YJ2781E0006ME	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:01 Pt:03
8316410YJ2781E0007QR	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:01 Pt:04
8316410YJ2781E0008WT	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:02 Pt:05
8316410YJ2781E0009EY	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:02 Pt:06
8316410YJ2781E0010QR	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:02 Pt:07
8316410YJ2781E0011WT	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:02 Pt:08
8316410YJ2781E0012EY	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:03 Pt:09
8316410YJ2781E0013RU	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:03 Pt:10
8316410YJ2781E0014TI	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:03 Pt:11
8316410YJ2781E0015YO	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:03 Pt:12
8316410YJ2781E0016UP	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:04 Pt:13
8316410YJ2781E0017IA	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:04 Pt:14
8316410YJ2781E0018OS	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:04 Pt:15
8316410YJ2781E0019PD	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:04 Pt:16
8316410YJ2781E0020IA	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:05 Pt:17
8316410YJ2781E0021OS	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:05 Pt:18
8316410YJ2781E0022PD	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:05 Pt:19
8316410YJ2781E0023AF	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:05 Pt:20
8316410YJ2781E0024SG	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:06 Pt:21
8316410YJ2781E0025DH	Residencial	1974	82 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:06 Pt:22
8316410YJ2781E0026FJ	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:06 Pt:23
8316410YJ2781E0027GK	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:06 Pt:24
8316410YJ2781E0028HL	Residencial	1974	79 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:07 Pt:25
8316410YJ2781E0029JB	Residencial	1974	79 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:07 Pt:26
8316410YJ2781E0030GK	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:07 Pt:27
8316410YJ2781E0031HL	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:07 Pt:28
8316410YJ2781E0032JB	Residencial	1974	107 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:08 Pt:29
8316410YJ2781E0033KZ	Residencial	1974	85 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:08 Pt:30
8316410YJ2781E0034LX	Residencial	1974	84 m	2,83%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:08 Pt:31
8316410YJ2781E0035BM	Residencial	1974	72 m	2,50%	CL HIGINIO NOJA (PROFESSOR) 26 Esc:1 Pl:09 Pt:32

6. Conclusiones

Tras la realización de este proyecto, hemos sacado una serie de conclusiones que exponemos en esta sección.

6.1. Dificultades encontradas

Durante la elaboración del proyecto se han encontrado algunas dificultades.

En primer lugar, pese a que Qgis es un software muy potente, la documentación disponible al respecto es menos clara y abundante que la existente para otros softwares de SIG.

Por otro lado, el acceso a la información de catastro no siempre es sencilla. La capa .shp descargada de Catastro no dispone apenas de información y la que tenemos, como las calles, están codificadas y por tanto no nos aportan demasiada información. Tampoco existe en catastro la posibilidad de descargar la información alfanumérica de manera directa, ya que esta se muestra únicamente en una página web, por lo que para obtener dicha información es necesario acceder al html fuente de esa página y buscar la información que necesitamos.

6.2. Posibles mejoras

Algunas limitaciones de la herramienta desarrollada y que podrían plantearse como línea futura de trabajo son:

- No limitar la aplicación a el municipio de Valencia, sino permitir al usuario que escoja el municipio en el que quiere trabajar.
- Trabajar tanto con parcelas como con otras capas de catastro como subparcelas, polígnos, etc.
- Añadir más puntos de interés.
- Permitir que el usuario escoja la simbología a aplicar según sus necesidades.