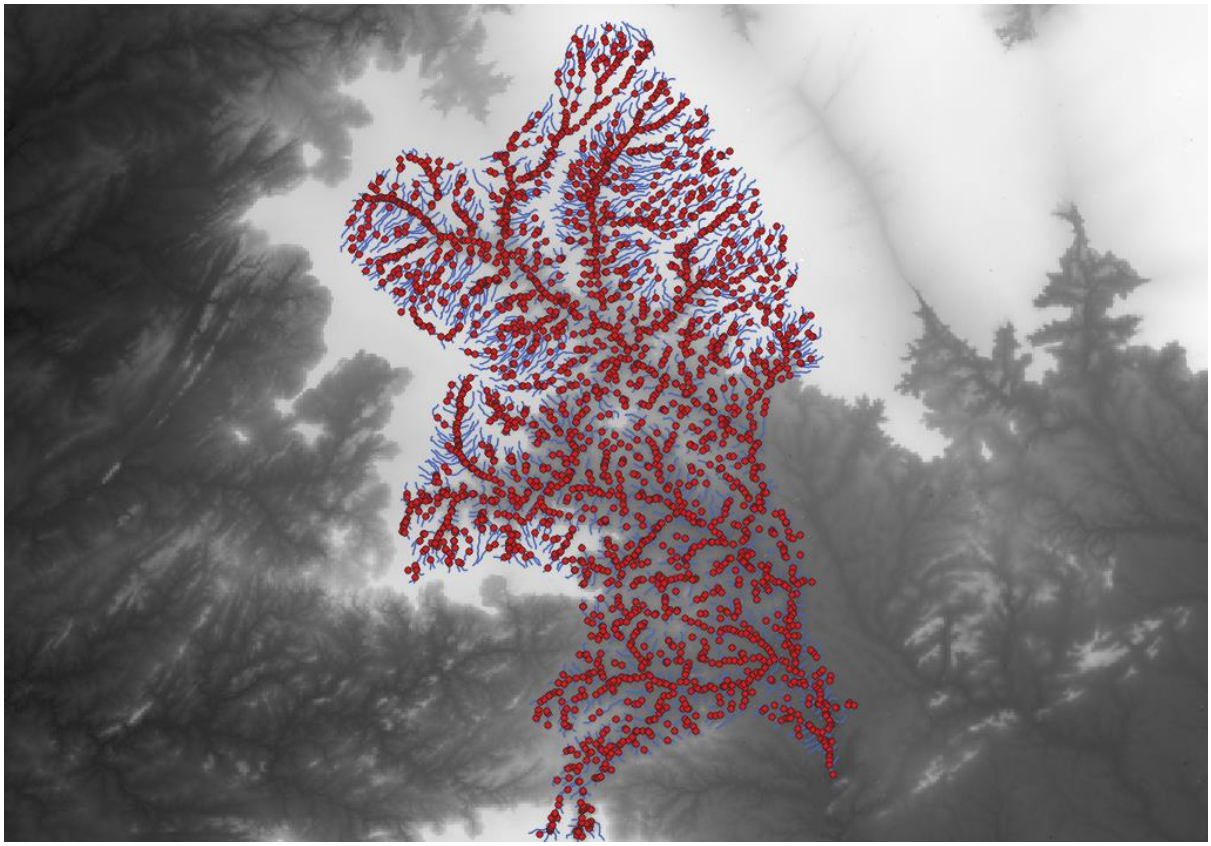


Criando Plugins QGIS com pyQGIS

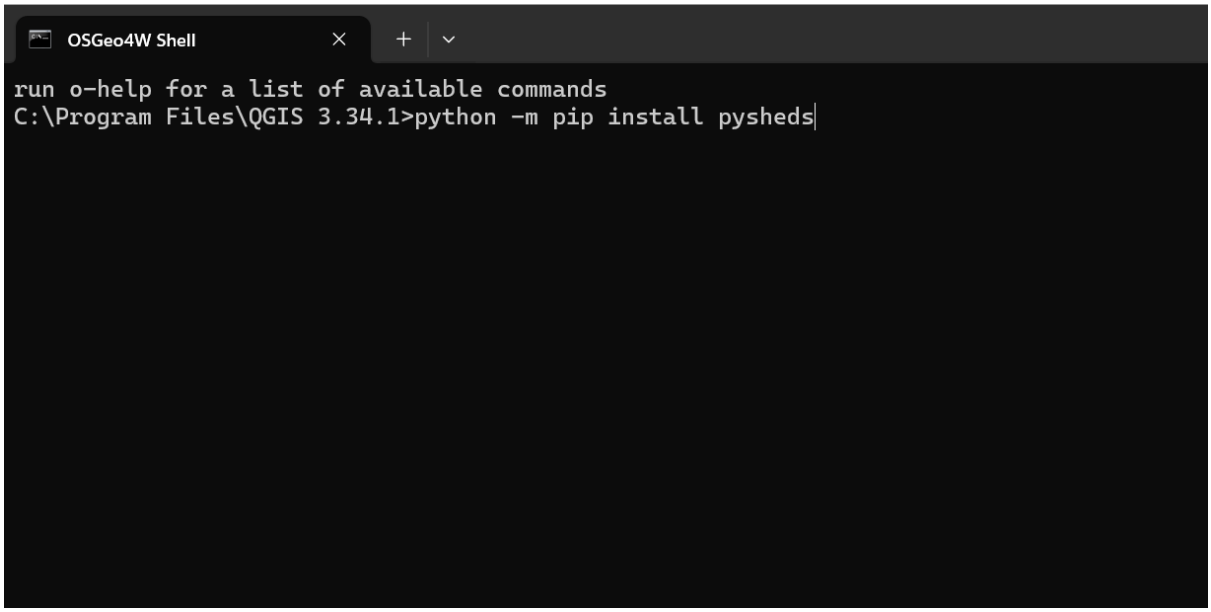
Módulo 3 – Usando bibliotecas externas no ambiente pyQGIS.
Novo plugin Stream Builder

1 - O ambiente pyQGIS

O QGIS possui um ambiente python dedicado e para instalar novas bibliotecas usamos o shell OSGeo4W. Nesse exemplo de plugin vamos utilizar as bibliotecas **pysheds** e **fiona**.

Inicie o shell e digite:

```
python -m pip install pysheds
```



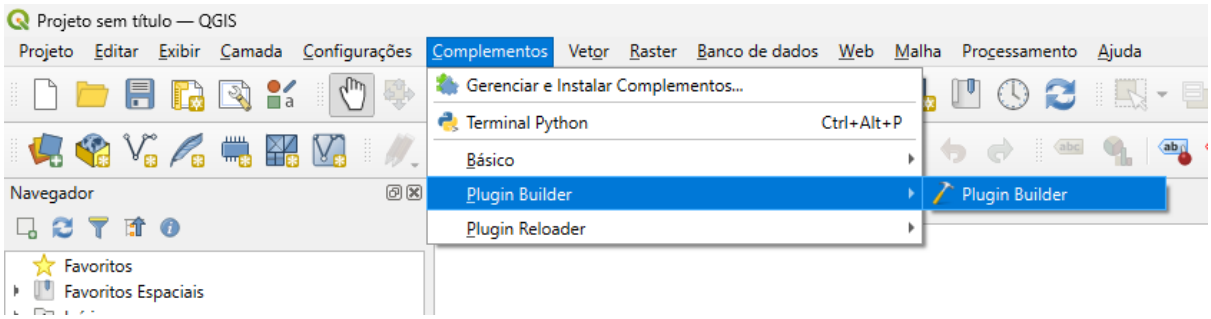
Após instalar o pysheds repita o procedimento instalando a biblioteca fiona:

```
python -m pip install fiona
```

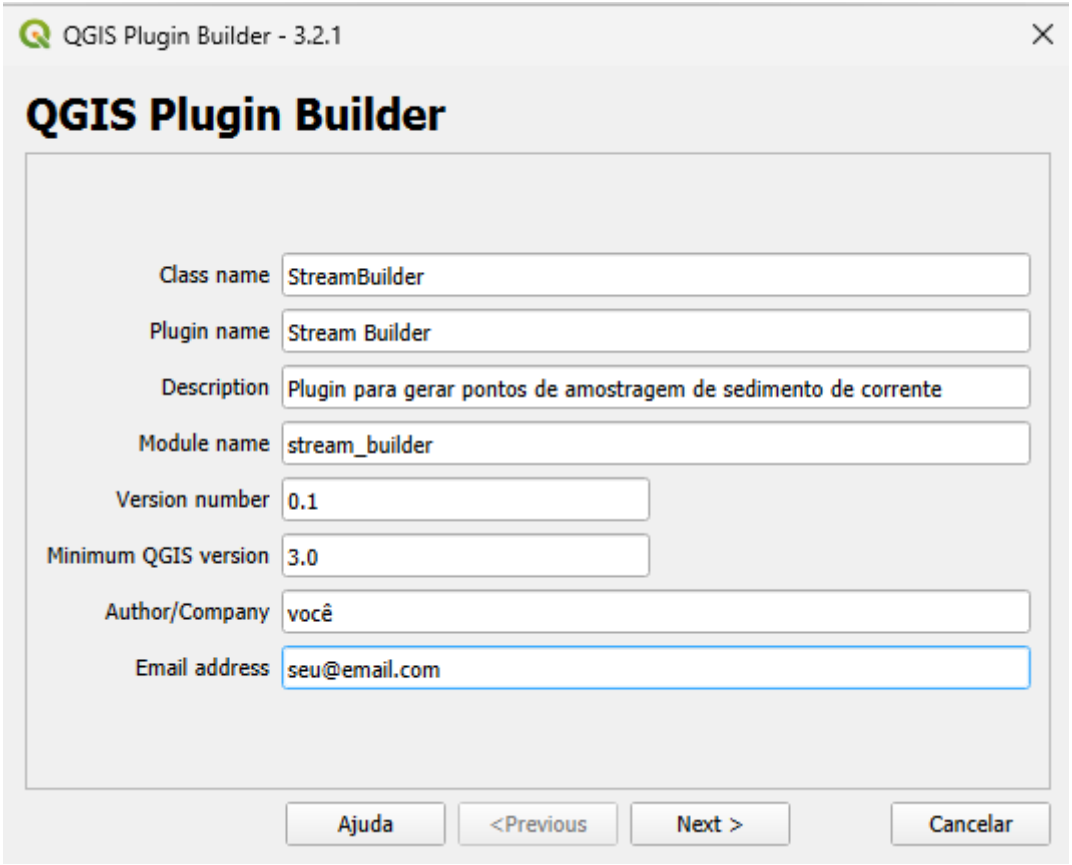
As bibliotecas necessárias para nosso plugin foram instaladas. Procederemos agora com a construção do plugin novo.

2 - Construindo o esqueleto do Stream_Builder no Plugin Builder 3

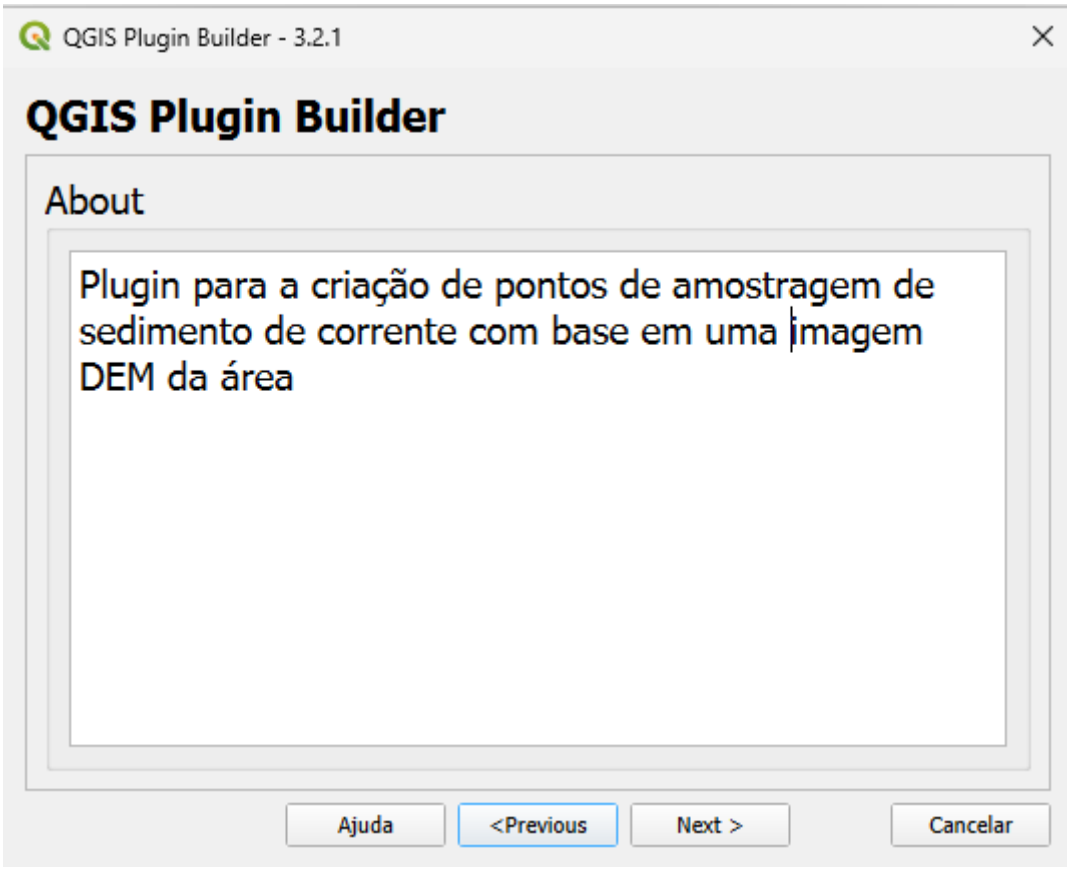
Inicie o Plugin Builder:



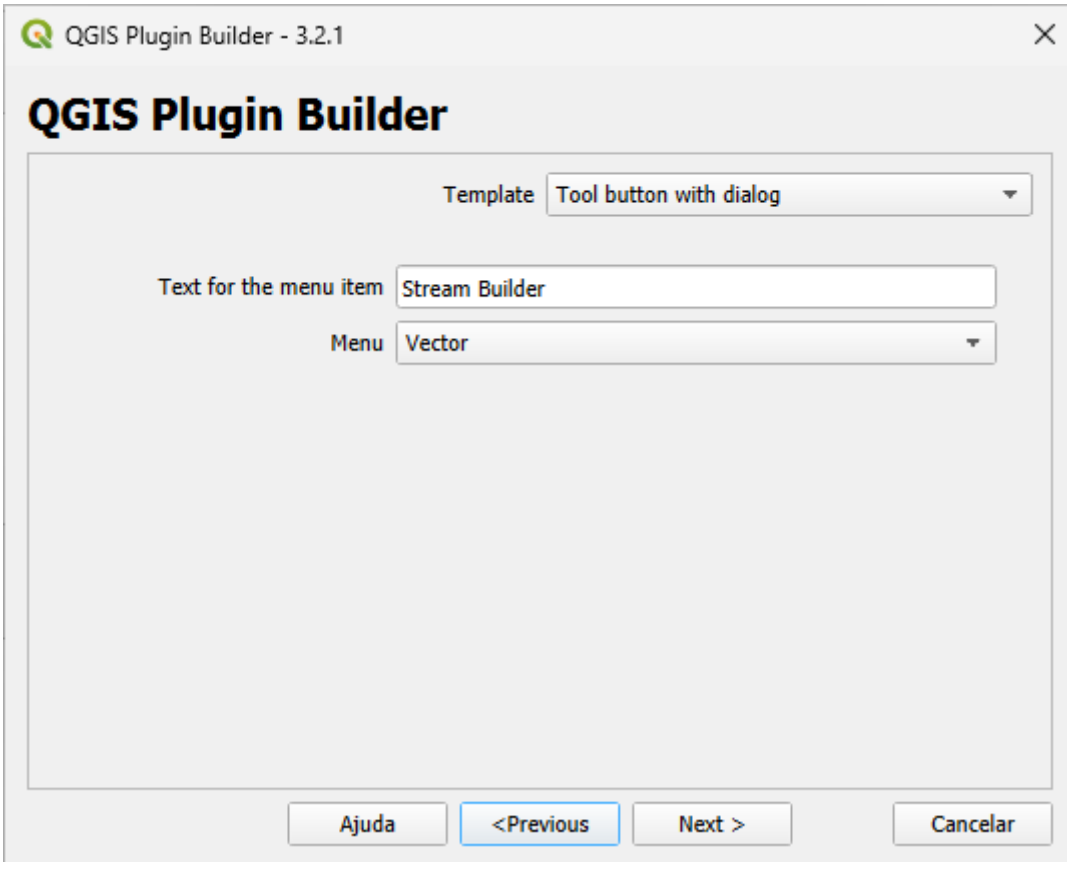
Preencha os campos dos formulários conforme as imagens a seguir:



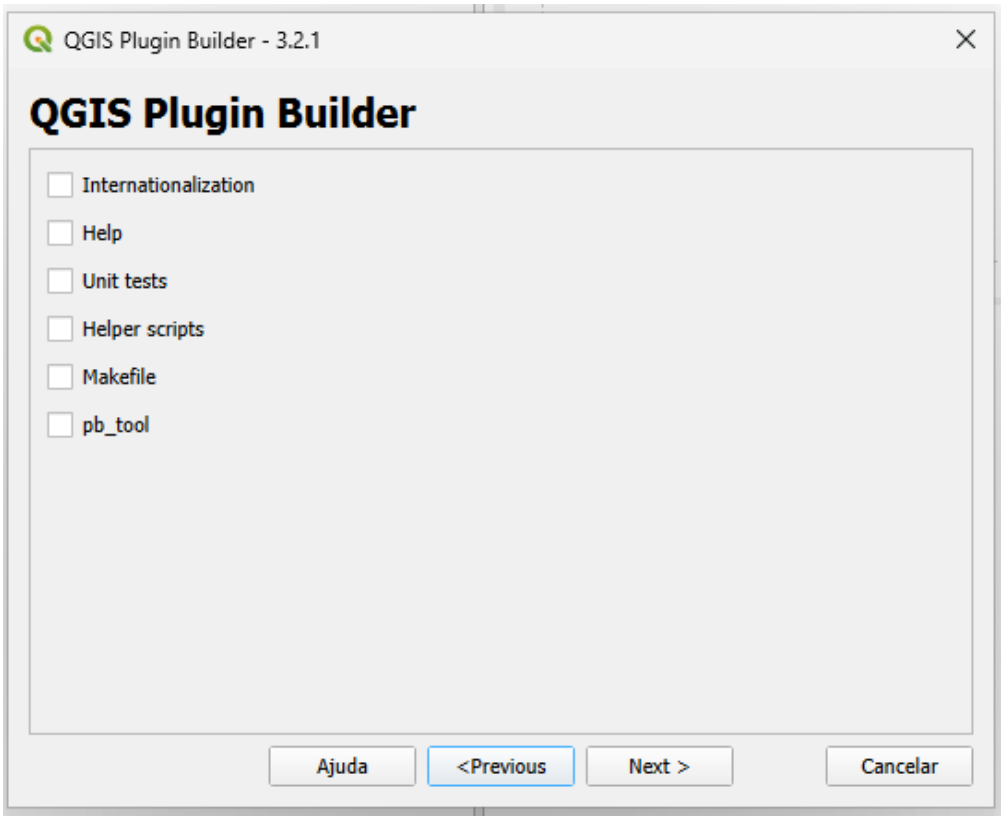
Esse primeiro formulário será usado na criação do arquivo metadata.txt e na definição do nome das classes do plugin.



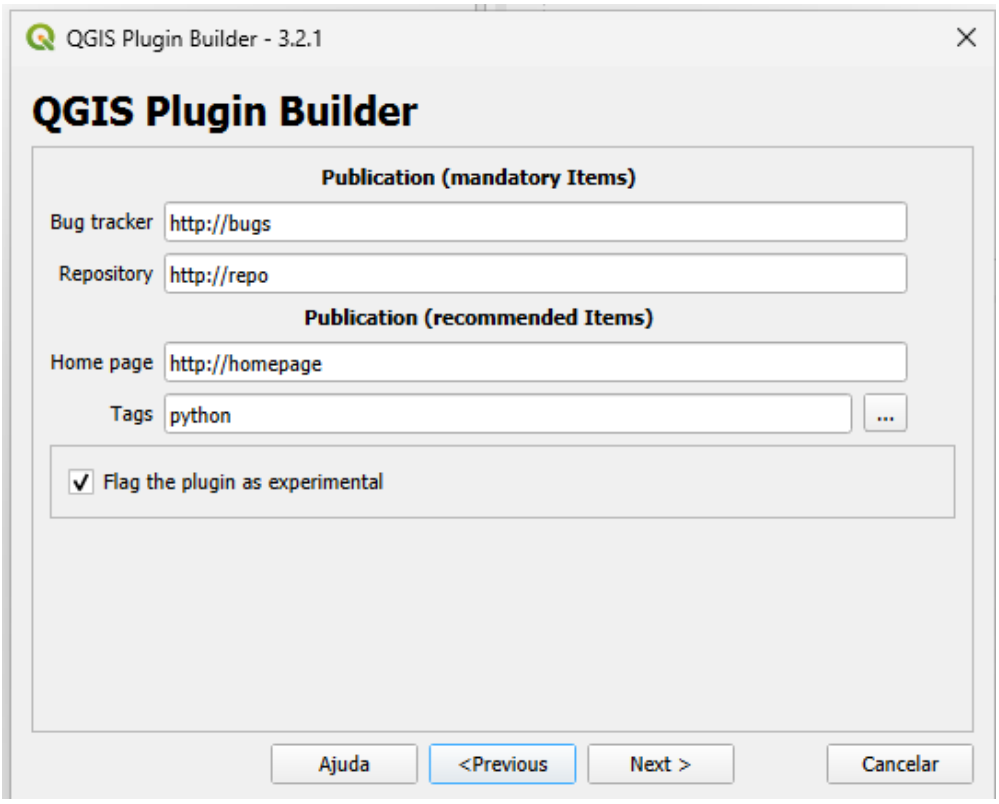
Descrição mais detalhada sobre o plugin que também será colocado no arquivo metadata.txt.



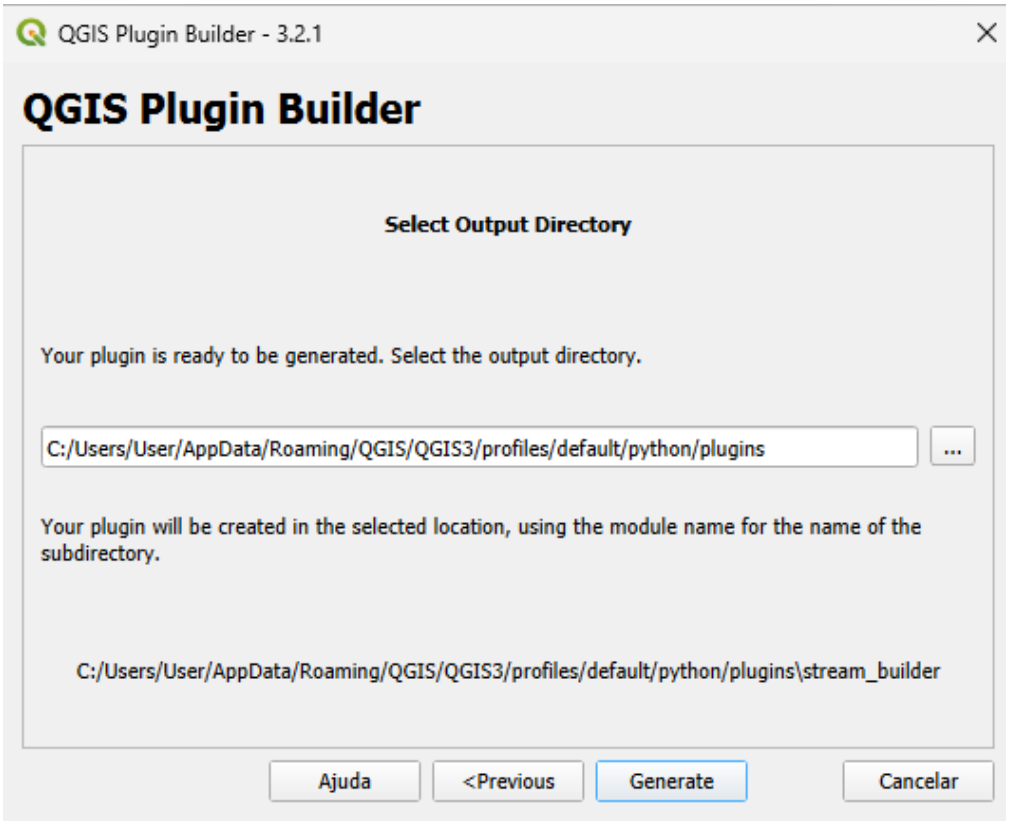
Template (tipo) do plugin, texto que vai aparecer no menu e em qual menu será listado o plugin.



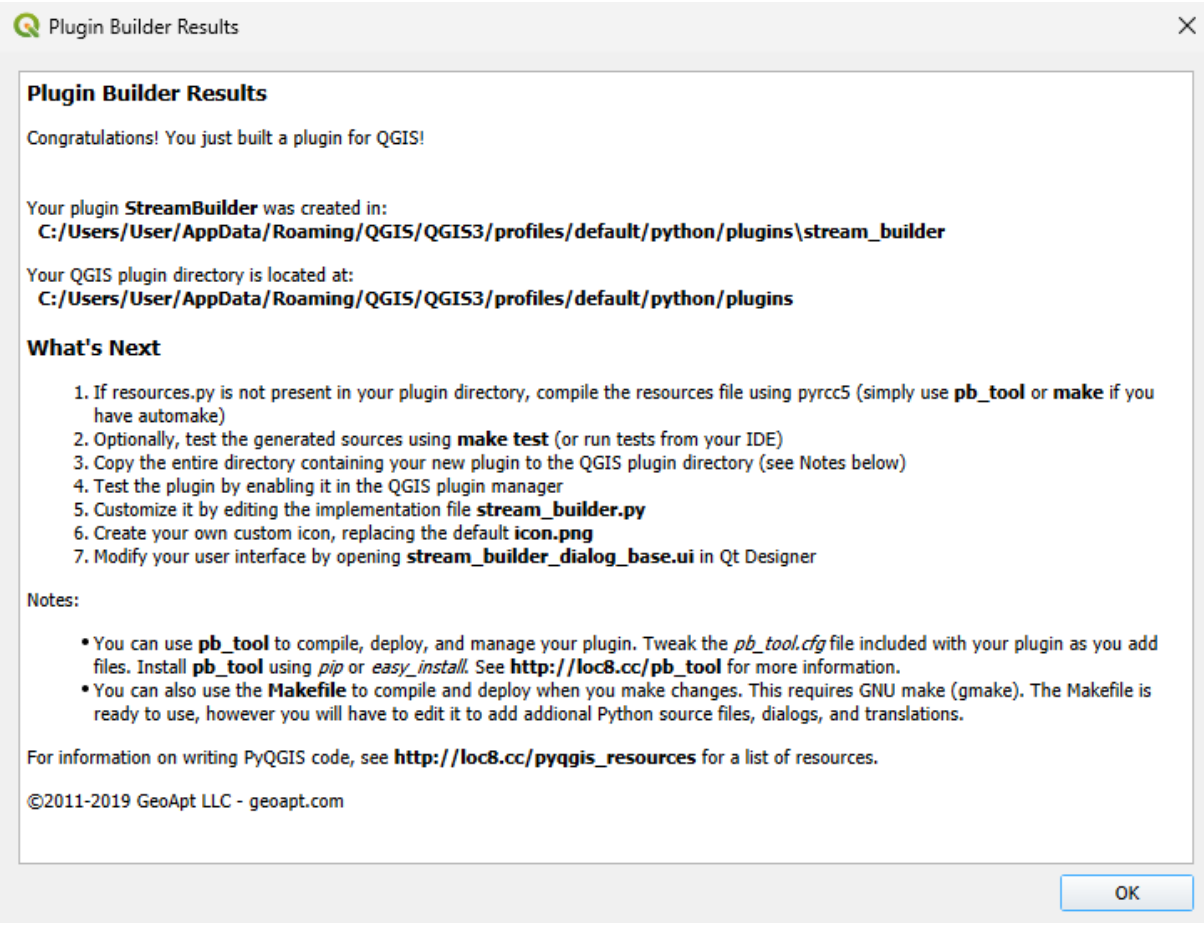
Desmarque todos para esse plugin,



Cheque a caixa de plugin experimental pois não iremos distribuir esse plugin no momento.

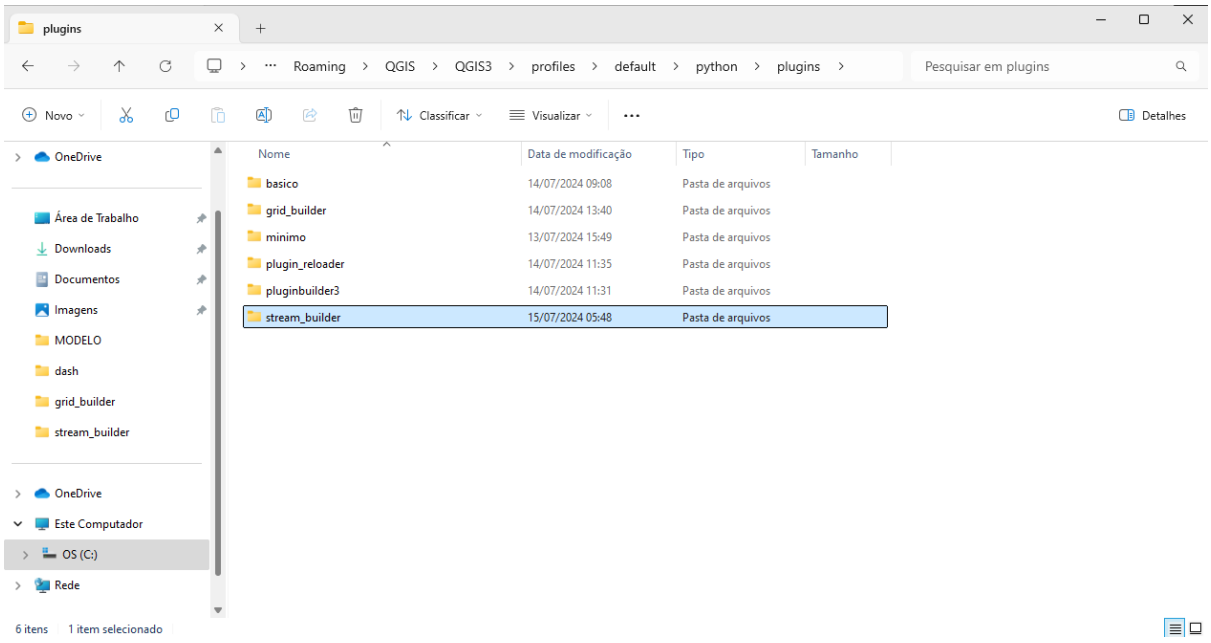


A pasta de plugin do sistema (nesse caso em sistema Windows). Clique **Generate** após selecionar o diretório de plugins.

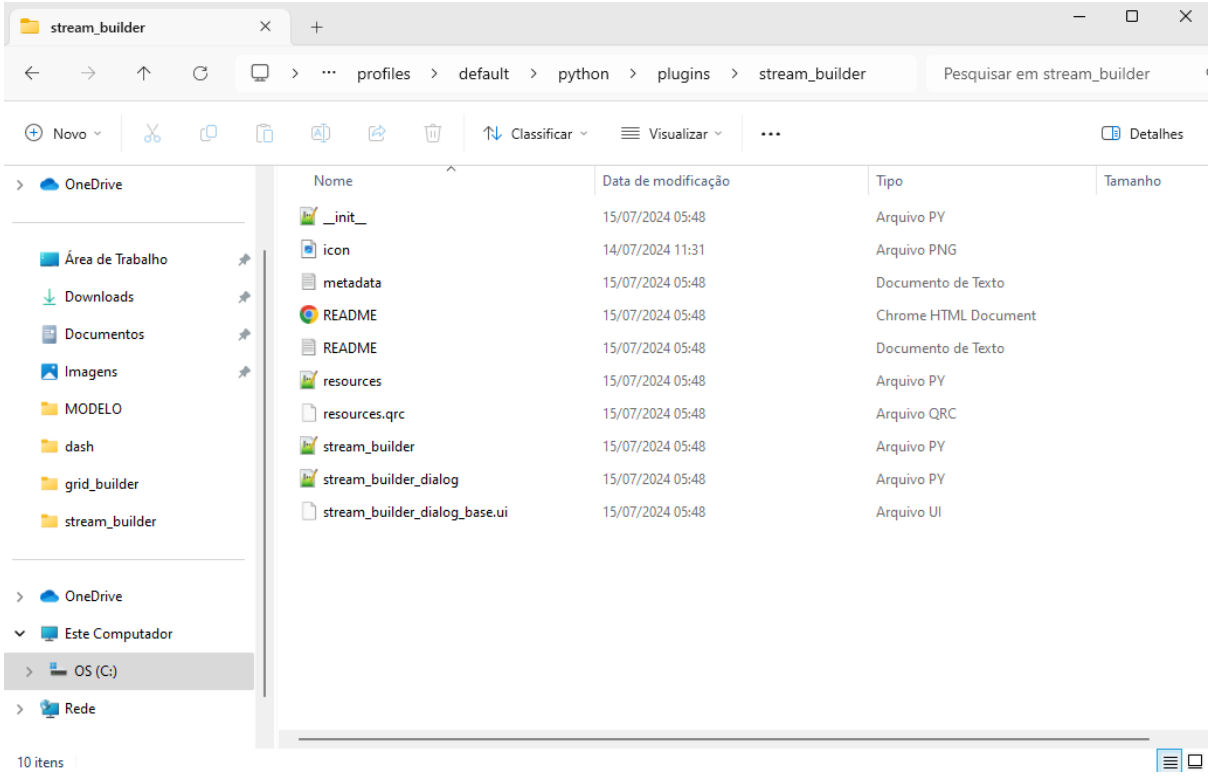


Pronto, os arquivos base de seu plugin foram criados na pasta:

C:/Users/User/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins\stream_builder

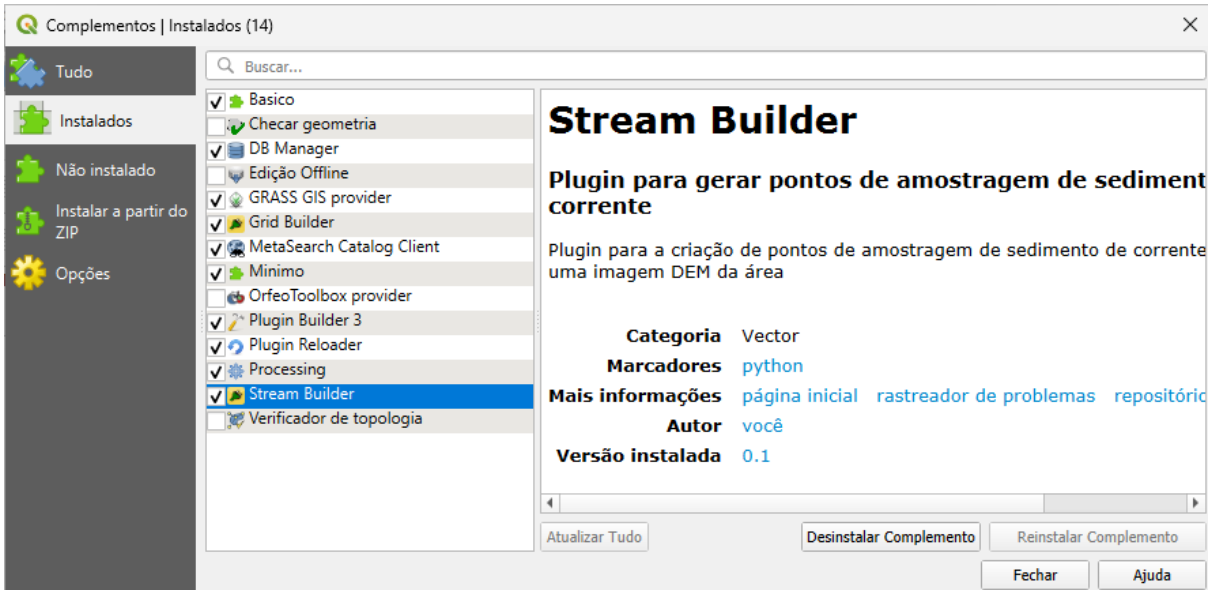


Os arquivos gerados:

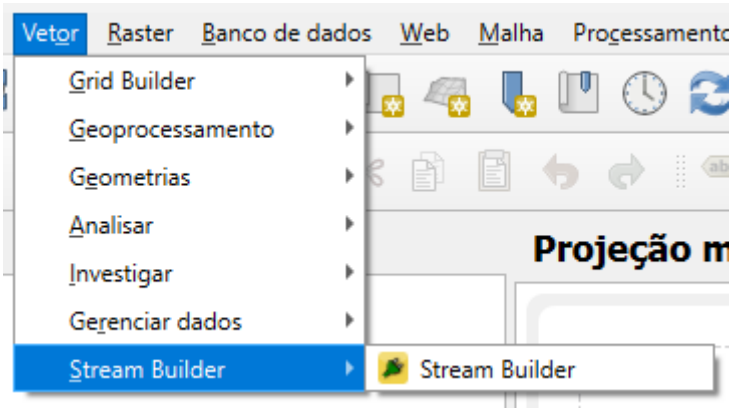


Os oito arquivos necessários mais dois arquivos README com instruções do PluginBuilder foram criados automaticamente.

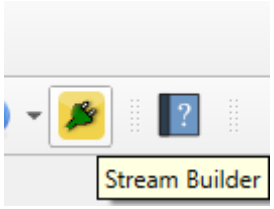
Vamos testar ele iniciando o QGIS e abrindo o **Complementos->Gerenciar e instalar Complementos**. Em Instalados vemos que ele não foi instalado ainda. Marque ele e instale para testarmos.



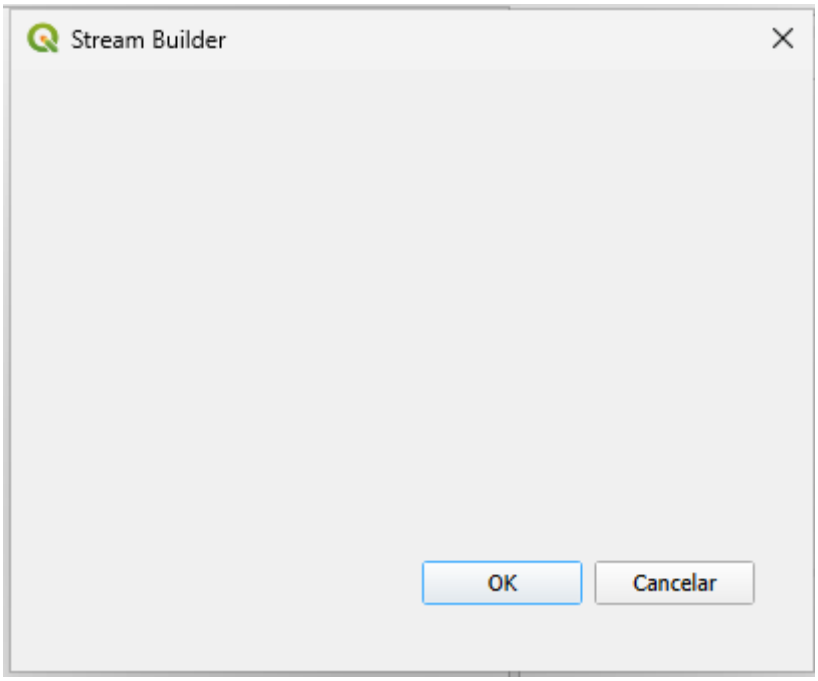
Inicie ele pelo Menu Vetor.



Ou pelo ícone na barra de ferramentas.



Funcionando, mas sem funcionalidade ainda.



Vamos construir a interface gráfica do usuário (GUI) e adicionar a funcionalidade agora na próxima seção.

3 - O plugin Stream_Builder

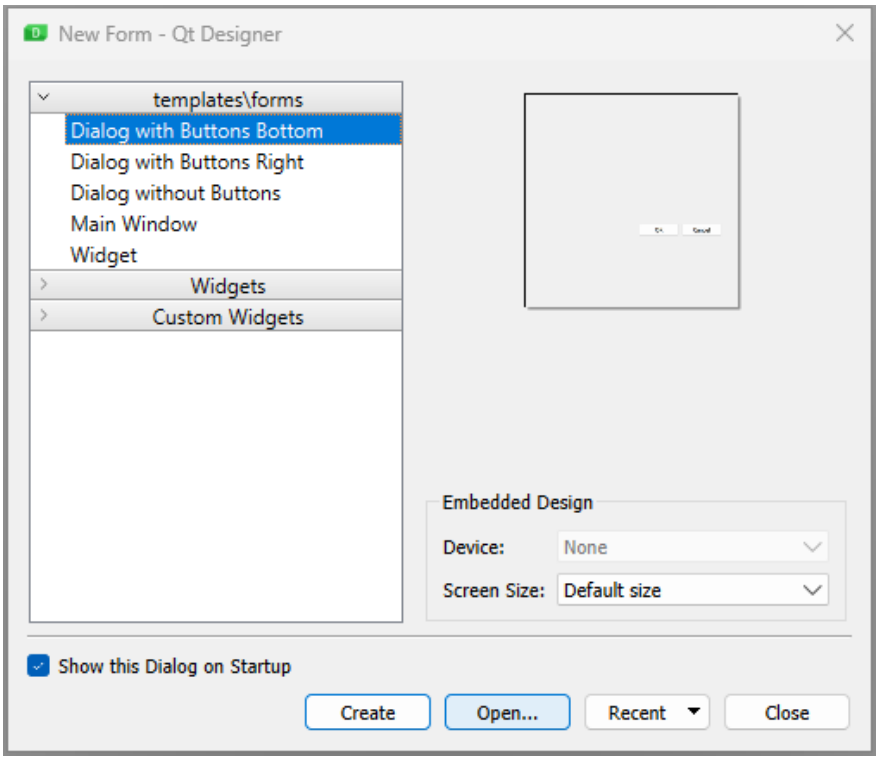
O plugin Stream Builder, executa a criação de pontos de amostragem de sedimento de corrente com base em uma imagem DEM em lat-long da área que deverá ser fornecida.

Abrir QtDesigner para criarmos a nossa interface gráfica de usuário (GUI).

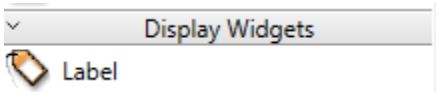
Abra o arquivo **stream_builder_dialog_base.ui** localizado em:

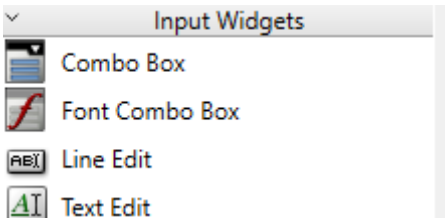
C:\Users\User\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\stream_builder

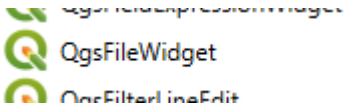
No primeiro diálogo do programa em **Open.**



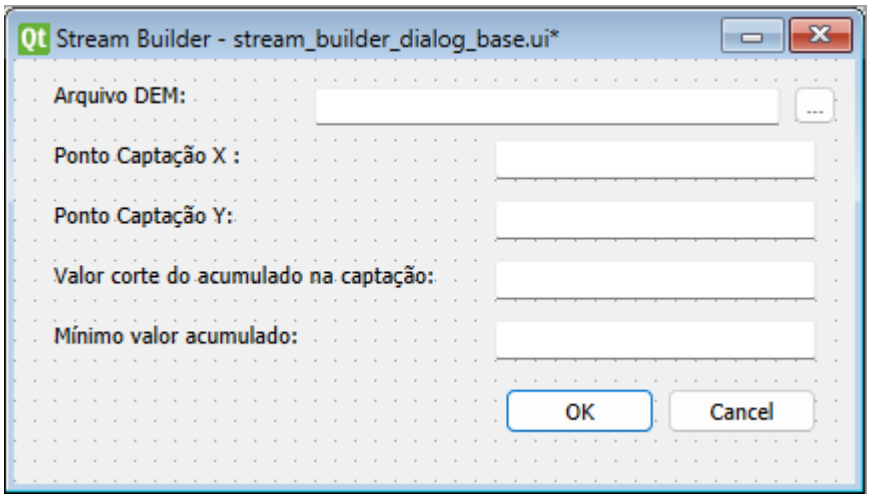
Vamos adicionar 5 widgets do tipo Label, 4 widgets do tipo Line Edit e um widget do tipo QgsFileWidget. Basta clicar no Widget e arrastar até a janela do diálogo.







A aparência final da interface deve ser:



Modifique o texto dos campos Label.

Deixo o campo QgsFileWidget inalterado.

Agora altere a propriedade objectName dos campos LineEdit na seguinte ordem:

lineEditX

lineEditY

lineEditSnap

lineEditAcc

Pronto, salve o diálogo e feche o QtDesigner.

Vamos agora editar o arquivo **stream_builder.py** para realizar a tarefa. Vamos ter de adicionar algumas bibliotecas de suporte via **import** e adicionar o código na função **run** que vai fazer a validação inicial dos campos e a criação da camada de pontos de amostragem de sedimento de corrente.

As bibliotecas serão (adicionar as faltantes):

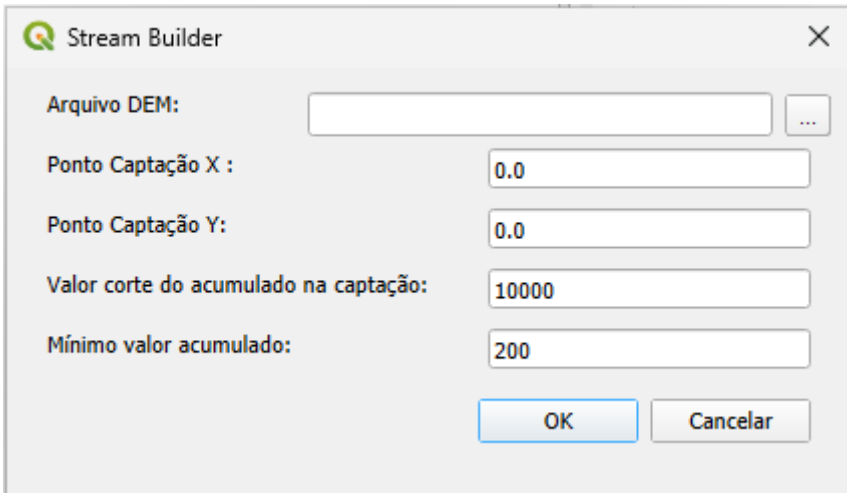
```
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication, QVariant
from qgis.PyQt.QtGui import QIcon, QIntValidator, QDoubleValidator
from qgis.PyQt.QtWidgets import QAction, QMessageBox
from qgis.core import QgsProject,  QgsRasterLayer, QgsMapLayerType, QgsWkbTypes,
QgsVectorLayer, QgsField
from .resources import *
from .stream_builder_dialog import StreamBuilderDialog
import os.path
import numpy as np
from pysheds.grid import Grid
from osgeo import gdal
from shapely import geometry, ops
import fiona
import itertools
import pandas as pd
import geopandas as gpd
import processing
```

A função **run** ficará assim:

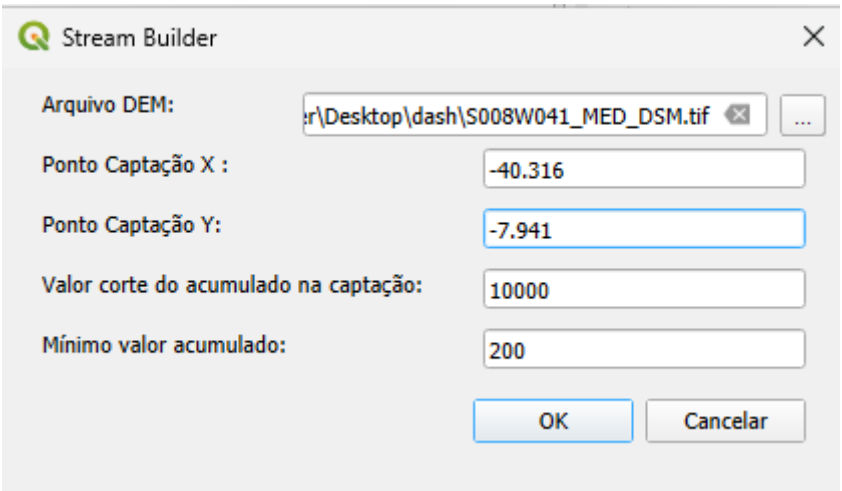
```
    def run(self):
        if self.first_start == True:
            self.first_start = False
            self.dlg = StreamBuilderDialog()
            onlyInt = QIntValidator()
            onlyDouble = QDoubleValidator()
            self.dlg.lineEditX.setText('0.0')
            self.dlg.lineEditX.setValidator(onlyDouble)
            self.dlg.lineEditY.setText('0.0')
            self.dlg.lineEditY.setValidator(onlyDouble)
            self.dlg.lineEditSnap.setText('10000')
            self.dlg.lineEditSnap.setValidator(onlyInt)
            self.dlg.lineEditAcc.setText('200')
            self.dlg.lineEditAcc.setValidator(onlyInt)

        self.dlg.show()
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            if self.dlg.lineEditY.text()=='' or self.dlg.lineEditX.text()=='' or
self.dlg.lineEditSnap.text()=='' or self.dlg.lineEditAcc.text()=='':
                QMessageBox.warning(self.iface.mainWindow(),
                        'Erro',
                        "Entrar todos os campo por favor \nSaindo...")
                return
            layer = self.dlg.mQgsFileWidget.filePath()
            y = float(self.dlg.lineEditY.text())
            x = float(self.dlg.lineEditX.text())
            snapp= float(self.dlg.lineEditSnap.text())
            accc= float(self.dlg.lineEditAcc.text())
            grid = Grid.from_raster(layer)
            dem = grid.read_raster(layer)
            pit_filled_dem = grid.fill_pits(dem)
            flooded_dem = grid.fill_depressions(pit_filled_dem)
            inflated_dem = grid.resolve_flats(flooded_dem)
            dirmap = (64, 128, 1, 2, 4, 8, 16, 32)
            fdir = grid.flowdir(inflated_dem, dirmap=dirmap)
            acc = grid.accumulation(fdir, dirmap=dirmap)
            x_snap, y_snap = grid.snap_to_mask(acc > snapp, (x,y))
            catch = grid.catchment(x=x_snap, y=y_snap, fdir=fdir,
dirmap=dirmap,xytype='coordinate')
            grid.clip_to(catch)
            clipped_catch = grid.view(catch)
            branches = grid.extract_river_network(fdir, acc > accc, dirmap=dirmap)
            schema = {
                'geometry': 'LineString',
                'properties': {}
            }

            with fiona.open('rivers.shp', 'w',
                driver='ESRI Shapefile',
                crs=grid.crs.srs,
                schema=schema) as c:
                i = 0
                for branch in branches['features']:
                    rec = {}
                    rec['geometry'] = branch['geometry']
                    rec['properties'] = {}
                    rec['id'] = str(i)
                    c.write(rec)
                    i += 1
            layeriv = QgsVectorLayer('rivers.shp', 'Drenagem', "ogr")
            QgsProject.instance().addMapLayer(layeriv)
            df = gpd.read_file('rivers.shp')
            s = pd.Series(df.geometry.values, index=df.index).to_dict()
            intersections = []
            for i in itertools.combinations(s, 2):
                i1, i2 = i
                if s[i1].intersects(s[i2]): #If they intersect
                    intersections.append([s[i1].intersection(s[i2])])

            dfinter = gpd.GeoDataFrame(pd.DataFrame(data=intersections, columns=['geometry']),
geometry='geometry', crs="EPSG:4326")
            dfinter.to_file('river_intersections.shp')
```

```
ptsau0 = QgsVectorLayer('river_intersections.shp', 'Pontos', "ogr")
parameters = {'INPUT': ptsau0,'OUTPUT': "temp.shp"}
processing.run('native:deleteduplicategeometries', parameters)
ptsau = QgsVectorLayer('temp.shp', 'Pontos', "ogr")
ptsau.startEditing()
id_idx = ptsau.fields().lookupField('FID')
ptsau.renameAttribute(id_idx, 'id')
ptsau.commitChanges()
QgsProject.instance().addMapLayer(ptsau)
QMessageBox.information(self.iface.mainWindow(),
             'Pronto',
             "Pontos de amostragem para sedimento de corrente criados!")
return
```

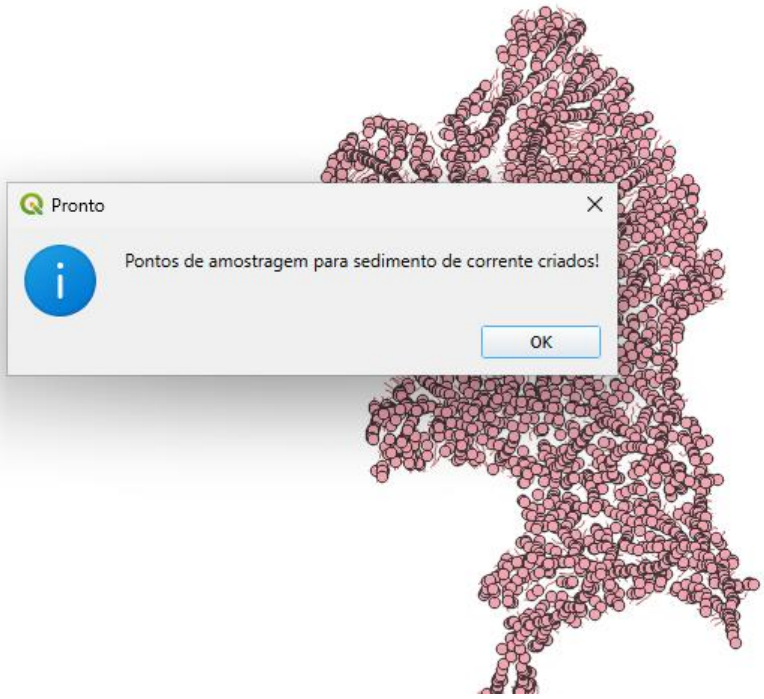
Antes de comentarmos o código adicionado vamos testar o plugin.
Baixe o arquivo DEM em https://gdatasystems.com/pyqgis/index.php **NOTA: O DEM deve estar em Latiyude-Longitude.**
Abra o QGIS e o plugin será carregado já com as alterações feitas. Ao iniciarmos o plugin teremos:



Execute o plugin com os seguintes parâmetros. A posição X e U da capitação deve ser sobre uma drenagem e a montante desse ponto será feita a análise:

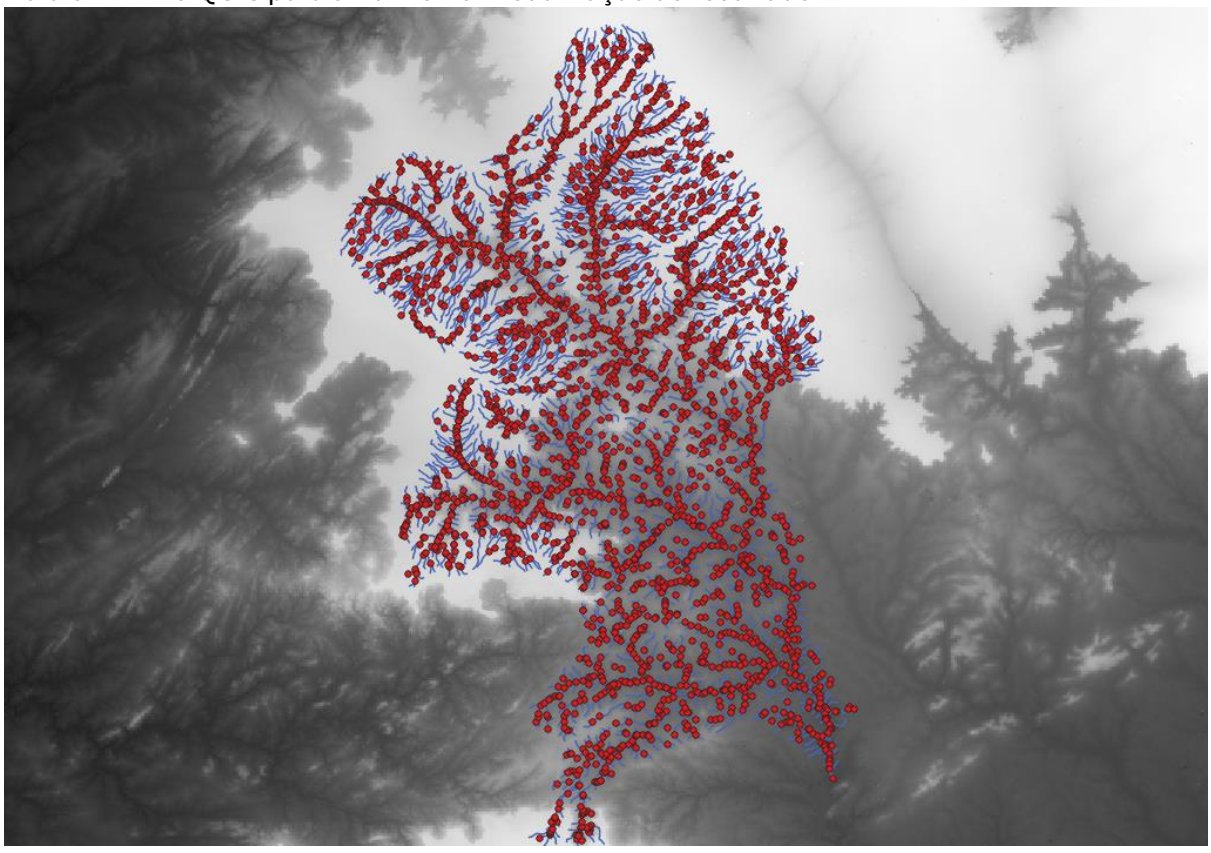


O processamento demora uns dois minutos e ao final teremos a mensagem:

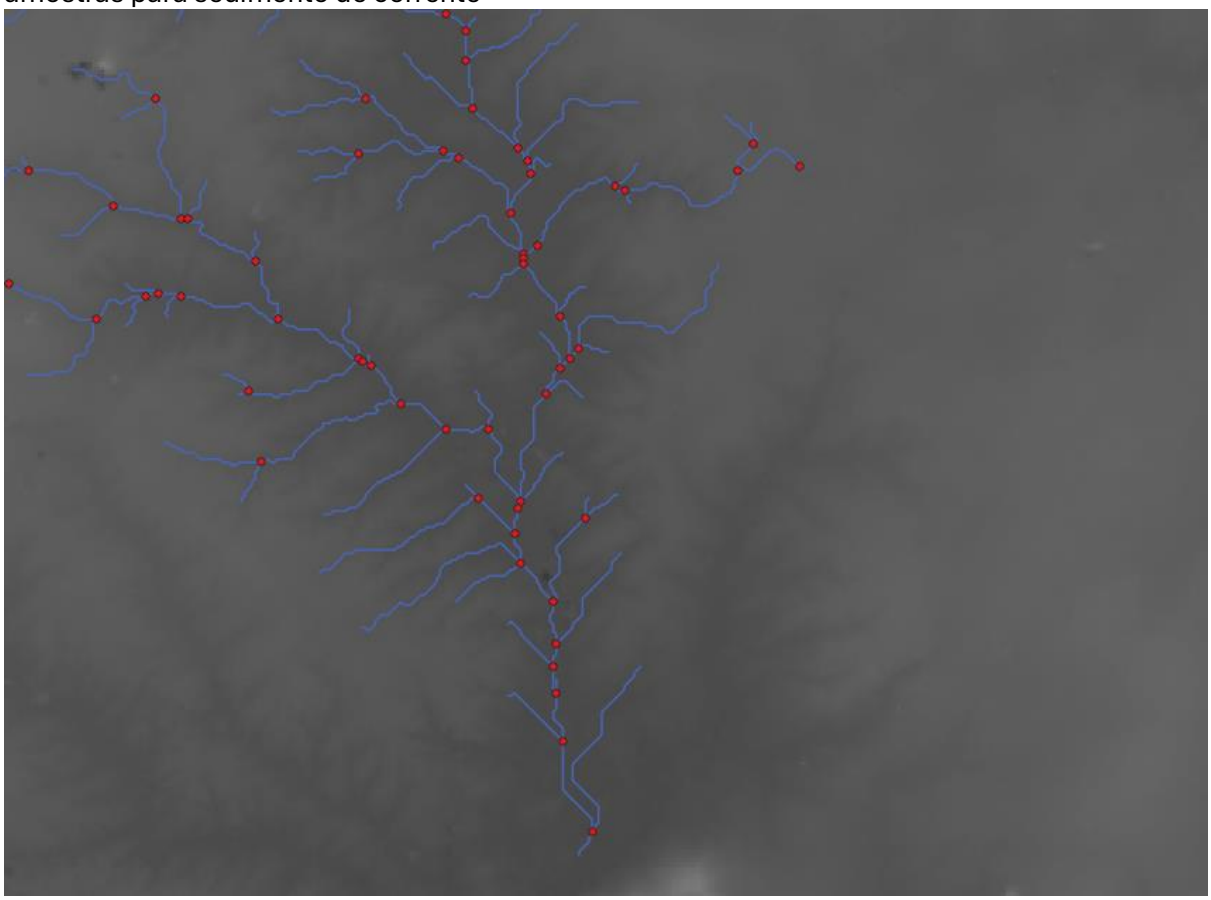


Agora, caso precise, é só salvar as camadas criadas na projeção desejada (drenagens e pontos de amostragem nas bifurcações).

Abra o DEM no QGIS para uma melhor visualização do resultado.



Zoom mostrando a drenagem extraída e os pontos de bifurcações para auxiliar na coleta de amostras para sedimento de corrente



Este bloco do código inicializa os widgets adicionando os valores iniciais nas linhas editáveis.

```
        if self.first_start == True:
            self.first_start = False
            self.dlg = StreamBuilderDialog()
            onlyInt = QIntValidator()
            onlyDouble = QDoubleValidator()
            self.dlg.lineEditX.setText('0.0')
            self.dlg.lineEditX.setValidator(onlyDouble)
            self.dlg.lineEditY.setText('0.0')
            self.dlg.lineEditY.setValidator(onlyDouble)
            self.dlg.lineEditSnap.setText('10000')
            self.dlg.lineEditSnap.setValidator(onlyInt)
            self.dlg.lineEditAcc.setText('200')
            self.dlg.lineEditAcc.setValidator(onlyInt)

        self.dlg.show()
        result = self.dlg.exec_()
```

O bloco seguinte checa se os campos estão todos preenchidos, assinala eles às variáveis do programa e lê o arquivo DEM.

```
            if self.dlg.lineEditY.text()=='' or self.dlg.lineEditX.text()=='' or
self.dlg.lineEditSnap.text()=='' or self.dlg.lineEditAcc.text()=='':
                QMessageBox.warning(self.iface.mainWindow(),
                         'Erro',
                         "Entrar todos os campo por favor \nSaindo...")
                return
            layer = self.dlg.mQgsFileWidget.filePath()
            y = float(self.dlg.lineEditY.text())
            x = float(self.dlg.lineEditX.text())
```

```
            snapp= float(self.dlg.lineEditSnap.text())
            accc= float(self.dlg.lineEditAcc.text())
            grid = Grid.from_raster(layer)
            dem = grid.read_raster(layer)
```

Aqui extraímos a drenagem do dem e criamos a camada **Drenagem**.

```
            pit_filled_dem = grid.fill_pits(dem)
            flooded_dem = grid.fill_depressions(pit_filled_dem)
            inflated_dem = grid.resolve_flats(flooded_dem)
            dirmap = (64, 128, 1, 2, 4, 8, 16, 32)
            fdir = grid.flowdir(inflated_dem, dirmap=dirmap)
            acc = grid.accumulation(fdir, dirmap=dirmap)
            x_snap, y_snap = grid.snap_to_mask(acc > snapp, (x,y))
            catch = grid.catchment(x=x_snap, y=y_snap, fdir=fdir,
dirmap=dirmap,xytype='coordinate')
            grid.clip_to(catch)
            clipped_catch = grid.view(catch)
            branches = grid.extract_river_network(fdir, acc > accc, dirmap=dirmap)
            schema = {
                'geometry': 'LineString',
                'properties': {}
            }
```

Finalmente criamos os pontos de amostragem nas bifurcações e criamos a camada de pontos **Pontos**.

```
    with fiona.open('rivers.shp', 'w',
                driver='ESRI Shapefile',
                crs=grid.crs.srs,
                schema=schema) as c:
                i = 0
                for branch in branches['features']:
                    rec = {}
                    rec['geometry'] = branch['geometry']
                    rec['properties'] = {}
                    rec['id'] = str(i)
                    c.write(rec)
                    i += 1
            layeriv = QgsVectorLayer('rivers.shp', 'Drenagem', "ogr")
            QgsProject.instance().addMapLayer(layeriv)
            df = gpd.read_file('rivers.shp')
            s = pd.Series(df.geometry.values, index=df.index).to_dict()
            intersections = []
            for i in itertools.combinations(s, 2):
                i1, i2 = i
                if s[i1].intersects(s[i2]): #If they intersect
                    intersections.append([s[i1].intersection(s[i2])])

            dfinter = gpd.GeoDataFrame(pd.DataFrame(data=intersections, columns=['geometry']),
geometry='geometry', crs="EPSG:4326")
            dfinter.to_file('river_intersections.shp')
            ptsau0 = QgsVectorLayer('river_intersections.shp', 'Pontos', "ogr")
            parameters = {'INPUT': ptsau0,'OUTPUT': "temp.shp"}
            processing.run('native:deleteduplicategeometries', parameters)
            ptsau = QgsVectorLayer('temp.shp', 'Pontos', "ogr")
            ptsau.startEditing()
            id_idx = ptsau.fields().lookupField('FID')
            ptsau.renameAttribute(id_idx, 'id')
            ptsau.commitChanges()
            QgsProject.instance().addMapLayer(ptsau)
            QMessageBox.information(self.iface.mainWindow(),
                        'Pronto',
                        "Pontos de amostragem para sedimento de corrente criados!")
                    return
```

No próximo vamos criar um plugin que efetua o desurvey de dados de furo de sondagem. Até lá!

Referência do pySheds

```
@misc{bartos_2020,
  title = {pysheds: simple and fast watershed delineation in python},
  author = {Bartos, Matt},
  url = {https://github.com/mdbartos/pysheds},
  year = {2020},
  doi = {10.5281/zenodo.3822494}
}
```