Work Package:5

Deliverable 5.1.1: Preparation of software used for the studies and the monitoring

Beneficiary: AUTH

INTERREG V-A COOPERATION PROGRAMME:

GREECE – BULGARIA 2014-2020

The Project is co-funded by the European Regional Development Fund and by national funds of the countries participating in the Interreg V-A “Greece - Bulgaria 2014-2020” Cooperation Programme.

About this document:

This deliverable presents the actions taken for the creation of a software, which is used for gathering, manipulating and presenting the data related to the pumps measurements.

Deliverable Version:v. 0.8 (Draft)

Date:18/06/2020

Deliverable Contributors:

1. Konstantinos Angelou

Has been reviewed and approved by: Michalis Maragakis

Table of Contents

Introduction

The Green Pump project purpose is to conserve water and energy by reducing the use of potable water in secondary uses, such as toilets and gardening, but also preserving the quality of the water. Its aim is to optimize the exploitation of row ground waters, in urban areas for secondary water uses, and also for heating purposes, while at the same time protecting basements of public buildings from flooding. In addition, by using row ground waters the need for chlorination will be reduced, which has impact on the environment. For this purpose, there will be 4 pilot pump constructions in Thessaloniki, Blagoevgrad, Petrich and Pylaia, while already existing pumps will be used for the study from TEICM and EMATECH in Serres and Drama respectively.

As there are 6 different type of pumps, it is very likely that the data and measurement that each one will produce will vary in type of data, format or even measurement units. However, it would be very useful for comparison and presentations purposes that all these data where of the same type. For this reason, a software needs to be developed that will homogenizes all the different pump measurements and will create a unified database.

It has been decided, that among all the various programming language that exist and can handle the required task, “Python” is the most suitable. Its advantage is that it provides all the necessary libraries that required to maintain a database, as well as libraries to handle and manipulate data and create graphs. The software that has been produced is located in one of the computers acquired for the current project and it will run on demand, each time new data is acquired. There is no point for the software to run constantly, as there is no continuous update of the data.

In a quick look, the software will download the data, keep the desired values, create a unified database, create graphs and upload them in a server so that they are available for the project website as well as publicly.

Software processes

Before running the software, each beneficiary with a pump is responsible for acquiring their pumps data and providing them to the lead beneficiary who will upload them on a specific folder in google drive that contains all the measurements. It is also possible for each beneficiary to directly upload their data to avoid any problems that may occur from the communication and sending of files.

Downloading and reading the data

The first task that the software does is to be able to access google drive. This is accomplished by using the “pydrive”, a python library that is making it possible to connect to google drive. Initially, it uses the administrator credentials in order to sign in the folder created especially for the Green Pump project. Next, it navigates through the folders in order to reach to the proper one. There exist folders for each of the beneficiaries where the data from the pumps are stored. After reaching the database

folder it searches through all the files in order to find new or updated files that have not been read before. Then, if there are new data, it downloads them in order to append them in the already created database.

Creating a unified database

After the new data have been downloaded, they need to be appended in the unified database. At first, the "pandas" python library will be used in order to read the data. In addition, the data may need to be edited or calculations may be needed to be done, in which cases pandas offer many useful tools. When all the necessary changes have been done, the data will be appended in the unified database.

For this purpose, we will use an SQL database by using "sqlite3" python library. The sql language has been selected, as it is one of the most common languages in manipulating data and it is widely used. If the database does not already exist (only the first time of execution), then two tables are created. The one holds information on the Universities such as their names and their location, while the other table holds the data. More specifically, it holds information derived from the pumps, such as the date of measurements, the energy consumed by the pump, the water temperature, and others. If the database exists, then it appends any new data that may exist.

The insertion process is different for each pump, as the data provided vary between them because the pumps are not the same. For this purpose, 6 different functions have been created, one for each beneficiary (with pumps). Each of these functions takes the respective data and converts them in proper format for the unified database. As a result, the final form of the database has the exact same format.

Manipulating data

After having created a unified database, there will be options to view the data. The user may be able to select to view the data for a specific location/pump or even for all of them. There will also be an option to extract these values into a text. In addition, there will be an option to create plots to present the values (e.g. energy consumption) of each pump, or even compare the pumps to each other. For the creation of figures, the "matplotlib" python library will be used, which offers a wide variety of tools to create and edit figures.

Uploading the database

Finally, after having created the unified database and any desired plots, they are exported in files. The unified database is then uploaded automatically to the Green Pump google drive folder, so that all the beneficiaries can access it. In addition, having the database uploaded to cloud allows for the website to present the updated version of the data automatically, whenever they are created/updated. This way, it also allows for the dissemination of our data to any interested party.

Conclusions

To sum up, a software has been developed that gathers the data from all beneficiaries with pump installations (from a common folder in Google Drive). Next, it defines the origin of the data (where the pump is located) and it appends the separated data to a unified SQL database. After the unified database has been created there will be options to view specific data (pump, location, measurement) and create plots. Finally, the unified database (local file) is being updated and immediately uploaded to Google Drive, where it is accessible by all beneficiaries and by where the website (www.greenpump.eu) is drawing the sample of data (for the tab “GreenPump Data”).

Appendix

The software that has been developed (and is still being developed) is presented here.

Main class

This is the main module of the software.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May  8 11:46:40 2020
@author: Angelou Konstantinos
Version: 0.9
Not full software. The real data are missing that
are useful for the completion of the software.
"""
import drive as gd
import mysql as ms
import matplotlib.pyplot as plt
import pandas as pd
import glob

drive = gd.authenticateGDrive()

sql_create_universities_table = """ CREATE TABLE
IF NOT EXISTS Universities (
                        id integer PRIMARY KEY,
                        name text NOT NULL,
                        place text NOT NULL
                    ); """

sql_create_info_table = """CREATE TABLE IF NOT
EXISTS Info (
                id integer PRIMARY KEY
AUTOINCREMENT,
                uni_id integer NOT NULL,
                date text NOT NULL,
                pump text,
                power real,
                FOREIGN KEY (uni_id) REFERENCES
Universities (id)
            );"""

database = r"database.db"

# create a database connection
conn = ms.create_connection(database)

# create tables
if conn is not None:
    # create projects table
    ms.create_table(conn,
sql_create_universities_table)

    # create tasks table
    #ms.create_table(conn,
sql_create_places_table)
    ms.create_table(conn, sql_create_info_table)
else:
    print("Error! cannot create the database
connection.")

with conn:
    # create participants
    ms.create_Participant(conn, (1, 'AUTh',
'Thessaloniki'))
    ms.create_Participant(conn, (2, 'SWU',
'Blagoevgrad'))
    ms.create_Participant(conn, (3, 'TEICM',
'Serres'))
    ms.create_Participant(conn, (4, 'EMATECH',
'Kavala'))
    ms.create_Participant(conn, (5, 'M.Pylaia',
'Pylaia'))
    ms.create_Participant(conn, (6, 'M.Petrich',
'Petrich'))
    ms.create_Participant(conn, (7,
'IHU','Thessaloniki'))

file_list = drive.ListFile({'q': "'root' in
parents"}).GetList()
folderid = ''
# Print all files and folders and keep the id of the
desired folder
for file1 in file_list:
    print('title: {}, id: {}, type: {}'.format(file1['title'],
file1['id'], file1['mimeType']))
    if file1['title']=='Databases':
        folderid = file1['id']

DatabasesID = gd.getId(file_list, 'Databases')
```

```
DatabasesID = '1Ei-
M2VuhnMymYJ7yLAhQBQ1WdsbZpF-Z'
dbfiles = gd.folder_files(drive, DatabasesID)

def importSerres(conn, filename):
   data = pd.read_csv(filename, ',', dtype=str)
   cols = data.columns
   for index, row in data.iterrows():
      ms.create_Info(conn,
(3,row['Timestamp'],'Inside',row[cols[1]]))
      ms.create_Info(conn,
(3,row['Timestamp'],'Outside',row[cols[2]]))

pumps = ['Thessaloniki', 'Serres', 'Petrich', 'Kavala',
'Blagoevgrad']
for city in pumps:
   print(city)
   cityID = gd.getId(dbfiles, city)
   cityFiles = gd.folder_files(drive, cityID)
   if len(cityFiles)==0:
      continue
   else:
      print(city)
      # get already downloaded files
      files=list(glob.glob('tmp/{}/*'.format(city)))
      # files to download
      newfiles = []
      for file in cityFiles:
         #print(file)
         # check which need to be downloaded
         if (file['title'] not in files) and (file['title'][-
3:]=='csv') and ('day' in file['title']):
            print(file['title'])
            gd.get_from_drive(drive, file['id'],
file['title'], 'tmp/{}'.format(city), 'csv')
            newfiles.append(file['title'])
      for file in sorted(newfiles):
         if city=='Serres':
            if 'day' in file:
               importSerres(conn,
'tmp/{}/{}'.format(city,file))

cur = conn.cursor()
cur.execute("SELECT * FROM Universities;")
print(cur.fetchall())
'''cur.execute("SELECT * FROM Places;")
print(cur.fetchall())
cur.execute("SELECT Universities.name,
Places.name, uni_id FROM Places,Universities
WHERE Universities.id==3 AND Places.uni_id==3;")
print(cur.fetchall())

cur.execute("SELECT * FROM Info;")
print(cur.fetchall())

cur.execute("SELECT Universities.name,
Places.name, Info.date, Info.pump, Info.power
FROM Universities id Places uni_id Info uni_id JOIN
;")
print(cur.fetchall())
'''

cur.execute("SELECT Universities.name,
Universities.name, Info.date, Info.pump,
Info.power FROM Info JOIN Universities ON
Info.uni_id=Universities.id ;")
print(cur.fetchall())

#SELECT n.firstname, l.latitude, l.longitude FROM
Names n JOIN Location l ON n.id = l.id WHERE
n.id=12

df = pd.read_sql_query("SELECT
Universities.name, Universities.place, Info.date,
Info.pump, Info.power FROM Info JOIN
Universities ON Info.uni_id=Universities.id ;",
conn)
df.to_csv('temp_db.txt', sep=' ', encoding='utf-8',
index=False, header=['University', 'City', 'Date',
'Pump', 'Power(kWh)'])
gd.UploadToDrive(drive, 'Databases', DatabasesID,
'1LvzNY_RVp1bhbKth5vgV9hWcuZf6KXk2',
'temp_db.txt')

df['date'] =  pd.to_datetime(df['date'], format='%Y-
%m-%d %H:%M:%S')
df2 =
df.sort_values(by=['date']).reset_index(drop=True)
df3 = df2.groupby(['uni_id', 'pump'])

for name, group in df3:
   print(name, group)
   tmpdf = group.reset_index(drop=True)
   tmpdf['cumulativePower'] =
tmpdf['power'].cumsum()
   fig, ax = plt.subplots()
   ax.plot(tmpdf['date'], tmpdf['cumulativePower'])
```

```
    ax.set_xlabel('Date')
    ax.set_ylabel('Cumulative Power (kWh)')
    ax.set_title('University:{},
pump:{}'.format(name[0],name[1]))

df4 = df3.get_group((3,
'Outside')).reset_index(drop=True)
df4['cumulativePower'] = df4['power'].cumsum()
fig, ax = plt.subplots()
ax.scatter(df4['date'], df4['cumulativePower'])

df4.to_csv('outsidepump.txt', sep=' ',
encoding='utf-8', index=False)
```

Google Drive functions

This module handles all the tasks relative to google drive. More specifically it handles the downloading, and the uploading of the files.

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from apiclient import discovery
from httplib2 import Http
import oauth2client
from oauth2client import file, client, tools

def authenticateGDrive():
    '''Authenitcate and save the credentials'''
    gauth = GoogleAuth()
    # scope = ['https://www.googleapis.com/auth/drive']
    # gauth.credentials =
ServiceAccountCredentials.from_json_keyfile_name('pydrive-
acct.json', scope)
    # gauth.LoadClientConfigFile('../files/client_secrets.json')
    # Try to load saved client credentials
    gauth.LoadCredentialsFile("mycreds.txt")
    if gauth.credentials is None:
        # Authenticate if they're not there
        gauth.LocalWebserverAuth()
    elif gauth.access_token_expired:
        # Refresh them if expired
        gauth.Refresh()
    else:
        # Initialize the saved creds
        gauth.Authorize()
    # Save the current credentials to a file
    gauth.SaveCredentialsFile("mycreds.txt")
    return GoogleDrive(gauth)

def folder_files(drive, folder_id):
    '''List all files and folders in a given folder id.
        drive: Auth object.
        folder_id: folder id to begin listing.
    '''
    file_list = drive.ListFile({'q': "'{}' in
parents".format(folder_id)}).GetList()
    return file_list

def create_folder(drive, parent_folder_id, folder_name):
    """Create a folder at the given folder id location with a folder
name.
        drive: Auth object.
        parent_folder_id: folder location you would like this folder
created.
        folder_name: name of the folder.
    Returns:
        folder_id: of this newly created folder.
    """
    try:
        folder = drive.CreateFile(dict(
            title=folder_name,
            parents=[dict(id=parent_folder_id)],
            mimeType="application/vnd.google-apps.folder"
        ))
    except:
        raise('problem creating folder')
    try:
        folder.Upload()
    except ApiRequestError:
        pass
    return folder['id']

def get_from_drive(drive, idv, filename, folder='', ext=''):
    '''Download a file from Google Drive'''
    myfile = drive.CreateFile({'id': idv})
    if ext!='':
        if filename.split('.')[-1]==ext:
            if folder=='':
                myfile.GetContentFile(filename)
            else:
                myfile.GetContentFile('{}/{}'.format(folder,filename))

def create_file(drive, upload_folder_id, fileName):
    #Create new file in the upload_folder
    my_file = drive.CreateFile({"parents":  [{"kind":
"drive#fileLink","id": upload_folder_id}]})
    my_file.SetContentFile(fileName) #Set the content to the
taken image
    my_file.Upload() # Upload it
    #This is the download URL of the file
    print('Download URL: ' + my_file['webContentLink'])

def UploadToDrive(drive, folderName, folderID, fileID,
fileName):
    '''Upload a file in a specific folder'''
            #Name of the folder where I'd like to upload images
    upload_folder = folderName
    #Id of the folder where I'd like to upload images
    upload_folder_id = folderID

    #Check if folder exists. If not than create one with the given
name
    #Check the files and folers in the root foled
    file_list = drive.ListFile({'q': "'root' in parents"}).GetList()
    for file_folder in file_list:
        if (file_folder['title']==upload_folder) and
(upload_folder_id==file_folder['id']):
            #upload_folder_id = file_folder['id'] #Get the matching
folder id
            dbfiles = folder_files(drive, folderID)
            for fi in dbfiles:
                if fi['id']==fileID:
                    fi.SetContentFile(fileName)
                    fi.Upload()
                    print('file set')
                    break
            #create_file(drive, upload_folder_id, fileName)
            print( 'File is uploaded to EXISTING folder: ' +
file_folder['title'])

def getId(files, fname):
    for file in files:
        if file['title']==fname:
            return file['id']
```

```
def printFiles(files):
    for file in files:
        print(file['title'])
```

SQL functions

This module handles all the tasks relative to the data handling.

```
import sqlite3
from sqlite3 import Error

def create_connection(db_file):
    """ create a database connection to the
SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)
    return conn

def create_table(conn, create_table_sql):
    """ create a table from the create_table_sql
statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE
statement
    :return:
    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)

def create_Participant(conn, project):
    """
    Create a new project into the projects table
    :param conn:
    :param project:
    :return: project id
    """
    sql = ''' INSERT INTO
Universities(id,name,place)
              VALUES(?,?,?) '''
    cur = conn.cursor()
    try:
        cur.execute(sql, project)
    except:
        print('Already exists')
    conn.commit()
    return cur.lastrowid

def create_Place(conn, place):
    """
    Create a new place
    """

    sql = ''' INSERT INTO Places(name,uni_id)
              VALUES(?,?) '''
    cur = conn.cursor()
    try:
        cur.execute(sql, place)
    except:
        print('Already exists')
    conn.commit()
    return cur.lastrowid

def create_Info(conn, info):
    sql = ''' INSERT INTO
Info(uni_id,date,pump,power)
              VALUES(?,?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, info)
    conn.commit()
    return cur.lastrowid

def importSerres(conn, filename):
    data = pd.read_csv(filename, ',', dtype=str)
    data['Date'] = data['Timestamp'].str.split(' ',
n=1, expand=True)[0]
    data['Hour'] = data['Timestamp'].str.split(' ',
n=1, expand=True)[1]
    cols = data.columns
    for index, row in data.iterrows():
        create_Info(conn,
(3,row['Date'],row['Hour'],'Inside',-100.0,-
100.0,row[cols[1]]))
```

```
    create_Info(conn,
(3,row['Date'],row['Hour'],'Outside',-100.0,-
100.0,row[cols[2]]))
```

The contents of the report are the sole responsibility of AUTH and can in no way be taken to reflect the views of the European Union, the participating countries, the Managing Authority and the Joint Secretariat.