

# METEO2MAP

## DAS PREVISÕES METEOROLÓGICAS AOS SERVIÇOS DE MAPAS

**Docente**

Prof. Hugo Martins

**Discente**

João Carlos Hermenegildo Oliveira  
g20201184

Software Aberto e Programação em SIG  
Mestrado em Ciência e Sistemas de Informação Geográfica  
Ano letivo 2020/2021

# METEO2MAP: das previsões meteorológicas aos serviços de mapas

João H. Oliveira

Mestrando do curso de Ciência e Sistemas de Informação Geográfica, NOVA IMS  
g20201184@novaims.unl.pt

Documento submetido para avaliação a 23/05/2021

**Resumo:** Fruto do atual contexto tecnológico, trabalhar hoje com dados geográficos é uma tarefa extremamente aliciante, suportada por ferramentas que auxiliam eficaz e eficientemente a coleção, armazenamento, processamento e disponibilização de informação geográfica. Este projeto propõe-se a simular um contexto de produção de serviços de mapas meteorológicos, automatizando o processo em 3 momentos distintos: i) coleção de dados; ii) armazenamento de dados e iii) disponibilização do serviço de mapas. Para corresponder ao objetivo proposto são integradas várias tecnologias de âmbitos diferentes, destacando-se a importância da base de dados objeto-relacional (BD) *PostgreSQL* e a sua extensão *PostGIS* para lidar com dados geográficos, e o *Geoserver*, através do qual se realiza a disponibilização dos serviços de mapas. O programa (meteo2map) é totalmente baseado na linguagem *Python*, incluindo-se alguns módulos desenvolvidos pela comunidade, tais como *GeoPandas*, *psycpg2*, *geoserver-rest*, entre outros. Este projeto caracteriza-se por ser 100% *open source*.

**Palavras-chave:** *software* aberto; programação em SIG; meteorologia; bases de dados espaciais; serviços de mapas; *Python*



<https://github.com/joacholiveira/meteo2maps>

# 1.Introdução

O domínio de um vasto conjunto de ferramentas tecnológicas aliado ao sólido corpo teórico da Ciência da Informação Geográfica confere ao analista de Sistemas de Informação Geográfica (SIG) uma promissora capacidade para lidar com os mais variados problemas existentes dentro da temática. Em concreto, algumas das competências que poderão ser valorizadas atualmente estão relacionadas com as habilidades de coleção de dados, construção e gestão de bases de dados espaciais e sistemas de *big data*, desenvolvimento de ferramentas de geoprocessamento, análise de dados e construção de *websig*. Cada um dos âmbitos enunciados exige individualmente um grande esforço para que se atinja um nível de proficiência relevante, consistindo em tópicos de especialização que subsistem *per si*.

Este leque de competências técnicas apresentadas depende intrinsecamente da adoção das ferramentas certas e da sua inteligente manipulação. Em SIG existem três grupos essenciais que distinguem o âmbito destas soluções de *software*: i) ferramentas desktop – possibilitam a visualização, manipulação e análise de informação geográfica através de uma interface gráfica (e.g. QGIS, GRASS ou ArcGIS Pro); ii) sistemas gestores de bases de dados espaciais (SGBDE) – como o nome indica, auxiliam em tarefas de construção e manipulação de bases de dados com capacidade para acolher informação geográfica e iii) ferramentas servidor e *web*, que tornam possível a conceção de serviços de mapas e o desenvolvimento *front-end* para a difusão da informação.

Num ambiente tecnológico tão denso, é frequente a necessidade de customização de processos e de criação de fluxos de trabalho adequados a enunciados complexos. Esta conjuntura particular suscita a necessidade de automação de procedimentos computacionais de natureza variada. Quando uma instituição ou empresa responsável por desenvolver um serviço ou produto reformula a sua lógica de trabalho, automatizando um conjunto de tarefas encadeadas, propõe-se necessariamente a evoluir uma vez que existe libertação de recursos que poderão ser alocados noutras frentes. Neste sentido, as linguagens de programação desempenham um papel fundamental, agindo como uma ponte entre o intelecto humano e os sistemas computacionais, i.e., é através delas que se efetivam instruções tendo por base um conjunto de premissas sintáticas e semânticas, interpretáveis por computadores.

O presente trabalho enquadra-se como uma simulação simplificada de um contexto de produção, onde se implementou a interação automática de várias tecnologias que atomicamente atuam ao nível da recolha e gestão de dados, geoprocessamento e disponibilização de mapas. Especificamente objetivou-se o desenvolvimento de um *script* que automatiza a recolha dados meteorológicos com desagregação ao nível dos distritos de Portugal Continental, faz o seu processamento e armazenamento em base de dados (BD), e finalmente disponibiliza essa informação através de um serviço de mapas. Este pequeno programa foi desenhado de modo a integrar algumas decisões que podem ser especificadas pelo utilizador durante a sua execução; houve também preocupação em concebê-lo da forma mais resiliente possível, especificando soluções para todos os cenários de execução identificados.

É também importante referir que todo o projeto é suportado por *software open-source*. Esta escolha prende-se com vários motivos, podendo-se destacar a vontade pessoal em conhecer melhor este tipo de soluções, uma vez que o percurso de aprendizagem formal incentiva principalmente o desenvolvimento de competências em *software* proprietário.

## 2. Metodologia

O desenvolvimento deste projeto inicia-se com uma reflexão sobre questões práticas, levantadas pelo objetivo central anteriormente apresentado. Apenas desta forma seria possível ter uma visão clara sobre os *inputs* de dados, tecnologias e respetivos *outputs* do programa a implementar. Após alguma investigação, conclui-se que a forma mais adequada de coletar os dados meteorológicos seria fazendo recurso a uma API (*Application Programming Interface*) que disponibilizasse dados de forma confiável e gratuita. Várias foram as plataformas identificadas, contudo, apenas duas foram seriamente consideradas: *AccuWeather* e *OpenWeatherMap*. A escolha acabou por recair sobre a *OpenWeatherMap*, uma vez que se considerou que as suas API's respondiam de forma mais direta às exigências do projeto. Deste modo, procedeu-se à inscrição na plataforma (<https://openweathermap.org/api>) de modo a obter um *token* de acesso. Esta temática será abordada com maior detalhe à frente, recorrendo-se ao código desenvolvido. Quanto aos elementos de *software* utilizados, destaca-se em primeiro lugar o *QGIS*, que esporadicamente auxiliou na visualização de dados geográficos; *PostgreSQL* e a sua extensão *PostGIS* foram empregues na tarefa de armazenamento de dados, fazendo-se uso pontual do *DBeaver* para executar *queries* simples de verificação; e o *Geoserver* para proceder à conceção e reposição dos serviços de mapas. Por sua vez, o *Visual Studio Code* foi o IDE escolhido para apoiar o desenvolvimento de todo o código que se apresentará ao longo das próximas secções.

Para o desenvolvimento do *script* recorreu-se inteiramente à linguagem *Python* (versão 3). Esta é atualmente uma das linguagens mais utilizadas a nível mundial e caracteriza-se pelo seu alto nível (i.e., a especificação de instruções é realizada de forma próxima à linguagem natural), interatividade, interpretação por máquina virtual e orientação a objetos. O sucesso do *Python* pode ser eventualmente justificado por dois motivos: o primeiro deve-se à sua natureza *open source*, havendo um desenvolvimento orientado às necessidades da comunidade de utilizadores; o segundo motivo, por ser uma linguagem adequada a múltiplos âmbitos aplicacionais, tais como o desenvolvimento de software de sistema (*shell tools*), desenvolvimento web, programação de BD, integração de componentes, programação científica e numérica, inteligência artificial, etc. (Martins, 2019).

Para promover o entendimento facilitado do desenvolvimento do trabalho, optou-se por subdividi-lo em 5 etapas lógicas. Ao longo da apresentação de cada etapa haverá um foco nas principais particularidades do código, complementadas com as devidas justificações metodológicas. O desenvolvimento do código faz-se acompanhar de pequenos comentários (#) e de *docstrings* documentando a finalidade de cada função. Todo o encadeamento lógico do programa poderá ser acompanhado através do fluxograma presente no anexo 1.

### 2.1 Procedimentos iniciais

Este é o momento que marca o início do *script*, onde se deu lugar i) à importação dos módulos com utilidade imediatamente identificável, tais como o *os* e *sys* (sendo que os restantes foram incluídos à medida que a sua necessidade foi sentida); ii) armazenamento em memória do *timestamp* relativo à inicialização do programa, útil para quantificar o tempo de execução do mesmo; iii) especificação dos diretórios úteis às várias funções do programa, havendo a indicação direta da pasta da "raiz" do projeto (*basePath*) e da pasta destinada a acolher os vários *outputs* derivados (*outputPath*); iv) seguidamente é criada a função *getMessageString* (fortemente baseada num dos materiais da disciplina), cujo o objetivo é integrar uma numeração sequencial

em todas as impressões direcionadas ao utilizador, apresentadas através da consola; e finalmente v) é construída uma função cujo objetivo prende-se com a verificação da existência do diretório de *outputs*. Caso este teste lógico devolva o valor *False*, é feito recurso à função *os.mkdir()* para proceder à sua criação.

```
# METEO2MAP
# Made by João H. Oliveira (2021), as final project for Software Aberto e Programação
  em SIG

# Licensed under GPLv3
# camelCase style adopted
# ### Stands for VSC interactive execution directly on .py, like a .ipynb

###

import os, sys, urllib.request, json, geopandas, pandas,\
psycopg2, re, time, warnings
from datetime import datetime, timedelta
from psycopg2 import extras as psy2extras
from geo.Geoserver import Geoserver

# Execution time (start)
startTime = time.time()

# To disable geopandas CRS warnings in terminal
warnings.filterwarnings('ignore')

os.environ['SHAPE_ENCODING'] = "utf-8"
basePath = 'C:\\saprog\\projeto'
outputPath = 'C:\\saprog\\projeto\\output\\'
os.chdir(basePath)

# Counter support variable for getMessageString
global counter
counter = {'counter': 0}

def getMessageString(msg):
    '''
    Função desenhada para substituir o simples "print", adicionando-lhe
    uma contagem incrementável ao longo da execução do programa.
    Deverá ser empregue sempre que se pretender estabelecer contacto com o
    utilizador pela linha de comandos.
    '''
    counter['counter'] += 1
    print('\n', str(counter['counter']) + '. ' + msg)

getMessageString('Working on initial steps... Please, wait a moment.')

def outputDir():
    if os.path.exists(outputPath) == False:
        os.mkdir(outputPath)
        getMessageString('Output dir. created.')
    else:
        getMessageString('Found output dir.')

outputDir()
```

## 2.2 Geoprocessamento

Nesta secção é feito recurso ao módulo *GeoPandas*, sendo esta uma variação do famoso módulo *Pandas*, com a particularidade de se orientar à gestão e manipulação de informação geográfica. Em primeiro lugar é feita a verificação da existência da *shapefile* "distritos" dentro da pasta de outputs - esta existiria caso já tivesse havido uma execução anterior do programa, e neste caso não seria necessário voltar a produzi-la. Caso não exista, é feito recurso à *shapefile* "caop" previamente disponibilizada no diretório principal do projeto (*basePath*), carregada através do método *geopandas.read\_file()*, sendo posteriormente executada a operação *dissolve* baseada no campo "Distrito". A *geodataframe* guardada na variável *districts* necessita então de ser convertida para o sistema de coordenadas projetadas ETRS89 PT TM06 (epsg/crs 3763) através do método *.to\_crs()* e guardada no diretório base sob o nome "districts". De seguida é implementada a função *getCoordTogether* que produzirá um dicionário para congregar o nome dos distritos de Portugal continental (chaves), adicionando-lhes um tuplo (valor) que contenha um valor de latitude na primeira posição e um valor de longitude na segunda posição, e.g. {'Aveiro': (-8.46838, 40.72369), 'Beja': (-7.94388, 37.82970), ...}. Destaca-se o uso do método *set\_index()*, de modo a possibilitar a indexação do campo "Distrito" presente na *geodataframe* produzida anteriormente, seguindo-se a extração da coordenadas do centroide do distrito através do método *geometry.centroid.x.to\_dict()* e *geometry.centroid.y.to\_dict()*.

```
###

# Working on districts
if os.path.exists(outputPath+'districts.shp') == False:
    getMessageString('districts.shp not found. Let\'s work on it.')
    # Reading main CAOP shapefile (it needs to already exist)
    caop = geopandas.read_file('caop.shp', encoding='utf-8')
    # Executing dissolve from parishes to districts
    districts = caop.dissolve(by = 'Distrito')
    districtsEtrs = districts.to_crs(3763)
    # Saving dissolve output as shapefile (WGS) and reading it
    districts.to_file(outputPath+'districts.shp', encoding='utf-8')
    districtsEtrs.to_file(outputPath+'districtsetrs.shp', encoding='utf-8')
    districts = geopandas.read_file(outputPath+'districts.shp', encoding='utf-8')
else:
    getMessageString('Found districts.shp')
    # Reading districts shapefile that already exists
    districts = geopandas.read_file(outputPath+'districts.shp', encoding='utf-8')
    districtsEtrs = districts.to_crs(3763)
    # # districts = districts.to_crs(3763)

def getCoordTogether(geoDataFrame):
    """
    Extração de coordenadas geográficas úteis ao harvest de dados meteorológicos.
    Devolve um dicionário onde as chaves são os nomes dos distritos,
    associando-se como valores um tuplo com as coordenadas Lat Long.
    """
    pre = districts.set_index('Distrito')
    coordDicX = pre.geometry.centroid.x.to_dict()
    coordDicY = pre.geometry.centroid.y.to_dict()
    coord = [coordDicX, coordDicY]
    coordDic = {}
    for i in coordDicX.keys():
        coordDic[i] = tuple(coordDic[i] for coordDic in coord)
    return coordDic
```

```
coordDist = getCoordTogether(districts)
getMessageString('Districts centroid coordinates compiled.')
```

## 2.3 Preparação da base de dados

Na fase da preparação da base dados estabeleceu-se o primeiro contacto com a BD *PostgreSQL* previamente criada para o efeito (através da clonagem da base de dados *default PostGIS*), à qual se deu o nome “meteo”. O apoio do módulo *psycopg2* foi crucial para estabelecer interação com a BD, sendo este o módulo mais conhecido para gerir uma BD *Postgres* a partir da linguagem *Python*.

Para efeitos da preservação da segurança do user *postgres*, a *password* é carregada a partir de um ficheiro de texto previamente criado e reposto no diretório base. Após ter em memória a *string* da *password*, é criada a variável *conn* de modo a possibilitar a posterior conexão à BD. Esta variável *conn* articula os vários parâmetros necessários (*user*, *password*, nome da BD, *host* e porta), recorrendo para isso ao método *psycopg2.connect()*. Seguidamente são construídas duas funções com âmbitos semelhantes, mas ainda assim distintas: i) a primeira (*checkPgDistrictsTable*) responsabiliza-se por fazer a verificação da existência da tabela geográfica dos distritos dentro da BD. Em concreto, após ser estabelecida a conexão *cur*, é executada uma *querie* que se propõe a verificar se o nome da tabela especificada através dos parâmetros da função existe dentro do *public schema* da BD. Se a resposta a esta *querie* for o valor *False*, é então executado um bloco de código que carrega dentro da BD a *shapefile* “distritos”, anteriormente processada e guardada no diretório de *outputs*. Este carregamento é efetuado com recurso a um comando de sistema executado através do método *os.system()*, que por sua vez especifica a inicialização do programa *shp2pgsql* já com os respetivos parâmetros de carregamento da *shapefile* definidos; ii) a segunda função contruída nesta secção (*checkPgForecastTable*), objetiva a verificação da existência da tabela principal, aquela que armazenará todos os dados meteorológicos sem qualquer tipo de geometria associada. À semelhança da função *checkPgDistrictsTable*, começa-se por criar um cursor de conexão à base de dados, e caso a tabela especificada nos parâmetros da função não exista, é criada uma nova tabela a partir de um bloco de código SQL predefinido para o efeito, executado a partir do método *execute()* do *psycopg2*.

```
###

# Check the existence of districts table in meteo PG database
# Reading hidden PostgreSQL password from txt file in dir
pgPassword = open(os.path.join('postgresPw.txt'), 'r').readline()
pgPassword = str(pgPassword)

# Defining PostgreSQL connection parameters
conn = psycopg2.connect(dbname='meteo', user='postgres', password=pgPassword,\
host='localhost', port='5432')

def checkPgDistrictsTable(connectionParameters, table):
    """
    Função para verificação de boleana de tabela geográfica dentro de BD PostgreSQL.
    Carrega shapefile se a tabela não existir na BD.
    """
    cur = connectionParameters.cursor()
    cur.execute("select * from information_schema.tables where table_name=%s",\
(table,))
    # Checking for existence of districts table
    if bool(cur.rowcount) == False:
        # Uploading districts shapefile in database
```



```
command = 'shp2pgsql -s 3763 C:\\saprog\\projeto\\output\\  
districtsetrs.shp public.districtsetrs\\  
| psql -q -U postgres -d meteo -h localhost -p 5432'  
os.system(command)  
time.sleep(5)  
getMessageString('{} table uploaded in meteo database.'.format(table))  
else:  
    getMessageString('{} table already exists in meteo database.'.format(table))  
  
# Checking for existence of forecast table  
def checkPgForecastTable(connectionParameters, table):  
    '''  
    Função para verificação de presença/ausência de tabela dentro de BD PostgreSQL.  
    Devolve True se existir; False se não existir.  
    '''  
    cur = connectionParameters.cursor()  
    cur.execute("select * from information_schema.tables where table_name=%s",\  
                (table,))  
    # Checking for existence of districts table  
    if bool(cur.rowcount) == False:  
        query = "CREATE TABLE forecast(\nforecast_id SERIAL, distrito VARCHAR(80), forecast_date DATE,\nforecast_time TIME, weather_desc VARCHAR(80),\ntemperature FLOAT, feels_like FLOAT, pressure INT, humidity INT,\ndew_point FLOAT, wind_speed FLOAT, wind_deg INT,\nrequest_type VARCHAR(80), CONSTRAINT forecast_pkey PRIMARY KEY (forecast_id))"  
        cur.execute(query)  
        conn.commit()  
        cur.close()  
        getMessageString('{} table created into meteo database.'.format(table))  
    else:  
        getMessageString('{} table already exists in meteo database.'.format(table))  
  
# Checking for districts table  
checkPgDistrictsTable(conn, 'districtsetrs')  
  
# Checking for forecast table  
checkPgForecastTable(conn, 'forecast')
```

## 2.4 Extração e armazenamento de dados

Esta penúltima etapa concentra em si uma grande importância, na medida em que é nesta instância que se procede à recolha dos dados meteorológicos recorrendo às API da *OpenWeatherMap*. De modo a tornar o exercício mais interessante, achou-se conveniente integrar algum dinamismo neste secção específica do trabalho, deixando ao critério do utilizador o momento para o qual será feita a recolha dos dados meteorológicos. Em concreto, desenvolveu-se a função *requestType* cuja finalidade é interagir com o utilizador através da linha de comandos, pedindo-lhe que especifique um *input* limitado para indicar o momento para o qual deseja que sejam recolhidos os dados. Decidiu-se possibilitar três escolhas diferentes: i) dados para as 12h do dia anterior; ii) dados para o momento da presente execução do programa e iii) dados para as 12h do dia seguinte. Estas escolhas podem ser indicadas através da introdução do carácter Y (para *yesterday*), N (para *now*) ou T (para *tomorrow*), e a sua validade é assegurada por uma verificação de expressão regular (habitualmente apelidada de *regex*, módulo *Python re*). Caso o *input* fornecido pelo utilizador não coincida com uma das destas opções limitadas (Y ou N ou T), será novamente pedido um *input*, oferecendo também opção para terminar a execução do programa através da introdução da palavra "exit". O tipo de pedido especificado será guardado em memória e a sua utilidade será demonstrada à frente.



A função *harvestOWM* é talvez uma das funções mais importantes de todo o *script*, na medida em que é responsável por recolher o conjunto de variáveis meteorológicas para cada distrito, possibilitando, mais à frente, a criação do mapa. Esta função exige a introdução de três parâmetros incontornáveis: o dicionário com as coordenadas dos centroides de cada distrito, a chave de acesso à API e o momento para o qual se pretende obter os dados. Em concreto, toda a extração e compilação dos dados acontece dentro de um grande ciclo *for* que iterará por 18 vezes, uma por cada distrito. No interior deste ciclo, inicia-se com a extração das coordenadas do centroide do distrito, convertendo-as para *string*. Estas coordenadas são a localização para a qual será feito o pedido de dados. De seguida segue-se com a avaliação do *input* dado anteriormente pelo utilizador através da função *requestType*, realizando-se uma sequência de testes lógicos para aferir a sua intenção e consequentemente construir o URL do pedido. Este URL tem particularidades específicas para cada API em utilização, as quais poderão ser consultadas com maior detalhe na documentação disponibilizada em <https://openweathermap.org/api>.

Ainda dentro do grande ciclo *for*, e após possuímos a variável *url* com a especificação do URL adequado ao pedido do utilizador, prossegue-se para o bloco de código responsável por recolher e compilar os dados meteorológicos. O método *urllib.request.urlopen()* é especificamente utilizado para abrir o URL produzido para contactar com a API, sendo imediatamente precedido do método *json.loads()* de modo a decodificar a resposta da API em formato *json* para um dicionário. Dentro deste dicionário existe toda a informação contida na resposta da API, contudo, apenas alguns elementos dessa resposta interessam ser armazenados e por isso é necessário levar a cabo alguns passos de pré processamento. Após uma análise cuidada que serviu para identificar o encadeamento das listas e dicionários dentro do dicionário da resposta da API, recorreu-se ao método *.get()* de modo a conseguir-se extrair as variáveis meteorológicas de interesse, organizando-as no dicionário "districtForecast". No fim de cada iteração do grande ciclo *for*, o dicionário "districtForecast" é concatenado a uma lista - "forecast" - que no fim das 18 iterações conterá 18 dicionários, cada um correspondendo a um distrito. Por motivos de comodidade à função seguinte, a lista "forecast" é então convertida para uma *dataframe Pandas*, a qual guardará todos os dados recolhidos num formato tabular.

De seguida é implementada a função *df2PgSQL*. O seu objetivo prende-se com o carregamento dos dados anteriormente recolhidos através da função *harvestOWM* para a BD "meteo", cujas entidades haviam sido previamente preparadas pelas funções *checkPgDistrictsTable* e *checkPgForecastTable*. De forma prática, a função *df2PgSQL* faz recurso aos cursores do módulo *psycopg2* para executar uma expressão SQL que carrega na tabela "forecast" a *dataframe* retornada pela função *harvestOWM*.

Após o carregamento dos dados meteorológicos naquela que pode ser vista como a tabela de "transações" da BD, é construída a função *geoViewExtraction*, responsável por extrair da tabela "forecast" os últimos 18 registos que nela foram carregados (os dados mais recentes para cada distrito). Simultaneamente confere-lhes um carácter espacial através da realização de um *join* com a tabela geográfica "distritos", respeitante à shapefile carregada na BD através da função *checkPgDistrictsTable*. A função *geoViewExtraction* recorre também aos cursores do módulo *psycopg2*, executando a *querie* necessária para a construção da *view*.

```
###

# Meteo request to API
def requestType():
    """
    Especificação do momento da previsão meteorológica. Exige um input ao
    user do tipo str().
    Devolve str() que identifica o momento da previsão meteorológica requerida.
    """
    # Requesting input
    while True:
        requestType = input(getMessageString('Specify the wanted type of meteomap \
request (Y for yesterday, N for now, T for tomorrow): ' ))
        if not re.match('[YNT]', requestType):
            if requestType == 'exit':
                sys.exit()
            else:
                getMessageString('Warning - please, limit your input to\
Y or N or T or exit with \'exit\'.')
                time.sleep(2)
        else:
            break
    return requestType

def harvestOWM(coordDic, apiKey, requestType):
    """
    Execução do tipo de pedido através da Open Weather Map One Call API.
    Devolve uma dataframe (pandas) com os principais parâmetros meteorológicos
    para cada distrito.
    """
    forecast = []
    for item in coordDic.items():
        lat = str(item[1][1])
        long = str(item[1][0])
        # For yesterday's weather
        if requestType == 'Y':
            yesterday = datetime.now() - timedelta(days=1)
            unixTimestamp = int(yesterday.timestamp())
            url = 'https://api.openweathermap.org/data/2.5/onecall/timemachine?'+\
                'lat={}&lon={}&dt={}&appid={}&units=metric'.format(lat, long,\
                unixTimestamp, apiKey)
            # example http://api.openweathermap.org/data/2.5/onecall/timemachine?
            # \lat=60.99&lon=30.9&dt=1616943972&appid=44cfad7f82ec61d3f522de3201b703d4
        # For current weather
        elif requestType == 'N':
            url = 'https://api.openweathermap.org/data/2.5/onecall?'+\
                'lat={}&lon={}&exclude=minutely,hourly,daily,alerts&appid={}&units=metric'.
                format(lat, long, apiKey)
            # example https://api.openweathermap.org/data/2.5/onecall?\
            # lat=33.441792&lon=-94.037689&exclude=minutely,hourly,daily,\
            # alerts&appid=44cfad7f82ec61d3f522de3201b703d4
        # For tomorrow's weather
        elif requestType == 'T':
            url = 'https://api.openweathermap.org/data/2.5/onecall?'+\
                'lat={}&lon={}&exclude=current,minutely,hourly,alerts&appid={}&units=metric'
                .format(lat, long, apiKey)
            # example https://api.openweathermap.org/data/2.5/onecall?\
            # lat=33.441792&lon=-94.037689&exclude=current,minutely,hourly,\
            # alerts&appid=44cfad7f82ec61d3f522de3201b703d4

        with urllib.request.urlopen(url) as url:
            data = json.loads(url.read().decode())
            if requestType == 'T':
```

```
        data = data['daily'][1]
    else:
        pass
    districtForecast = {}
    districtForecast['distrito'] = item[0]
    if requestType == 'Y' or requestType == 'N':
        districtForecast['forecast_date'] =
            datetime.datetime.utcnow().timestamp(data.get('current')\
                .get('dt')).strftime('%d-%m-%Y')
        districtForecast['forecast_time'] =
            datetime.datetime.utcnow().timestamp(data.get('current')\
                .get('dt')).strftime('%H:%M:%S')
        main = data.get('current').get('weather')
        for item in main:
            districtForecast['weather_desc'] = item.get('main')
            districtForecast['temperature'] = data.get('current').get('temp')
            districtForecast['feels_like'] = data.get('current').get('feels_like')
            districtForecast['pressure'] = data.get('current').get('pressure')
            districtForecast['humidity'] = data.get('current').get('humidity')
            districtForecast['dew_point'] = data.get('current').get('dew_point')
            districtForecast['wind_speed'] = data.get('current').get('wind_speed')
            districtForecast['wind_deg'] = data.get('current').get('wind_deg')
            if requestType == 'Y':
                districtForecast['request_type'] = 'yesterday'
            elif requestType == 'N':
                districtForecast['request_type'] = 'now'
    else:
        districtForecast['forecast_date'] =
            datetime.datetime.utcnow().timestamp(data.get('dt'))\
                .strftime('%d-%m-%Y')
        districtForecast['forecast_time'] =
            datetime.datetime.utcnow().timestamp(data.get('dt'))\
                .strftime('%H:%M:%S')
        main = data.get('weather')
        for item in main:
            districtForecast['weather_desc'] = item.get('main')
            districtForecast['temperature'] = data.get('temp').get('day')
            districtForecast['feels_like'] = data.get('feels_like').get('day')
            districtForecast['pressure'] = data.get('pressure')
            districtForecast['humidity'] = data.get('humidity')
            districtForecast['dew_point'] = data.get('dew_point')
            districtForecast['wind_speed'] = data.get('wind_speed')
            districtForecast['wind_deg'] = data.get('wind_deg')
            districtForecast['request_type'] = 'tomorrow'
    forecast.append(districtForecast)
forecast_df = pandas.DataFrame(forecast)
return forecast_df

def df2PgSQL(connectionParameters, dataframe, table):
    """
    Utilização da função psycopg2.extras.execute_values()
    para carregamento da data frame colhida na tabela PostgreSQL "forecast"
    """
    # Create a list of tuples from the dataframe values
    tuples = [tuple(x) for x in dataframe.to_numpy()]
    # Comma-separated dataframe columns
    cols = ','.join(list(dataframe.columns))
    # SQL query to execute
    query = "INSERT INTO public.%s(%s) VALUES %s" % (table, cols)
    cur = connectionParameters.cursor()
    try:
        psycopg2.extras.execute_values(cur, query, tuples)
        conn.commit()
```

```
except (Exception, psycopg2.DatabaseError) as error:
    getMessageString("Error: %s" % error)
    conn.rollback()
    cur.close()
    return 1
getMessageString('Meteorological data has been successfully loaded into DB.')
cur.close()

# Reading hidden Open Weather Map API key from txt file
apiKey = open(os.path.join('apikey.txt'), 'r').readline()
apiKey = str(apiKey)

# Requesting type of meteo ((Y)esterday, (N)ow or (T)omorrow)
request = requestType()
getMessageString('Your request has been successfully validated.')

# Harvesting data from Open Weather Map
meteoDataFrame = harvestOWM(coordDist, apiKey, request)
getMessageString('Meteorological data has been successfully harvested.')

# Loading meteorological dataframe into DB
df2PgSQL(conn, meteoDataFrame, 'forecast')

# Isolating the harvested data in a new PG view, adding the districts geometry
def geoViewExtraction(connectionParameters):
    '''
    Construção de view dentro da BD, respeitante ao último request.
    Devolve view que associa uma componente espacial às variáveis meteorológicas
    por distrito.
    '''
    query = "DROP VIEW IF EXISTS forecast_map;\n\
    CREATE VIEW forecast_map as\n\
    select forecast.*, districtsetrs.geom\n\
    from forecast, districtsetrs\n\
    where forecast.distrito = districtsetrs.distrito\n\
    order by forecast.forecast_id desc limit 18"
    cur = connectionParameters.cursor()
    cur.execute(query)
    conn.commit()
    cur.close()
    getMessageString('PostgreSQL forecast_map geoview created.')

geoViewExtraction(conn)
```

## 2.5 Disponibilização do serviço de mapas

A última secção do programa diz respeito à disponibilização automática do serviço de mapas, recorrendo à implementação de quatro funções que farão interação com o *Geoserver* a diferentes níveis. O *Geoserver*, na sua natureza, disponibiliza uma REST API que possibilita a escrita e leitura através de quatro primitivas: *GET* para ler; e *PUT*, *POST* e *DELETE* para escrever, evitando qualquer contacto manual com a interface de *web administration*. No entanto, encontram-se disponíveis algumas implementações que objetivam facilitar o uso da REST API numa *framework Python*, contornando as repetitivas execuções de comandos (*GET*, *PUT*, *POST* e *DELETE*) através do método *os.system()* ou *subprocess.Popen()*. Em concreto, adotou-se a utilização do módulo *geoserver-rest* (<https://github.com/gicait/geoserver-rest>).

Após fazer leitura da *password* do *Geoserver* a partir de um ficheiro de texto (mesmo procedimento da leitura da *password* do *PostgreSQL*) e de criar um dicionário com as respetivas credenciais, é construída a função *initializeGeoserver*, a qual tem como único objetivo a

inicialização do *Geoserver* através da execução do seu ficheiro de binários responsável pelo *startup*. Após inicialização, é implementada a segunda função (*checkWorkspace*), responsável por fazer a verificação da presença do *workspace* que acolherá o serviço de mapas. Caso o método *.get\_workspace()* devolva o valor *"None"*, é feita a criação do *workspace* por meio do método *.create\_workspace()*.

À semelhança da função *checkWorkspace*, a terceira função desenvolvida (*createFeatureStore*) faz uma verificação de conteúdo dentro do *Geoserver*, mas desta vez orientada às *stores*. Se o método *.get\_featurestore()* devolver uma *string* descritora do erro *"Expecting value: line 1 column 1 (char 0)"*, é dada entrada na secção lógica que fará uso do método *create\_featurestore()*, de modo a criar a *store* dentro do *workspace* anteriormente preparado. Esta *store* tem a particularidade de estar associada à BD "meteo", onde existe já preparada a *view* derivada a partir da função *geoViewExtraction*.

A última função criada neste *script* (*publishFeatureStore*) materializa o derradeiro objetivo do projeto: disponibilizar o serviço de mapas com os dados meteorológicos solicitados pelo utilizador. Ao contrário das funções atrás apresentadas, o código aqui presente executará sempre uma instrução do tipo *POST* de modo a carregar os dados recolhidos momentos antes. No caso do programa *meteo2map* ter sido executado anteriormente, é expectável que exista na *store* um WMS relativo a essa execução. Nesse contexto é necessário eliminá-lo através do método *delete\_layer()*, procedendo-se à criação do novo WMS através do carregamento da *view* em BD. Caso seja a primeira vez que o programa *meteo2map* esteja a ser executado ou o WMS anterior tenha sido eliminado manualmente, a expressão lógica de verificação devolverá uma *string* com o erro *"get\_layer error: Expecting value: line 1 column 1 (char 0)"*, e o novo WMS é criado a partir da *view* presente na BD "meteo", sem necessidade de haver eliminação de um serviço antecedente.

O *script* termina com a avaliação do seu próprio tempo de execução, fazendo recurso ao *timestamp* inicialmente guardado na variável *startTime*.

```
###

# Reading hidden Geoserver password from txt file in dir
gsPassword = open(os.path.join('geoserverPw.txt'), 'r').readline()
gsPassword = str(gsPassword)

geoserverCred = {'username': 'admin', 'password': gsPassword}

def initializeGeoserver():
    """
    Inicialização do Geoserver através da linha de comandos.
    """
    getMessageString('Starting Geoserver. Please, wait a moment.')
    # Geoserver starting in a new command line
    # cmd = 'start C:\\\"Program Files\"\\geoserver\\bin\\startup.bat'
    # subprocess.Popen(cmd, shell=True)
    os.system('start C:\\\"Program Files\"\\geoserver\\bin\\startup.bat')
    # To give time to Geoserver startup correctly
    time.sleep(20)

initializeGeoserver()

def checkWorkspace(geoserverCredentials, workspaceName):
    """
    Verificação de ausência/presença de workspace dentro do Geoserver.
```

```
Caso o workspace especificado não exista, a função cria-o.
'''
# Initialize the library
geo = Geoserver('http://localhost:8080/geoserver',\
username=geoserverCredentials.get('username'),\
password=geoserverCredentials.get('password'))
# Checking for workspace
if geo.get_workspace(workspace=workspaceName) == None:
    getMessageString('{} workspace not found. Let's create it.'\
        .format(workspaceName))
    geo.create_workspace(workspace=workspaceName)
else:
    getMessageString('Found {} workspace.'.format(workspaceName))

checkWorkspace(geoserverCred, 'saprog_meteo')

postgresCred = {'dbname':'meteo', 'user':'postgres', 'password':pgPassword,\
'host':'localhost', 'port':'5432'}

def createFeatureStore(geoserverCredentials, postgresCredentials, workspaceName, storeName):
    '''
    Verificação de ausência/presença de featurestore dentro do Geoserver.
    Caso a featurestore especificado não exista, a função cria-a.
    '''
    geo = Geoserver('http://localhost:8080/geoserver',\
username=geoserverCredentials.get('username'),\
password=geoserverCredentials.get('password'))
    if geo.get_featurestore(workspace=workspaceName, store_name=storeName)\
== 'Error: Expecting value: line 1 column 1 (char 0)':
        getMessageString('{} featurestore not found. Let's create it.'\
            .format(storeName))
        geo.create_featurestore( workspace=workspaceName, store_name=storeName,\
db=postgresCredentials.get('dbname'),\
host=postgresCredentials.get('host'), pg_user=postgresCredentials.get('user')
, pg_password=postgresCredentials.get('password'))
    else:
        getMessageString('Found {} featurestore.'.format(storeName))

createFeatureStore(geoserverCred, postgresCred, 'saprog_meteo', 'meteomap')

def publishFeatureStore(geoserverCredentials, workspaceName, storeName, pgTableName):
    '''
    Função para carregamento da view em base de dados para o featurestore.
    Necessita que exista a referida view dentro da base de dados Postgres,
    e do workspace e featurestore definida dentro do Geoserver.
    '''
    geo = Geoserver('http://localhost:8080/geoserver',\
username=geoserverCredentials.get('username'),\
password=geoserverCredentials.get('password'))
    if geo.get_layer(workspace=workspaceName, layer_name=pgTableName)\
== "get_layer error: Expecting value: line 1 column 1 (char 0)":
        .format(pgTableName, storeName):
            getMessageString('Forecast WMS successfully uploaded.')
            geo.publish_featurestore(workspace=workspaceName, store_name=storeName,\
pg_table=pgTableName)
    else:
        getMessageString("There was an old forecast WMS stored. Let's overwrite it.")
        geo.delete_layer(layer_name=pgTableName, workspace=workspaceName)
        geo.publish_featurestore(workspace=workspaceName, store_name=storeName,\
pg_table=pgTableName)
```



```
publishFeatureStore(geoserverCred, workspaceName='saprog_meteo', storeName='meteomap',  
pgTableName='forecast_map')
```

```
# %%  
# Execution time (finish)  
exTime = time.time() - startTime  
getMessageString("meteo2map executed in {} seconds".format(exTime))  
  
# End of METEO2MAP script
```

### 3. Resultados

Os primeiros resultados da execução do código apresentado remetem para a verificação das tabelas da BD "meteo", onde estão armazenados todos os dados meteorológicos recolhidos até ao momento (tabela "forecast"), bem como a *view* "forecast\_map" que contém os dados mais recentes de forma isolada.

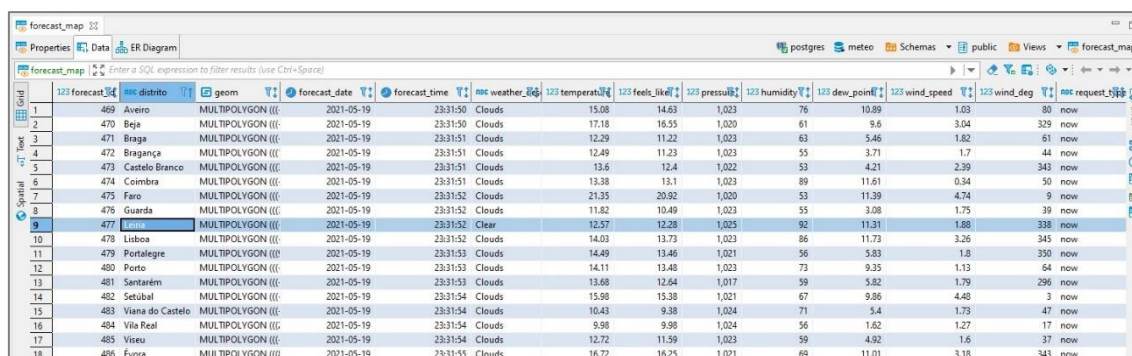
O número de ocorrências na tabela *forecast* (figura 1) dependerá sempre do número de vezes que o programa *meteo2map* tenha sido executado, sendo que por cada execução são adicionadas 18 novas linhas à tabela. O número de colunas manter-se-á necessariamente estático, correspondendo às 8 variáveis meteorológicas consideradas somando-se outras 5 relativas à chave-primária e outros campos descritivos.

forecast_id	distrito	forecast_time	forecast_date	weather_desc	temperature	feels_like	pressure	humidity	dew_point	wind_speed	wind_deg	request_type
217	Aveiro	2021-05-18	10:56:52	Clouds	16.23	16.31	1,022	92	14.93	2.24	239	yesterday
218	Beja	2021-05-18	10:56:52	Clouds	20.18	19.14	1,021	34	3.84	4.98	333	yesterday
219	Braga	2021-05-18	10:56:53	Clouds	14.9	14.98	1,021	97	14.43	0.95	231	yesterday
220	Bragança	2021-05-18	10:56:53	Clouds	15.72	15.18	1,020	70	10.27	3.38	259	yesterday
221	Castelo Branco	2021-05-18	10:56:53	Clouds	22.96	22.46	1,020	44	10.07	4.39	273	yesterday
222	Coimbra	2021-05-18	10:56:53	Clouds	19.71	19.38	1,018	63	12.40	4.1	267	yesterday
223	Faro	2021-05-18	10:56:54	Clouds	20.35	19.25	1,021	31	2.68	4.8	354	yesterday
224	Guarda	2021-05-18	10:56:54	Clouds	16.37	15.52	1,020	56	7.58	5.74	263	yesterday
225	Leiria	2021-05-18	10:56:54	Clouds	19.35	19.53	1,023	84	16.58	1.79	266	yesterday
226	Lisboa	2021-05-18	10:56:54	Clouds	18.43	18	1,022	64	11.5	4.16	291	yesterday
227	Portalegre	2021-05-18	10:56:55	Clouds	19.64	18.94	1,021	49	8.64	3.98	293	yesterday
228	Porto	2021-05-18	10:56:55	Clouds	15.33	15.4	1,021	95	14.53	2.85	234	yesterday
229	Santarém	2021-05-18	10:56:55	Clouds	21.27	21.33	1,018	72	16.02	1.34	300	yesterday
230	Setúbal	2021-05-18	10:56:56	Clear	19.77	19.4	1,022	61	12.04	3.8	300	yesterday
231	Viana do Castelo	2021-05-18	10:56:56	Clouds	12.71	12.46	1,021	93	11.61	1.24	188	yesterday
232	Vila Real	2021-05-18	10:56:56	Clouds	12.57	12.36	1,021	95	11.79	2.01	237	yesterday
233	Viseu	2021-05-18	10:56:56	Rain	17.21	17.23	1,021	86	14.85	2.94	235	yesterday
234	Evora	2021-05-18	10:56:57	Clear	19.72	18.9	1,022	44	7.14	4.79	322	yesterday
235	Aveiro	2021-05-18	11:07:46	Clouds	16.2	16.28	1,022	92	14.9	2.24	239	now
236	Beja	2021-05-18	11:07:46	Clouds	25.27	24.74	1,021	34	3.8	4.98	333	now
237	Braga	2021-05-18	11:07:47	Clouds	14.89	14.97	1,021	97	14.42	0.95	231	now
238	Bragança	2021-05-18	11:07:47	Clouds	15.72	15.18	1,020	70	10.27	3.38	259	now
239	Castelo Branco	2021-05-18	11:07:47	Clouds	23.52	23.08	1,020	44	10.57	4.39	273	now
240	Coimbra	2021-05-18	11:07:48	Clouds	19.95	19.67	1,018	64	12.94	4.1	267	now
241	Faro	2021-05-18	11:07:48	Clouds	20.35	19.25	1,021	31	2.68	4.8	354	now
242	Guarda	2021-05-18	11:07:48	Clouds	18.76	18.15	1,020	56	9.81	5.74	263	now
243	Leiria	2021-05-18	11:07:48	Clouds	20.09	20.24	1,023	80	16.53	2.24	284	now
244	Lisboa	2021-05-18	11:07:49	Clouds	18.45	18.02	1,022	64	11.52	4.16	291	now
245	Portalegre	2021-05-18	11:07:49	Clouds	20.22	19.58	1,021	49	9.18	3.98	293	now
246	Porto	2021-05-18	11:07:49	Clouds	15.48	15.56	1,021	95	14.68	2.85	234	now
247	Santarém	2021-05-18	11:07:49	Clouds	21.41	21.49	1,018	72	16.16	0.89	281	now
248	Setúbal	2021-05-18	11:07:50	Clear	19.82	19.45	1,022	61	12.09	3.8	300	now
249	Viana do Castelo	2021-05-18	11:07:50	Clouds	12.71	12.46	1,021	93	11.61	1.24	188	now
250	Vila Real	2021-05-18	11:07:50	Clouds	12.57	12.36	1,021	95	11.79	2.01	237	now
251	Viseu	2021-05-18	11:07:51	Clouds	15.02	14.82	1,021	86	12.7	2.94	235	now
252	Evora	2021-05-18	11:07:51	Clear	23.51	23.07	1,022	44	10.57	4.79	322	now

Figura 1 - Extrato da tabela "forecast" retirado do software DBeaver. Apresentam-se 36 ocorrências relativas a duas execuções distintas do programa *meteo2map* (a primeira orientada a um pedido de dados para o dia anterior; a segunda orientada a um pedido de dados para o momento).

O segundo resultado que se apresenta diz respeito à *view* "forecast\_map" (figura 2). Como explicado anteriormente, esta *view* é produzida pela função *geoViewExtraction*, e isola os dados mais recentes da tabela "forecast", associando-lhes o polígono de cada distrito. Esta tabela limita-se a 18 ocorrências (distritos) e para além das 13 colunas extraídas da tabela "forecast" é adicionada uma 14ª onde se integra a geometria poligonal do distrito.





id	forecast_id	distrito	geom	forecast_date	forecast_time	weather_desc	temperature	feels_like	pressure	humidity	dew_point	wind_speed	wind_deg	request_type
1	469	Aveiro	MULTIPOLYGON ((	2021-05-19	23:31:50	Clouds	15.08	14.63	1,023	76	10.99	1.03	80	now
2	470	Beja	MULTIPOLYGON ((	2021-05-19	23:31:50	Clouds	17.18	16.55	1,020	61	9.6	3.04	329	now
3	471	Braga	MULTIPOLYGON ((	2021-05-19	23:31:51	Clouds	12.29	11.22	1,023	63	5.46	1.82	61	now
4	472	Bragança	MULTIPOLYGON ((	2021-05-19	23:31:51	Clouds	12.49	11.23	1,023	55	3.71	1.7	44	now
5	473	Castelo Branco	MULTIPOLYGON ((	2021-05-19	23:31:51	Clouds	13.6	12.4	1,022	53	4.21	2.39	343	now
6	474	Coimbra	MULTIPOLYGON ((	2021-05-19	23:31:51	Clouds	13.38	13.1	1,023	89	11.61	0.34	50	now
7	475	Faro	MULTIPOLYGON ((	2021-05-19	23:31:52	Clouds	21.35	20.92	1,020	53	11.39	4.74	9	now
8	476	Guarda	MULTIPOLYGON ((	2021-05-19	23:31:52	Clouds	11.82	10.49	1,023	55	3.08	1.75	39	now
9	477	Lisboa	MULTIPOLYGON ((	2021-05-19	23:31:52	Clear	12.57	12.28	1,025	92	11.31	1.88	338	now
10	478	Lisboa	MULTIPOLYGON ((	2021-05-19	23:31:52	Clouds	14.03	13.73	1,023	86	11.73	3.26	345	now
11	479	Portalegre	MULTIPOLYGON ((	2021-05-19	23:31:53	Clouds	14.49	13.46	1,021	56	5.83	1.8	350	now
12	480	Porto	MULTIPOLYGON ((	2021-05-19	23:31:53	Clouds	14.11	13.48	1,023	73	9.35	1.13	64	now
13	481	Santarém	MULTIPOLYGON ((	2021-05-19	23:31:53	Clouds	13.68	12.64	1,017	59	5.82	1.79	298	now
14	482	Setúbal	MULTIPOLYGON ((	2021-05-19	23:31:54	Clouds	15.98	15.38	1,021	67	9.86	4.48	3	now
15	483	Viana do Castelo	MULTIPOLYGON ((	2021-05-19	23:31:54	Clouds	10.43	9.38	1,024	71	5.4	1.73	47	now
16	484	Vila Real	MULTIPOLYGON ((	2021-05-19	23:31:54	Clouds	9.98	9.98	1,024	56	1.62	1.27	17	now
17	485	Viseu	MULTIPOLYGON ((	2021-05-19	23:31:54	Clouds	12.72	11.59	1,023	59	4.92	1.6	37	now
18	486	Évora	MULTIPOLYGON ((	2021-05-19	23:31:55	Clouds	16.72	16.25	1,021	69	11.01	3.18	343	now

Figura 2 - View "forecast\_map", retirada do software DBBeaver. Contempla os dados meteorológicos coletados para um pedido do tipo "now", executado no dia 19/05/2021 entre as 23:31:50 (primeiro pedido - Aveiro) e as 23:31:55 (último pedido - Évora).

Por último apresenta-se o mapa (figura 3) que apresenta o WMS com os dados meteorológicos. Por motivos de limitação de tempo, produziu-se este mapa com o auxílio do *plugin* QGIS *qgis2web* (<https://plugins.qgis.org/plugins/qgis2web>), derivando um HTML que integra elementos controlados pela biblioteca *Leaflet*.

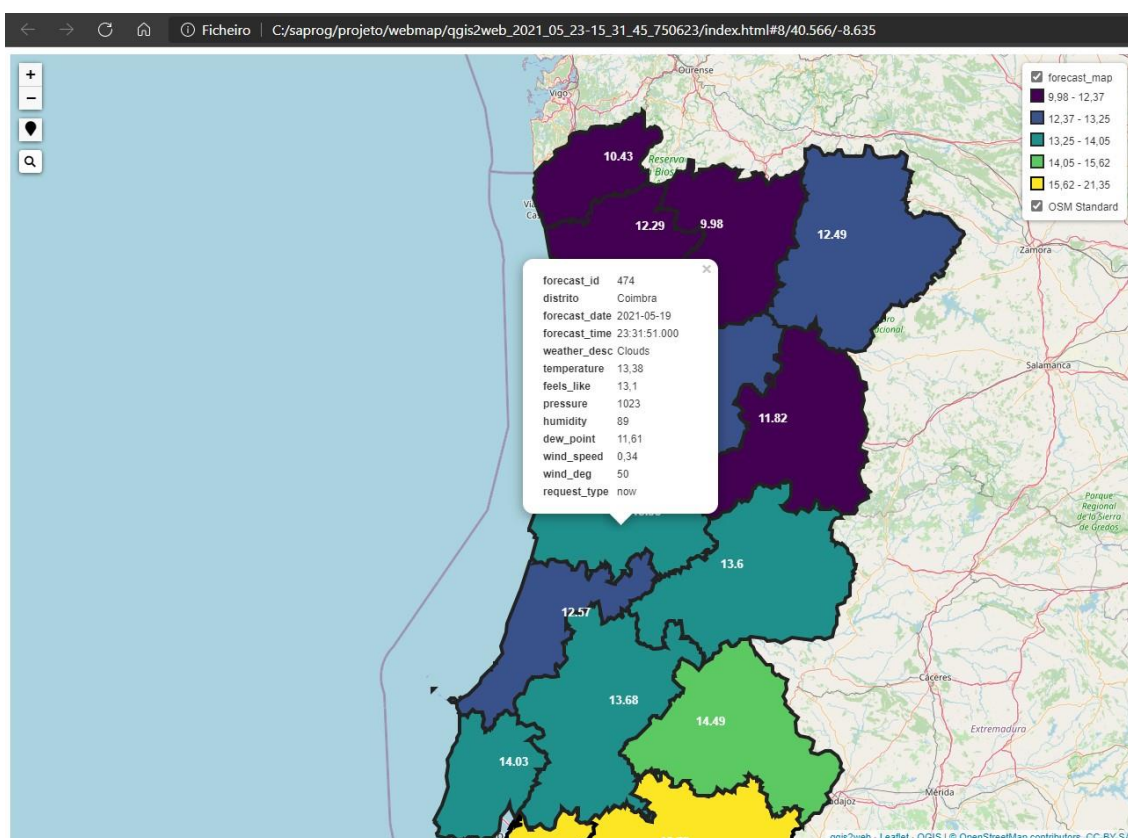


Figura 3 - Webmap com a apresentação do serviço WMS, ilustrando um popup com os dados meteorológicos recolhidos no dia 19/05/2021 às 23:31:51 para o distrito de Coimbra.

## 4. Conclusões

Após o desenvolvimento deste projeto, conclui-se que o ambiente *open source* fornece ferramentas com valências robustas, que dão resposta a todo o tipo de desafios propostos. Contudo, a instalação de um sistema de produção poderá ser uma tarefa desafiante: dada a multiplicidade de soluções existentes para o mesmo tipo de tarefas (e.g., *Geoserver* vs *Mapserver*, *OpenLayers* vs *Leaflet*, ou os múltiplos módulos *Python* para processar informação geoespacial), é exigido ao programador uma sensibilidade subjetiva, de modo que sejam selecionadas as ferramentas com melhor grau de adequabilidade ao enunciado em questão. Quanto mais vasta a experiência do indivíduo, melhores escolhas serão feitas e por consequência o sistema de produção sairá beneficiado.

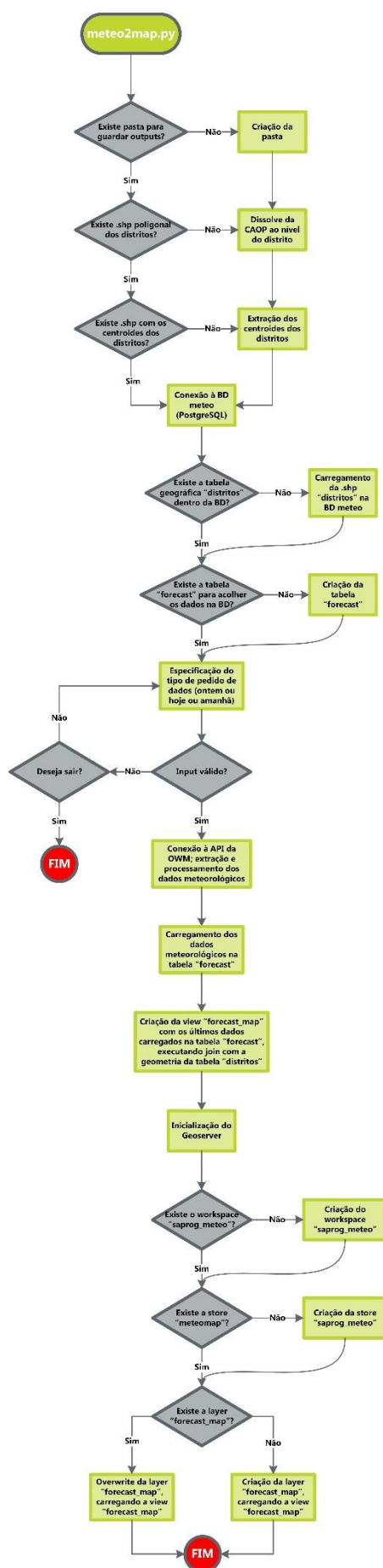
Em particular, as maiores dificuldades técnicas foram sentidas na última fase do projeto, na qual se automatizou a interação com o *Geoserver*. Foram adotadas várias abordagens (passando pelas instruções por linha de comandos até à realização de experiências com vários módulos *Python*) até que se encontra-se uma forma ótima para a disponibilização automática do serviço de mapas. Estas dificuldades decorrem também do facto de esta ser a tecnologia com a qual havia menos contacto à data.

Outras questões têm ainda margem para melhoria, tais como: i) inicialização do programa *meteo2map* através de uma função *start()* sem parâmetros, que contenha no seu interior a especificação de um grupo de variáveis globais de modo evitar possíveis redundâncias, bem como o encadeamento de todas as funções e respetivos parâmetros necessários para a correta execução do programa; ii) melhoramento do modelo de dados simplista a partir do qual se contruiu a BD “meteo”; e por fim iii) o desenvolvimento de um *webmap* mais adequado à apresentação de um mapa meteorológico desta natureza, possibilitando a especificação de filtros e alterações de simbologia mediante a seleção de uma variável meteorológica.

## 5. Bibliografia

1. Bitzer, J.; Schrettl, W.; Schröder, P.J.H. Intrinsic motivation in open source software development. *J. Comp. Econ.* 2007, 35, 160–169.
2. Boston GIS shp2pgsql Command Line Cheatsheet Disponível online: [https://www.bostongis.com/pgsql2shp\\_shp2pgsql\\_quickguide.bqg](https://www.bostongis.com/pgsql2shp_shp2pgsql_quickguide.bqg) (consultado a 8 de abril de 2021).
3. bruceh48 Geoserver REST example calls Disponível online: <https://gist.github.com/bruceh48/b725e2c10d36442d8cd26a16e44e5e79> (consultado a 15 de maio de 2021).
4. Corti, M.; Park, S.; PostGIS Cookbook - Store, organize, manipulate, and analyze spatial data; 2018; ISBN 9781788299329.
5. GeoPandas developers GeoPandas 0.9.0 documentation Disponível online: <https://geopandas.org/docs.html> (consultado a 28 de maio de 2021).
6. gicait geoserver-rest: Python library for management for geospatial data in GeoServer Disponível online: <https://github.com/gicait/geoserver-rest> (consultado 15 de abril de 2021).
7. Iacovella, S. GeoServer Beginner's Guide; Packt, 2017; ISBN 978-1-78829-737-0.
8. Martins, H. E-book de Software Aberto e programação em SIG; 2019;
9. Mikiewicz, D.; Mackiewicz, M.; Nycz, T. Mastering PostGIS; 2017; ISBN 978-1-78439-164-5.
10. Moreno-Sanchez, R. Free and Open Source Software for Geospatial Applications (FOSS4G): A Mature Alternative in the Geospatial Technologies Arena. *Trans. GIS* 2012, 16, 81–88.
11. Psycopg Team Psycopg 2.8.7. documentation Disponível online: <https://www.psycopg.org/docs/> (consultado a 10 de abril de 2021).
12. Westra, E. Python Geospatial Development; 3rd edition.; Packt, 2016; ISBN 9781785288937.

## 6. Anexos



Anexo 1 - Fluxograma da execução de `meteo2map`