# Progetto Estate – Documentation

## Table of Content

Contents

[Progetto Estate - Documentation 1](#_Toc111324291)

[Table of Content 1](#_Toc111324292)

[Project Aim 1](#_Toc111324293)

[Project Scope 1](#_Toc111324294)

[Data flows – chart 2](#_Toc111324295)

[How to get an API access ? 2](#_Toc111324296)

[FMI 2](#_Toc111324297)

[AERIS 3](#_Toc111324298)

[ACCUWEATHER 5](#_Toc111324299)

[YR 8](#_Toc111324300)

[THE WEATHER CHANNEL 8](#_Toc111324301)

[METEOMATICS 9](#_Toc111324302)

[FAQ 9](#_Toc111324303)

[How to add or delete a new bbox to start gathring the forecast data? 9](#_Toc111324304)

[How to add new Observation source? 10](#_Toc111324305)

[How to add a new Forecast source? 12](#_Toc111324306)

[How to add a new metric for tracking? 12](#_Toc111324307)

[How to change a time horizon, how far do we look? 13](#_Toc111324308)

[How to change a frequency of the reloading? 14](#_Toc111324309)

[How to debug, in case some Forecast or some observations were not loaded? 14](#_Toc111324310)

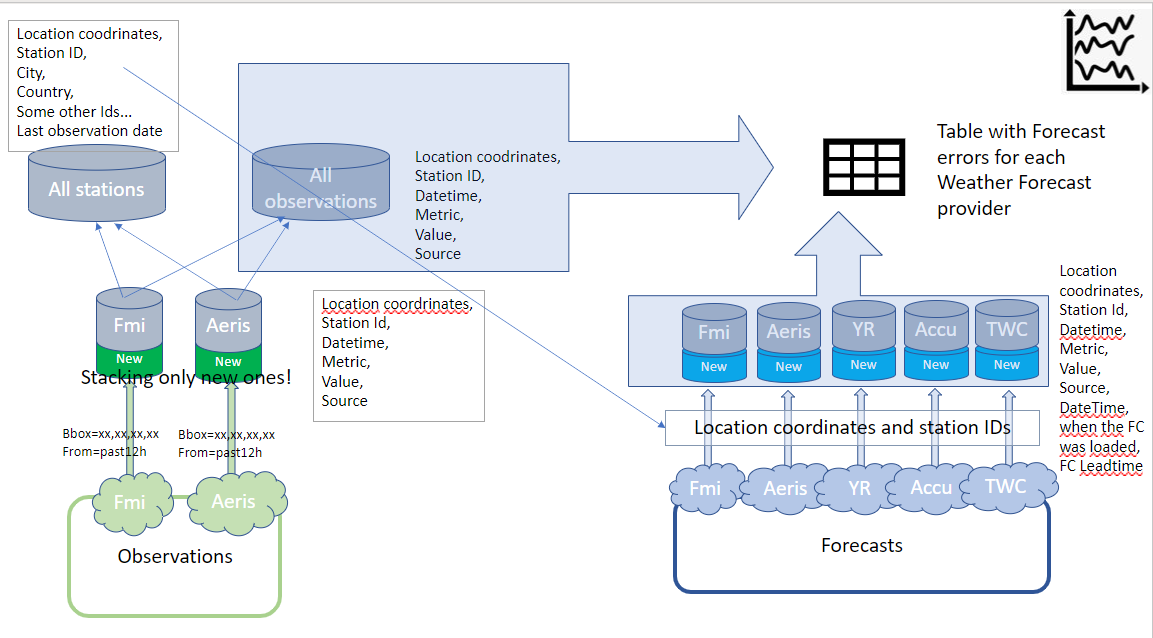
## Project Aim

Build flexible data flows and reports for benchmarking Vaisala’s vs competitive forecasting capabilities, to support argumentation of NowCasting and NearCasting offering.

## Project Scope

|  |  |
| --- | --- |
| **Tasks** | **Deliverables** |
| * To get familiar with Vaisala internal verification tool​ * To map, identify and bundle of ground-truth observation stations to the relevant chosen physical area​ * To find and purchase if needed the external competitive forecasts\* and own Vaisala forecasts. Identify the limitation and possible risks. To setup the connection to data streams/APIs​ * To build up the data flows and controls architecture​ * To make the documentation of built stack for eventual auditing, or modification by Vaisala in a wiki format​ * Compare all available forecasts with the ground truth using Vaisala verification tool, analyze and interpret the results, prepare the visualizations driving insights and pre-sales argumentations​ | * List of Helsinki public and Vaisala private stations and their coordinates. Connection strings to gather the ground truth observations​ * Credentials and instructions how to connect to APIs and load necessary data, listed tariffs. Connection strings to all selected APIs. Listed and described limitations and possible risks​ * Application(written in Python) that loads, processes, formats the data in desired way. Its output is a dataset that should be easily fetched into the verification tool​ * Full documentation in a wiki format​ * Analysis and interpretation of the comparison results with various visual content that drives the insights​ |

## Data flows – chart



## How to get an API access?

During this project only Free accounts were used. However, they might be not sufficient in the future. Often there are certain limitations for the number of calls per day and other limitations. All weather forecast providers do provide different account extensions to developers on the commercial basis. Contact details of sales are usually mentioned on their webpages.

### FMI

FMI service does not require any registration, their API is open. FMI has a set of saved queries, their list can be found here:

<http://opendata.fmi.fi/wfs?service=WFS&version=2.0.0&request=listStoredQueries&>

Within a framework of this project we are interested in the following 3 queries:

fmi::ef::stations – stations

fmi::observations::weather::multipointcoverage – observations

fmi::forecast::harmonie::hybrid::point::timevaluepair – forecasts

There is one project available in github, called fmiopendata:

<https://github.com/pnuu/fmiopendata/>

This project makes the data loading easier. Hovewer, at the moment when this project was going there was only 1 suitable query – query for observations. Forecast and stations can be retreived through the XML parsing.

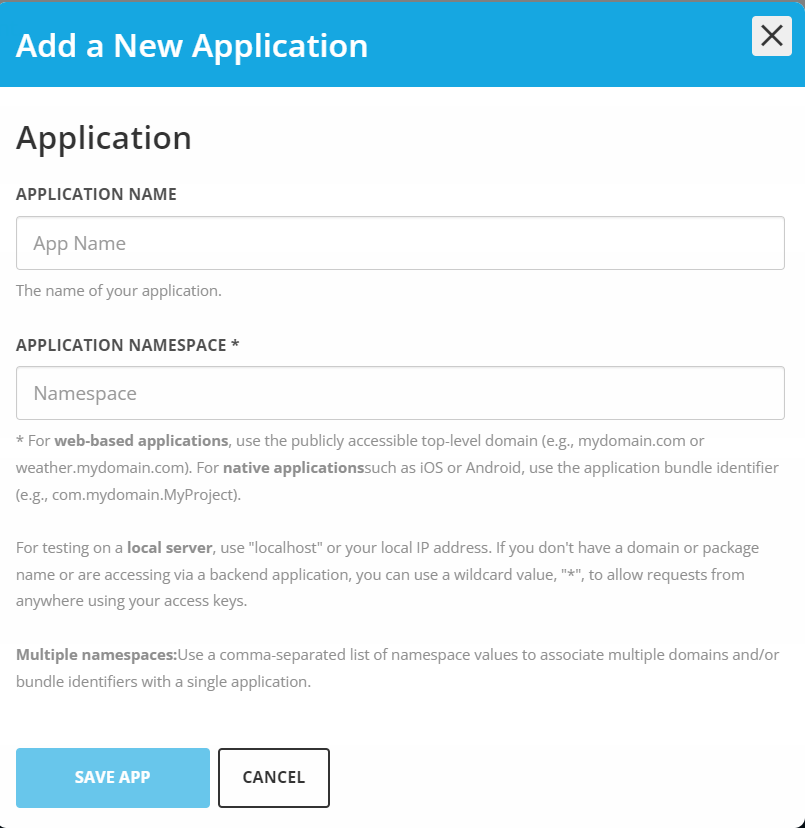
### AERIS

Aeris API requires the registration. Aeris asks to provide your full name, company name, email and password. Free Trial allows you to test-drive our most popular features and products at no cost for thirty (30) days.

<https://www.aerisweather.com/signup/developer/>

After the account is created – you need to create an application in Aeris, here:

<https://account.aerisweather.com/account/apps>



When an application is created – you can copy an secret-key and client-id and use it in the application. These 2 parameters are used in *get\_data()* functions of observations and forecast classes. Namely, class *Aeris\_observations in the stations\_observations.py* and class *aeris\_forecasts in forecasts.py file.*

class Aeris\_observations():  
 def \_\_init\_\_(self, bbox, from\_date, limit):  
 self.bbox = bbox  
 self.from\_date = from\_date  
 self.limit = limit  
  
 def get\_data(self):  
 url\_base = 'https://api.aerisapi.com/observations/archive/within?p=' + self.bbox + '&from=' + self.from\_date + '&format=json&filter=allstations&limit=' + str(self.limit) + '&'  
 secret\_key = 'O2U57aPy30qn6TRhMogxUNPgYs6zDX23rXTvOKuy'  
 client\_id = '0aeGF7FKmqjeuiTlyxuvN'  
 url = url\_base + 'client\_id=' + client\_id + '&client\_secret=' + secret\_key  
 response = req.get(url=url)  
 json\_file = json.loads(response.text)  
 return json\_file

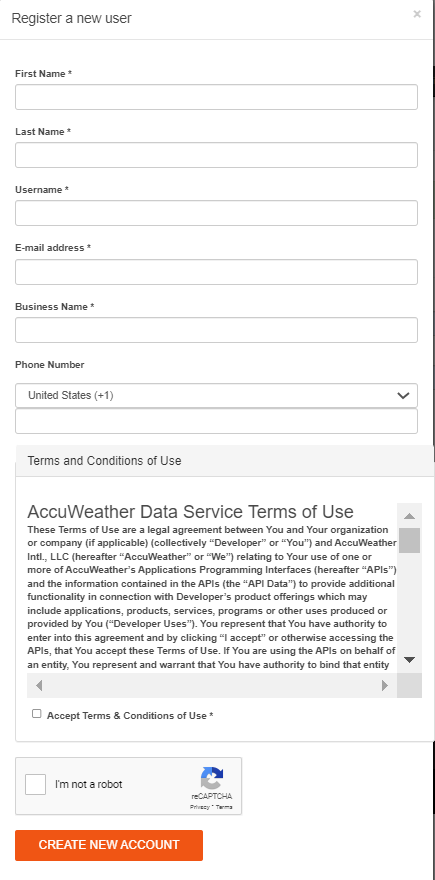
class aeris\_forecasts:  
  
 def \_\_init\_\_(self, lat, lon, limit):  
 self.lon = lon  
 self.lat = lat  
 self.limit = limit  
  
 def get\_data(self):  
 url\_base = 'https://api.aerisapi.com/forecasts/' + str(self.lat) + ',' + str(self.lon) + '?format=json&filter=1hr&limit=' + str(self.limit) + '&'  
 secret\_key = 'O2U57aPy30qn6TRhMogxUNPgYs6zDX23rXTvOKuy'  
 client\_id = '0aeGF7FKmqjeuiTlyxuvN'  
 url = url\_base + 'client\_id=' + client\_id + '&client\_secret=' + secret\_key  
 response = req.get(url=url)  
 json\_file = json.loads(response.text)  
 return json\_file

User cabinet at Aeris web-page also provides the the data about the API usage, it shares an info about remaining API requests.

### ACCUWEATHER

Accuweather also requires the registration: Full name, business name, contact information.

<https://developer.accuweather.com/user/login?destination=node/55&autologout_timeout=1>

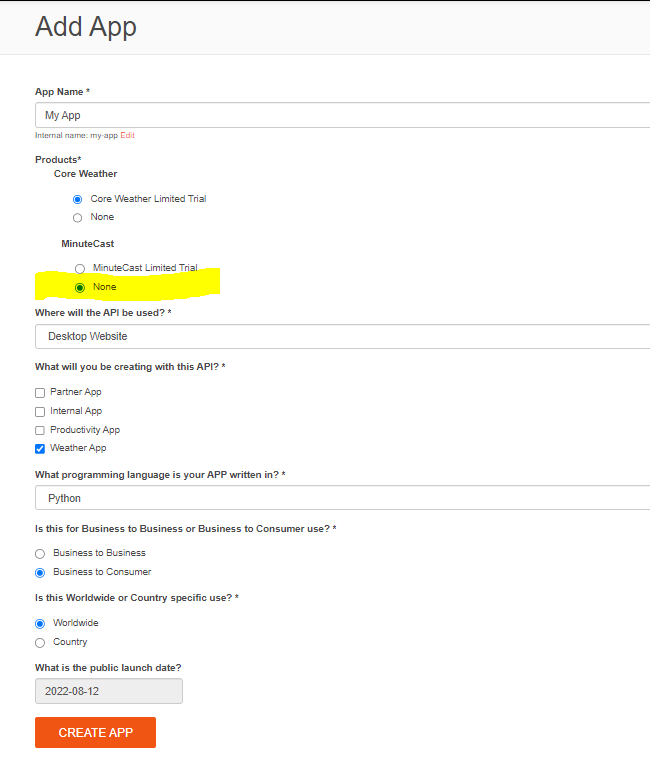


There are different tariffs to load the data from accuweather:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | **Free** | **Standard** | **Prime** | **Elite** |
| **Locations** | ✔ | ✔ | ✔ | ✔ |
| **Current Conditions** | ✔ | ✔ | ✔ | ✔ |
| **24 Hours Historical Current Conditions** | ✔ | ✔ | ✔ | ✔ |
| **Daily Forecast** | 5 Days | 5 Days | 10 Days | 15 Days |
| **Hourly Forecast** | 12 Hours | 12 Hours | 72 Hours | 120 Hours |
| **Indices** | 5 Days | 5 Days | 10 Days | 15 Days |
| **Alarms** |  | 5 Days | 10 Days | 15 Days |
| **Translations** |  | ✔ | ✔ | ✔ |
| **Tropical** |  |  | ✔ | ✔ |
| **Alerts** |  |  | ✔ | ✔ |
| **Imagery** |  |  | ✔ | ✔ |
| **Price** | **Free** | **$25/mo** | **$250/mo** | **$500/mo** |
| 50 calls/day, Limit 1 key/developer | $0.12 CPM over 225,000 calls per month | $0.15 CPM over 1,800,000 calls per month | $0.22 CPM over 2,400,000 calls per month |

Free tariff allows to send only 50 requests per day and is limited to 12 hours. Unfortunately, it won’t be enough for the normal work of the application as 1 run for one location aready requeries 3 calls (observations, location id and the forecast), if we run an app 8 times per day – it will mean that one location will consume 24 calls per day.

When a new account is created with Free pricing package, you need to create an application in Accuweather user cabinet. It is important to select Core Weather = Core Weather Limited Trial, MinuteCast = None.



When an application is created, from “My APPs” API Key can be copied and pasted into the *get\_data()* function of *Accuweather\_forecast* class in *forecasts.py* file.

class Accuweather\_forecast:  
  
 def \_\_init\_\_(self, lat, lon):  
 self.lon = lon  
 self.lat = lat  
 #self.api\_key = "1YjFUtWv4U4Ag9xxEAOquo539uwAE50D"  
 self.api\_key = "CkoeDLfPcgAN9zLv5xtjFUNHLN9TpR5n"  
  
  
 def code\_location(self):  
 url\_location\_key = f'http://dataservice.accuweather.com/locations/v1/cities/geoposition/search?apikey={self.api\_key}&q={self.lat},{self.lon}&language=en-us'  
 response = req.get(url\_location\_key, headers={"APIKey": self.api\_key})  
 json\_data = json.loads(response.text)  
 code = json\_data['Key']  
 return code

It i also possible to create multiple Free accounts in Accuweather API and switch them when the usage is over.

### YR

YR does not require the registration, however, it is necessary to pass user-agent property to API requests.

class yr\_forecasts:  
  
 def \_\_init\_\_(self, lat, lon):  
 self.lon = lon  
 self.lat = lat  
  
 def get\_data(self):  
 url\_weather = f'https://api.met.no/weatherapi/locationforecast/2.0/complete?lat={self.lat}&lon={self.lon}'  
 headers = {'User-Agent': 'ProgettoEstate/1.0 github.com/ProgettoEstate'}  
 response = req.get(url=url\_weather, headers=headers)  
 json\_data = json.loads(response.text)  
 return json\_data

### THE WEATHER CHANNEL

The Weather Channel requires the registration that gets confirmed by some people from IBM. In order to get an access to 30 days Free Trial at Weather IBM API – Users will need to create an IBMid and electronically sign terms and conditions:

<https://www.ibm.com/account/reg/us-en/signup?formid=urx-19776&target=https%3A%2F%2Flogin.ibm.com%2Foidc%2Fendpoint%2Fdefault%2Fauthorize%3FqsId%3D9537a894-6dfa-4977-beb2-54d4761f16f0%26client_id%3DMyIBMLondonProdCI>

In order to get 30 days subscription to the Weather API of IBM you need to have a company name and ID to be mentioned in your application. I used my Toiminimi and my Business Id.

Free Trial can be requested here:

<https://developer.ibm.com/apis/catalog/weather--weather-forecast-apis/api/API--weather--hourly-forecast-api/#v3HourlyForecast>

Once the program agreement is completed, the request will be routed to our API provisioning team and provisioning will follow within 48 hours.

In my case I was not contacted automatically, so I wrote a support request. After we have settled everything via emailing – I have received my API key in a separate email.

Once you get an new API key, it needs to be inserted to the function *get\_data()* of the class *weather\_channel\_forecast* in *forecasts.py* file.

class weather\_channel\_forecast:  
  
 def \_\_init\_\_(self, lat, lon):  
 self.lat = lat  
 self.lon = lon  
  
 def get\_data(self):  
 import http.client  
 conn = http.client.HTTPSConnection("api.weather.com")  
 headers = {'accept': "application/json; charset=UTF-8"}  
 days = 2  
 latlon = str(self.lat)+','+str(self.lon)  
 units = 'm'  
 #units = ''  
 language = 'en-US'  
 format = 'json'  
 api\_key = '28d0d590186f456190d590186f0561ff'  
 URL\_base = f'https://api.weather.com/v3/wx/forecast/hourly/{days}day?geocode={latlon}&format={format}&units={units}&language={language}&apiKey={api\_key}'  
 conn.request("GET", URL\_base, headers=headers)  
 res = conn.getresponse()  
 data = res.read()  
 data\_json = json.loads(data.decode("utf-8"))  
 return data\_json

### METEOMATICS

Meteomatics also requires a simple registration here: <https://www.meteomatics.com/en/sign-up-weather-api-test-account/>

Free account allows to test our technology and data quality for 2 weeks free of charge. Limitations: 1,000 queries (50 queries / minute, 10 queries / parallel).

Username and password will be delivered to your email mentioned in the registration form. It needs to be copied to *\_\_init\_\_()* of the class *Meteomatics\_forecast()* in *forecasts.py*.

class Meteomatics\_forecast:  
 def \_\_init\_\_(self, coordinates):  
 self.coordinates = coordinates  
 self.username = 'valentinasmelova\_smelova'  
 self.password = '629RuCVs9q'  
 self.model = 'mix'  
 self.startdate = datetime.datetime.utcnow().replace(minute=0, second=0, microsecond=0)  
 self.enddate = datetime.datetime.utcnow() + datetime.timedelta(days=3)  
 self.interval = datetime.timedelta(hours=1)

## FAQ

### How to add or delete a new bbox to start gathring the forecast data?

The list of rectangular areas (bboxes) for which data is actively collected is in the main project directory and is called *“bbox\_list.csv”*. In order to add a new bbox – you just need to add a new line to the table.

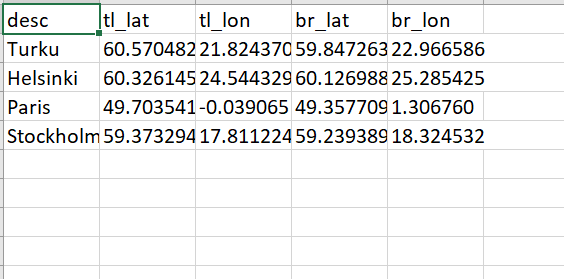


Table has 5 columns:

* desc – description, it can be whatever
* tl\_lat – top left corner latitude
* tl\_lon – tol left corner longitude
* br\_lat – bottom right corner latitude
* br\_lon – bottom right corner longitude

If you want to stop tracking the weather data for some bbox – you just need to delete corresponding line from *“bbox\_list.csv”* file.

### How to add new Observation source?

There are 2 data sources with observation added: FMI(Finnish Meteorological Institude) and Aeris. FMI is local Finnish one, has data only for Finnish locations and Aeris is global. All available aeris stations can be found on [HERE](https://www.pwsweather.com/map/).  Please note the as you zoom into a location more and more stations will become visible to not create too much congestion in the standard view.

In order to add a new observation source you need:

1. You need to create a new class in *stations\_observations.py* file.
2. New class should have some methods that help to get the response from API for one specific coordinate.
3. The final output of get\_observations method should have the folowing format:

dataframe with the folowing columns:

'id', 'lat', 'lon', 'source', 'timezone', 'datetime', 'temp\_2', ‘rhum\_2’, ’winds\_10’, ‘windd\_10’

where:

'id' - id of the station

‘lat’ – latitude of the station

‘lon’ – longitude of the station

‘source’ – just some string that specify the name of the new observation source (example: ‘FMI’, ‘Aeris’)

‘timezone’ – timezone of the station

‘datetime’ - datetime of the observations (required format is ‘YYYY-MM-DD hh:mm:ss’)

'temp\_2' – air temperature in C

‘rhum\_2’ – relative humidity in %

’winds\_10’ – wind speed in ms

‘windd\_10’ - wind direction in degrees

In case there will be some more metrics added for tracking – they need to be added to this list as well.

As a side comment, it is beneficial to add *“metrics” and startdate* parameters to *get\_observations* method, because the scope of weather metrics and time horizon might change in the future.

1. New class also should be added to *observation\_requests.py* file. This class should have a method *export\_observations* that allows to query observations for multiple bboxes in cycle or at once (depending on the API). This method should return the dataframe of the same format as described above for all listed bboxes for the past 12 hours.
2. In addition to observations you also need to export available properties of the stations, whose observations were loaded. Some APIs allow to export directly all the stations at once from a separate URL, some APIs do not have such possibility.

For example, FMI allow to export all the stations and their properties at once, that is why there is a separate class called *Stations\_fmi* in *stations\_observations.py* file. This class has a method *get\_stations(),* this method returns a dataframe with all listed stations: 'id','name','location','properties','source'. Then, class *Observations\_FMI in observation\_requests.py* has a method *export\_stations()* that simply exports all the stations.

Another example is Aeris, it does not allow to export available stations separately, stations’ properties are stored inside the observations response. That is why class *Aeris\_observations* in *stations\_observations.py* file has a separate method *get\_stations()* that gathers the properties of the stations from the given bbox. Then, class *Observations\_Aeris in observation\_requests.py* file has a method *export\_stations()* for a list of bboxes.

1. The last step related to the new observations and stations – method *import\_observations()* of an *app* class in main.py file should be padded with new lines of code. This code should try to load past observations and earlier queried stations from the same source and stack only the new ones. Example from FMI is below:

# FMI Observations  
 fmi = Observations\_FMI()  
 fresh\_observations\_fmi = fmi.export\_observations(bbox\_list)  
 os.chdir(os.getcwd() + '\Observations')  
 try:  
 old\_observations\_fmi = pd.read\_csv('fmi\_observations\_full.csv')  
 old\_observations\_fmi['datetime'] = pd.to\_datetime(old\_observations\_fmi['datetime'])  
 except:  
 old\_observations\_fmi = pd.DataFrame()  
 combined\_observations\_fmi = pd.concat([old\_observations\_fmi, fresh\_observations\_fmi]).drop\_duplicates()  
 combined\_observations\_fmi.to\_csv('fmi\_observations\_full.csv', index=False)  
 os.chdir(oldpwd)  
 fmi.export\_stations()

### How to add a new Forecast source?

There are 6 distinct Forecast sources added to an application already: FMI, Aeris, YR, The Weather Channel, Accuweather, Meteomatics. All services except FMI are global.

In order to add a new Forecast source you need:

1. You need to create a new forecast class in *forecasts.py.* This class should have some methods that return a dataframe(forecast for one coordinate) that has the folowing columns: 'metric\_name', 'datetime', 'metric\_value', 'collection\_datetime', 'fc\_leadtime', 'timezone'

Datetime columns should have a format of ‘YYYY-MM-DD hh:mm:ss’.

As a side comment, it is beneficial to add *“input\_metrics”* parameter to *get\_forecasts* method, because the scope of weather metrics might change in the future.

1. Then a new Forecast\_X class needs to be created *in forecast\_requests.py file.* This class should have a method *export\_forecasts().* This method exports the forecast for the list of coordinates. Returned dataframe should have the folowing columns: 'id', 'lat', 'lon', 'source', 'timezone', 'collection\_datetime', 'datetime', 'fc\_leadtime', "temp\_2", "rhum\_2", "winds\_10", "windd\_10". Datetime columns must have a format of ‘YYYY-MM-DD hh:mm:ss’.
2. The last step is to add a few more lines to the method *import\_forecast()* of an *app* class in *main.py.* Example is below:

#METEOMATICS FORECAST  
try:  
 meteomatics = Meteomatics()  
 meteomatics\_new\_forecast = meteomatics.export\_forecasts(locations\_list)  
 os.chdir(oldpwd)  
 os.chdir(os.getcwd() + '\Forecasts')  
 now = datetime.datetime.utcnow().strftime("%Y%m%d%H%M%S")  
 meteomatics\_new\_forecast.to\_csv(f'meteomatics\_new\_forecasts\_full{now}.csv', index=False)  
except:  
 print("Meteomatics forecast was not reloaded")

Csv-file must contain a datetime stamp in the filename.

### How to add a new metric for tracking?

There are 4 metrics included into this project: temperature, relative humidity, wind speed and wind direction. In order to add a new metric first of all you need to get its exact name in each observation and forecast source. List of available metrics should be available in API documentations.

Exact metric names should be added to:

* Method *export\_observations* of class *Aeris\_Observations* in *observation\_requests.py +* in the rename function

class Aeris\_Observations:  
 def export\_observations(self, bbox\_list):  
 limit = 100  
 metrics = ['humidity', 'tempC', 'windDirDEG', 'windSpeedKPH']

fmi\_observations\_dataframe = fmi\_observations\_dataframe.rename(columns={'Air temperature': "temp\_2", 'Relative humidity': "rhum\_2", 'Wind speed': "winds\_10", 'Wind direction': "windd\_10"})  
return fmi\_observations\_dataframe

* Method *export\_forecasts()* of class *Forecast\_Aeris* in *forecast\_requests.py* file *+* in the rename function
* class Forecasts\_Aeris:  
    
   def export\_forecasts(self, locations\_list):  
   limit = 48  
   metrics = ['humidity', 'tempC', 'windDirDEG', 'windSpeedKPH']

aeris\_observations\_dataframe = aeris\_observations\_dataframe.rename(columns={"tempC": "temp\_2", "humidity": "rhum\_2", "windSpeedKPH": "winds\_10", "windDirDEG": "windd\_10"})  
aeris\_observations\_dataframe['winds\_10'] = aeris\_observations\_dataframe['winds\_10']\*1000/3600

* Method *export\_forecasts()* of class *Forecasts\_YR* in *forecast\_requests.py* file **+ in the rename function**
* Method *export\_forecasts()* of class *Forecasts\_Accuweather* in *forecast\_requests.py* file **+ in the rename function**
* Method *export\_forecasts()* of class *Forecasts\_Weather\_Channel* in *forecast\_requests.py* file **+ in the rename function**
* Method *export\_forecasts()* of class *Forecasts\_Meteomatrics* in *forecast\_requests.py* file **+ in the rename function**

New added metric should have a suitable name that is commonly used by Vaisala verification tool.

### How to change a time horizon, how far do we look?

Developed solution loads the weather forecast for the next 48 hours. If you need to change this time horizon - you need to dothe folowing changes:

* Method *get\_forecasts* of a class *yr\_forecasts* in *forecasts.py.*
* def get\_forecasts(self, input\_metrics):  
   timeseries = self.get\_data()['properties']['timeseries']  
   fc\_period\_i = []  
   fc\_metric\_value\_i = []  
   fc\_metric\_name\_i = []  
   enddate\_yr = datetime.datetime.utcnow() + datetime.timedelta(hours=48)
* Method *get\_data()* of a class *Accuweather\_forecast* in *forecast.py*

def get\_data(self):  
 url\_location\_key = f'http://dataservice.accuweather.com/forecasts/v1/hourly/12hour/{str(self.code\_location())}?apikey={self.api\_key}&language=en-us&details=true&metric=false'  
 response = req.get(url\_location\_key, headers={"APIKey": self.api\_key})  
 json\_data = json.loads(response.text)  
 return json\_data

* *Method \_\_init\_\_() of a class Meteomatics\_forecast in forecast.py*

class Meteomatics\_forecast:  
 def \_\_init\_\_(self, coordinates):  
 self.coordinates = coordinates  
 self.username = 'valentinasmelova\_smelova'  
 self.password = '629RuCVs9q'  
 self.model = 'mix'  
 self.startdate = datetime.datetime.utcnow().replace(minute=0, second=0, microsecond=0)  
 self.enddate = datetime.datetime.utcnow() + datetime.timedelta(days=2)  
 self.interval = datetime.timedelta(hours=1)

* Method get\_data() of a class weather\_channel\_forecast in forecasts.py

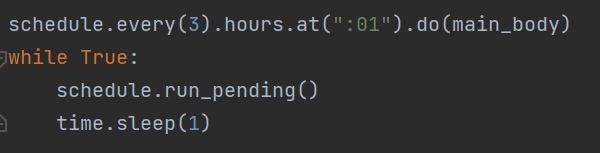
def get\_data(self):  
 import http.client  
 conn = http.client.HTTPSConnection("api.weather.com")  
 headers = {'accept': "application/json; charset=UTF-8"}  
 days = 2  
 latlon = str(self.lat)+','+str(self.lon)

* Method *expot\_forecasts* of a class *Forecast\_Aeris* in *forecast\_requests.py* file
* def export\_forecasts(self, locations\_list):  
   limit = 48  
   metrics = ['humidity', 'tempC', 'windDirDEG', 'windSpeedKPH']

If there have been already added some new forecast sources – time horizon parameter there should also be changed.

### How to change a frequency of the reloading?

For scheduling I have used *scheduler* library of Python.



If you want to change the frequency of the loading - adjust patameter of the method “*every*” of scheduler class the *main.py* file.

### How to debug, in case some Forecast or some observations were not loaded?

Logging library of Python was used to write possible error messages to the text file. Basically, as this application heavily depends on external APIs, all possible error messages are about observations or forecasts that were not loaded for some reason from external APIs. Application main run won’t stop in case observations or forecast was not loaded for some reason from API X for a specific coordinate, but there was a line written to the log file. Here are some examples:

ERROR:root:2022-08-15 12:01:03.739303: Observations for [49.703541, -0.039065, 49.357709, 1.30676] were not loaded from FMI.

ERROR:root:2022-08-15 12:01:03.773033: Observations for [59.373294, 17.811224, 59.239389, 18.324532] were not loaded from FMI.

ERROR:root:2022-08-15 12:01:41.338943: Forecast for LFOH of Aeris observation source was not loaded from FMI

ERROR:root:2022-08-15 12:01:41.453768: Forecast for MID\_E6240 of Aeris observation source was not loaded from FMI

ERROR:root:2022-08-15 12:01:44.286484: Forecast for MID\_E8226 of Aeris observation source was not loaded from FMI

ERROR:root:2022-08-15 12:01:44.451592: Forecast for MID\_G1111 of Aeris observation source was not loaded from FMI

There will be always a new log file created after each 24 hours. This timestep (24 hours, 1 d) can be also changed in the function *“create\_timed\_rotating\_log”* in *main.py file.*