McView WebDashboard

Release 1

RAMDE ISMAIL & RIVERA HECTOR

Jul 15, 2021

CONTENTS:

CHAPTER

ONE

MCVIEW WEBDASHBOARD

1.1 app module

The *app.py* module is the main script of our Dash application. This script is composed of two parts : the **layout** (app.layout) and the **Dash Callbacks**. It also contains package loads and imports of other modules that are necessary for the application to work, several independent commands (loading and preprocessing of data, etc.) and several functions.

- **app.layout** describes what the application looks like and is a hierarchical tree of components. the dash_html_components library provides classes for all HTML and keyword arguments describing HTML attributes such as style, className, and id. the dash_core_components library generates higher level components such as controls and graphics.
- The **callback function** allows to update the *output* component on the page (the HTML div) each time the value of the *input* component (the text zone) is modified.

`filter_dataframe`(*df1*, *year_slider*, *month_slider*)

Parameters

- **df1** – the data frame containing PT (Pressure), TT (Temperature Transmission), FT (Flow Transmission) and PCV (Pressure Control Valve) data. This data set has been loaded in the top part of the code.
- **year_slider** – recovers the selected year
- **month_slider** – recovers the selected month

Returns filtered_df (filtered data).

The inputs and outputs of following functions are retrieved in the decorator **@app.callback**.

`update_figure`(*year_slider*, *month_slider*, *curve_selector*, *graphe_selector*, *station_selector*)

In this function we use two functions. The first one is the **filter_dataframe** function to filter the data according to the user's selection. And the second one is the **update_figure_general** function implemented in another script (*update_figure.py*). This allows us to simplify our main script (*app.py*).

Parameters

- **year_slider** – which retrieves the selected year
- **month_slider** – which retrieves the selected month
- **curve_selector** – which retrieves the selected curve
- **graphe_selector** – which retrieves the selected graph
- **station_selector** – which retrieves the selected station

Returns figure (graph of data evolution over time)

update_figure_energy(*year_slider*, *month_slider*, *station_selector*)

Here we also use the **filter_dataframe** function and the **update_figure_energy_general** function, another function implemented in the *update_figure.py* script.

Parameters

- **year_slider** – which retrieves the selected year
- **month_slider** – which retrieves the selected month
- **station_selector** – which retrieves the selected station

Returns figure (graph of energy consumption)

update_figure_motors(*year_slider*, *month_slider*, *station_selector*)

As the two previous functions, we use the **filter_dataframe** function and the **update_figure_motors_general** function contained in the *update_figure.py* script.

Parameters

- **year_slider** – which retrieves the selected year
- **month_slider** – which retrieves the selected month
- **station_selector** – which retrieves the selected station

Returns figure (barplot graph of motors)

update_offline(*data*)

Parameters **data** – data of stations status

Returns value (number of offline stations)

update_not_available(*data*)

Parameters **data** – data of stations status

Returns value (number of not available stations)

update_coming_soon(*data*)

Parameters **data** – data of stations status

Returns value (number of coming soon stations)

update_available(*data*)

Parameters **data** – data of stations status

Returns value (number of available stations)

1.2 update_figure

In this script three functions are implemented, namely *update_figure_general*, *update_figure_energy_general* and *update_figure_motors_general*. It was created in order to make the main *app.py* script less busy and more fluid. It is imported into the *app.py* script with the following command: **import update_figure as upfig** As an example, *upfig.update_figure_general([parameters])* allows to use the *update_figure_general* function in the *app.py* script.

Translated with www.DeepL.com/Translator (free version)

`update_figure_general`(*year_slider*, *month_slider*, *curve_selector*, *graph_selector*, *filtered_df*)

Parameters

- `year_slider` – retrieves the selected year
- `month_slider` – retrieves the selected month
- `curve_selector` – retrieves the selected curve
- `graphe_selector` – retrieves the selected graph
- `filtered_df` – data filtered according to the user's selection.

Returns figure (graph)

`update_figure_energy_general`(*year_slider*, *month_slider*, *filtered_df1*)

Parameters

- `year_slider` – retrieves the selected year
- `month_slider` – retrieves the selected month
- `filtered_df` – filtered_df: data filtered according to the user's selection

Returns figure (graph)

`update_figure_motors_general`(*year_slider*, *month_slider*, *station_selector*)

param year_slider retrieves the selected year

Parameters

- `month_slider` – retrieves the selected month
- `filtered_df` – filtered_df: data filtered according to the user's selection

Returns figure (graph)

1.3 read_write_google_API module

This script is used to establish the connection to the Google cloud via an API. It also contains functions used to write in the Google spread Sheet, to open a database of tables from the Server and to structure our different data frame.

`Insert_DF_inGsheets`(*gSheetKey*, *gsheetindex*, *df*)

This function allows to write in the Google spread sheet.

Parameters

- `gSheetKey` – Google spread Sheet key

- **gsheetindex** – Google spread Sheet index (0 - first sheet, 1 - second sheet etc)
- **df** – Data Frame

Returns Do not return but export the dataframe to the Google Sheet.

openDF(*gSheetKey*, *index*)
openDF function allows to open a Table Database from Server.

Parameters

- **gSheetKey** – Google spread Sheet key
- **index** – Index value (0 - first sheet, 1 - second sheet etc.)

Returns Data Frame

SortH2MDistributionData()
This function allows us to structure the data according to our needs in the web application for the construction of the first graph. The function takes nothing as input but uses the **openDF** function.

Returns It returns the Data Frame in the following form:

| Date | Year | Month | Day | Time | PT | TT | FT | PCV |

SortH2MEnergyData()

Returns Data Frame in the following form:

| TimeStr | Year | Month | Day | Hour | kVA | KVAR | kW |

SortH2MotorsData()

Returns Data Frame in the following form:

| TimeStr | Year | Month | Day | Hour | Air_Compressor | D1_Cooling_Unit |

Data_MAP()
The Data_MAP function takes nothing as input but uses **openDF** function. It retrieves the location data of McPhy stations and their current status. This data is used in the satellite overview presentation.

Returns It returns the Map Tableau in the following form:

| Latitud | Longitud | Country | City | Project | Status | Status Value |

CHAPTER

TWO

THE WEB APPLICATION IN PRACTICE

2.1 The data

To feed our dashboard , we need to access Google Cloud through an API. In this database is historicized the data from the stations every one hour. Each data is then modeled according to our needs and recorded in Google spread Sheet and this, in real time. Thus, 4 Google spread Sheet data sets will be created:

- the transmission pressure (PT), the temperature (TT), the transmission flow (FT) and the valve control pressure (PCV) according to time (year, month, day, hour)
- the location of all McPhy stations and their status
- energy consumption (kVA, KVAR, kW) as a function of time (year, month, day, hour) - motor data (air compressor, D1 cooling unit, air extractor) as a function of time (year, month, day, hour)

2.2 Graphics

In the dashboard, five (05) types of graphs are used to illustrate the data.

- Time series to follow in a first graph the evolution of the transmission pressure, temperature, transmission flow and control pressure of the valves over time, and in a second graph the evolution of the energy consumption.
- A Mapbox that allows a statistical representation of numerical data of transmission pressure, temperature, transmission flow and valve control pressure over time through their quartiles and median.
- A Mapbox Maps, which is a map that displays in a web browser the location of all hydrogen stations and their status (Offline, Not Available, Coming Soon and Available) by attaching individually requested vector data.
- The Sunburst graph, equivalent to a pie chart, visualizes the hierarchical location data of the different stations.
- A bar chart to represent the data of the motors (air compressor, D1 cooling unit, air extractor)

2.3 How to run the app locally?

First create a virtual environment with conda or venv inside a temp folder, then activate it.

virtualenv venv

Windows
venvScriptsactivate
Or Linux

source venv/bin/activate

Download the folder on your computer or Clone the git repo, then install the requirements with pip

In your terminal type :

- git clone [the link of the project on github]
- cd [position yourself in the folder]
- pip install -r requirements.txt or :
  - pip install dash
  - pip install jypyter-dash
  - pip install pandas

Run the app

python app.py

After execution of the script app.py we obtain : *$ python app.py . . . Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)*

open this link: http://127.0.0.1:8050/ to see the application

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX