



Hogeschool van Amsterdam

Technical User Manual

Project: Sport Data Valley Baseball
Semester: Big Data
Version: 0.1
Date: 11-01-2021

Erenay Aykilic	500801090	BD1-B1
Daniel IJsbrandij	500803211	BD1-B1
Bram Minderhoud	500714605	BD1-B1
Shayeed Sohabatali	500760757	BD1-B1
Michelle Spong	500832751	BD1-B1
Hassan Adnane	500805193	BD1-B2
Marloes Flier	500806454	BD1-B2
Kasper Wiepking	500804715	BD1-B2
Koen Zwaagstra	500770009	BD1-B2
Michael Visser	500823669	BD1-B1
Abdellah Amnad	500799703	BD1-B1
Shan Hu	500684659	BD1-B1
Semih Simsir	500830526	BD1-B1
Wang Di Jiang	500826958	BD1-B1



Table of contents

Introduction	3
1.- Web scraping	4
1.1 Libraries	4
1.2 External sources	4
1.2.1 URLs	4
1.3 Web scraping scripts	5
1.3.1 CSV	5
2.- Storing the data	6
2.1 Database	6
2.1.1 Set up	6
2.1.2 PHPMyadmin	8
2.1.3 Tables in database	8
2.2 Login credentials	9
2.3 Fetch & Store data in tables	10
2.3.1 Fill database with MASTER_SCRIPT.py	10
2.3.2 Import data from CSV-files	11
2.3.3 SQL Statements for retrieving data to the dashboard	11
2.4 GIT	11
3.- Visualisation	11
3.1 Required library	11
3.2 Dash	13
3.2.1 How to run the dashboard app	13
3.2.2 Adding a page and URL Support	15
3.2.3 Layout and CSS	16
3.2.4 Working with callbacks	16
4.- Docker	18
References	20



Hogeschool van Amsterdam

Introduction

This document will cover all the technical documentation that provides information about the technical parts of the project.

The topics that will be covered are:

- Web scraping
- Storing the data in a database
- Visualizing the data
- Dashboard files and folder structure
- Docker



1.- Web scraping

Web Scraping is the technique employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database in a table (spreadsheet) format.

Web Scraping can be divided into several steps, where the first step is to send HTTP-requests to the URL of the web page that you want to access. Once you've accessed the content, you use a HTML parser which creates a nested/tree structure of the HTML data.

1.1 Libraries

Beautifulsoup is a Python library for pulling data out of HTML & XML-files where the structure of doing this is defined by the parentheses which contains the raw HTML content and the specified HTML parser. For this project, Python's own html-parser was the one that was used.

1.2 External sources

The sources from which the data was extracted from the dutch baseball league (KNBSBstats) together with MLB (Major League) and MiLB (Minor League) of the american baseball leagues. However, other sources have been used in combination with the URL's from these. These can be found under the URLs topic.

1.2.1 URLs

URL to scrape all the player names:

<https://www.honkbalsite.com/profhonkballers/>

For each name that is scraped from honkbalsite, the corresponding player page is scraped from baseball-reference (for the american Minor Leagues and Major Leagues)

URL for scraping the tables from Major League: <https://www.baseball-reference.com/players/gl.fcgi?id=>

URL for scraping the tables from Minor League: <https://www.baseball-reference.com/register/player.fcgi?id=>

Dutch players that have been playing the dutch league, the player statistics were scraped from KNBSBstats (which is the Dutch baseball league).

URL for players from the dutch league:

<https://www.knbsbstats.nl/archief/>

In addition to the websites that have been scraped, external csv-files have been used to get the Team names by using the teamID, year and teamIDBR. These can be found and downloaded from the following URL from github:

<https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Teams.csv>



However, the URL is not used alone (since that would make the project dependent on the URL), but in addition, downloaded as an external csv-file with all the team-names to prevent any unexpected events that might occur if the website shuts down or isn't accessible for some reason.

1.3 Web scraping scripts

There are 12 web scraping files which can all be found in the scraper_files → Scripts folder

The KNBSB.py file scrapes all the data from the Dutch Baseball League. It also contains the majority of the cleaning.

The Honkbalsite.py file scrapes the names of all active, international Dutch baseball players found on <https://www.honkbalsite.com/profhonkbatters/>. These names are then used to search on <https://www.baseball-reference.com> for the corresponding pages of all players.

With the pages scraped by the Honkbalsite.py file, scraping of the actual data is possible. MajorLeague.py and MinorLeague.py scrape the MLB and MiLB data respectively. It also contains the majority of the cleaning. Because of missing data, the MiLB data does not include fielding statistics

A small amount of scraping is also done by the KNBSB_CleanNames_BatFld.py and KNBSB_CleanNames_Pitching.py files. These files scrape all the names of the players in the Dutch Baseball League in the regular season between 2010 and 2020. These names are then matched with the raw but partial names scraped by the KNBSB.py file. This ensures the data contains the full name of all the baseball players in the whole database.

The MLB and the MiLB scrapers scrape all the players with their given id. These players will be presented in the "Add Players" page.

The Playerscraper will scrape all the statistics and data from a specific player. This scraper will be triggered once a player is added via the "Add Players" page.

There are 5 tournaments scrapers. They gather all the statistics from the players and store them into a csv.

1.3.1 CSV

Most of the data-scraping files output temporary CSV files to the Data_retrieval → Results folder.

These files will later be split and stored on the SQL server by the CombiningCSVsToSql.py file



Hogeschool van Amsterdam

2.- Storing the data

In order to run the web application and visualize the data with graphs, the data is first scraped from external sources and then stored in a database. The storing is done in a private database, which in turn was stored on the Oege server provided by Hva.

2.1 Database

Oege is a server to support student and staff projects and courses. If you study or teach at one of the FDMCI courses. Access to Oege can be requested via the following link:

<https://oege.ie.hva.nl/registratie/>

It is also possible to set up a different server, locally or on another server.

2.1.1 Set up

Diensten



Mijn account

Met de Aanvragen-knop krijg je een UNIX en MySQL account. De Aanvragen-knop is ook om alleen het unix wachtwoord te resetten als je dat bent vergeten.

[Aanvragen](#) [Voorwaarden](#)

Figure 1 - Oege-overview



Hogeschool van Amsterdam

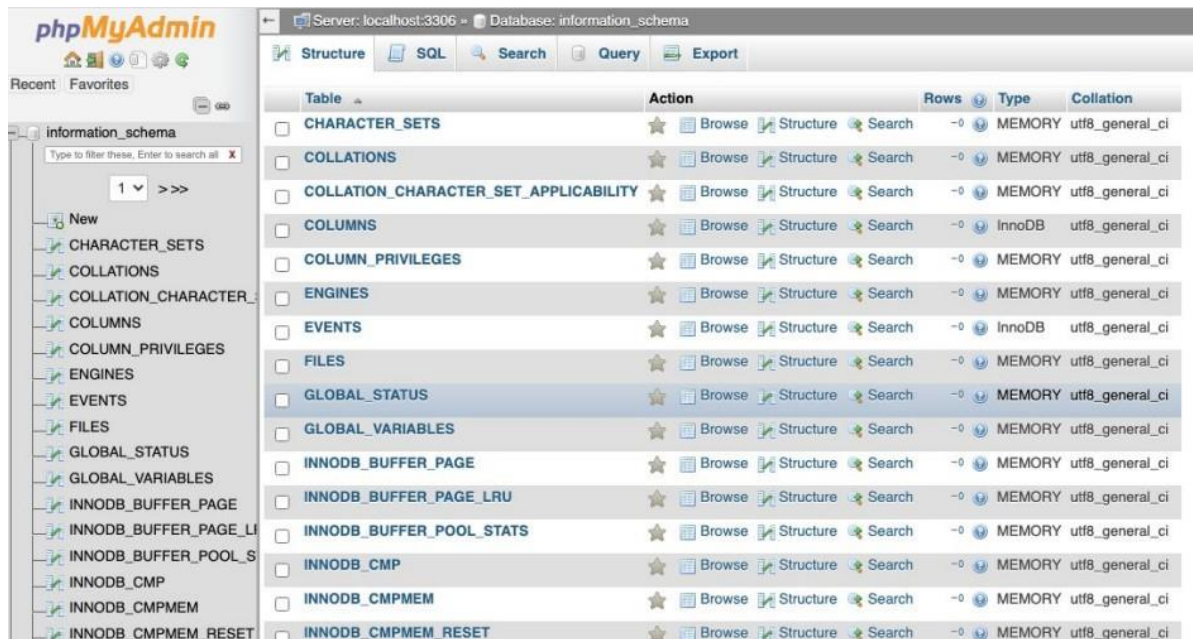
In order to set up a database, the first step is to request for an account in UNIX and MySQL. The username that is provided is an automatically generated username according to the student's email address generated by HvA. This means that the username is the first part before the '@' sign in the email address. Upon successful registration, a password for Oege is automatically generated and sent by email. The registration is only possible if you have an AUAS-ID account with a valid password.



2.1.2 PHPMYadmin

When the account has been set up, log into the Oege server by using your username and the password that was sent to you by email.

The database that will be found now is the information_schema which contains tables from multiple categories. In here, you can see information about plugins, character sets, engines etc. These in turn are selected and displayed with SQL queries.



The screenshot shows the phpMyAdmin interface with the 'information_schema' database selected. The left sidebar lists various tables, and the main area displays a table with columns: Table, Action, Rows, Type, and Collation. The tables listed include CHARACTER_SETS, COLLATIONS, COLLATION_CHARACTER_SET_APPLICABILITY, COLUMNS, COLUMN_PRIVILEGES, ENGINES, EVENTS, FILES, GLOBAL_STATUS, GLOBAL_VARIABLES, INNODB_BUFFER_PAGE, INNODB_BUFFER_PAGE_LRU, INNODB_BUFFER_POOL_STATS, INNODB_CMP, INNODB_CMPMEM, and INNODB_CMPMEM_RESET.

Table	Action	Rows	Type	Collation
CHARACTER_SETS	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
COLLATIONS	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
COLLATION_CHARACTER_SET_APPLICABILITY	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
COLUMNS	★ Browse Structure Search	~0	InnoDB	utf8_general_ci
COLUMN_PRIVILEGES	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
ENGINES	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
EVENTS	★ Browse Structure Search	~0	InnoDB	utf8_general_ci
FILES	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
GLOBAL_STATUS	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
GLOBAL_VARIABLES	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_BUFFER_PAGE	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_BUFFER_PAGE_LRU	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_BUFFER_POOL_STATS	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_CMP	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_CMPMEM	★ Browse Structure Search	~0	MEMORY	utf8_general_ci
INNODB_CMPMEM_RESET	★ Browse Structure Search	~0	MEMORY	utf8_general_ci

Figure 2 - Data tables-overview

In addition, you'll find information regarding the server and the web socket on the home page to the right of your screen.

However, an additional database is created and provided for each user (zminderb002 in our case). It is in this database the user is able to create new tables and import data.

2.1.3 Tables in database

From the CSV-files that were extracted from the web scraped sources, the following tables were created in the database (zminderb002):

- allPlayers
- db_Batting_Career
- db_Batting_Gamelogs
- db_Batting_Summary
- db_Batting_Yearly
- db_Fielding_Career
- db_Fielding_Gamelogs
- db_Fielding_Summary
- db_Fielding_Yearly



Hogeschool van Amsterdam

- db_Pitching_Career
- db_Pitching_Gamelogs
- db_Pitching_Summary
- db_Pitching_Yearly
- tournaments

These tables are divided into three categories; batting, pitching and fielding, where each category has multiple tables that each belong to a sub-category. The career-tables contain data for distinct players whereas the summary tables summarize the data according to the teams in which different players have played in.

The gamelogs and yearly tables are quite self-explanatory, where the first subcategory stores gamelogs data and the second one contains yearly data.

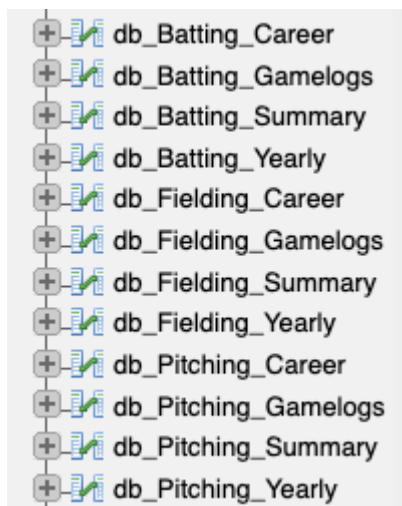


Figure 3 - Database tables layout

2.2 Login credentials

The login credentials for the database (as in MySQL_data.py) are as follows:

```
host = "oege.ie.hva.nl"
username = "minderb002"
password = "+#/tRfRM5zwhnN"
database = "zminderb002"
```



2.3 Fetch & Store data in tables

2.3.1 Fill database with MASTER_SCRIPT.py

It is best to set up your own database instead of using the existing one, for this you have to follow the steps above to get access to your own Oege server. It is still possible to use the existing one.

When you have set up your own database edit the SQL credentials in the MySQL_data.py file to the credentials you have received.

In order to upload the CSV-files to the database on the Oege server, the first step is to import a Python library that handles SQL requests and queries. In python, you can import create_engine from sqlalchemy, which creates a new connection to a specified database.

When the SQL credentials are updated to your own credentials, you have to adjust the MASTER_SCRIPT.py file in order to fill the database. The adjustment is as follows:

```
# dataCollection()
# Run every night at 3 AM docker uses UTC time (1 hour earlier)
schedule.every().day.at("04:00").do(dataCollection)
while True:
    schedule.run_pending()
    now = datetime.datetime.now()
    nextRun = schedule.next_run()
    print('Current Time: ', now.strftime("%H:%M:%S"), 'Next Run at: ',
nextRun.strftime("%H:%M:%S"))
    time.sleep(60)
```

The script will in this state only update the database every night at 3 AM. To run the script comment out the lines 43 till 49 and uncomment line 41. The script will look as follows:

```
dataCollection()
# Run every night at 3 AM docker uses UTC time (1 hour earlier)
# schedule.every().day.at("04:00").do(dataCollection)
# while True:
#     schedule.run_pending()
#     now = datetime.datetime.now()
#     nextRun = schedule.next_run()
#     print('Current Time: ', now.strftime("%H:%M:%S"), 'Next Run at: ',
nextRun.strftime("%H:%M:%S"))
#     time.sleep(60)
```

Now save the file and run the MASTER_SCRIPT.py and wait until the script finishes, this can take up to 3 hours.



2.3.2 Import data from CSV-files

If you are using another instance of PHPmyadmin (locally) it is possible to export the tables in .csv file format from the existing database and upload them to your own PHPmyadmin.

This is, unfortunately, not possible with the Oege server due to the size of the files. Oege only accepts files with a maximum of 2,048 kb. Therefore this way of filling the database is not possible for the Oege server.

2.3.3 SQL Statements for retrieving data to the dashboard

In order to retrieve data from the Oege server, SQL queries are used in the dashboard scripts to fetch data directly from the database. Regular SQL-statements can be used to select and retrieve data with specific conditions and characteristics.

2.4 GIT

The code for the dashboard and the webscraper (including the technical documentation and user manual) can be found by following the following link, for access to the Git please send an email to the email address below :

Gitlab	https://gitlab.fdmci.hva.nl/minderb002/baseball-hva-dashboard
Contact for access to GIT repository	bram.minderhoud@hva.nl



Hogeschool van Amsterdam

3.- Visualisation

3.1 Required library

The required libraries are listed in the requirements.txt file. The following libraries are used:



astroid==2.4.2
chromedriver == 2.24.1
webdriver-manager==3.3.0
beautifulsoup4==4.9.1
selenium==3.141.0
Brotli==1.0.7
bs4==0.0.1
certifi==2020.6.20
chardet==3.0.4
click==7.1.2
colorama==0.4.3
dash==1.13.3
dash-bootstrap-components==0.10.2
dash-core-components==1.10.1
dash-daq==0.5.0
dash-html-components==1.0.3
dash-renderer==1.5.0
dash-table==4.8.1
DateTime==4.3
Flask==1.1.2
Flask-Compress==1.5.0
Flask-Login==0.5.0
Flask-SQLAlchemy==2.4.3
future==0.18.2
gunicorn==20.0.4
idna==2.9
isort==4.3.21
itsdangerous==1.1.0
Jinja2==2.11.2
lazy-object-proxy==1.4.3
MarkupSafe==1.1.1
mccabe==0.6.1
mysql==0.0.2
mysql-connector-python==8.0.20
mysqlclient==1.4.6
numpy==1.18.5
pandas==1.0.5
plotly==4.8.1
protobuf==3.12.2
pylint==2.5.3
python-dateutil==2.8.1
pytz==2020.1
requests==2.24.0
retrying==1.3.3
six==1.15.0
soupsieve==2.0.1
SQLAlchemy==1.3.17



Hogeschool van Amsterdam

toml==0.10.1

typed-ast==1.4.1

urllib3==1.25.9

Werkzeug==1.0.1

wrapt==1.12.1

zope.interface==5.1.0

schedule

To install the required libraries navigate to the project folder in your terminal where the requirements.txt file is stored. Then use pip to install the requirements by typing the following command: **pip install -r requirements.txt**

Without these libraries the application won't run.



3.2 Dash

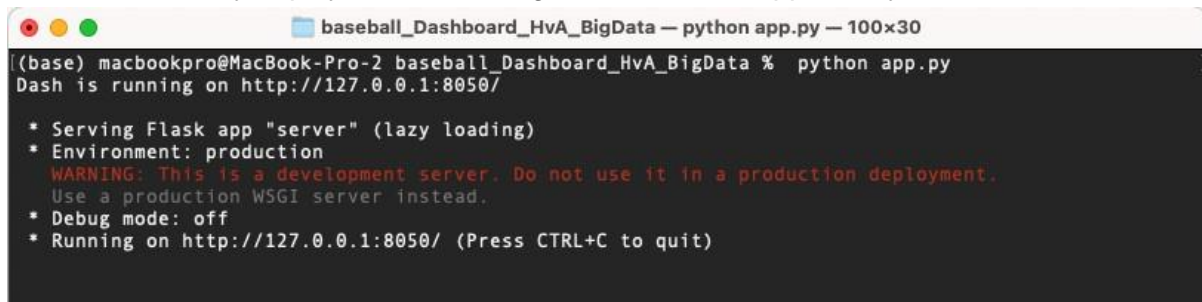
3.2.1 How to run the dashboard app

There are two ways to run the application. One is by using the terminal to run the application and the other way is to run it through an IDE (like Visual Studio Code). Both will be shown below.

Using the terminal (best applicable for Linux and OS X users)

To run the application via the terminal, navigate to the project folder in your terminal. When in the project folder type in the following command: **python app.py**

When successfully deployed the following information will appear in your terminal:



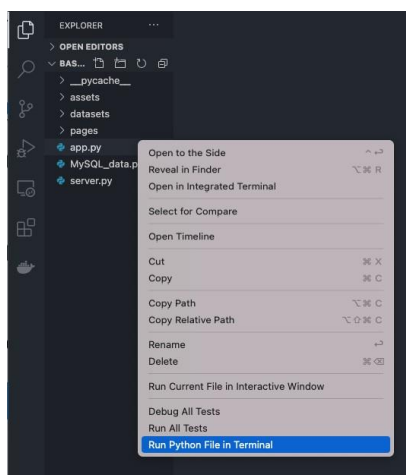
```
baseball_Dashboard_HvA_BigData — python app.py — 100x30
(base) macbookpro@MacBook-Pro-2 baseball_Dashboard_HvA_BigData % python app.py
Dash is running on http://127.0.0.1:8050/

* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

As shown in the terminal the dashboard can be accessed via the browser on the following url: 127.0.0.1:8050

Using an IDE (applicable for Windows users, also works on Linux and OS X)

To run the application in an IDE (in this case Visual Studio Code) open the project folder in Visual Studio Code. In the left explorer pane right click on the app.py file and select “*run Python file in terminal*” (as shown in **figure 5: running app in VS Code**).





Hogeschool van Amsterdam

When successfully deployed the following information will appear in your terminal:

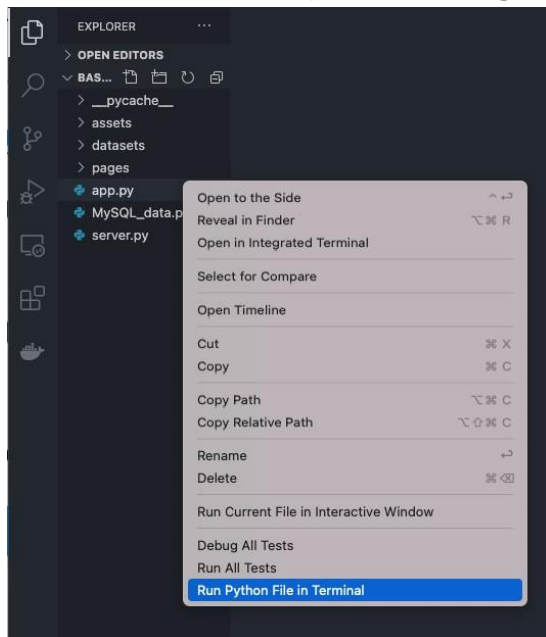
```
baseball_Dashboard_HvA_BigData — python app.py — 100x30
(base) macbookpro@MacBook-Pro-2 baseball_Dashboard_HvA_BigData % python app.py
Dash is running on http://127.0.0.1:8050/

* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

As shown in the terminal the dashboard can be accessed via the browser on the following url: 127.0.0.1:8050

Using an IDE (applicable for Windows users, also works on Linux and OS X)

To run the application in an IDE (in this case Visual Studio Code) open the project folder in Visual Studio Code. In the left explorer pane right click on the app.py file and select “run Python file in terminal” (as shown in figure 5: running app in VS Code).



When successfully deployed the following information will appear in the terminal in your IDE:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
(base) macbookpro@MacBook-Pro-2 baseball_Dashboard_HvA_BigData % /Users/macbookpro/anaconda3/bin/python /Users/macbookpro/Desktop/baseball_Dashboard_HvA_BigData/app.py
Dash is running on http://127.0.0.1:8050/

* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```




Hogeschool van Amsterdam

As shown in the terminal the dashboard can be accessed via the browser on the following url: 127.0.0.1:8050

Editing while deployed

If you want to edit the code while the application is running, the file first has to be saved before the changes will be visible in the dashboard. When an error has been made while the application is deployed the application will crash and show the error in the terminal. Fix the error and rerun the app.py file.

3.2.2 Adding a page and URL Support

Dash uses a “single page app”. This means that the application does not completely reload when the user navigates the application, making the browser very fast.

The picture below shows the index file. In this file you specify the routing for the application

```
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

from app import app
from apps import app1, app2

app.layout = html.Div([
    dcc.Location(id='url', refresh=False),
    html.Div(id='page-content')
])

@app.callback(Output('page-content', 'children'),
              Input('url', 'pathname'))
def display_page(pathname):
    if pathname == '/apps/app1':
        return app1.layout
    elif pathname == '/apps/app2':
        return app2.layout
    else:
        return '404'

if __name__ == '__main__':
    app.run_server(debug=True)
```

when making a multipage app you can specify the path name. For this example the pathname is “/apps/app1”. So when this path name is being used it returns the layout of the “app1” page.

A useful website that can be used when working with dash url routing is: <https://dash.plotly.com/urls>



3.2.3 Layout and CSS

The layout determines what the app will look like. Dash provides classes for all visual components of the application. By importing `dash_core_components` and `dash_html_components`, you can set up the layout.

The library `dash_core_components` (<https://dash.plotly.com/dash-core-components>) allows you to add special functions, such as a dropdown or a button.

The library `dash_html_components` (<https://dash.plotly.com/dash-html-components>) allows you to create Div and H1 tags.

To style the layout, a css stylesheet is used. To style the created dash components, the `className` must be created in the component by adding `className='Name of the className'`. To actually style the component, the `className` must also be called in the CSS.

3.2.4 Working with callbacks

A callback function is a function that is automatically called by Dash whenever an input of a component property changes. See the example below.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),
])

@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)

if __name__ == '__main__':
    app.run_server(debug=True)
```



Figure 6 - Callback function

By writing the `app.callback` decorator, dash is being told to call this function whenever the value of the “input” component (in this case the textbox) changes in order to update the children of the “output” component on the page.

You must use the same id you gave a Dash component in the `app.layout` when referring to it as either an input or output of the `@app.callback` decorator.



Whenever an input property changes , the function that the callback decorator wraps will get called automatically . Dash provides the function with the new value of the input property as an input property as input argument and Dash updates the property of the output component with whatever was returned by the function.

When the dash app starts, it automatically calls all of the function of the callbacks with the initial values of the input component in order to populate the initial state of the output components.

A useful website that can be used when working with dash callbacks is the following:

<https://dash.plotly.com/basic-callbacks>.



Hogeschool van Amsterdam

4.- Docker

The webscraper is dockerized the process of running the webscraper in a docker container will be explained below.

First install Docker Desktop by following the steps on this page:

<https://www.docker.com/products/docker-desktop>

In the WEBSCRAPER_BASEBALLDASHBOARD folder the Dockerfile can be found. This is build as follows:

```
FROM python:3

# define working directory to be created in docker container
WORKDIR /application

# copy requirements file to workdir
COPY requirements.txt .

# install all requirements in the requirements.txt file
RUN pip install --no-cache-dir -r requirements.txt

# copy the contents of the application folder to the workdir
COPY . .

# run the master script
CMD [ "python", "MASTER_SCRIPT.py" ]
```

To build the docker image open a terminal and navigate to the project directory. When in the project directory in a terminal type the following:

“docker build -t dockerpython .”

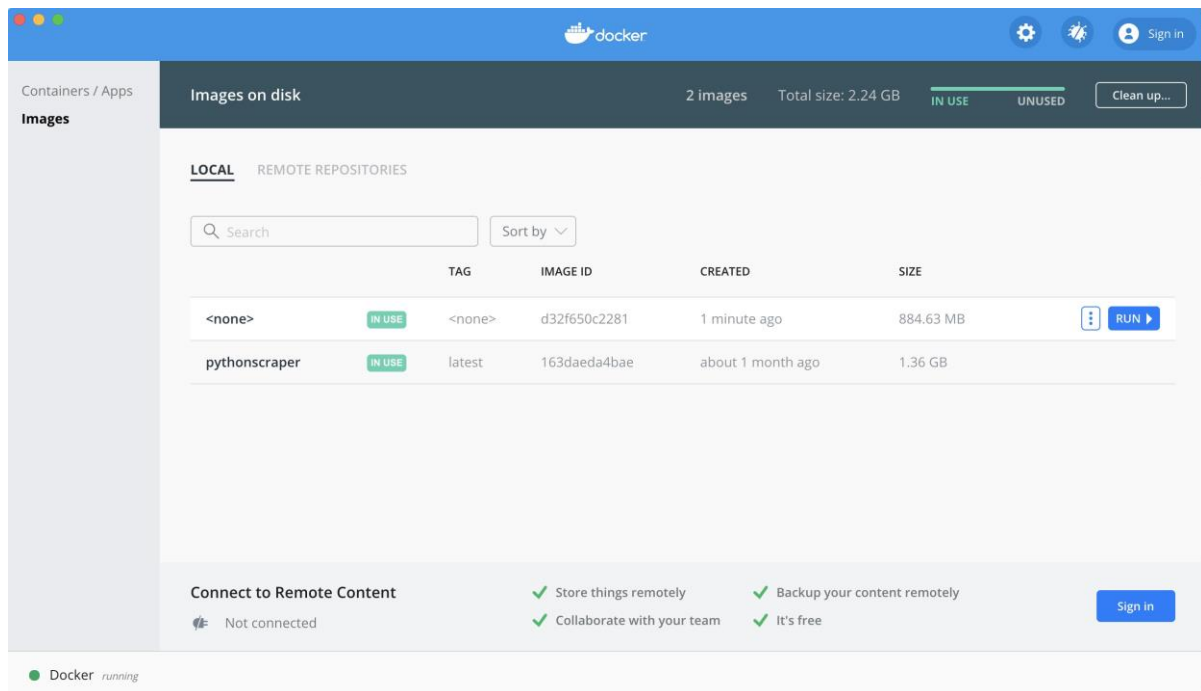
The name of the image is in this case “dockerpython” Docker will now build the container, this may take some time (about 15 minutes).

When completed the following message will be shown in the terminal and the container will appear in Docker Desktop:

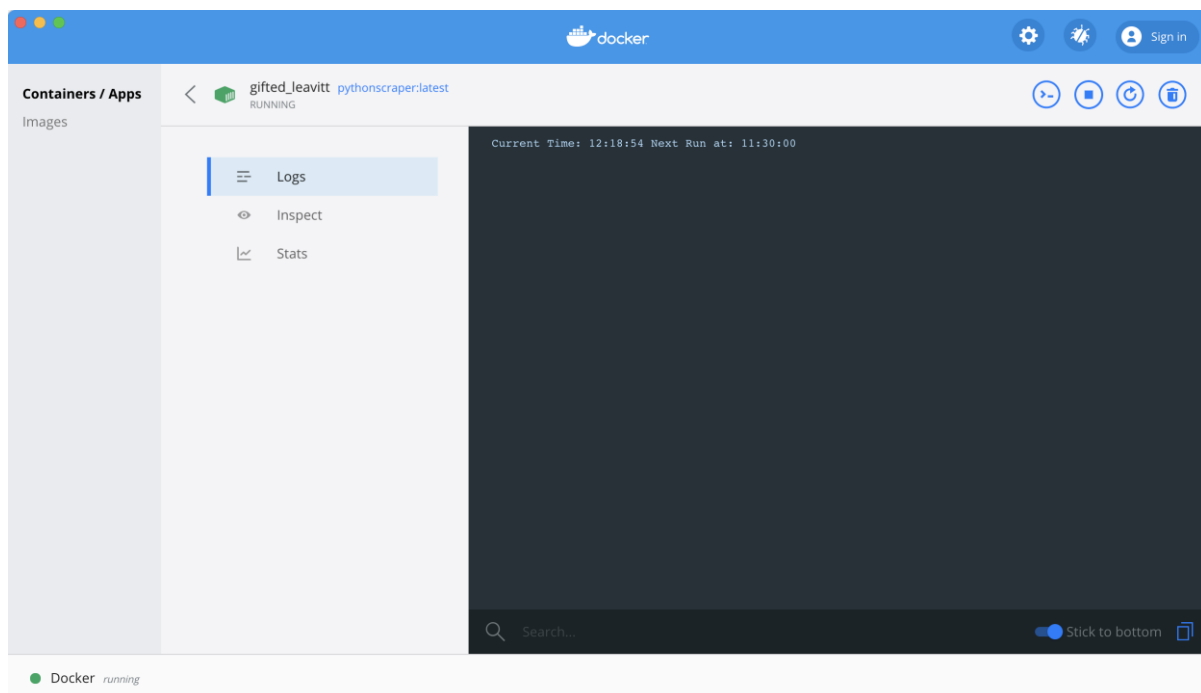
```
Successfully built 6a0f473750a7
Successfully tagged dockerpython:latest
```



In Docker Desktop the image will appear:



Run the just created image. When the image is runned it will appear in the Container/ Apps tab:



In the logs it shows when the MASTER_SCRIPT will be run next, it will run every night at 3 AM.



Hogeschool van Amsterdam

References

baseball-reference.com. (n.d.). Barry Bonds 2007 Batting Game Logs. Retrieved January 6, 2021, from <https://www.baseball-reference.com/players/gl.fcgi?id=>

Baseball-Reference.com. (n.d.-a). MLB Stats, Scores, History, & Records. Retrieved January 6, 2021, from <https://www.baseball-reference.com>

Baseball-Reference.com. (n.d.). Register Players Encyclopedia. Retrieved January 6, 2021, from <https://www.baseball-reference.com/register/player.fcgi?id=>

baseball-2-wasa. 2020. Sport Data valley Baseball app .GitHub; [accessed 2021 January 07]. <https://gitlab.fdmci.hva.nl/heijnew/baseball-2-wasa>

baseball-2-wasa technical documentation. Sport Data valley Baseball app .GitHub; [accessed 2021 January 07]. https://gitlab.fdmci.hva.nl/heijnew/baseball-2-wasa/-/blob/master/Documentation/Technical%20Documentation_Wasa%20.pdf

dash.plotly.com. (n.d.-a). Basic Callbacks | Dash for Python Documentation | Plotly. <https://dash.plotly.com/basic-callbacks>

dash.plotly.com. (n.d.). Dash Core Components | Dash for Python Documentation | Plotly. Retrieved January 7, 2021, from <https://dash.plotly.com/dash-core-components>

dash.plotly.com. (n.d.-b). Dash HTML Components | Dash for Python Documentation | Plotly. Retrieved January 7, 2021, from <https://dash.plotly.com/dash-html-components>

honkbalsite. (n.d.). Profhonkballers. Retrieved January 6, 2021, from <https://www.honkbalsite.com/profhonkballers/>

Honkbalsite. (n.d.). Profhonkballers. Retrieved January 6, 2021, from <https://www.honkbalsite.com/profhonkballers/>

KNBSB Stats. (n.d.). Archief – KNBSB Stats. Retrieved January 6, 2021, from <https://www.knbsbstats.nl/archief/>