MAUSTATS Developer KT Documentation

# Project Setup and Execution Guide

## 1. Running the Python Backend

To start the backend server:

* Navigate to the directory:  
  cd maustats/datahudi
* First Open the virtual environment

source venv/bin/activate

* Run the Django server:  
  python manage.py runserver
* Open another terminal, navigate to the same directory:  
  cd maustats/datahudi
* Start Huey for background tasks:  
  python manage.py run\_huey

## 2. Running the Frontend Interfaces

**Data Manager Interfaces**

* Navigate to the following directories and start each component:
  + **Data Import**  
    cd maustats/dataimport  
    npm start
  + **Data Supplier**  
    cd maustats/datasupplier  
    npm start
  + **Data Manager**  
    cd maustats/datamanager  
    ng s

**Dashboard Interface**

* Navigate to:  
  cd maustats/dashboard  
  ng s

**Note:** Only one of dashboard or data manager can run on the default port at a time. If you need both, start data manager first, then dashboard—the latter will automatically use a different port.

**Note:** for using Pyspark we need jar files which are needed for pyspark to work.

Path for jar file is stored in SparksessionCreator.py file. And this is the jar hudi-spark3.3-bundle\_2.12-0.13.1.jar file.

**Note:** collection used for Primary data is metadata and for generate report is report. Both can be found in .env file and in django project for maustats for primary data we are using mongo db and the file where it is present is CalculationConnectionCreator.py in utilities folder.

## Primary Data Processing

The **Primary Data** involves the following five key steps:

1. **Ingest**
   * Import .csv or .xlsx files into the system.
   * Files are converted into .parquet format using PySpark and stored in:  
     primaryData/{ID} folder.
2. **Filter**
   * Apply filters to the ingested .parquet file.
   * Save the filtered file back to the same folder.
3. **Clean**
   * On clicking "Clean Data", the system processes the file from the ID folder and saves the cleaned version in the CD (Cleaned Data) folder.
   * The "View Data" option under Clean step reads from the CD folder, whereas "Clean" action always reprocesses the original from the ID folder.
4. **Calculate**
   * System retrieves the cleaned file from the CD folder.
   * Performs calculations and saves the output in the AD (Analyzed Data) folder.
5. **Complete**
   * The final processed file is available for **download**.

**Dashboard Integration:**  
Once a file has passed through all steps and reaches the **Complete** stage, it becomes available on the **Dashboard** for visualization.

## Primary Data Processing Workflows

The Primary Data processing workflows involve the following key steps. These workflows are accessible at:

🔗 [**http://localhost:4200/manage-workflow**](http://localhost:4200/manage-workflow)

Each workflow represents a complete primary data process, encapsulated in a single step-by-step sequence.

### How to Achieve It

1. Ingest  
   When a file is successfully ingested, users have the option to "Save as Template". This saves the exact settings used during the ingestion process for future use.
2. Clean  
   After successful cleaning, users can save the cleaning configuration. This ensures repeatable cleaning actions for similar files.
3. Calculate  
   Similarly, users can save the calculation steps, making it easy to reapply the same logic to other files of the same structure.

💡 Templates saved in the above steps can be used in workflows to ingest and process multiple files that share the same structure.

1. Complete
   * The final processed file is made available for download.
   * Once a workflow completes, the file appears in the "Complete" section of the Primary Data.

### Dashboard Integration

After a file completes all workflow steps and reaches the Complete stage, it becomes available on the Dashboard for visualization and analysis.

## Generate Report Process

This process retrieves data stored in **MongoDB**, applies transformations, and generates pivot tables. It comprises **5 steps**—three in **Node.js** and two in **Python**.

**Step 1: Layout**

* Users select hierarchical levels for applicable columns.
* Python script HierarchyService.py handles the logic:
  + Extract data
  + Identify levels from DB
  + Process level relationships
  + Populate corresponding columns
* The system uses the payload to determine highest and lowest levels of the associated values. A column is created for each level accordingly.
* get\_filtered\_unique\_key() is used to fetch associated data from other linked columns.

Columns generated for selected levels are automatically marked for use in the pivot layout.

## Using Layout Scripts

For cases where the built-in report generator fails to meet structure requirements, a **CSV file** (pre-pivot) can be downloaded via an option in the report module. Scripts can then be run on this file to generate the required format.

These scripts cater to custom client structures and reporting needs.

## Report Modules and Scripts

There are four primary modules, each with its corresponding reports and transformation scripts:

* **Demography**
* **Trade**
* **Labour**
* **SBR (Statistical Business Register)**

A reference spreadsheet listing all scripts used is available here:  
**[Script Reference Sheet](https://docs.google.com/spreadsheets/d/1bcFEixrfBL4BHAiWGrjSq5ZvTRFGlW9pTSjh36HEoMU/edit?pli=1&gid=1973377292" \l "gid=1973377292)**

## Standard Rules Followed by All Scripts

1. **Input File Prompt**
   * Ask the user for the **input file name** and **path**.
2. **Report Title**
   * Prompt the user for the **report title**.
   * This title appears in **row 1** of the final Excel output and is included in the output filename.
3. **Output Location**
   * The processed output is saved to the **same path** as the input file by default.

## Maustats Chatbot

Access the project here:  
🔗 <https://www.partnershipsforthegoals.org/maustats/chatbot>

### Overview

The Maustats Chatbot is designed to understand user queries and provide relevant answers using two core models:

* A spaCy model for natural language understanding.
* A TF-IDF vectorization model for semantic matching.

Each module within the chatbot is supported by its own trained model and a corresponding JSON file.

### Modules Supported

1. SBR – Establishment, Enterprise
2. Trade
3. Demography – Live Birth, Death, Still Birth, Marriage
4. Labour

The JSON files contain example questions and associated tags used to train the models for each specific module.

### Usage Requirements

* To use the chatbot, users must log in using their Maustats website credentials.
* Authentication is handled through the ITM Cloud, where both the chatbot and the main Maustats site are hosted.

### How the Chatbot Works

* Users do not need to select a specific module before asking a question.
* Instead, the chatbot follows this two-step process:
  1. The first model determines the intent of the question and identifies the relevant module.
  2. The second model provides the most appropriate response based on the identified module and question context.

## PLAYSA Documentation

### Project Setup Steps

#### 1. Install Python 3.7

Ensure your system is updated and install Python 3.7:

* Update the system:
  + sudo apt update
* Install Python:
  + sudo apt install python3.7
* Confirm installation:
  + python3.7 --version

#### 2. Create a Virtual Environment

* Create the environment:
  + virtualenv -p python3.7 ve
* Verify it exists:
  + ls ve
* Activate it:
  + source ve/bin/activate
* A successful activation will show (ve) at the beginning of your terminal prompt.

#### 3. Install Python Dependencies

* Install project requirements:
  + pip install -r requirements/requirements.txt

#### 4. Install Node.js (v18.16.1)

* Install using nvm:
  + nvm install 18.16.1

#### 5. Install Node Dependencies and Build Assets

* Install dependencies:
  + yarn install
* Build assets:
  + yarn run build

#### 6. Setup PostgreSQL Database

* Ensure PostgreSQL version 16 (and pg\_restore v16) is installed.
* Restore the database:
* pg\_restore -v -c -O -d postgresql://postgres:itm2016@127.0.0.1:5432/powerofplay < powerofplay-prod.pg\_dump

#### 7. Finalize Django Setup

* Run migrations:
  + python manage.py migrate
* Create a superuser:
  + python manage.py createsuperuser
* Start the development server:
  + python manage.py runserver

#### 8. S3 Storage Dependency

* Install required package:
  + pip install django-storages==1.14.3 --no-deps

### Admin Panel Access

* URL: [play.org/admin](http://play.org/admin)
* Username: administrator
* Password: @ccess@2024

## Monthly Report Generation

### Steps

1. Access the Database Server:
   * sudo -u postgres psql
   * \c playsa
2. Run the Queries Below and fill the values in the provided Excel sheet.

### **GRADE R** Query

SELECT

COUNT(reg.id) AS registration\_count,

SUM(CASE WHEN el.completed\_at IS NULL THEN 0 ELSE 1 END) AS completion\_count,

reg.province

FROM

registration\_powerofplayuser reg

JOIN

elearning\_userprogress el ON reg.id = el.user\_id

WHERE

(reg.created BETWEEN '2025-04-01' AND '2025-04-27') AND el.course\_id = 3

GROUP BY

reg.province;

### ECD Query

SELECT

COUNT(reg.id) AS registration\_count,

SUM(CASE WHEN el.completed\_at IS NULL THEN 0 ELSE 1 END) AS completion\_count,

reg.province

FROM

registration\_powerofplayuser reg

JOIN

elearning\_userprogress el ON reg.id = el.user\_id

WHERE

(reg.created BETWEEN '2025-04-01' AND '2025-04-27') AND el.course\_id IN (1, 2)

GROUP BY

reg.province;

### Grade 1 to 3 Query

SELECT

COUNT(reg.id) AS registration\_count,

SUM(CASE WHEN el.completed\_at IS NULL THEN 0 ELSE 1 END) AS completion\_count,

reg.province

FROM

registration\_powerofplayuser reg

JOIN

elearning\_userprogress el ON reg.id = el.user\_id

WHERE

(reg.created BETWEEN '2025-04-01' AND '2025-04-27') AND el.course\_id = 4

GROUP BY

reg.province;

Note: To confirm course IDs, refer to the Course module under the Data Manager/Administrator section.

### Modules to Include in the Monthly Report

* ECD
* Grade R
* Grade 1 to 3

### Verifying Email Service After Code Changes

After making any code changes and accessing the server, it is important to ensure that the email service is functioning correctly.

To do this:

* Start the Celery worker process for the project.

Command -> celery worker --app=project --loglevel=info --concurrency=1 --max-tasks-per-child=1000 &

The process runs with the following settings:

* + Application: project
  + Log level: info (to monitor detailed logs)
  + Concurrency: 1 (processes one task at a time)
  + Max tasks per child: 1000 (ensures the worker is restarted after handling 1000 tasks to prevent memory leaks)

Once the Celery worker is running, the system will be able to send emails as intended via background processing.

This was for the server if you want to run celery for local then open virtual environment in another terminal and run

command -> celery -A project worker -l info