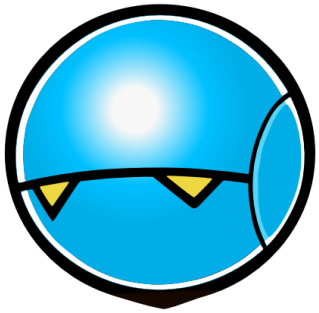


Marvin

automated Data Gov.

Marvim Quick User's guide v1.01

Marvim is the Data Governance & Data Lineage solution integrated into:

TIMi

CREATIVITY THROUGH EFFICIENCY

TIMi Company headquarters:

Address: Chemin des deux Villers, 11 - 7812 Ath (V.N.D.) - Belgium

Phone (global): +32 479 99 27 68

Phone (Americas): + 57 300 675 13 69

Creation date: September 2026

Last Edited: September 2026

Last Edited by Frank Vanden Berghen

1. Introduction

Welcome to the Quick User's Guide to Marvim.

Marvim is Data Governance, Data Lineage & Documentation Solution.

Marvim is an Acronym that stands for: **M**aster, **A**utomated, **R**epository for **V**isual **I**ntelligent **M**etadata management

Marvim's main characteristics are :

- **Automated** : When Marvim is used in combination with Anatella/TIMi, Marvim becomes Self-documenting. The content of the documentation stored in Marvim is filled-in automatically by Anatella/TIMi
- **Agnostic** : Marvim is built to be used in combination with Data Management tools. Obiously, if you use Anatella, everything works directly "out-of-the-box".
- **Standalone & Open** : Marvim only uses the standard open stack: Apache+PHP+Sqlite (that is free & open source): i.e. Marvim does not rely on any third-party component or technology. Also, there are no burdensome commercial license attached to Marvim. **Marvim is 100% free to use**.

Marvim is standalone. However, when Marvim is used in combination with Anatella/TIMi, Marvim becomes Self-documenting (this is a totally optional combination): i.e. No need any more to spend days (or even months) encoding stuffs inside your data governance tool: It's now done for you, automatically. The content of the documentation stored in Marvim is filled-in automatically by Anatella/TIMi and always match closely with the "real thing" that is really going on with your data (no "unsynchronized" documentation anymore!).

A bit of History

Marvim is integrated into the data platform named "TIMi". "TIMi" is an acronym that stands for "The Intelligent Mining Machine (TIMi)" . TIMi is a 100% Sovereign solution. TIMi is created in Europe and is 100% GDPR compliant. TIMi was originally created in 2007 to solve the most complex Machine Learning problems (using the first tool that we created at the time: an Auto-ML engine named "Modeler"). In 2010, we added into TIMi a new tool named "Anatella". Anatella is a 100% self-service (no-code), high performance "data preparation"/ETL and "big data" solution. In 2026, we created Marvim and integrated it into Anatella.

Why is "Automation" important in the context of a data governance tool?

Most Data Governance tools are very time-consuming to setup. Indeed, it's common to spend several months, even sometimes years, to encode everything inside the Data Governance tool and thereafter to properly document all the data sources, all the data flows and all the reports that are used inside your company. …And when it's finished, you must (almost) restart from scratch again because the documented processes and the documented datasets have evolved further away. Encoding documentation like that is a very boring and repetitive task. Not only is this very time consuming (and thus very expensive), it's also very misleading for the users of your Data Governance solution. Indeed, since it takes an enormous time to document everything, it means that the documentation is always

be "*lagging behind*" the true processes or datasets that are really used in production. This "lag" can range from a few weeks to a few years and it quickly becomes the source of a lot of frustration and the source of a complete distrust in the content of the Data Governance tool. *..And a tool that is not trusted is a useless tool.* The solution to this problem is, obviously, automation. We created Marvim so that, in a few mouse clicks, we can synchronize, everything that is happening "in production" (i.e. what's happening inside Anatella) and what is stored in Marvim. No more delays. No more distrust. Reliable (meta-)data that you can explore using the dedicated user-friendly Marvim interface.

Why "Standalone & Open" is important?

We built Marvim so that it can be "plugged-in" directly inside any data management/data storage tool, including the ones that you are already using now (even if it's not yet Anatella! 😉). ..And if something is not exactly the way that you want it, you can always modify Marvim to make it better suited to your particular situation, since Marvim is 100% open source. We know that all companies work in a slightly different way and it seemed important to us to give you the possibility to easily tailor Marvim to your specific needs. Marvim's source code is hand-crafted by a human and easily readable & extendable by a human (we only used AI to create the CSS files).

And, as usual with *all* TIMi tools, Marvim is a tool that is **100% sovereign**. Use Marvim "On premise" for maximum security or use Marvim inside any cloud (including European clouds such as Hetzner, Scaleway, etc.) for flexibility. Regain control of your hardware infrastructure. Avoid the enormous & uncontrollable costs of the clouds. Get fixed, reasonable and controllable fees. Instead of handing over the "keys" of your wallet to the cloud providers, get leverage to regain your autonomy/sovereignty.

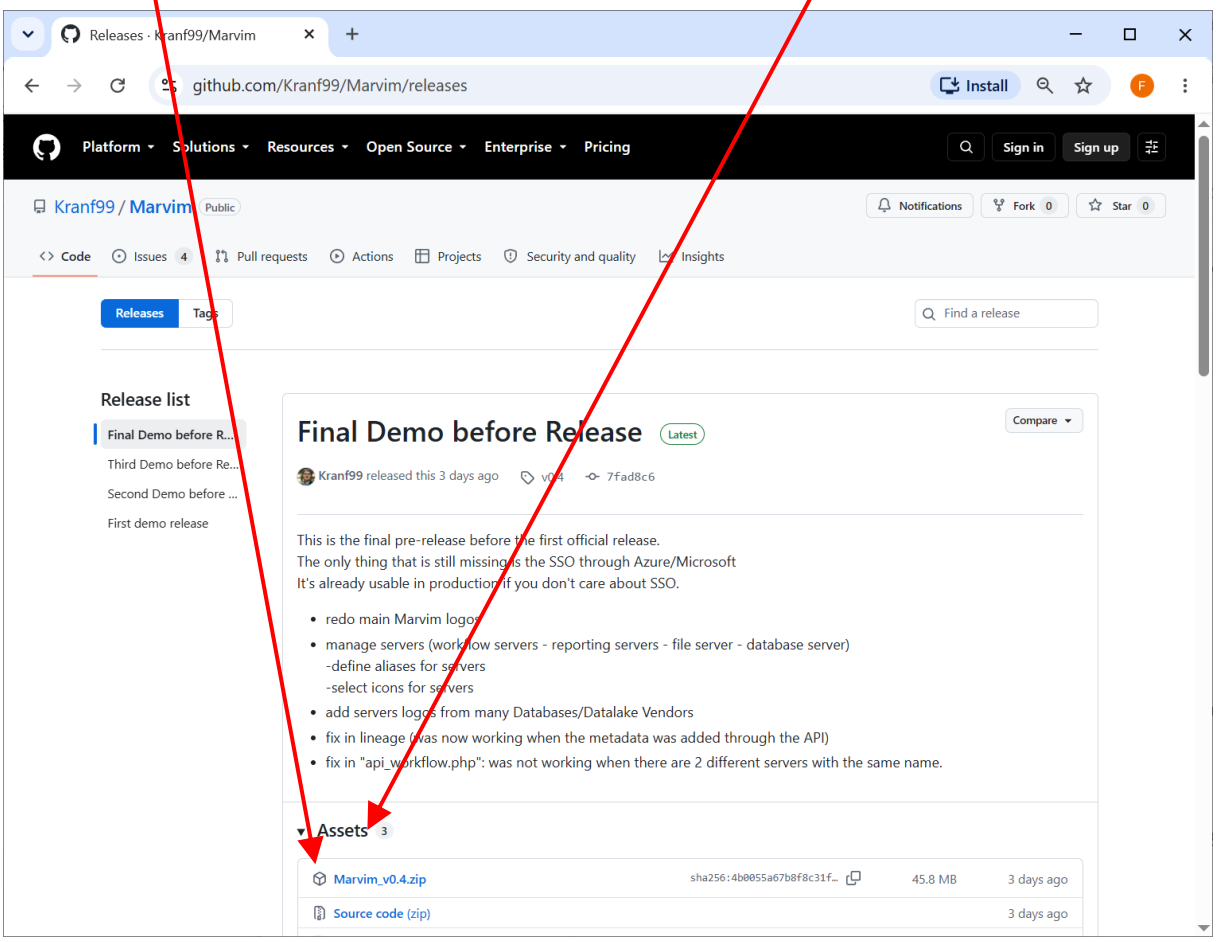
Marvim is an alternative to: Data Galaxy, Collibra, etc.

2. Marvim Installation

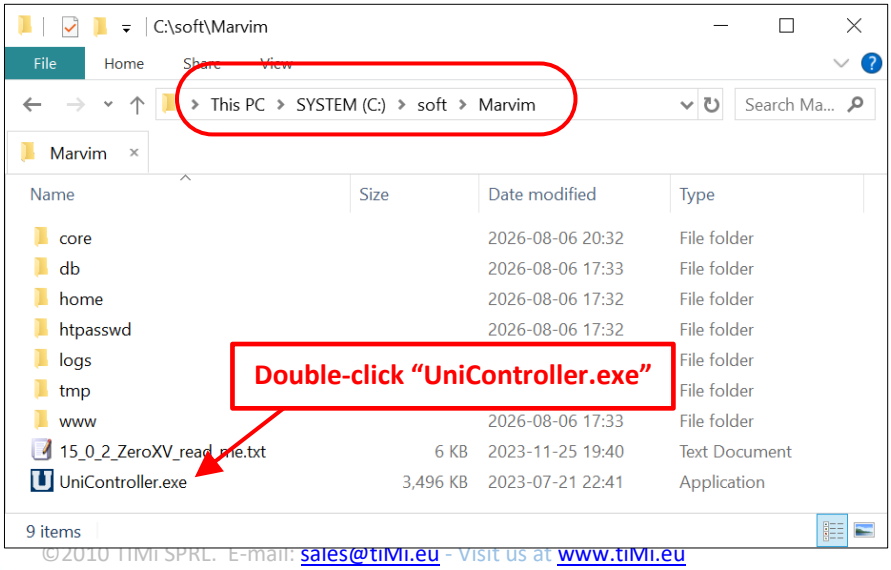
Since Marvim only uses the standard classical open stack “Apache+PHP+Sqlite” (that is free & open source), it can be installed in any server or OS where this stack is available. i.e. Marvim can be installed in: any Linux box, any Windows box. Marvim can be installed almost everywhere since you don’t need any administrative rights to install & run it.

The minimum Marvim installation procedure for Windows is:

1. Go to https://github.com/Kranf99/Marvim/releases , click on “Assets”, and download the first zip file:

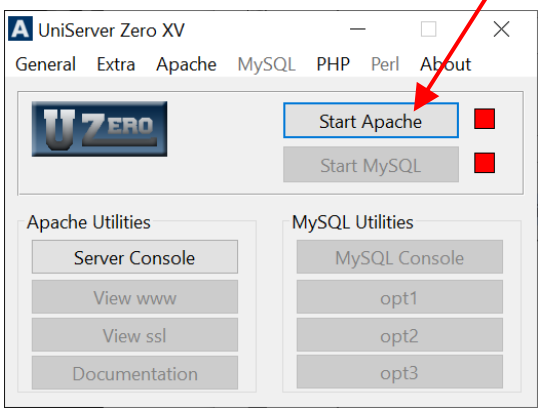


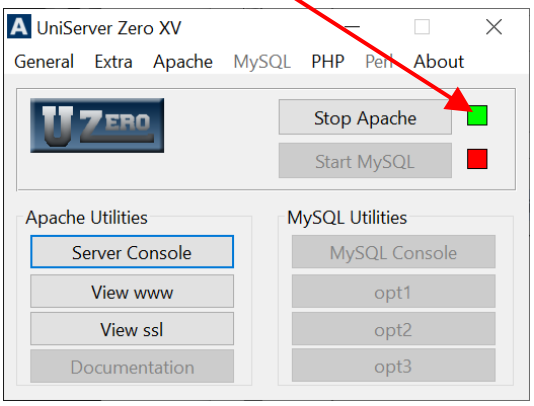
2. Unzip the downloaded file (typically inside “c:\soft\Marvim”). You should now see:



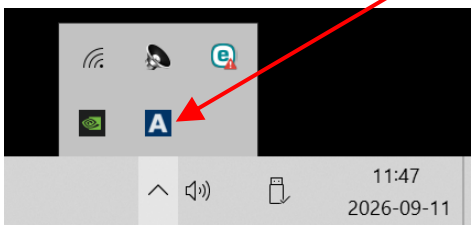
3. Double-Click on the “UniController.exe” executable (inside the Marvim directory) to run it.
4. Click on the “Start Apache” button.

The red light turns green and your browser opens.

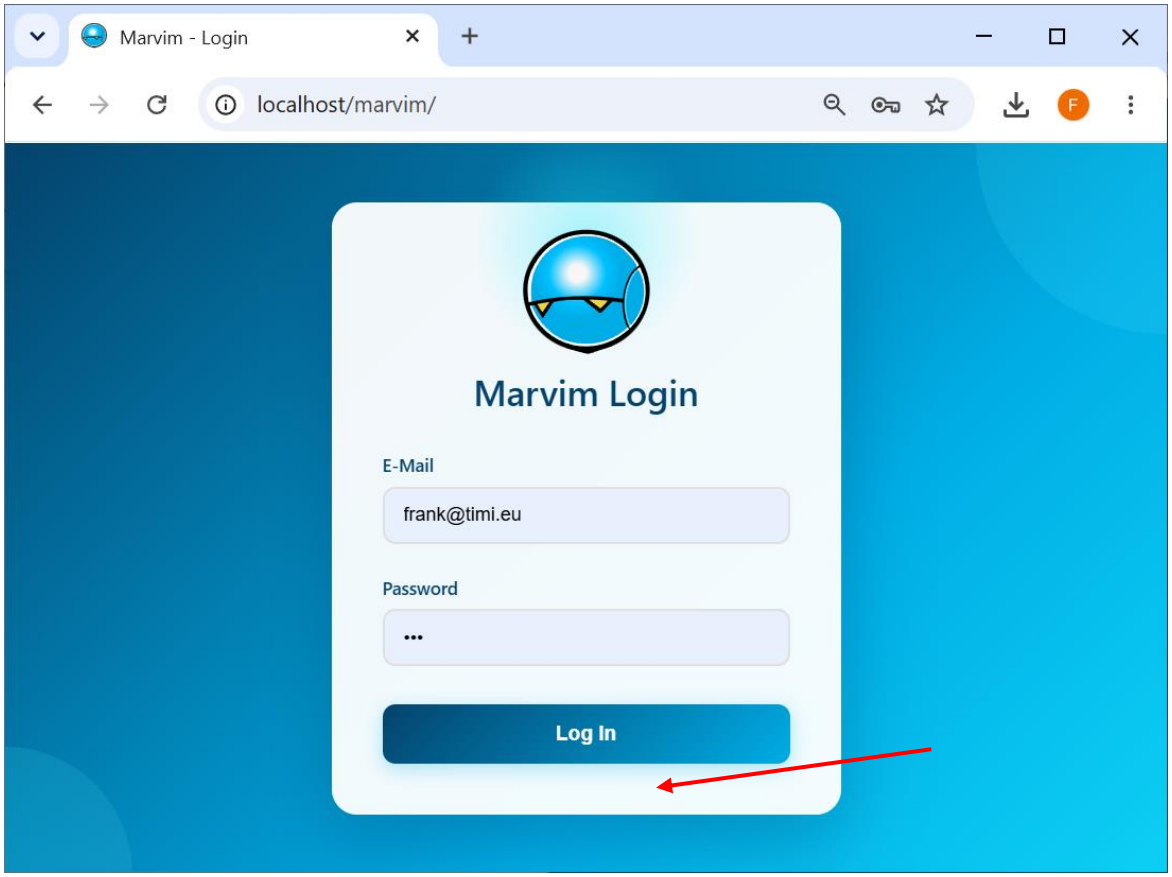




5. You can now reduce the window “UniServer Zero XV”.
   If you need it back, this window now lives inside bottom-right corner of your screen here:



6. Inside your browser, log into Marvim. The default credentials (at the first run) are:
   Login: frank@timi.eu
   Password: 123



3. Marvim Quick Guide

3.1. Intro

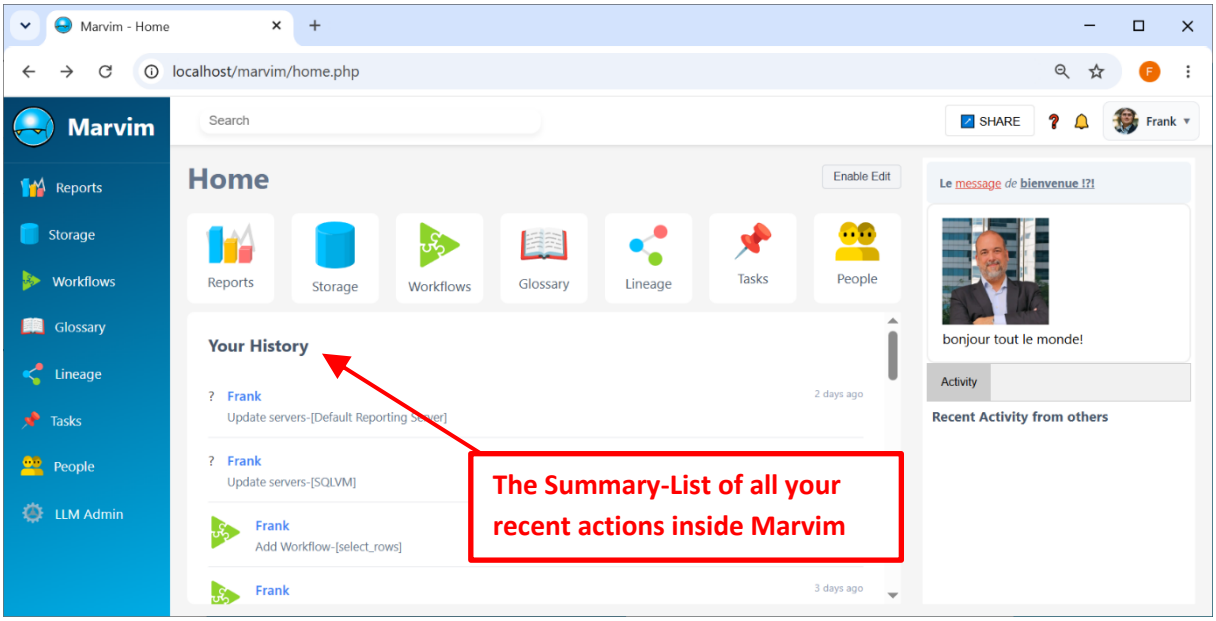
From the Marvim home page, you have access to 9 different sections:

1. **Reports**
   This section documents all the reports of the company (in PowerBI, Qlick, Tableau, Etc.)
2. **Storage**
   This section documents all the datasets accessible to the company
3. **Workflows**
   This section documents the data flows from one storage to another to finally end-up into a report, a predictive model or an external server
4. **Glossary**
   All the documentation texts from the Report, Storage and Workflow sections contain a very special vocabulary. The technical words are explained & defined in the Glossary.
5. **Lineage**
   Allows you to have access the large dependency graph where you can do “Impact analysis” (what happens at the end of the workflows if I change this source) or “Lineage Analysis” (what are the raw data sources that allowed you to compute this result at the end of the workflows?)
6. **Tasks**
   A list of tasks assigned to the different Marvim users. One important set of tasks is the “Validation Tasks” where the owner of a data-asset X can accept (or reject) the modifications on the documentation of the data-asset X that were proposed by another Marvim user.
7. **People**
   The section for the management of Marvim users. The Marvim users are classified in 5 groups:
   - Super-user
   - Admin
   - Creators
   - Documentors
   - Viewers
8. **LLM Admin**
   You can optionally use an LLM to help you write Marvim documentation. This is where you setup the access to the LLM.
9. **Search**
   To search globally across all sections: reports, storage, workflows, glossary, columns, kpis. Press [Enter] in the search box to open the dedicated “search” section.

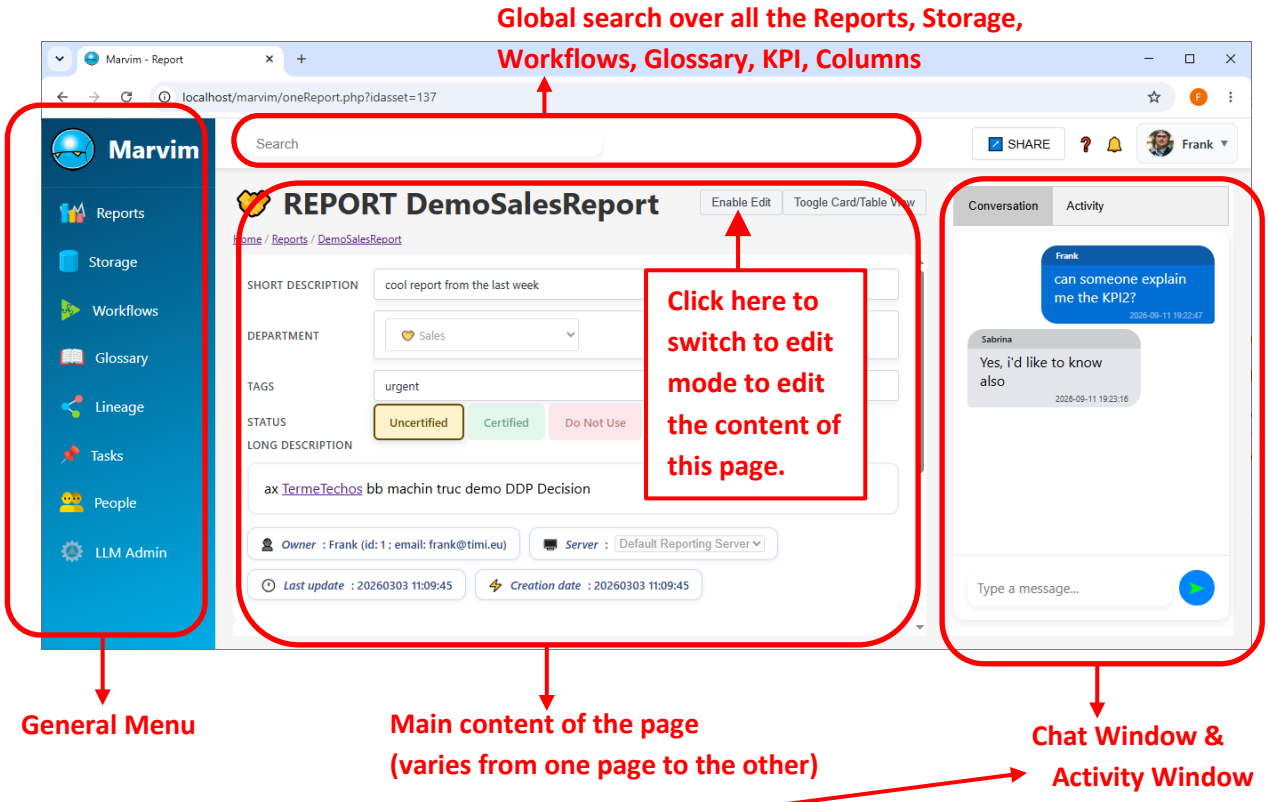
There is one hidden additional section:

10. **Servers**
    In this section, the Marvim users manage all the servers from the company (and the servers from the providers and the partners), how they appear in Marvim (their logo) and their (multiple) aliases.

Here is a screenshot of the Marvim home page:

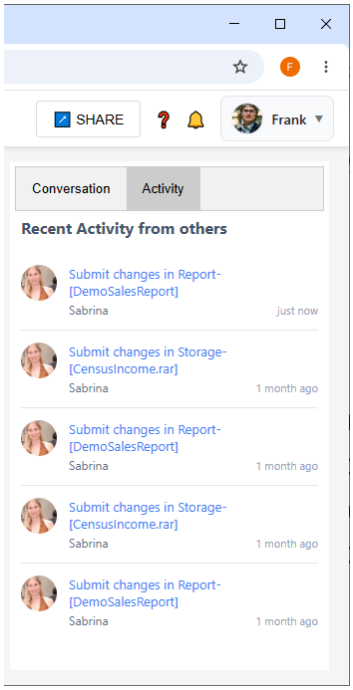


Inside Marvin, most pages follow the same structure:

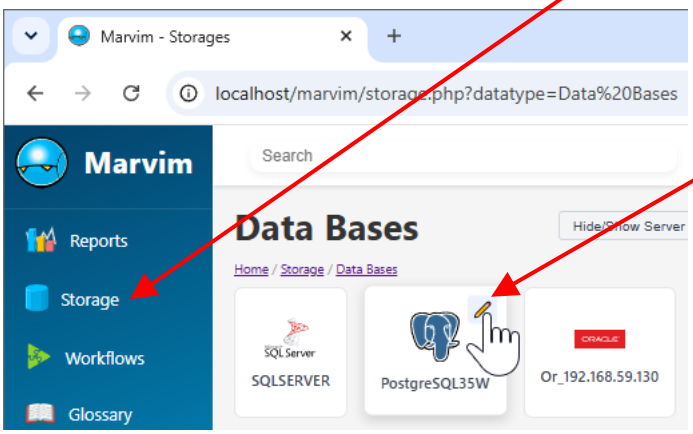


This "Chat Window" is always "**contextualized**": it's the conversation that is linked to the current asset (the current report, dataset table or workflow). This allows to automatically "attach" the knowledge contained inside the chat to a specific asset. Thanks to the contextualized chat, no more question such as: "*I remember that zzz gave me the info that I need about this asset but I don't know where I got this info: Was it Whatsapp, Signal, Teams ??*"

The Activity Window gives a summary list of the recent activities of all the **other** Marvim users (this is the same list as the one visible on the Home page, but for **other** users, rather than for you). For example:

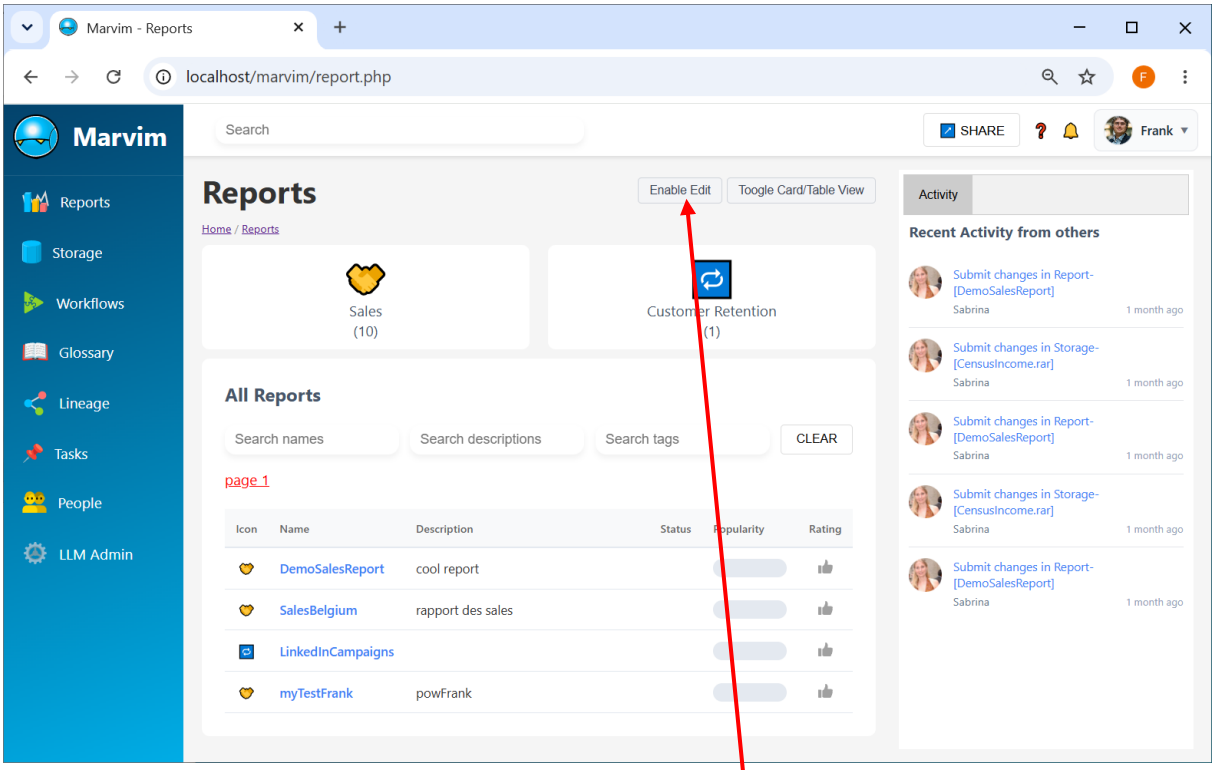


To access the "Server" sections: Click on the "Storage" section and then click on the pencil here: (you must have super-admin rights)

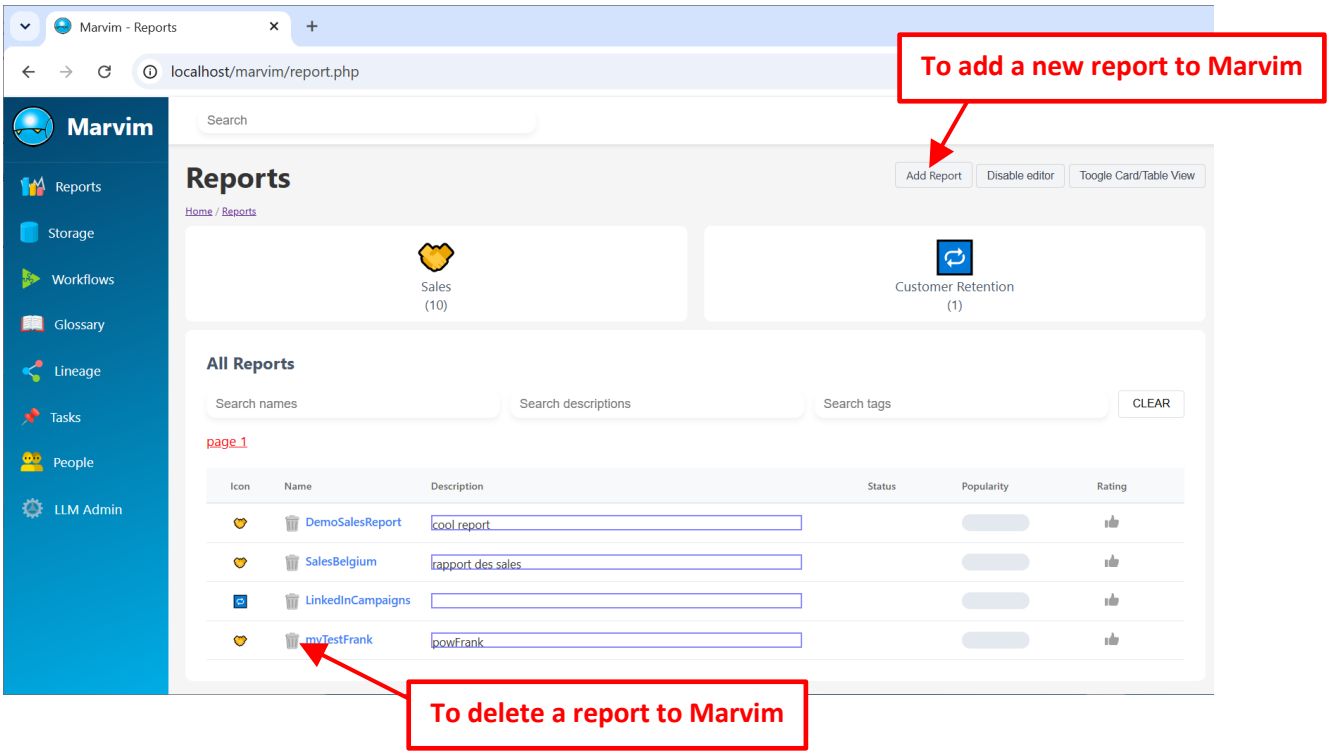


3.2. Reports

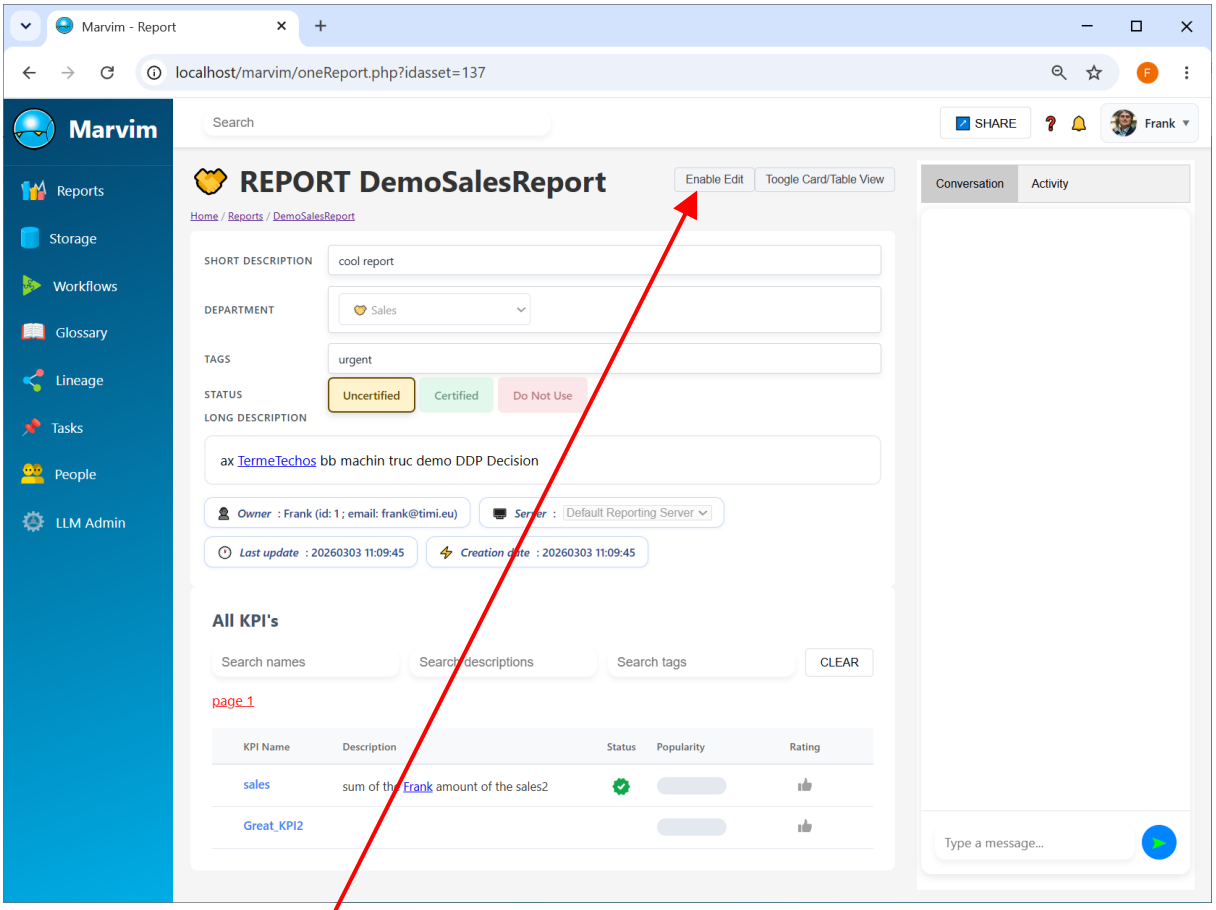
All changes made in the text descriptions of the Reports are submitted to the approval procedure described in section 4.3.



If you want to add or delete a report, click on “Enable Edit”:

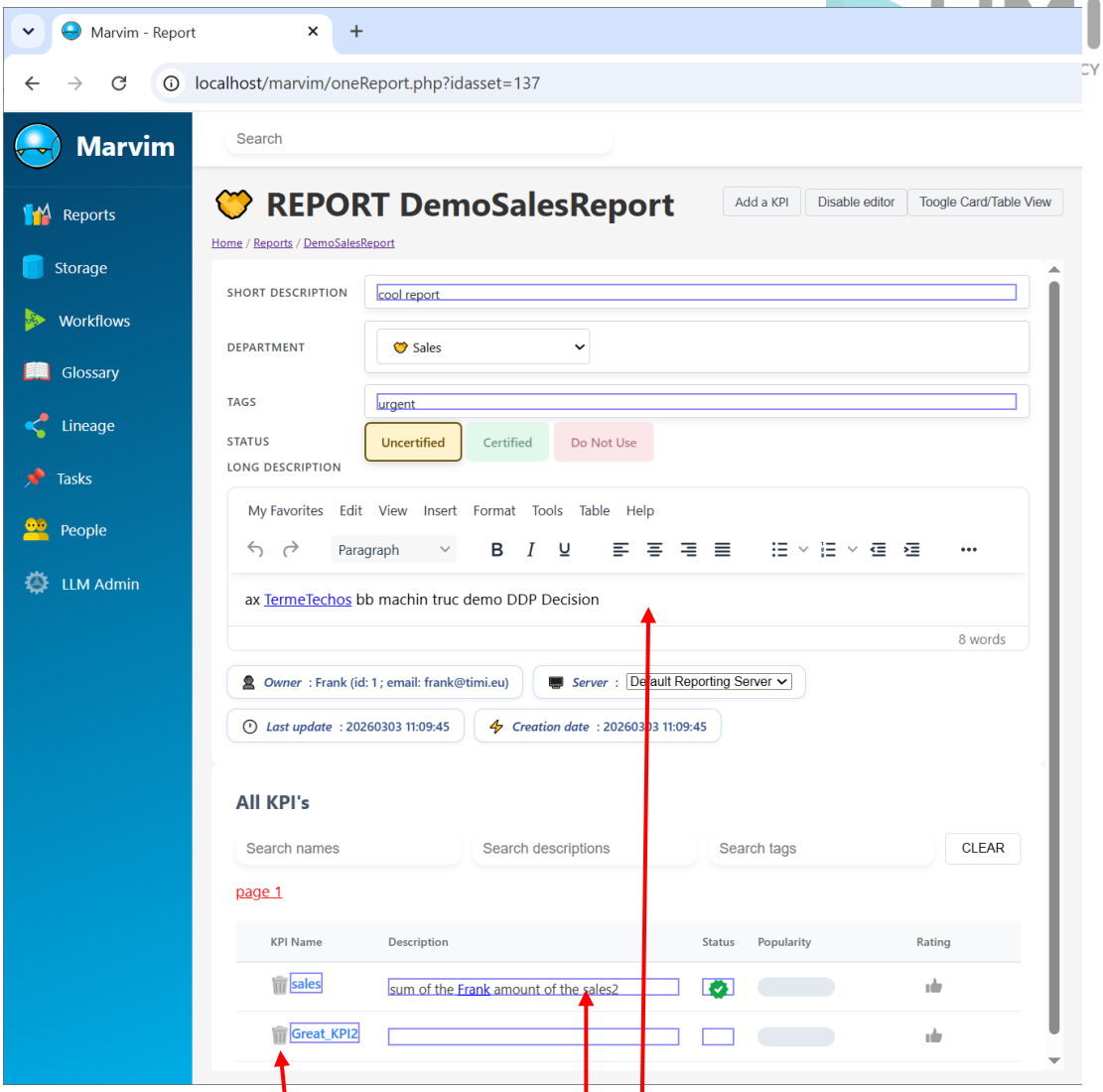


Click on the Report's Name (in blue) to open a report: For example:



Click on the "Enable Edit" , to edit this page (if you have at least the "Documentor" rights).

You now see:



You can edit all the texts that are inside a blue rectangle ( ):

Click the icon to delete an asset

The main documentation section opens-up here:

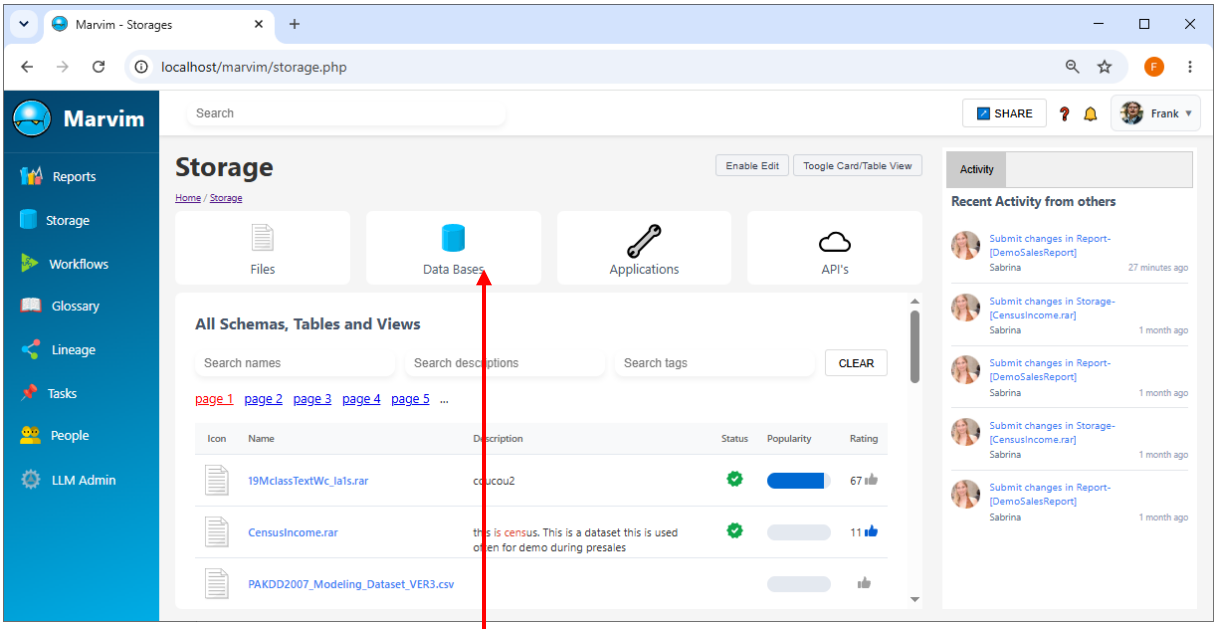
If you are a super-user, you can edit the fields directly, without requiring the approval of the owner of this webpage (about the report “DemoSalesReport”). Otherwise, you’ll need to follow the normal approval procedure to get your “changes” approved/rejected: see section 4.3. for more details on this procedure.

3.3. Storage

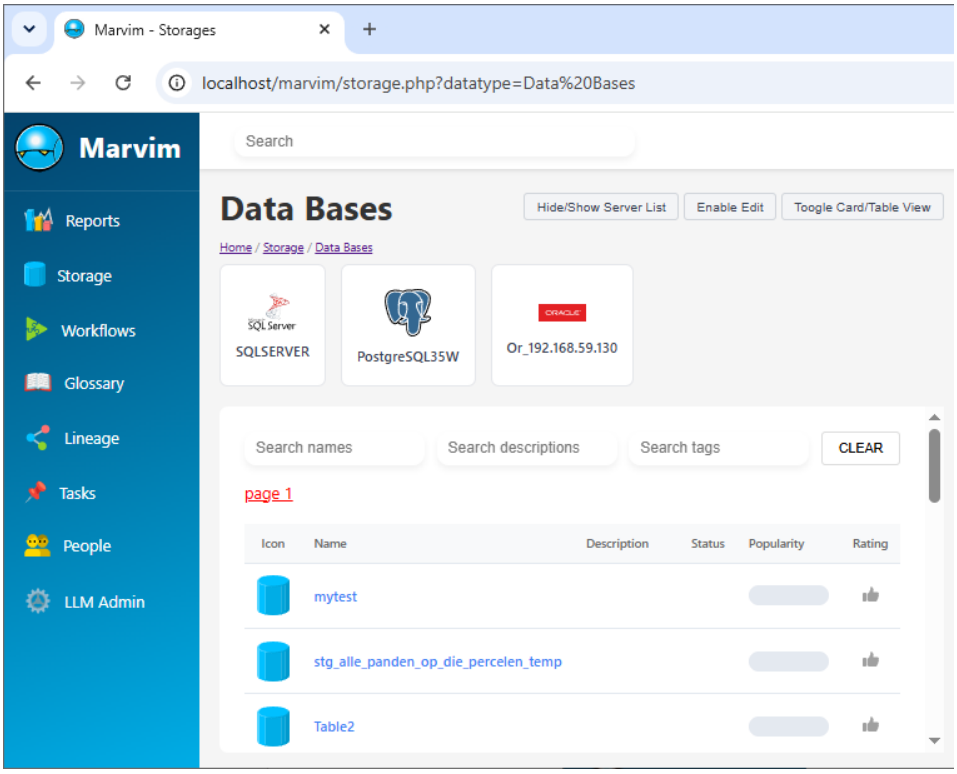
All changes made in the text descriptions of the Storage are submitted to the approval procedure described in section 4.3.

See the section 4.5. to know how to use a AI/LLM to automatically create the text description of your tables/datasets.

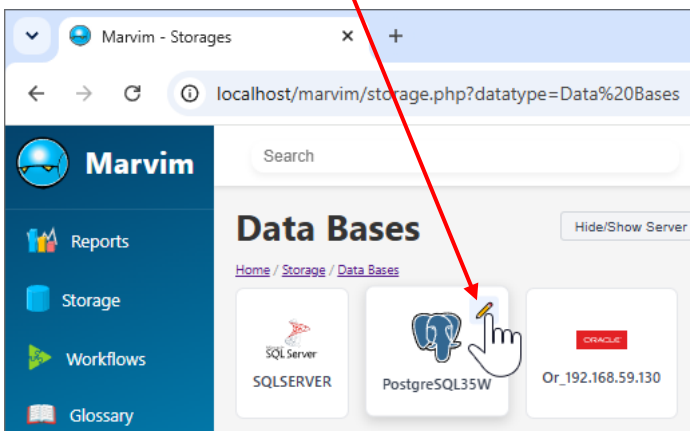
The first screen in the “storage” section lists all the storages (that your user is allowed to see):



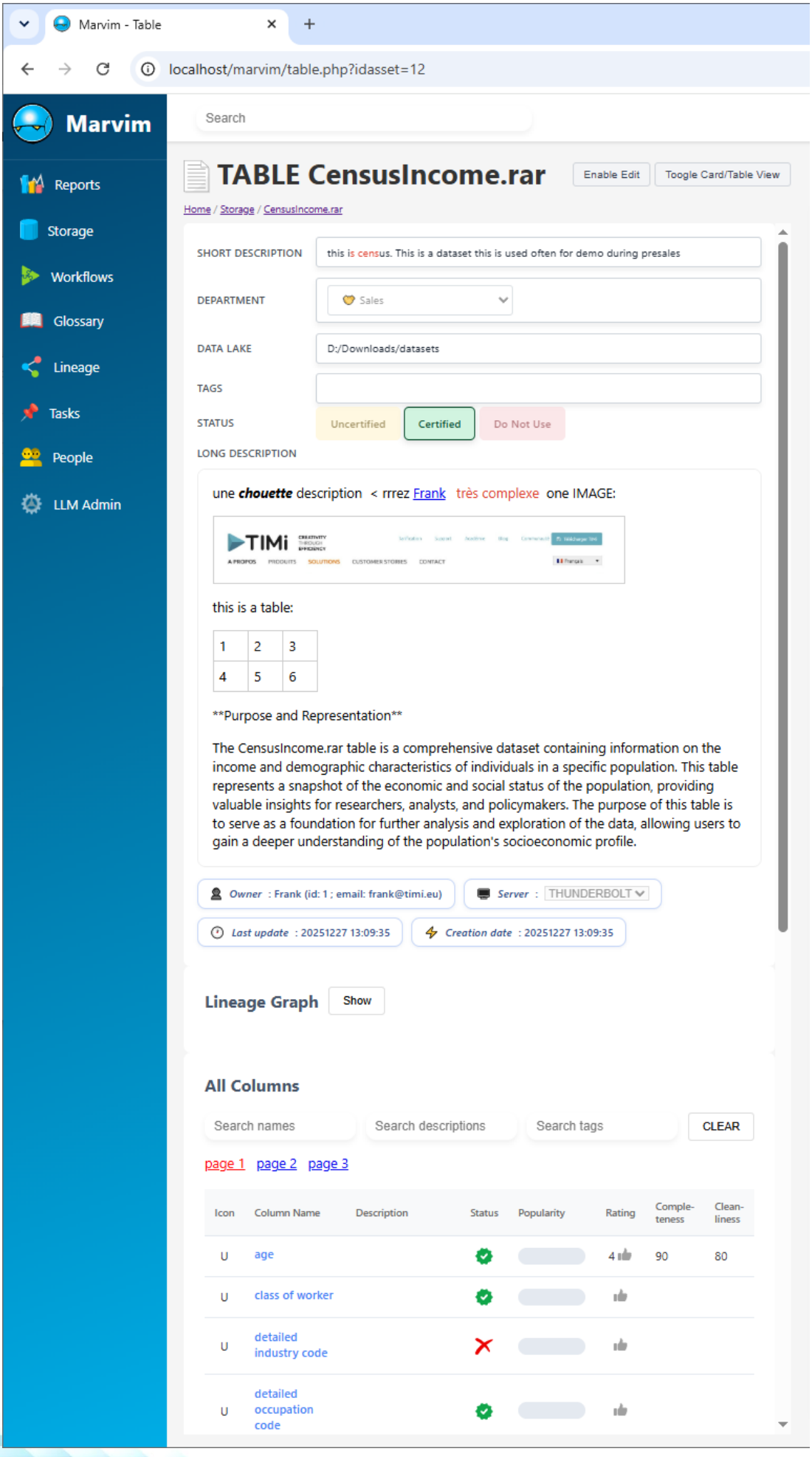
For example, let’s click on “Data Bases” We get:



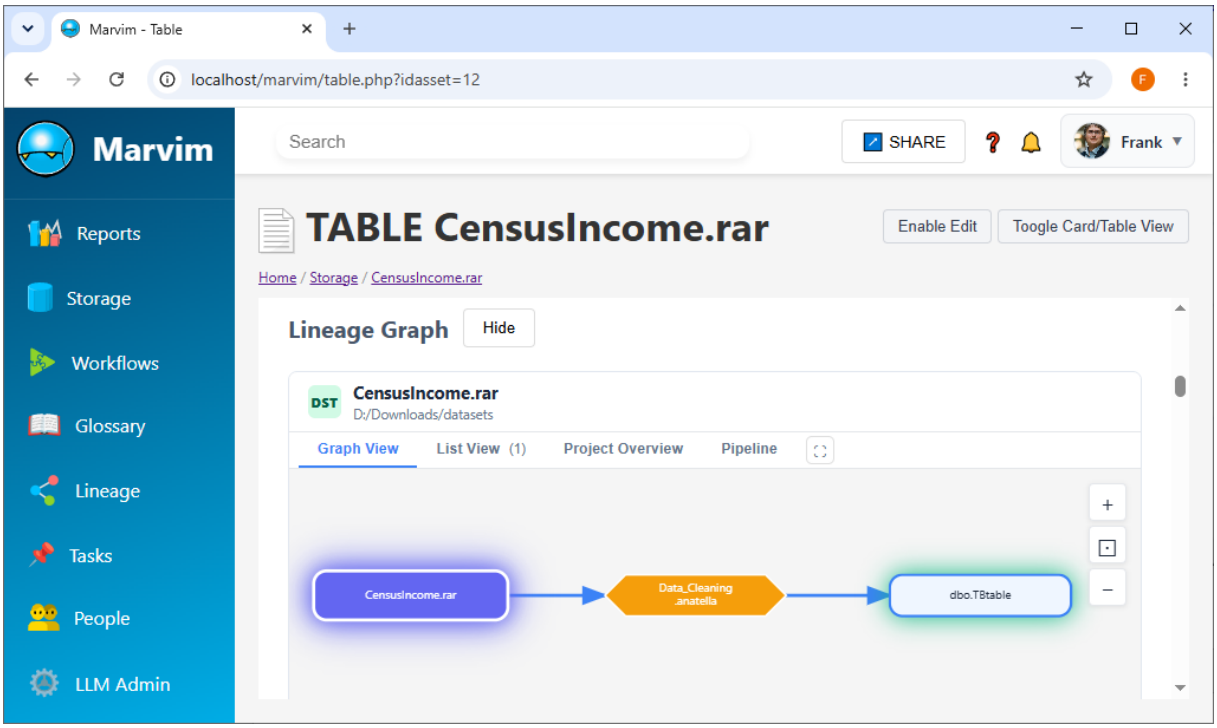
To access the screen that allows to manage the servers, click here:



Click on a Storage's Name (in blue) to open a storage: For example:



If you click on the “Show” button next to the “Lineage Graph” title, a local Lineage display appears:



When you open the detail page for a specific table inside the “Storage” section, you get plenty of information about the selected table:

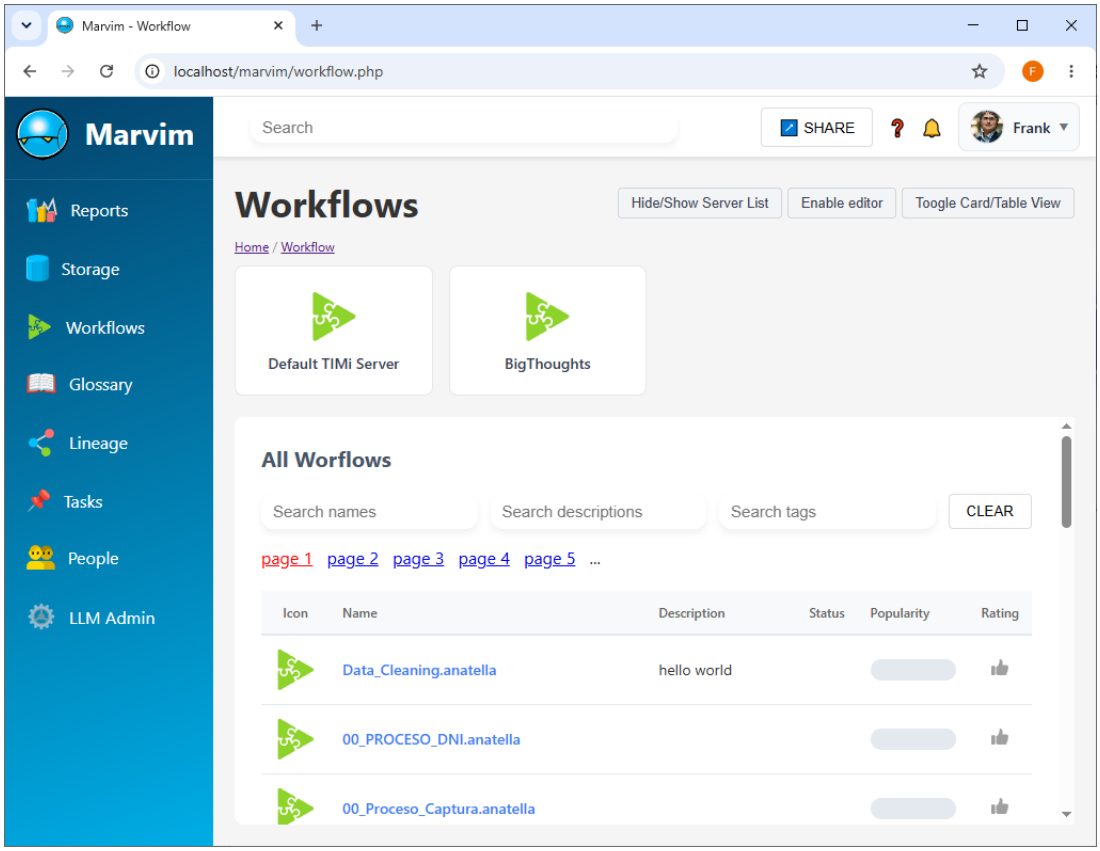
- The table name
- The table “Short Description” and “Long Description”
- The department that manages that table
- The name of the “Data Lake” to which this table belongs.
- The table status (uncertified, certified, do not use)
- The tags associated with the table
- The table owner (that approves all changes made on the table)
- The server that stores the table
- The last update time and the creation time of the table inside Marvim
- The lineage graph: a graph that illustrates where this table occurs inside a complex system that might involve thousands of workflows.
- The list of all the columns inside the table.
  For each of these columns, you get:
  - The column’s icon: U/K/F (represents the type of the column: Unknown, Key, Float): see the section 5.1.2. of the “AnatellaQuickGuide.pdf” for more details about these types.
  - The column’s Name
  - The column’s Short Description
  - The column’s status: undefined, ok, do not use
  - The column’s popularity: that’s a score based on the number of times this column is used inside a workflow, the number of “likes” that this column received and other indexes. This help new “hires”, fresh data scientists find the most relevant columns to use when investigating your databases.
  - The column’s rating: i.e. the number of “likes” that this column received.
  - The column’s completeness: a value of 90% means that 10% of the column is filled-in with NULL values

- The column's cleanliness: a value of 80% means that 20% of the column is either filled-in with NULL values or filled-in with non-sensical values (e.g. some examples of non-sensical values are: negative ages, phone numbers that are 100 characters long, a "sex" column filled with invalid characters such as "LGBTQIAAP2S+" rather than "Male"/"Female"/M/F, etc.)

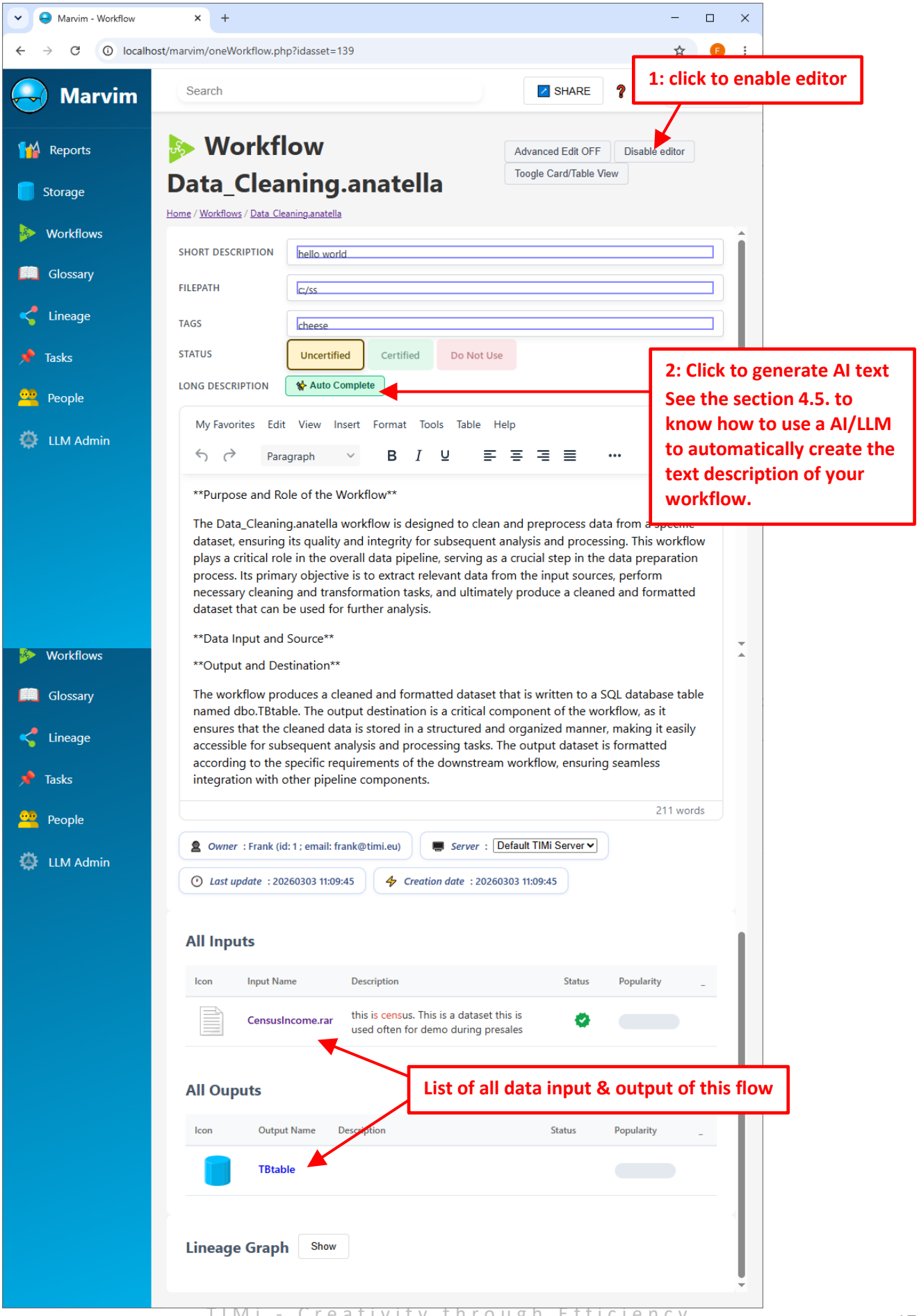
The column's completeness and the column's cleanliness are two KPI's that are typically updated through a little Anatella graph running at regular intervals (every day, every week, etc.). This Anatella graph can be very simple (simply running a SQL "count") or quite complex (when working on data tables that spans over thousands of files stored inside a complex distributed storage).

3.4. Workflows

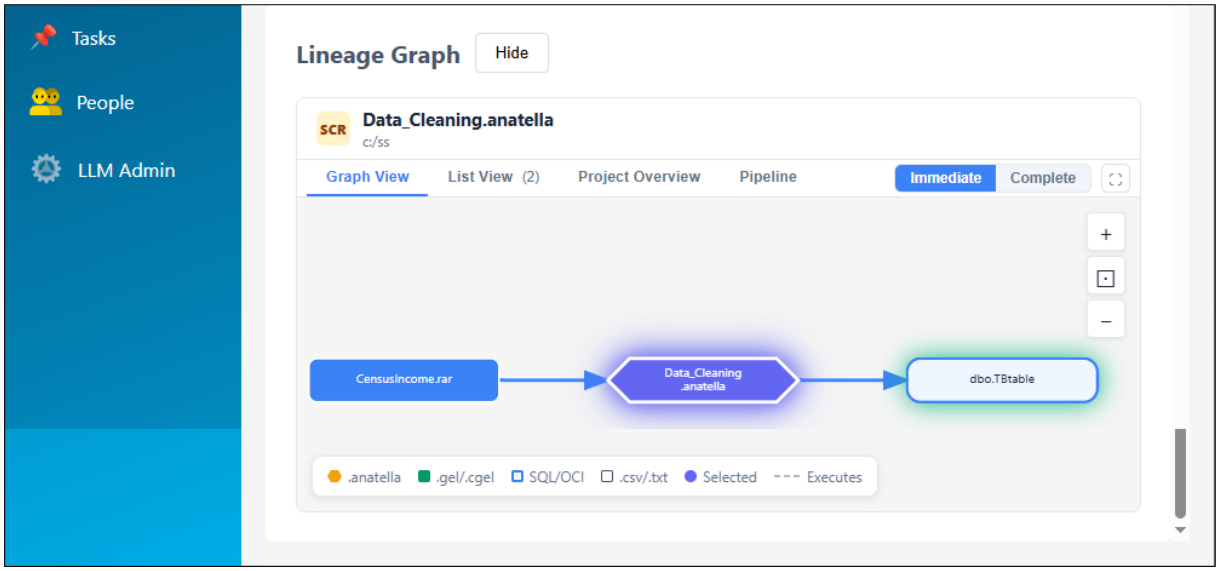
All changes made in the text descriptions of the workflows are submitted to the approval procedure described in section 4.3.



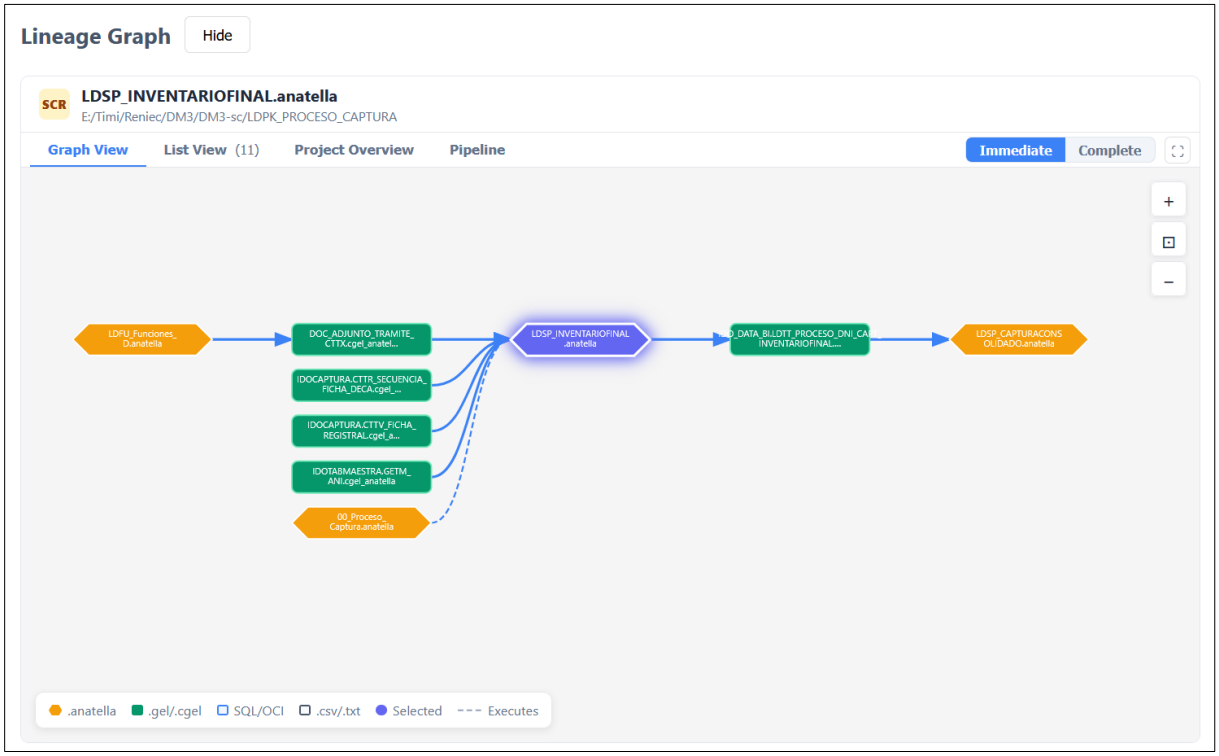
Click on a Workflow's Name (in blue) to open a Workflow: For example:



If you click on the “Show” button next to the “Lineage Graph” title, a local Lineage display appears:



..or:

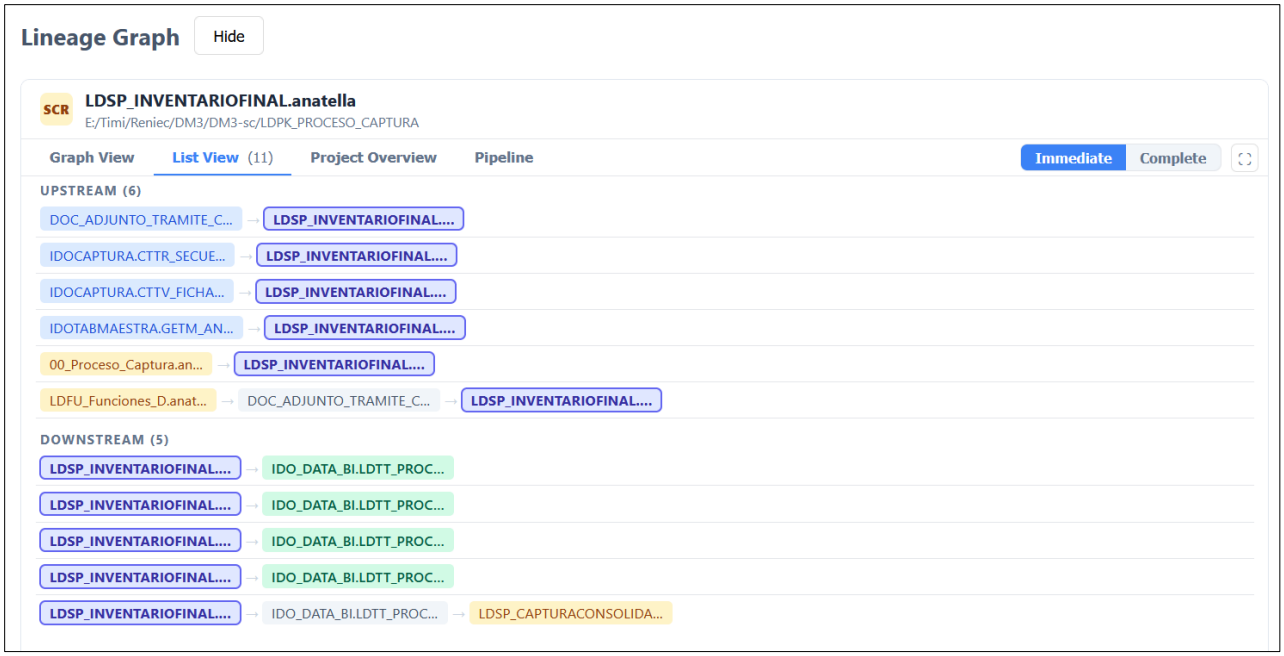


Hereabove, the yellow boxes are the Workflows (i.e. the .anatella files).
The blue box represents the currently selected Anatella graph.
The green boxes are the tables (i.e. the .gel_anatella files, the CSV files, the database tables, etc.)

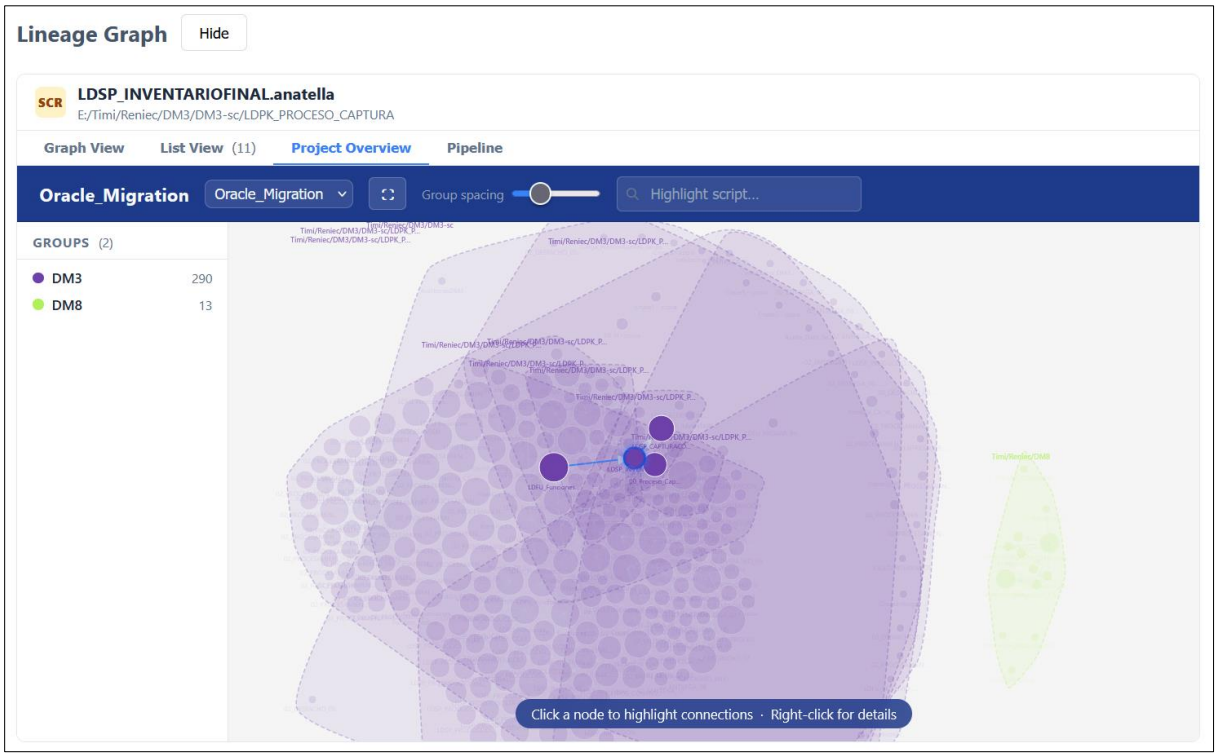
The lineage section has 4 tabs:

1. **Graph View** (that we just saw hereabove)

2. **List view**



3. **Global Project Overview**

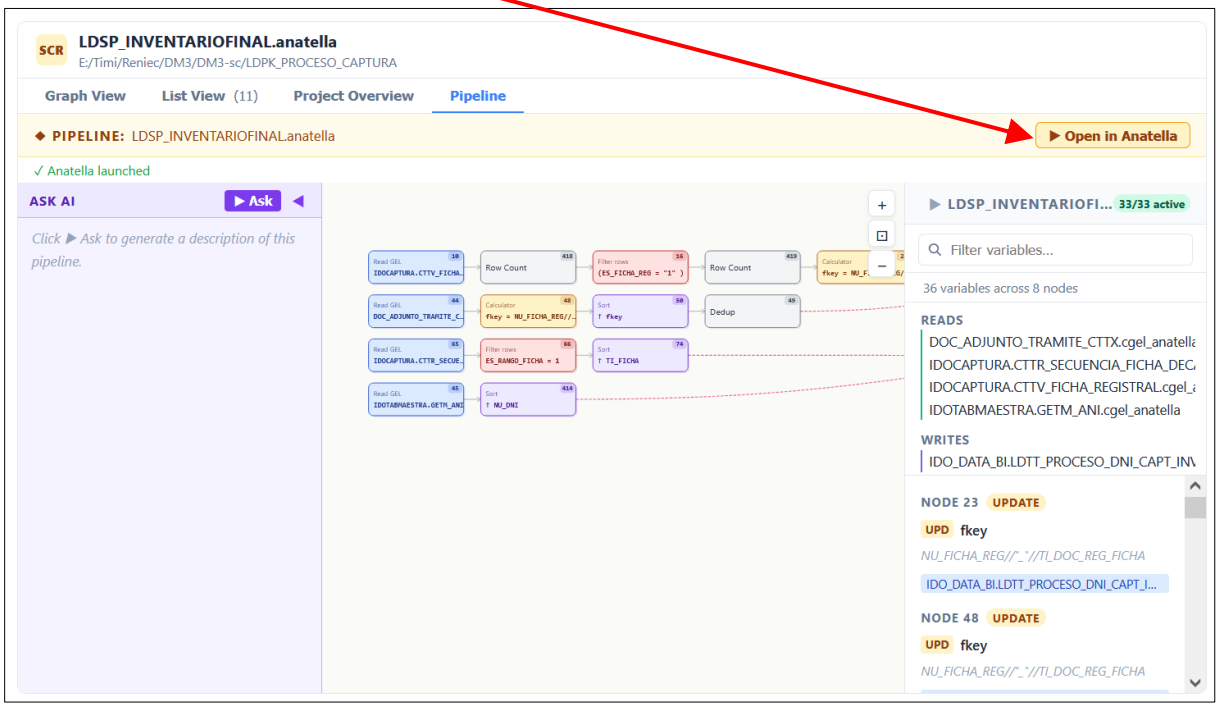


4. Pipeline

This last tab only works if you decided to save a copy of your .anatella file inside Marvim (this is optional).

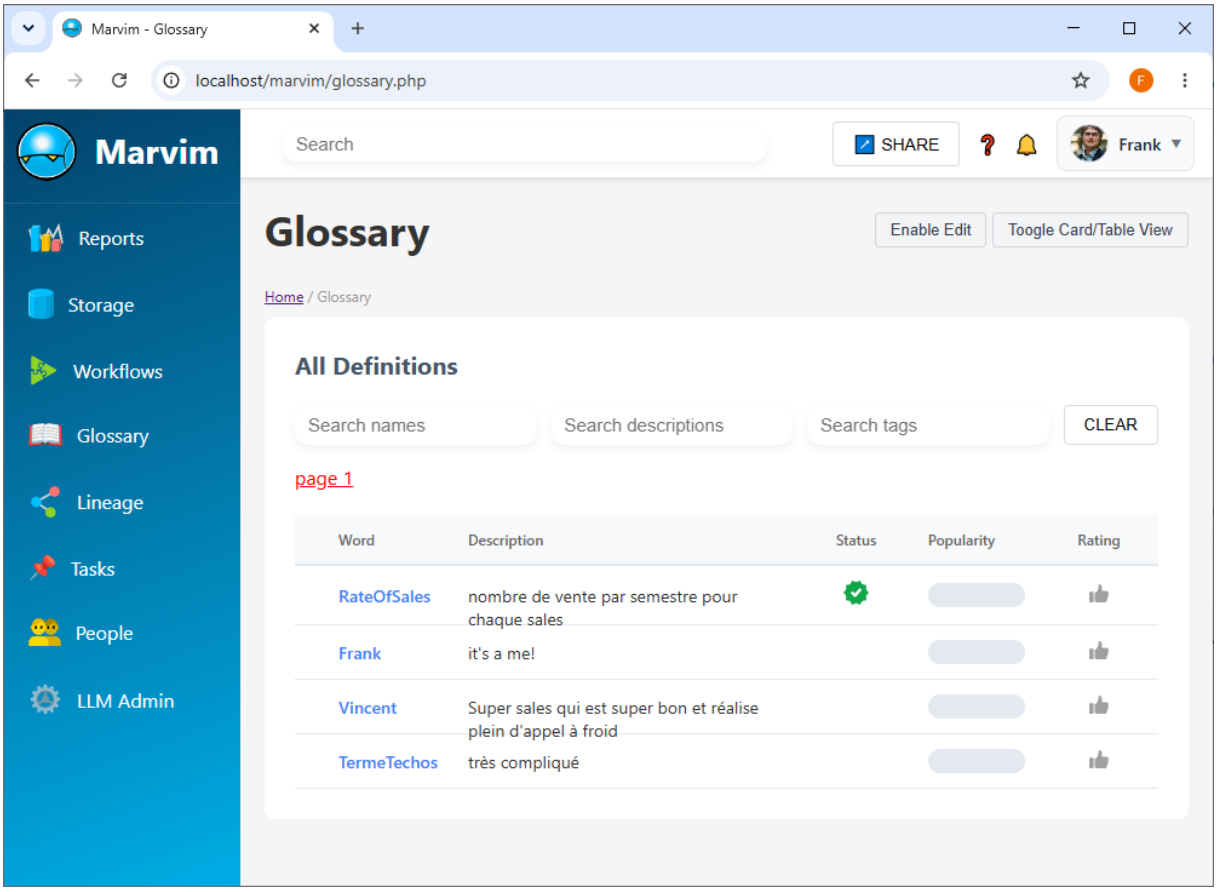
You can see here your Anatella graph directly rendered inside your browser.

You can also click here to open the graph inside Anatella.

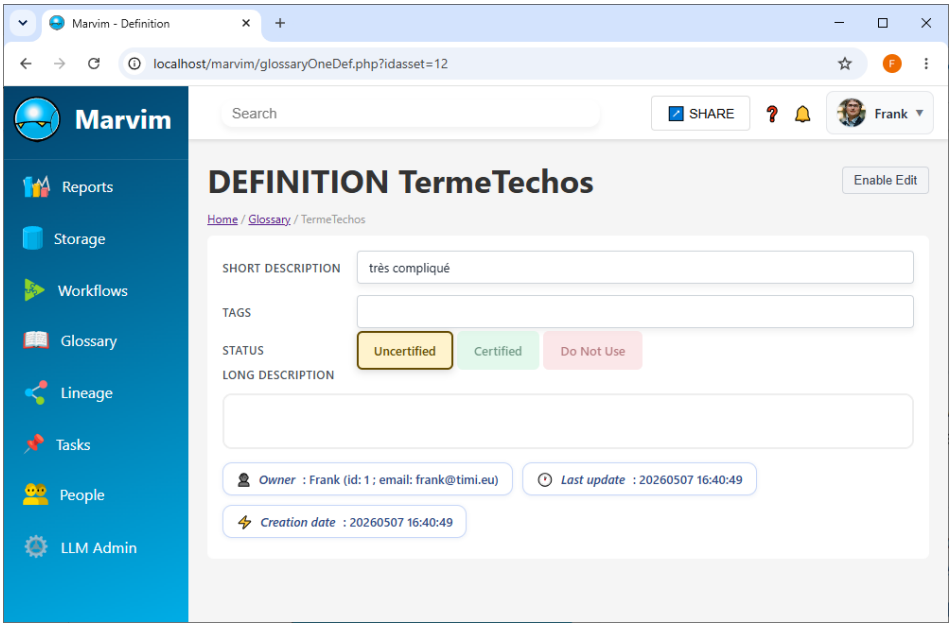


3.5. Glossary

All changes made in the text descriptions of the Glossary are submitted to the approval procedure described in section 4.3.

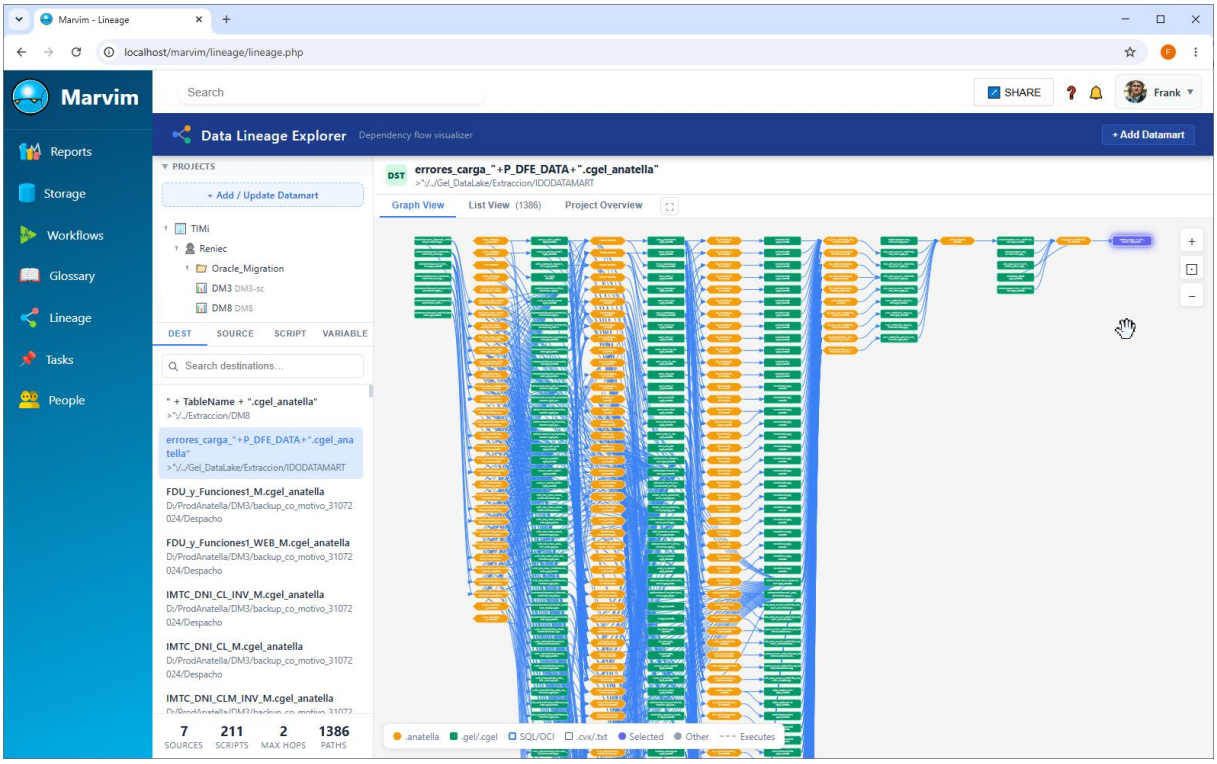


The words inside the glossary are automatically converted to URL link inside all the text descriptions visible inside the Reports, Storage and Workflows webpages.

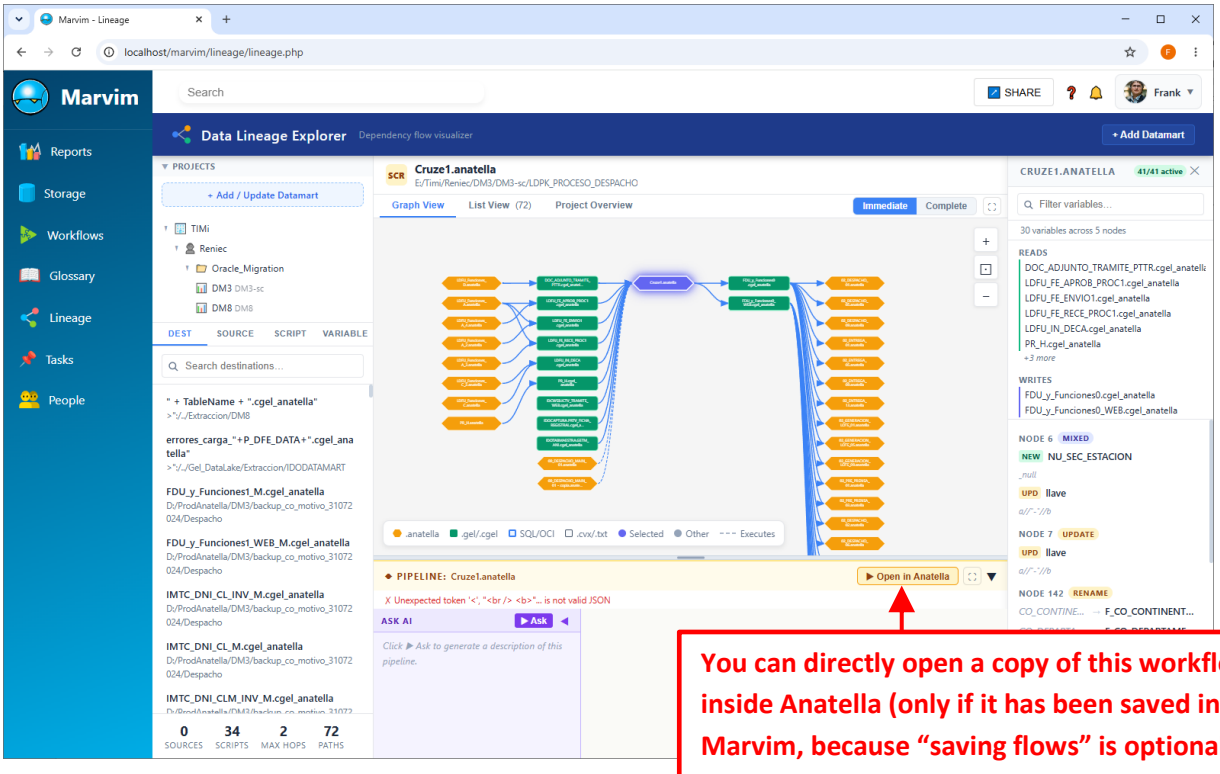


3.6. Lineage

A section dedicated to the exploration of data lineage inside your projects



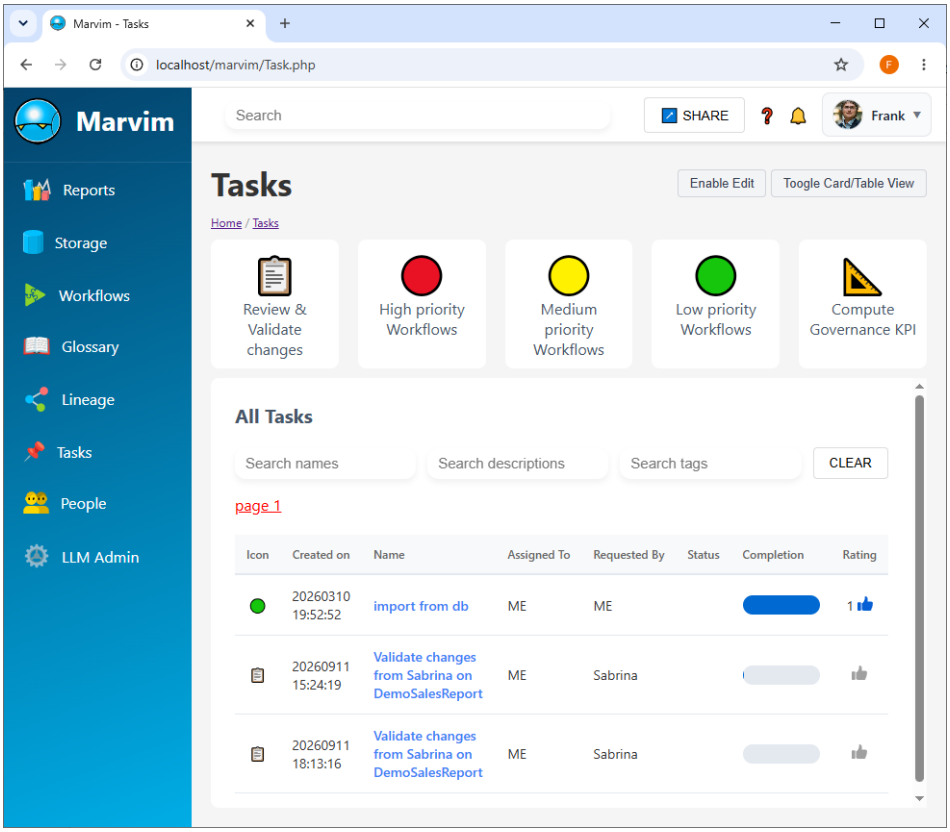
Click on a **Green** rectangle (that represents a Workflow) to zoom on it:



You can directly open a copy of this workflow inside Anatella (only if it has been saved inside Marvim, because "saving flows" is optional)

3.7. Tasks

List of the tasks for the current users:



The most common task is the “Validate Change” task: see section 4.3. for more details about this task.

Another common task is to compute the a “Governance KPI” for a column inside the dataset (and then publish the result of this computation inside Marvin through the Marvin API). This “Governance KPIs” give an idea how reliable a column is. By default, in Marvim, we typically have these two “Governance KPIs”:

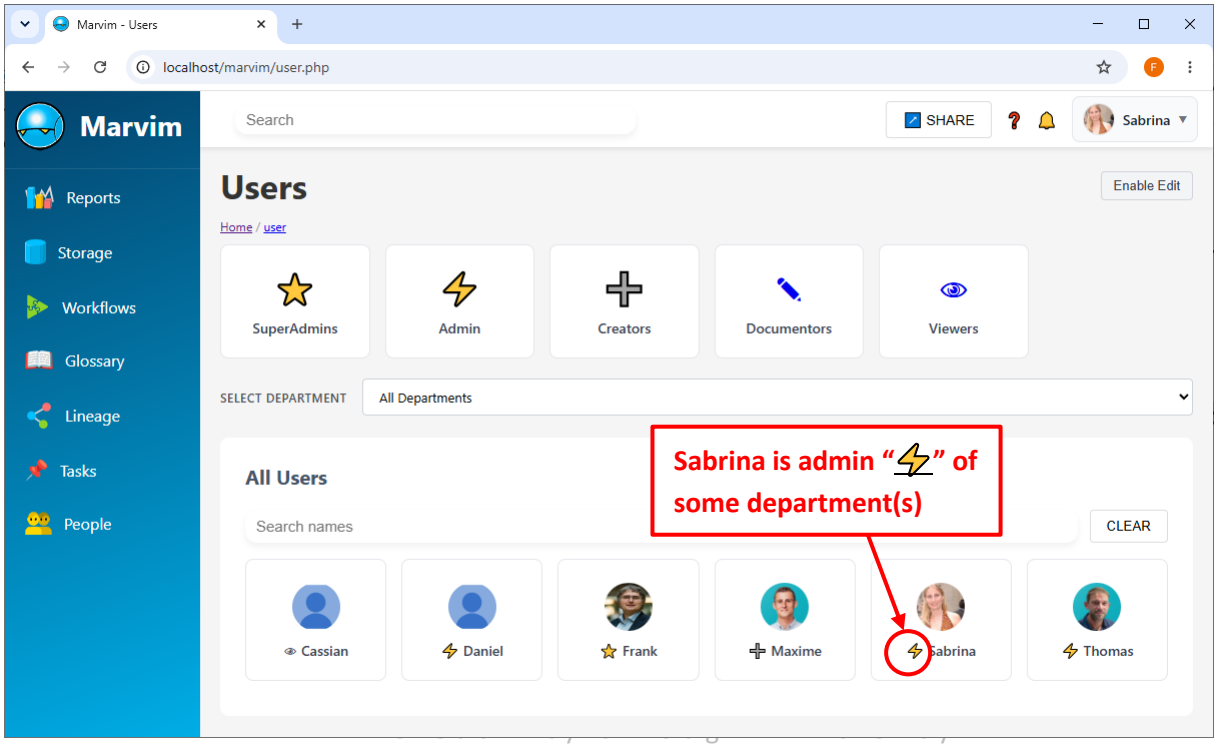
- the percentage of non-null value inside a column,
- the percentage of valid data in a column. By “valid”, we mean, for example:
    - For the column “customer age”: The percentage of numbers that are between 18 and 120
    - For a column “Date”: does it follow the standard date formatting such as yyyy-MM-dd?
    - Etc.

3.8. People

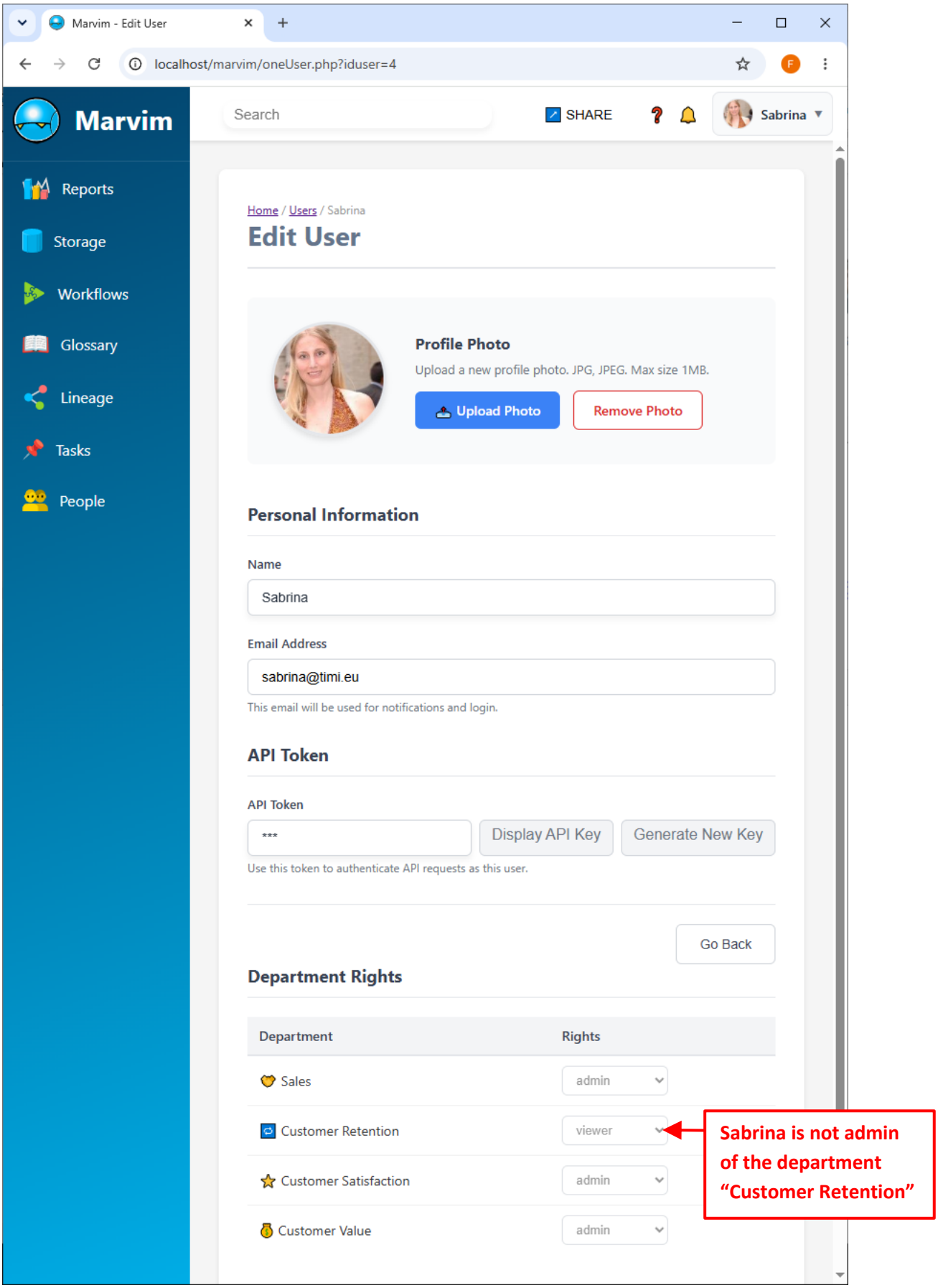
Manage the Marvim users.
The Marvim users are classified in 5 groups. Each group has different “rights”:

- **Viewers**
  Can only view the content of the documentation.
  The consultation is limited to the departments where the user has “View” rights.
- **Documentors**
  Rights: all what the previous group can do PLUS:
  Can edit the content of the documentation but not add or remove assets (e.g. these users cannot add or remove columns inside a table).
  The edition is limited to the departments where the user has “Documentor” rights.
- **Creators**
  Rights: all what the previous group can do PLUS:
  Can add or remove assets
  The add/remove action is limited to the departments where the user has “Creator” rights.
- **Admin**
  Rights: all what the previous group can do PLUS:
  Can change the user’s rights for the departments on which they are admin.
  For example, if you are admin of the “Sales” department, you can give “Documentors” rights to the assets from the “Sales” department to some specific users.
- **Super-user**
  Rights: everything.
  Can do everything directly (e.g. no Approval/Rejection procedure).
  Can assign any rights for any department to any user.
  Can reset passwords.
  Can edit LLM settings.

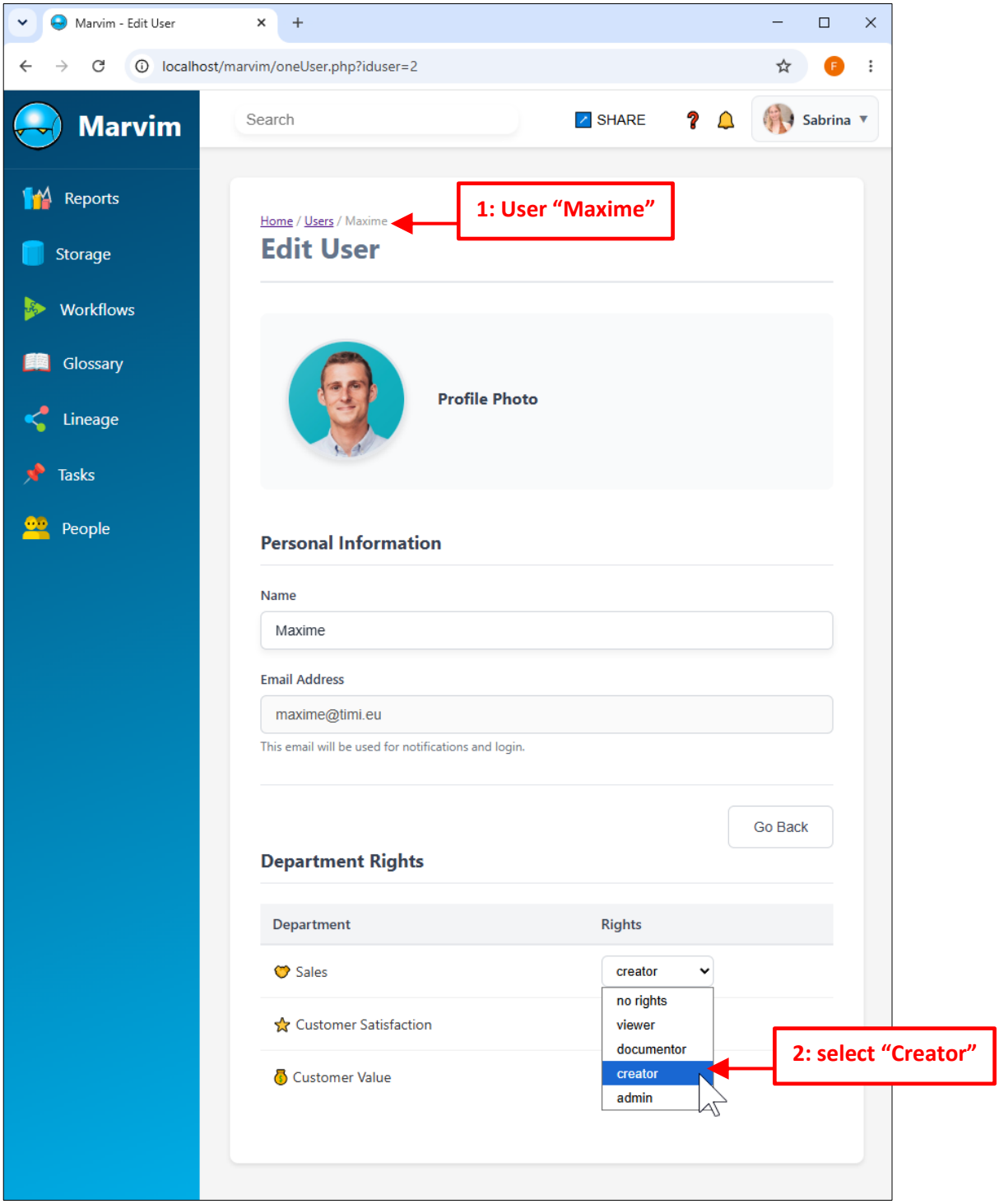
To be able to use the Marvim API to add/update the meta-data inside Marvim for a specific department X, you need at least the “Creators” rights for this department X.



Let's click on Sabrina:
Sabrina is "Admin" in 3 departments: "Sales", "Customer Satisfaction", "Customer Value" :



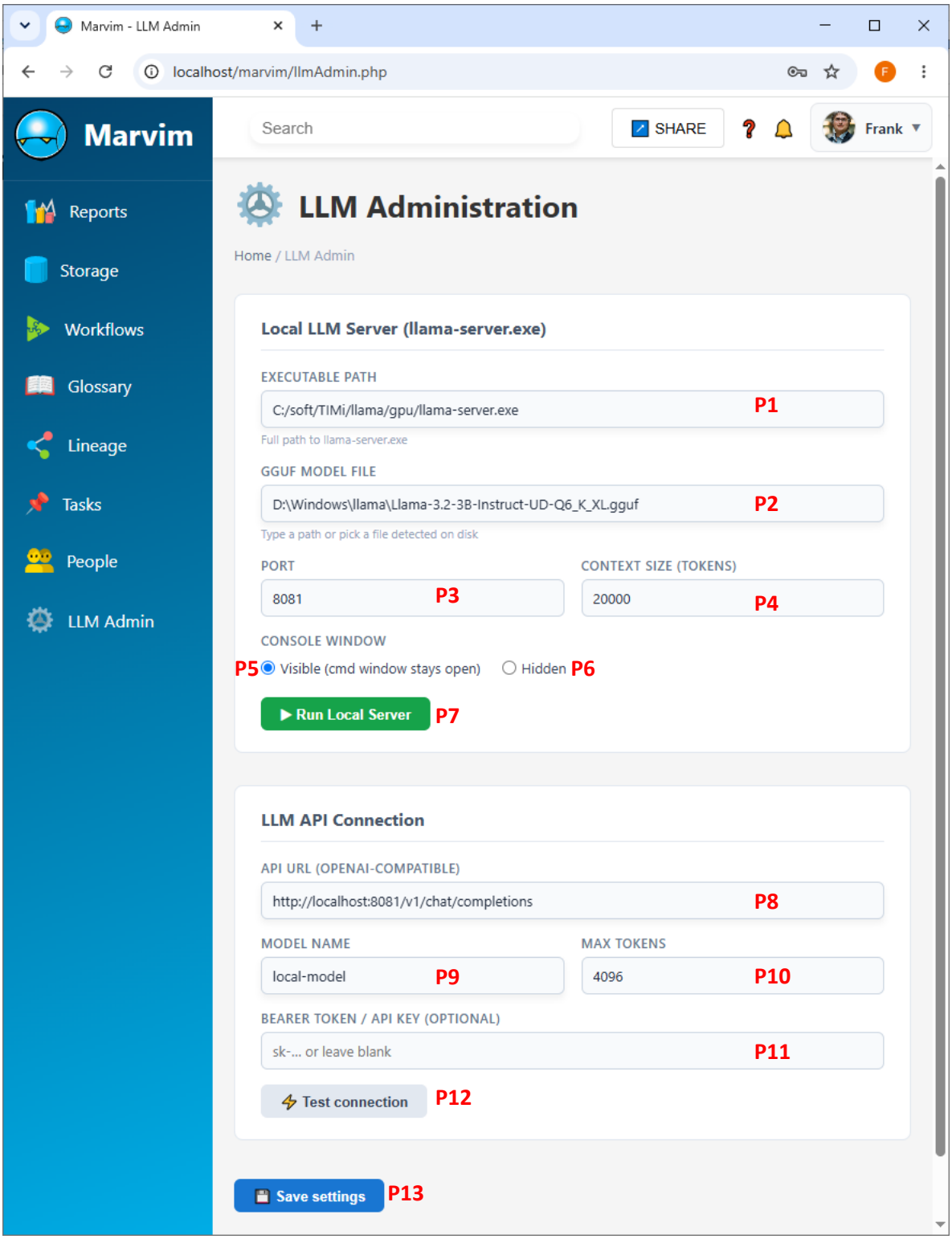
Let's continue to use the user "Sabrina" and let's give the "Creator" rights to Maxime. Click on the icon from Maxime in the user list and then:



3.9. LLM Settings

This section is only accessible for users with Super-Admin rights (otherwise it's hidden).

There are 13 parameters in this section: from parameter **P1** to parameter **P13**:

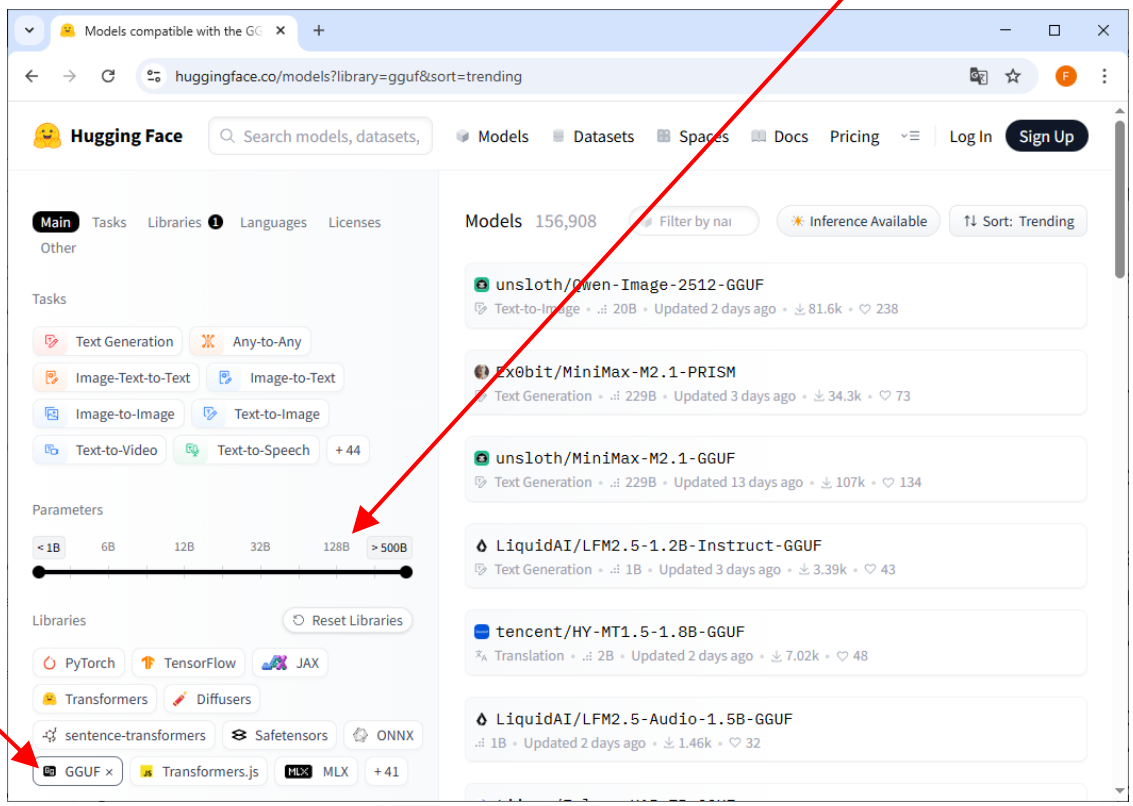


The “Auto Complete” button inside the Storage and Workflow section (that allows to use a LLM to automatically write the description of the storage/workflow) is only visible if the connection to the LLM engine is properly configured. With Marvim, you can use a local LLM/AI (to be 100% sovereign and secure!) or you can use an LLM somewhere in the Cloud (OpenAI-compatible).

3.9.1. Local LLM Setup.

Marvim cannot run LLM models by itself: Instead, Marvim connects to another executable named “Llama.exe” that is installed locally on your server. The “Llama.exe” server is the actual software that runs the LLM models “behind the scene”. Each time you want to use a “Local” LLM model, you first have to launch the “Llama.exe” server executable. When the “Llama.exe” executable starts, it first loads into your RAM (more precisely: into the RAM from your CPU if you are using the CPU version of “Llama.exe” or into the RAM from your GPU is you are using the GPU version of “Llama.exe”) a file with the GGUF extension. This GGUF file is actually the “brain” of your AI. You can download GGUF files for free from here: https://huggingface.co/models?library=gguf

A GGUF file represents the “brain” of your AI. The most important parameter of these ”brains” is the number of parameters inside the model. The higher the number of parameters, the smarter your AI will be. So, it’s advised to use GGUF files with as many parameters as possible. On HuggingFace, you can search for GGUF based on the criteria “number of parameters” using this slider:



Make sure to always select the “GGUF” filter here:

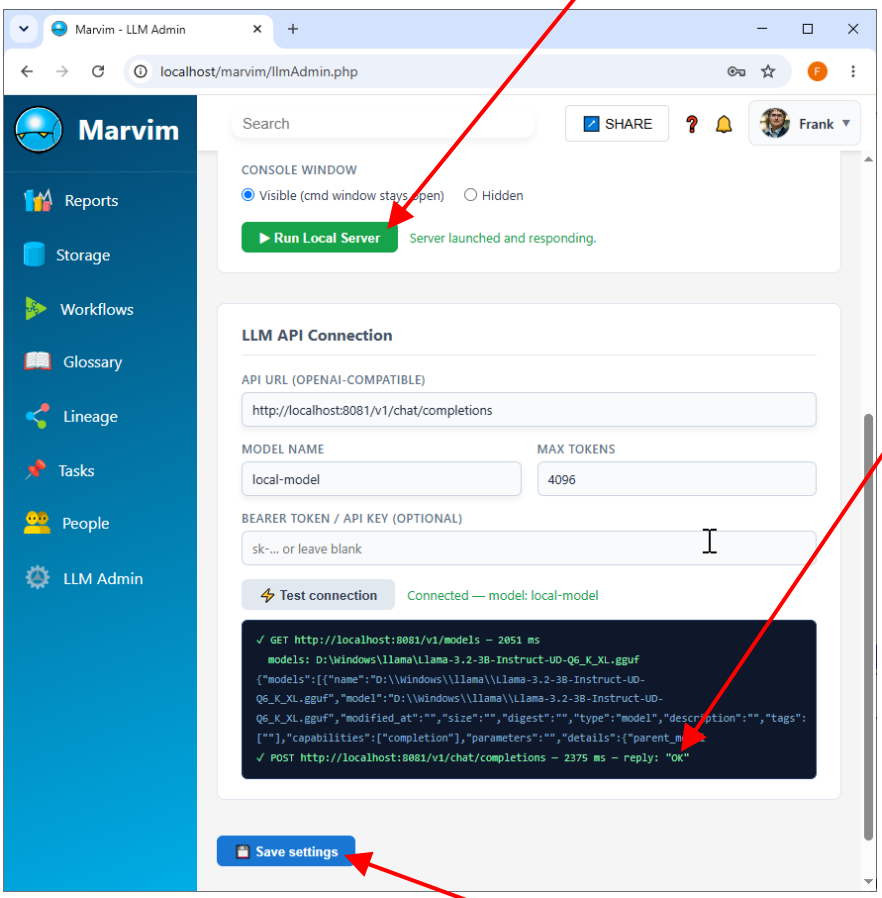
The number of parameters of a LLM model is expressed in “billions” (B) of parameters. For example, a model with 12 billion parameters will be noted “12B”.

If you want more general background information on: (1) how to use “Llama.exe”, (2) how to select the proper GGUF file for your hardware, I would suggest to read the section 5.1.9. of the AnatellaQuickGuide.pdf that is available here:
https://download.timi.eu/docs/_AnatellaQuickGuide.pdf

The procedure to use a local LLM is the following:

- Install “Llama-server.exe”:
  Download and unzip:
  - For a LLM running on your GPU: https://download.timi.eu/LLM/llama-cuda-x64.zip
  - For a LLM running on your CPU: https://download.timi.eu/LLM/llama-cpu-x64.zip

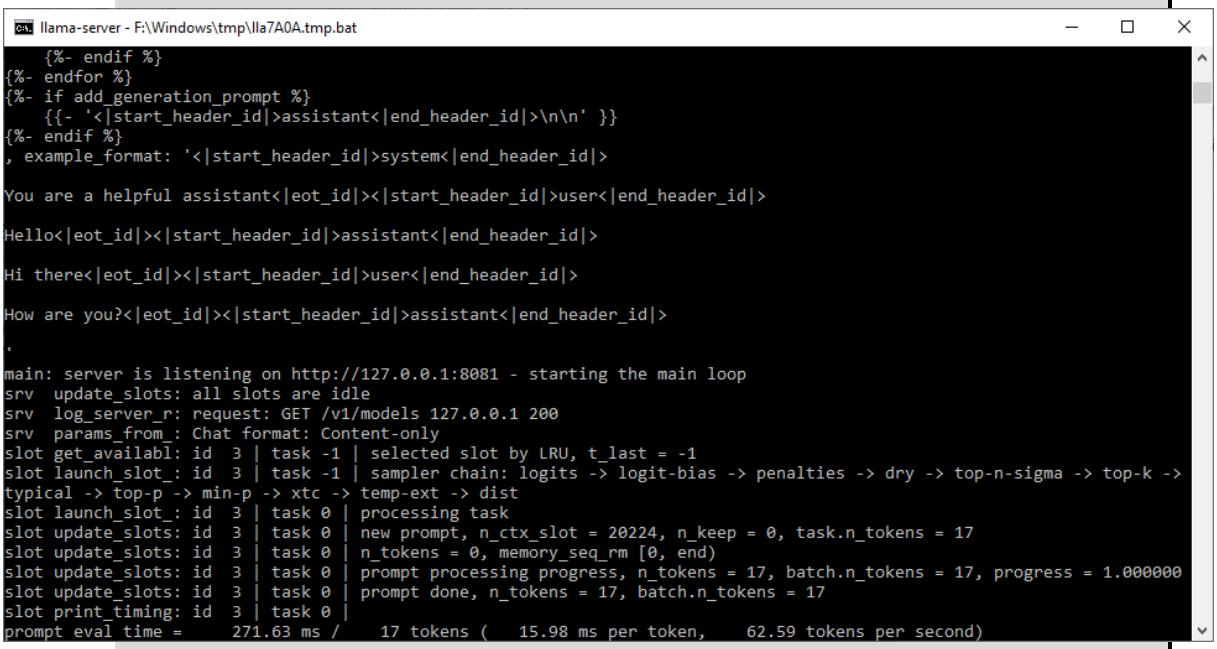
  I would suggest to unzip the zip file inside c:\soft\TIMi\llama\gpu
  Alternatively, you can also download “Llama-server.exe” directly from the source repository on Github: https://github.com/ggml-org/llama.cpp/releases
- Setup the parameter **P1** to point to your local installation of “Llama-server.exe”
- Download a GGUF file:
  You can find here: https://download.timi.eu/LLM/Llama-3.2-3B-Instruct-UD-Q6_K_XL.gguf
  … a default GGUF file (“Llama-3.2-3B-Instruct-UD-Q6_K_XL.gguf”) that was obtained from this webpage: https://huggingface.co/unsloth/Llama-3.2-3B-Instruct-GGUF
  This LLM model has only 3B parameters so that it’s “small enough” to work both on CPU and on GPU. I would strongly advise you to use a larger model (with more parameters): Please refer to the section 5.1.9. of the AnatellaQuickGuide.pdf (pdf available here) to know more about this subject.
- Setup the parameter **P2** to point to your local GGUF file
- You can let all other parameters at their default value.
- Click the green button “▶ Run Local Server”. A console appears. Wait a little.
  You should now see “OK” in green here:



- Click on the button **P13** “Save settings”: You are good to go!

The "Llama-server.exe" is running as long as the black&white Console is running. When you don't need to use any LLM service to generate text anymore, you can close the console (**but not before**!).

The "Llama-server.exe" console looks like this:



3.9.2. Cloud LLM Setup

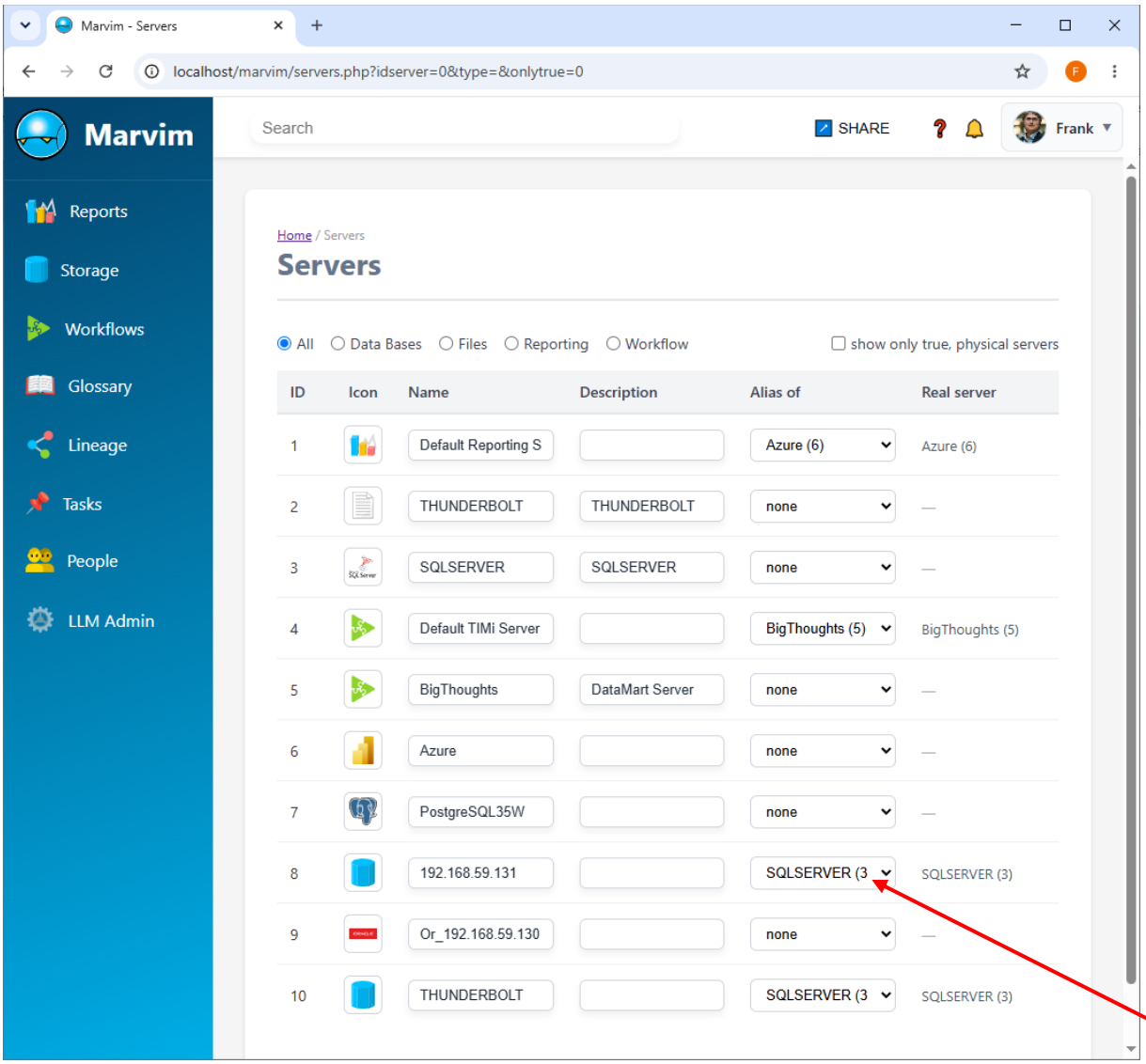
You only need 3 parameters (You can let all other parameters at their default value) :

- Parameter **P8**: To use the LLM's from OpenAI, please enter: https://api.openai.com
- Parameter **P9**: the Model to use
  For example: "GPT-5.2"
- Parameter **P10**: the API key to access OpenAI server

To get the parameters **P9** and **P10**, follow the procedure given in section 5.9.1.1. of the AnatellaQuickGuide.pdf (this pdf is available here)

Once these 3 parameters are properly set, click the button "Test Connection" **P12**. If everything is ok, click on the button **P13** "Save settings" and you are good to go!

3.10. Servers

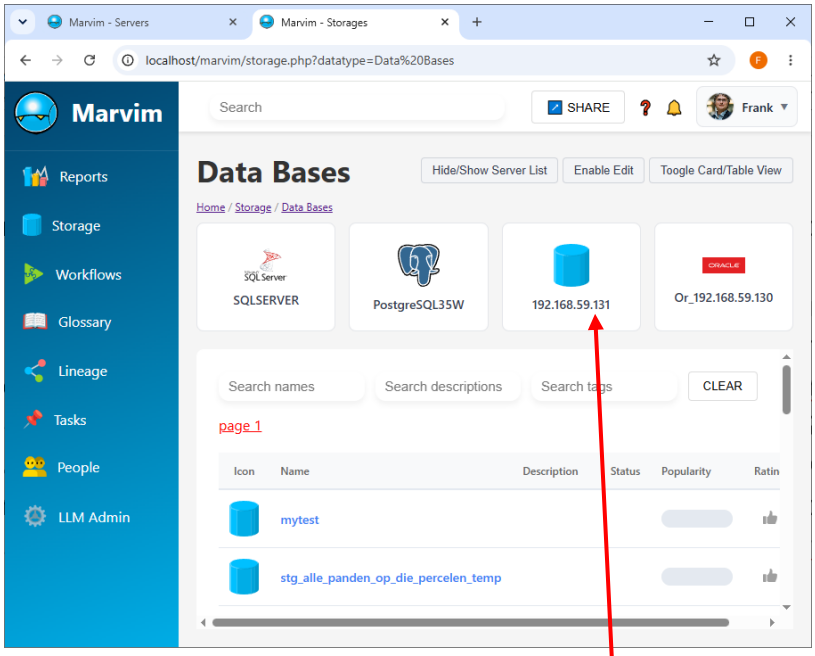


There are 4 different classes of servers: (1) Data Base servers, (2) File servers, (3) Reporting server, (4) Workflow Servers.

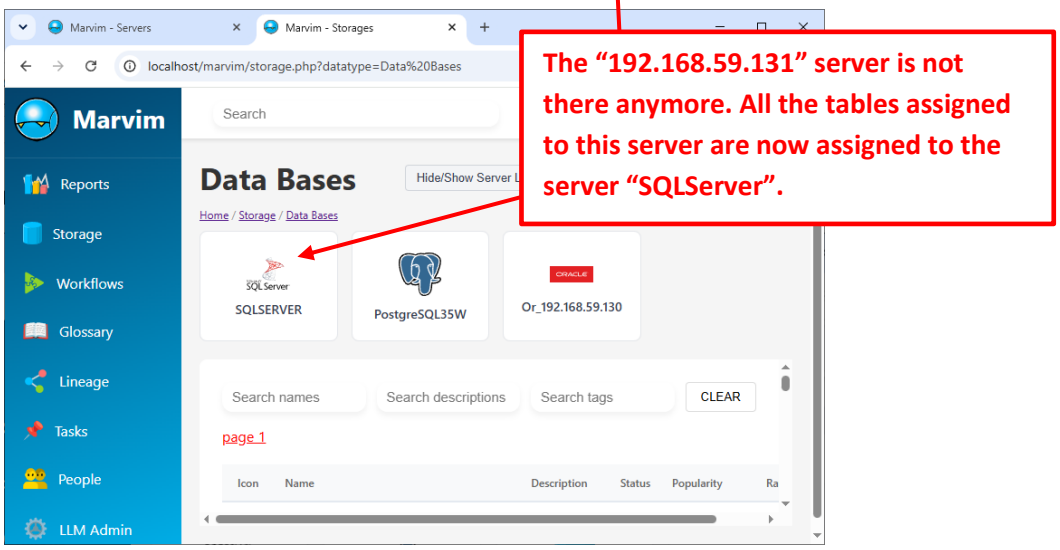
When you use the Marvim API to automatically populate & update the Marvim database, you always need to provide a server name (e.g. the server name of the database server where the table is stored). If this server does not exist yet inside the Marvim database, Marvim will automatically add it to the list of known servers.

Because of this automatic addition, it can happen that we end-up with a slightly "dirty" Marvim database (that contains way too many different servers). Let's take an example. Let's assume that we have 2 workflow servers. Inside the first workflow server, the connection to our SQLServer is named "SQLServer3". Inside the second workflow server, the connection to SQLServer is named "192.168.59.131". We use the Marvim API to "push" to Marvim all the meta-data-informations from each of the 2 workflow servers. We'll end-up with 2 DB servers (that are named "SQLServer3" and "192.168.59.131") that are both defined inside Marvim. …While, in reality, we have only one unique SQLServer server (named "SQLServer3") ! . This is what happened in the screenshot hereabove. To correct this situation, we said: "*the 192.168.59.131 server is an alias of the SQLServer server.*"

Before the correction (before creating the alias), we could see:

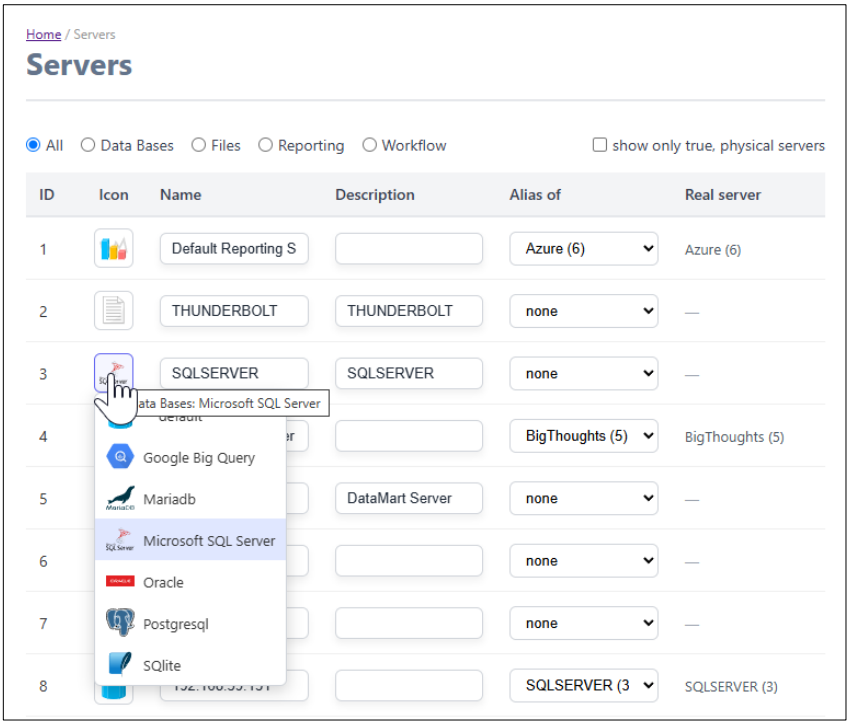


Now, after the correction (after creating the "alias" that says: "*the 192.168.59.131 server is actually in reality the SQLServer server*"):



The possibility to create "Server Aliases" is paramount when using the Marvim API in order to clear the clutter due to some possible inconsistent naming conventions on different workflow servers.

You can change the icon of a server, just by clicking the current icon: For example, let's click the icon on the 3rd row:



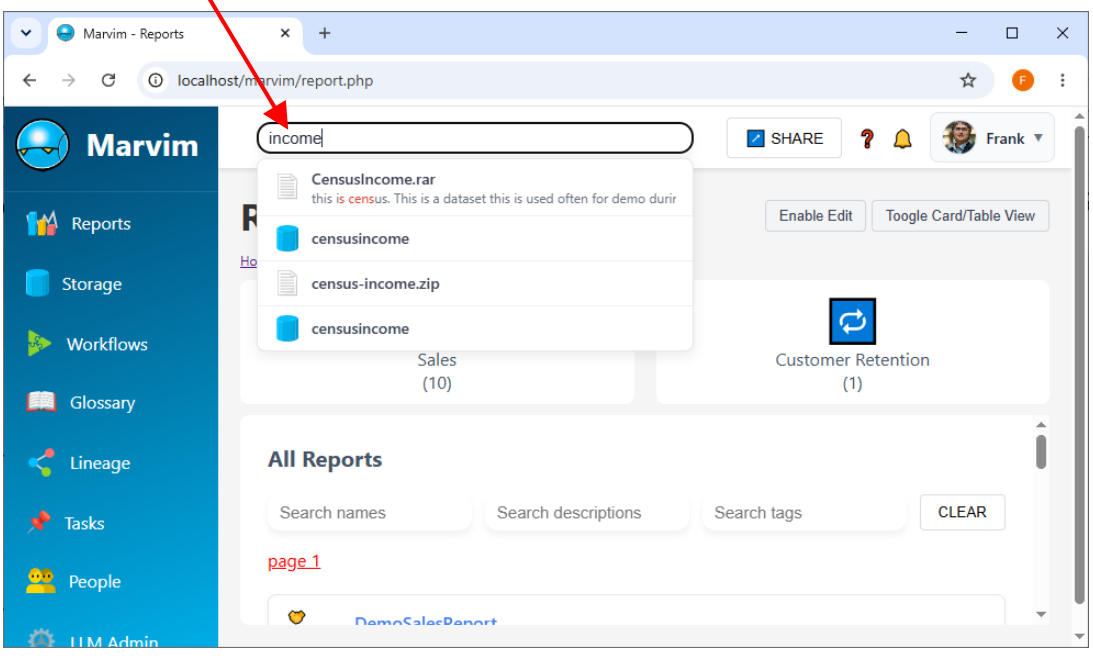
You can easily add new icons to use inside Marvim: Just add more icon images inside the directories:

- <Marvin_root>/www/marvin/ressources/ReportingServers
  …for the Reporting servers
- <Marvin_root>/www/marvin/ressources/StorageServers
  …for the Storage servers (Database+File+App+API servers)
- <Marvin_root>/www/marvin/ressources/WorkflowServers
  …for the Workflow servers

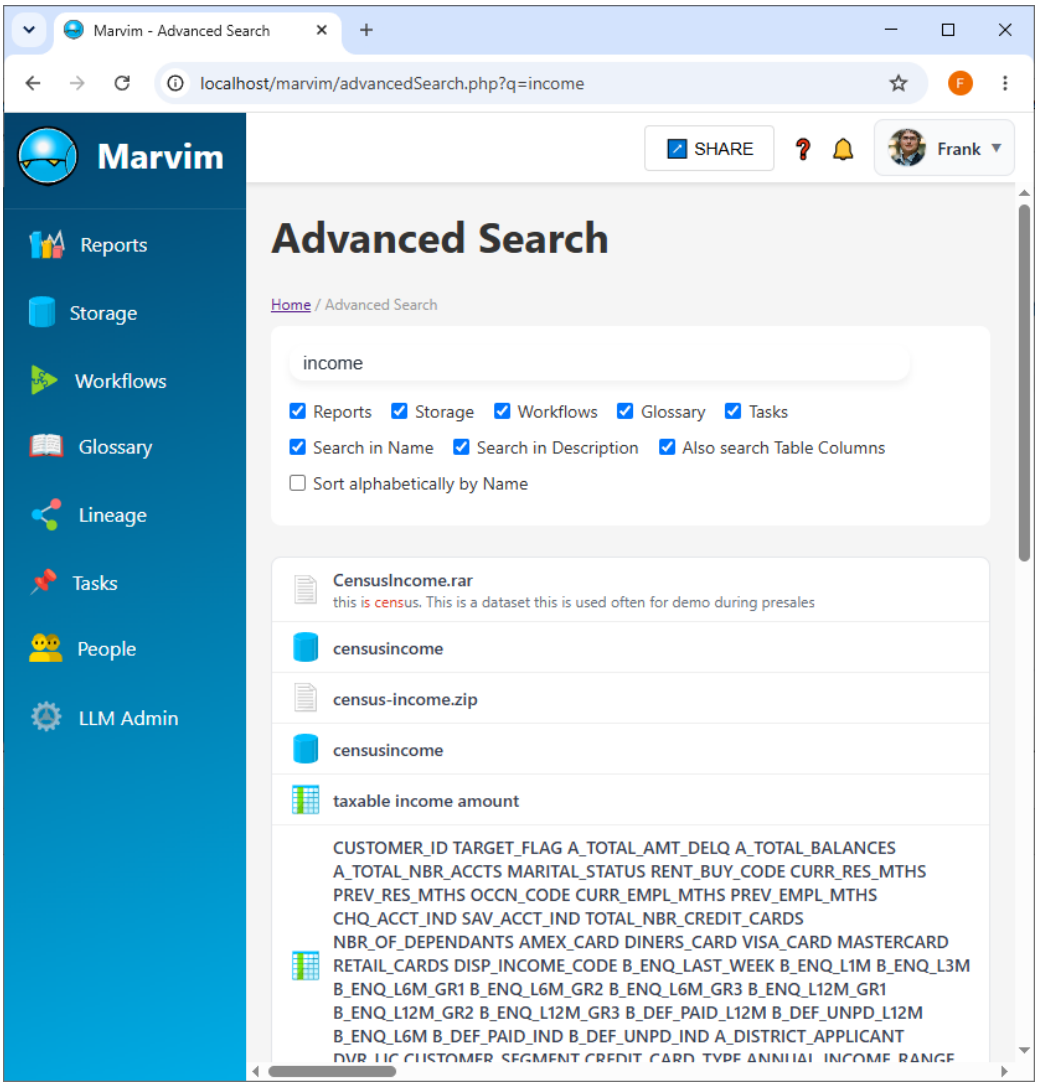
..and they'll magically appear as a new choice inside the icon drop-down-combo-box inside Marvim.

3.11. Global Search

At any time, write here some keywords to perform a "global search" over all terms and definition defined inside Marvim:



Press [Enter] to go to a dedicated “Global Search” Page:

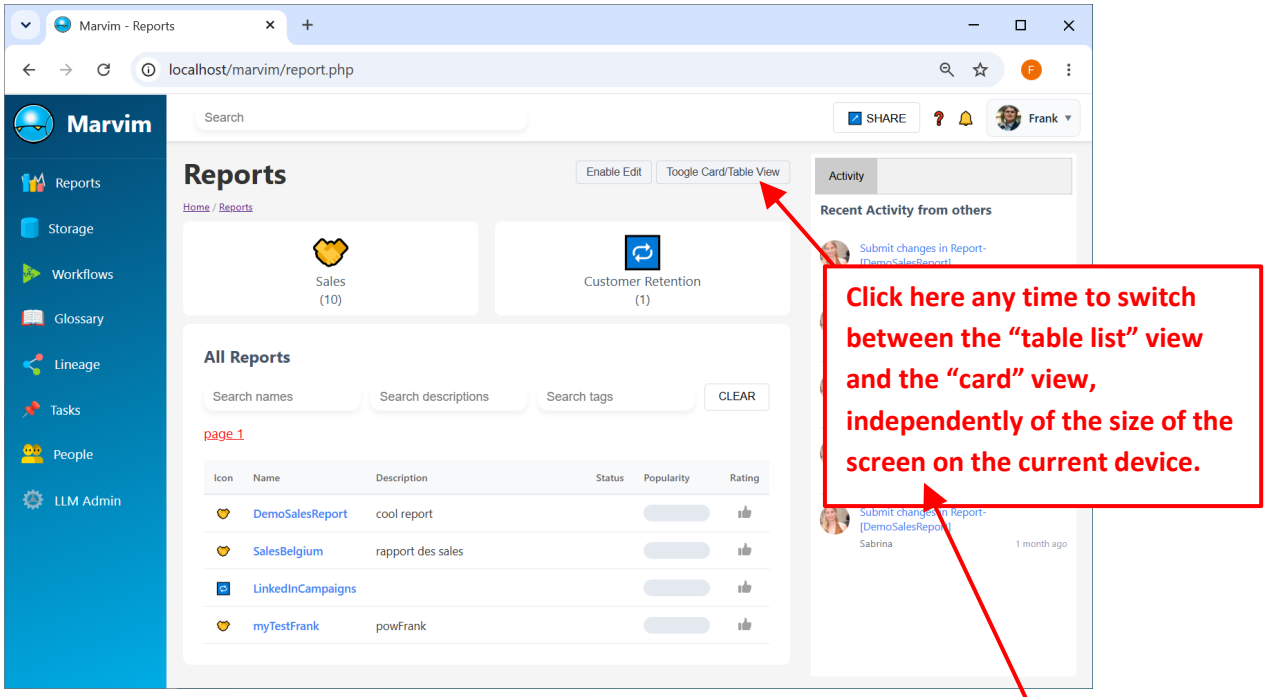


4. Marvim Technical notes

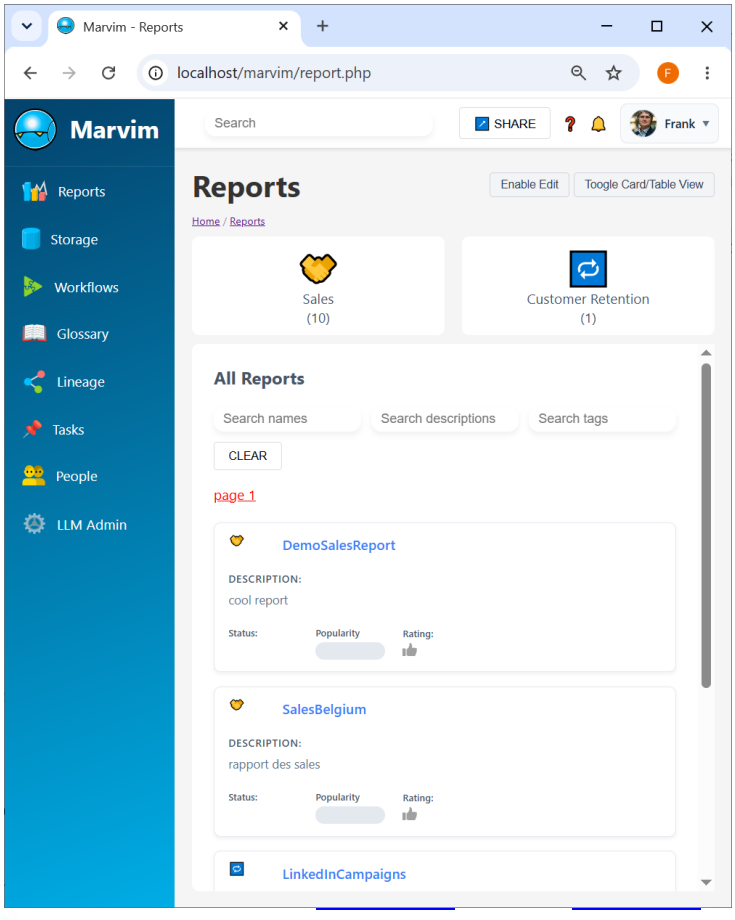
4.1. Responsive Design

Marvim is a website that is using a “Responsive Design”. You can consult Marvim using a laptop, a tablet or a Mobile phone and Marvim will adapt the display so that it remains readable on all devices. For example, the “Reports” list is:

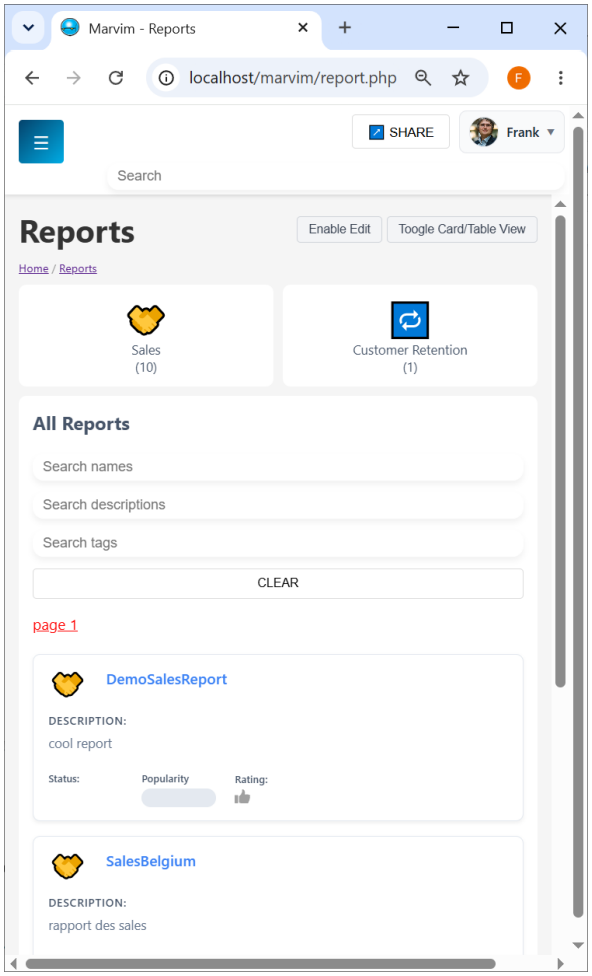
- On a wide-screen laptop:



- On a Tablet screen (on smaller screen, the “table list” is replaced by some “cards”):



- On a tight Mobile phone screen (the normal menu is replaced by a “burger menu”):



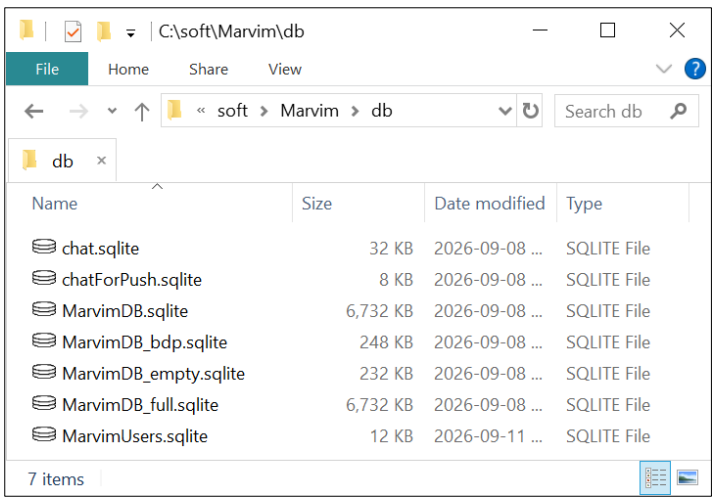
4.2. Marvim Databases

Inside the directory “<marvin_root>/db”, you’ll find all the Marvim databases: These are sqlite files.

There exist many open&free tools to open and see the content of SQLite files.

For example, you can use: “TIMiSqliteExplorer.exe”
..that is always installed with TIMi/Anatella.
(it’s inside the directory “C:\soft\TIMi\bin”).

Here is a screenshot of this directory “<marvin_root>/db”:



To backup Marvim, you just need to make a copy of the files inside this directory. You only need to copy the .sqlite files. You can backup the files at **any time**.

The most interesting file in this directory is the file named "MarvimDB.sqlite" that contains all the Meta-data and Documentation accessible in the Marvim web interface. In addition to "MarvimDB.sqlite", there exist 3 other versions of this same file:

- **"MarvimDB_empty.sqlite":** Overwite "MarvimDB.sqlite" with this file to reset all Marvim data to nothing (this is the empty, default Marvim database)
- **"MarvimDB_full.sqlite":** Overwite "MarvimDB.sqlite" with this file to get a large demo of Marvim that contains hundreds of tables, flow and reports (this is the big demo Marvim database)
- **"MarvimDB_bdp.sqlite":** Overwite "MarvimDB.sqlite" with this file to get a small demo of Marvim that just contains 3 tables and 1 flow (this is the small demo Marvim database)

The file "chat.sqlite" contains all the (contextualized) chat discussion visible in Marvim.

The file "chatForPush.sqlite" is a temporary file not very useful (it's just used to temporarily store some messages to broadcast the new message to all the people currently chatting in Marvim).

The "MarvimUsers.sqlite" contains the credential data of all users.

4.3 Approval/rejection procedure

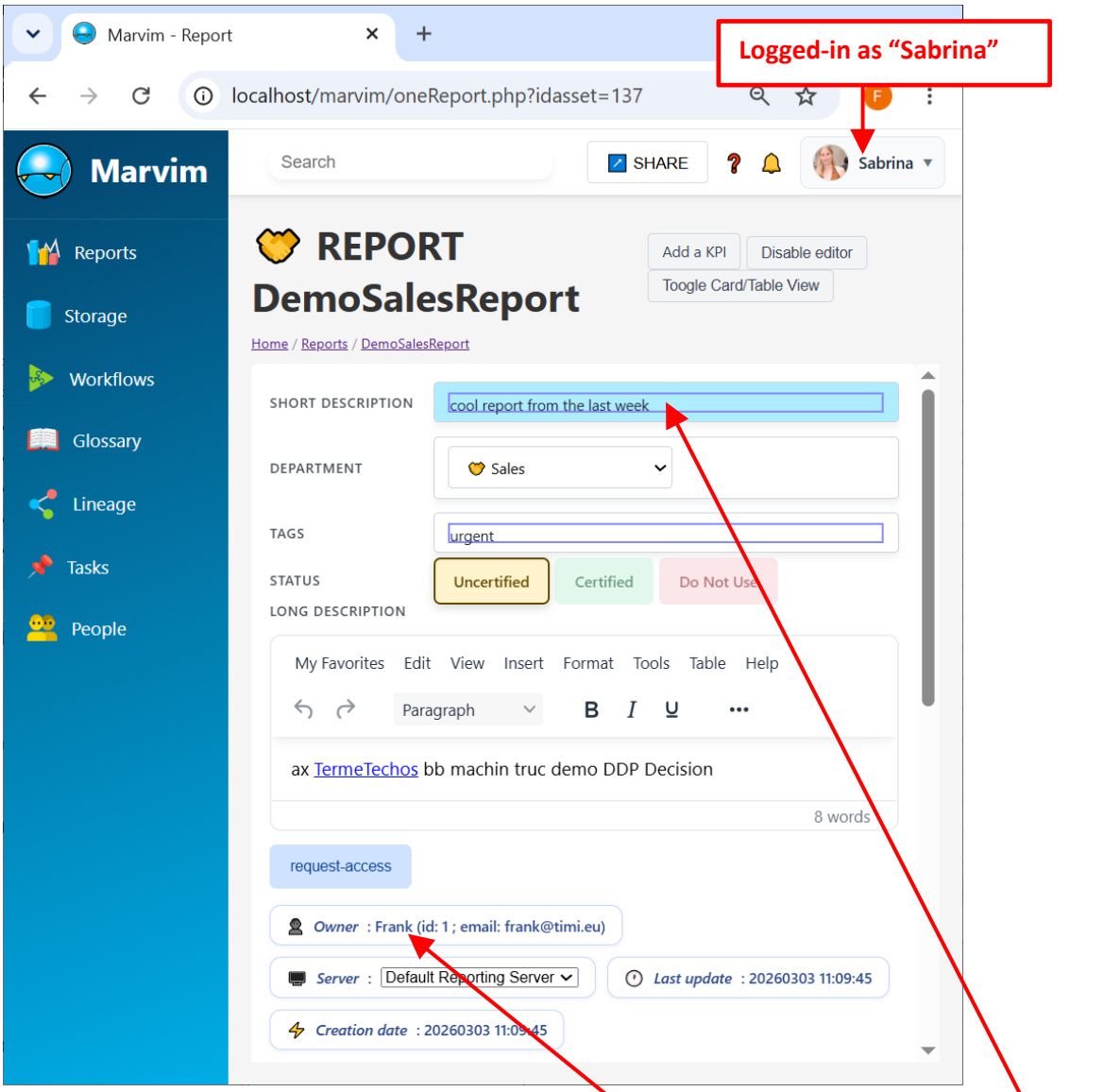
Each time some user edits some documentation page inside Marvin, the proposed "edition" must be approved by the owner of the page.

There is one exception to this rule: If you are a super-user, you can edit the fields directly, without requiring the approval of the owner of any page.

Here is a typical flow:

1. Open the page that you want to edit.

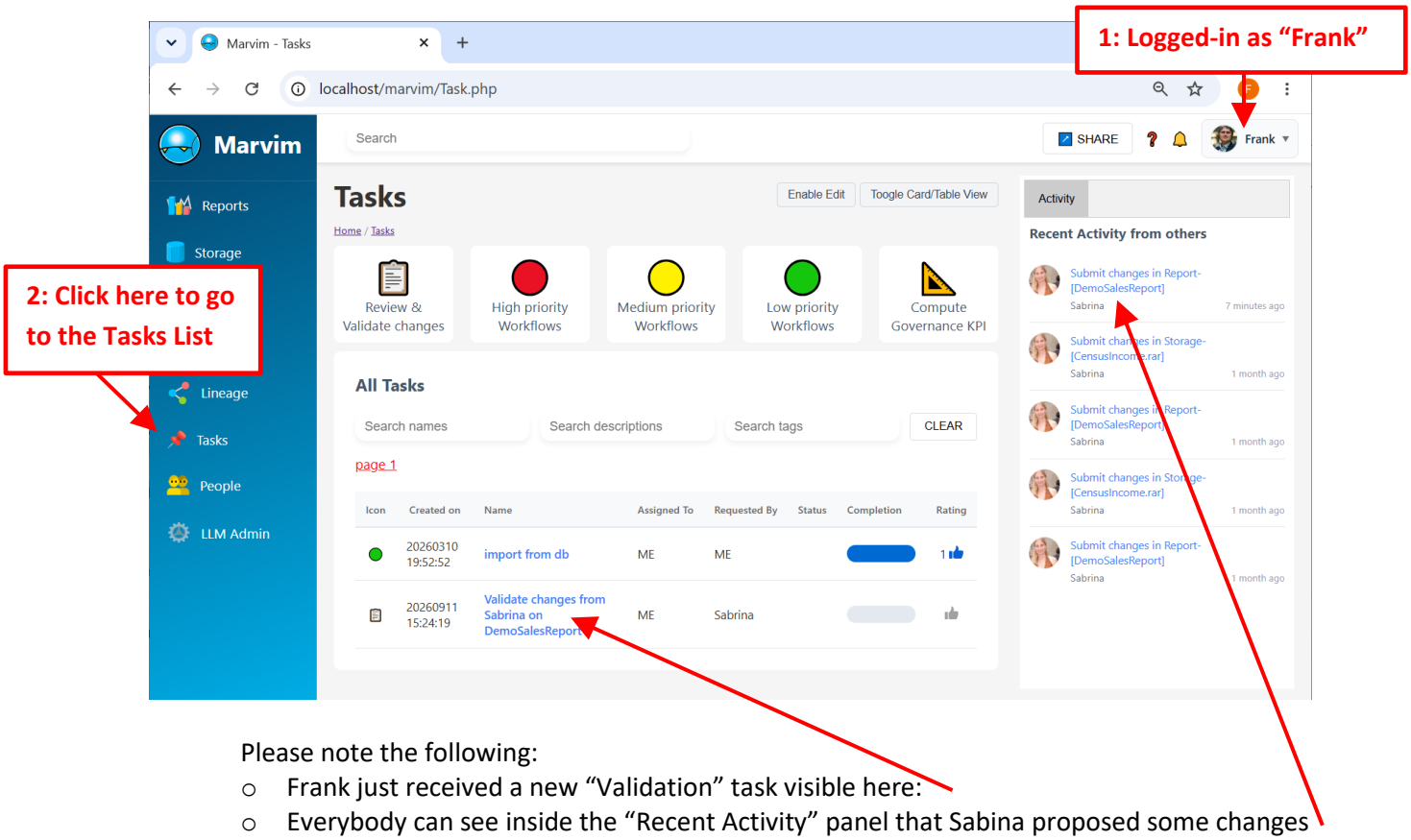
   In the example below, Sabrina just edited the field "Short Description" inside the report "DemoSalesReport":



Please note the following:

- The owner of the “DemoSalesReport” report is Frank. This means that Frank will receive a new validation task to approve (or reject) this edition.
- The background-color of the “Short Description” field is now “blue”
  This is to emphasize the fact that this field has been edited **by you**: i.e. you are the **only person** that is seeing the new proposed value.
  If the background-color stays white (i.e. you didn’t see any blue color at any time), it means that there was a problem when saving the new content inside the Marvim database.
  When you press F5 to refresh the webpage: the “Short Description” field stays “blue”: it stills requires approval (if you are super-admin, the new content is approved directly and the color returns to white after pressing F5).

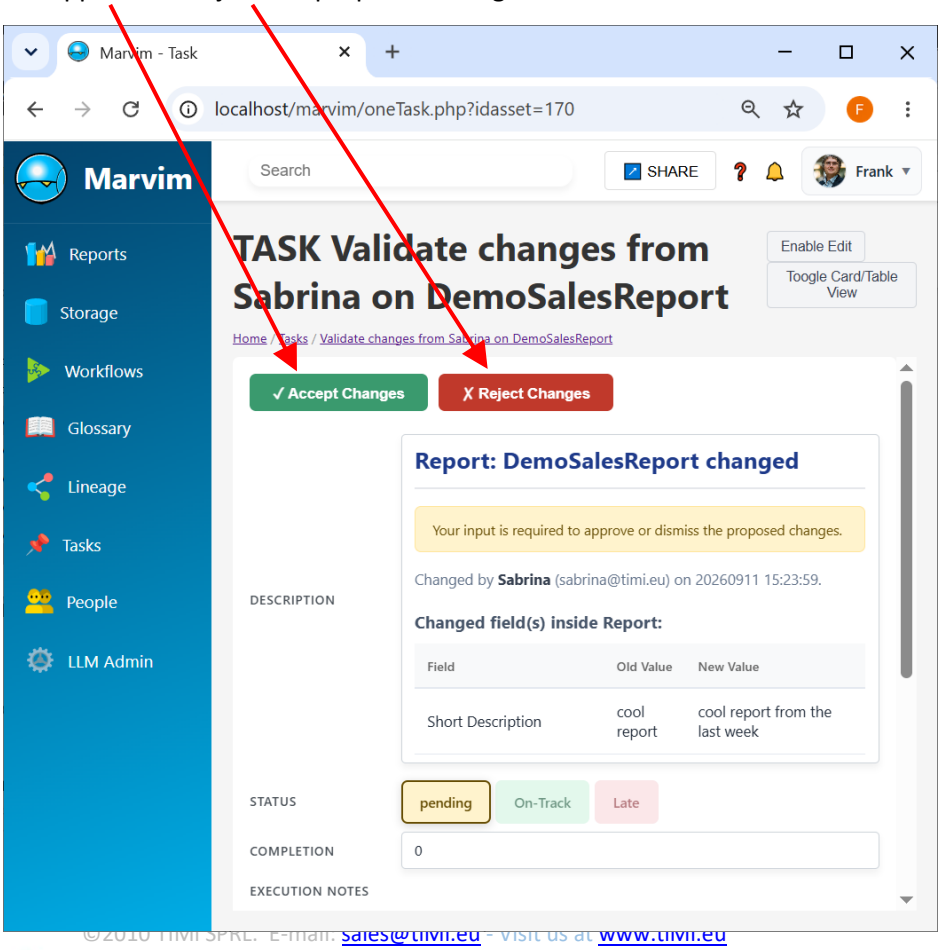
2. Let's log-in as "Frank" (i.e. as the owner of the page that we edited) and check the "Tasks List":



Please note the following:

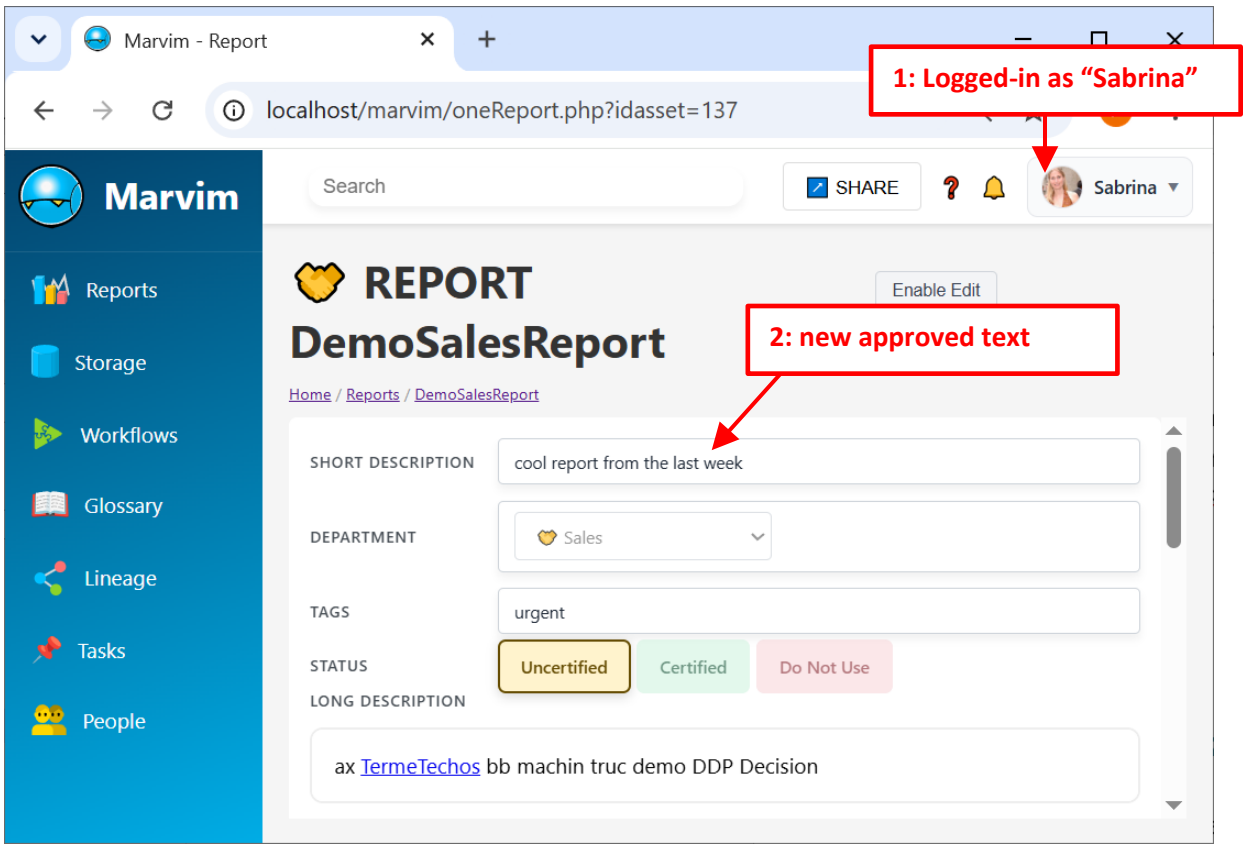
- Frank just received a new "Validation" task visible here:
- Everybody can see inside the "Recent Activity" panel that Sabina proposed some changes

3. Click on the new "validation task".
You can now Approve or Reject the proposed change:



Please note here that Marvin is "clever" in the sense that if will only show you one single "Validation" task that regroups *ALL* the changes proposed by Sabrina, despite the fact that Sabrina might have made all these different changes at different moment in time. All these changes are consolidated in a single task, so that no "information overload" occurs.

4. Let's click on the **green** button "Accept Changes".
5. Let's login again as Sabrina and look at the field "Short Description" inside the report "DemoSalesReport" that we just approved:



Please note that the background color of the "Short Description" field is now white: i.e. this means that the change has been approved and everybody now sees the new approved version.

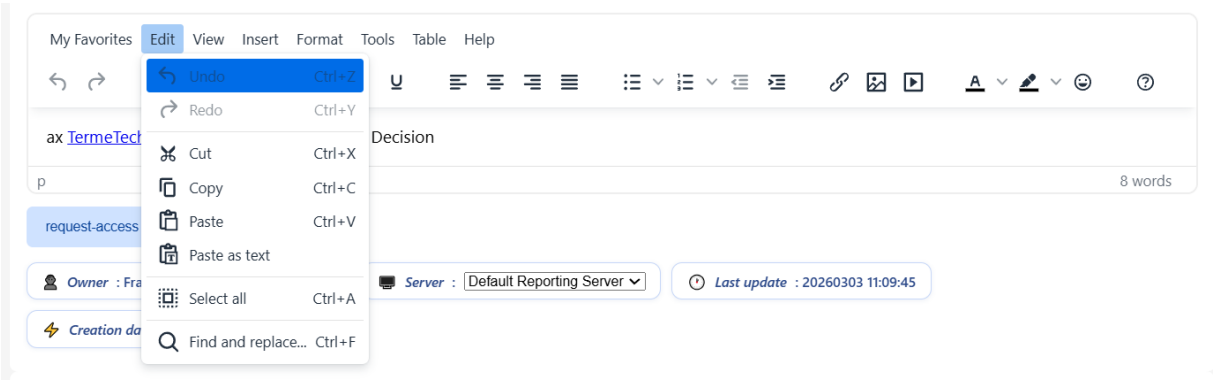
4.4. Text Editor

The WYSIWYG Marvim text editor is very powerful.

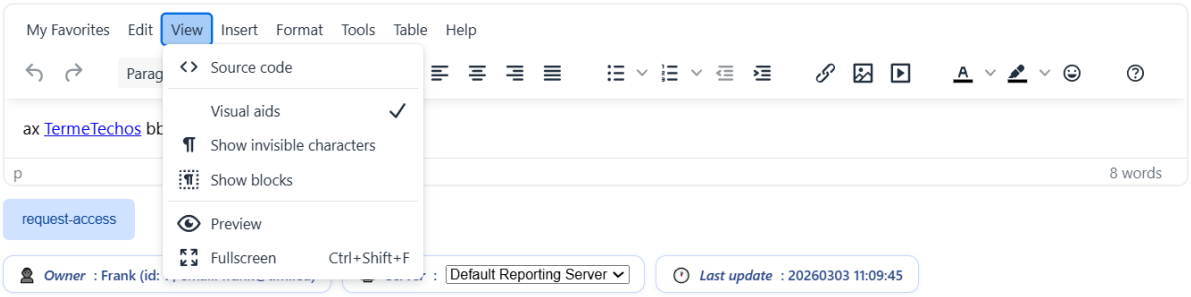
The words that are inside the Glossary are automatically converted to clickable blue URL's (that redirect to the Glossary).

Some functionalities:

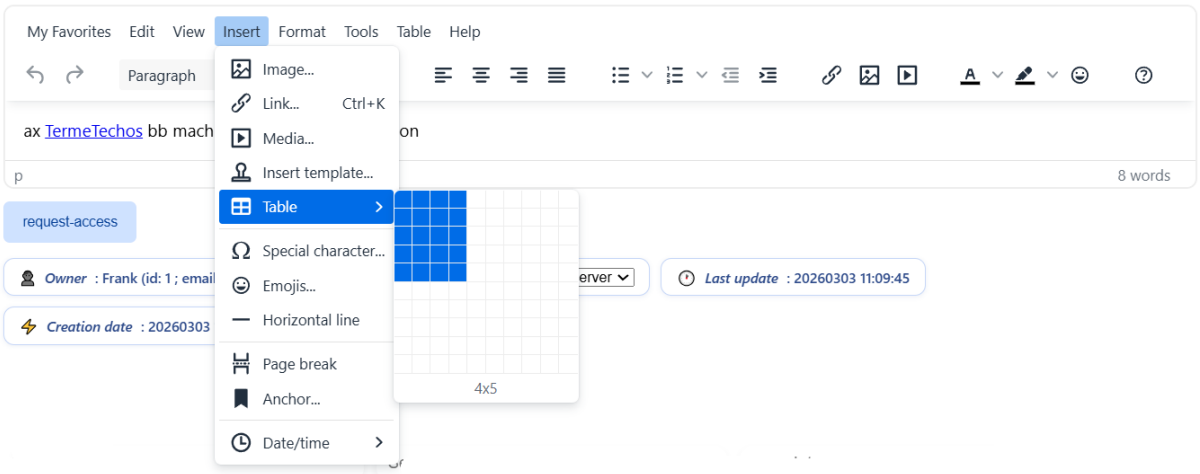
- Undo, redo, copy, paste:



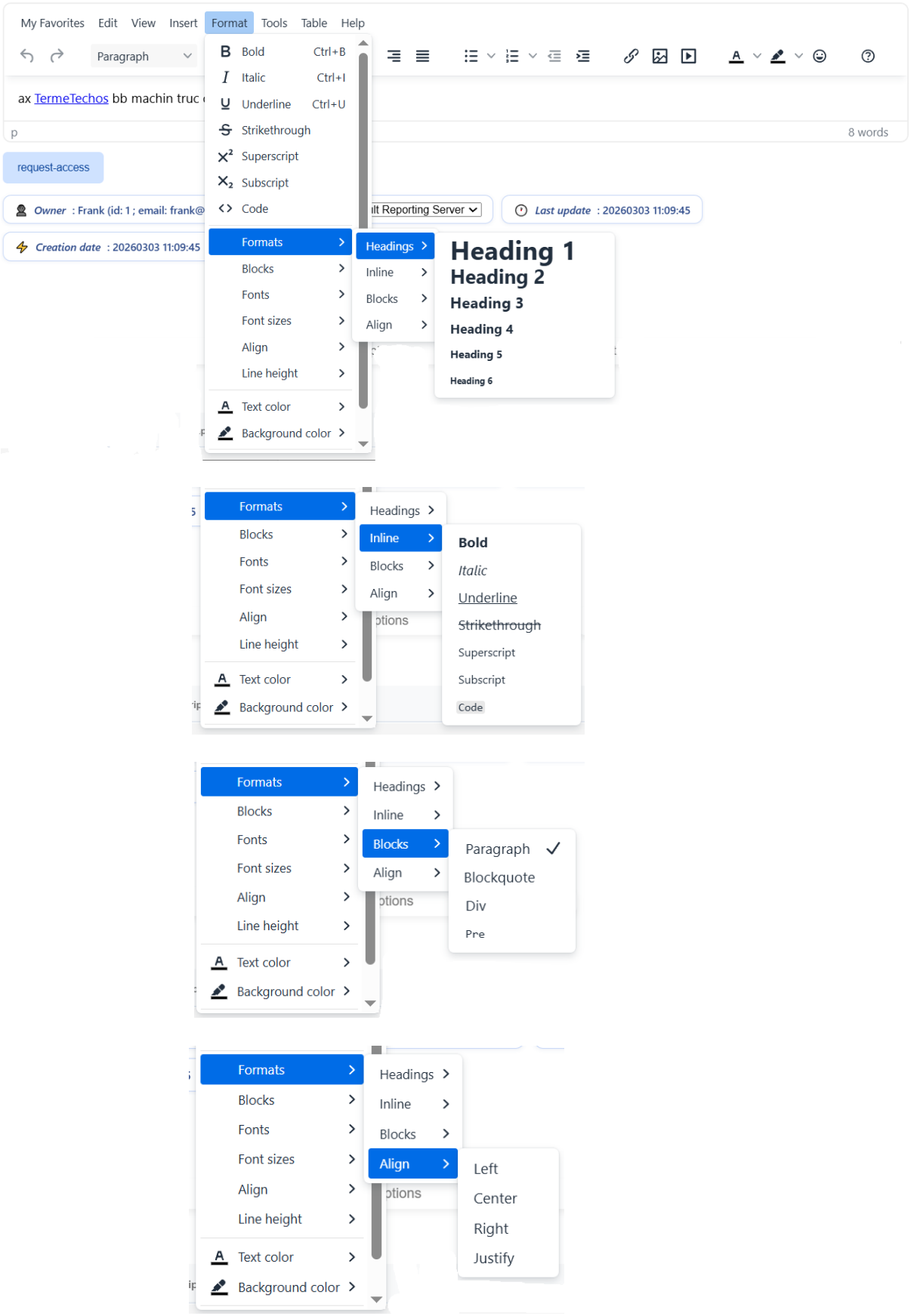
- Visual aids for better design:



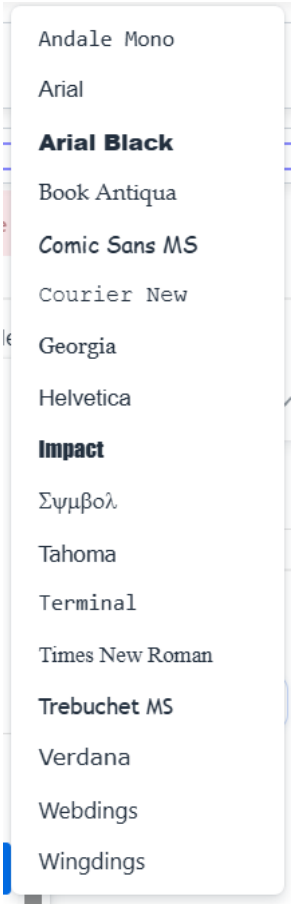
- Insert image, link, media (youtube), table, emojis, page break, date/time:



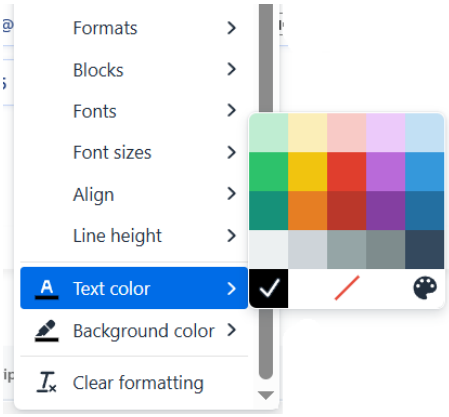
- Offers a lot of text - formatting options:



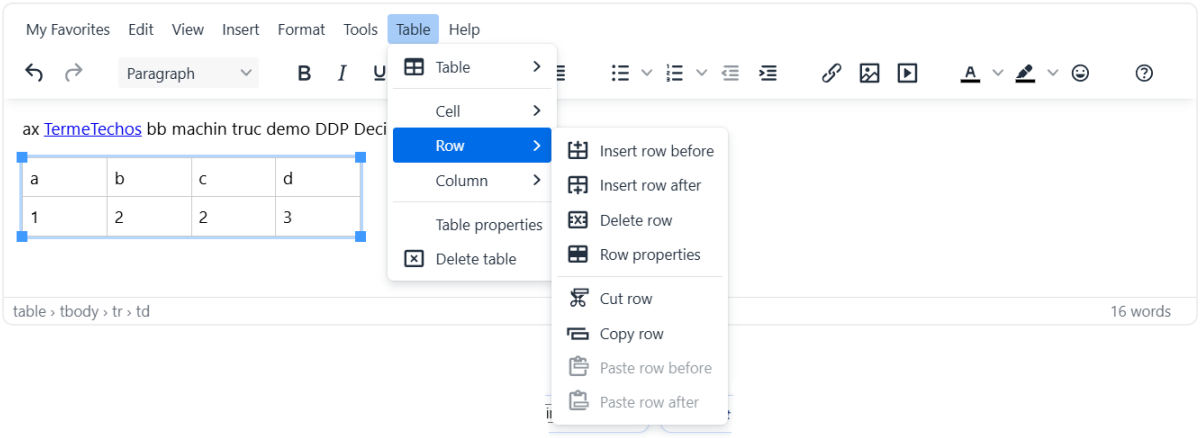
- Many fonts:



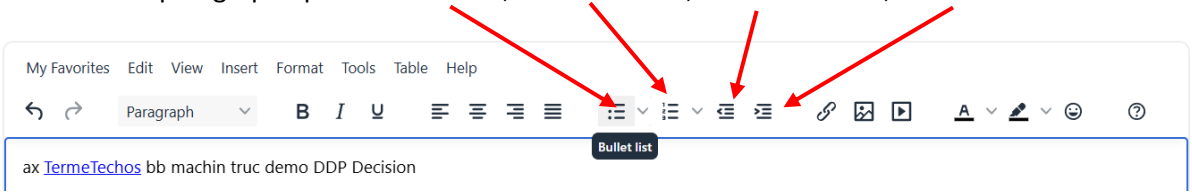
- Easy text & background color selection:



- Include powerful Table/Array editor:

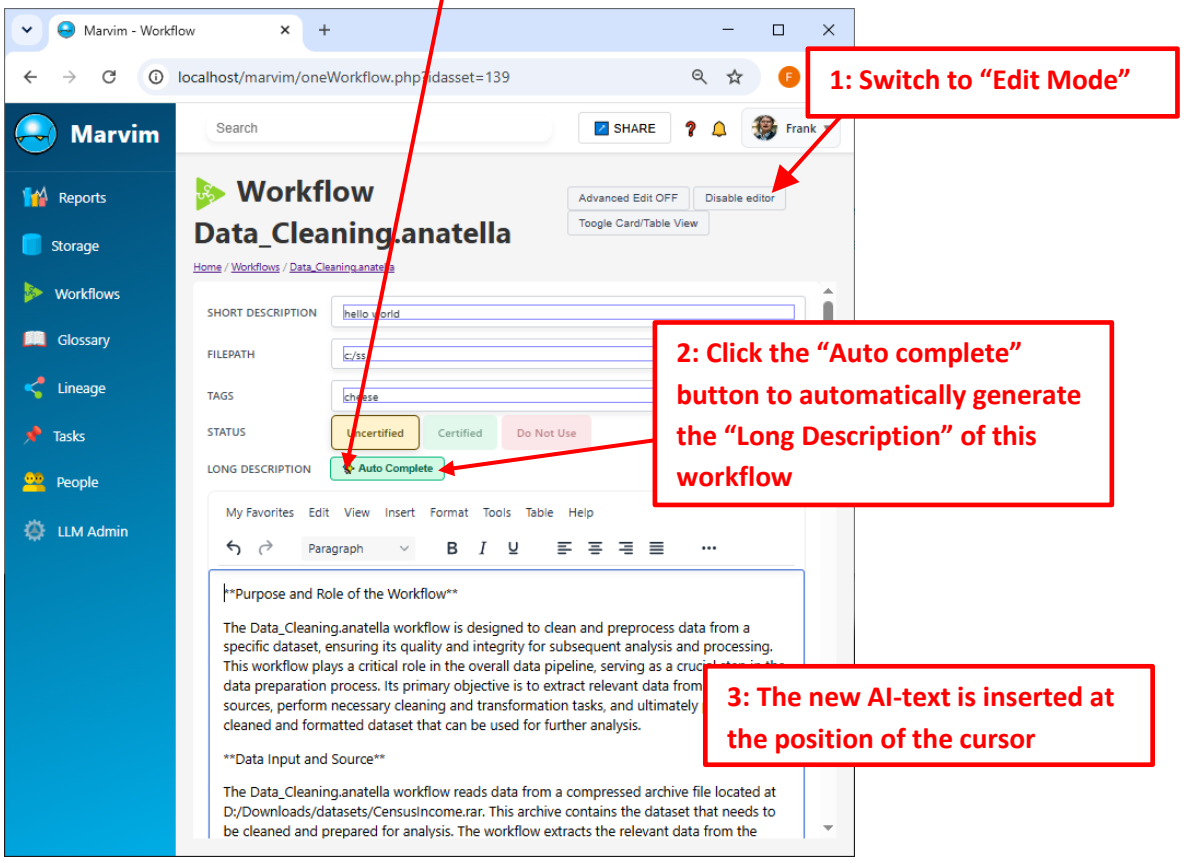


- More paragraph options: Bullet list, Numbered list, increase indent, decrease indent:



4.5. Automated Text Generation (Generative AI - LLM)

If the connection to the LLM has been properly configured (see section 3.9.), then the “Auto complete” button appears (when in “edit mode”):



5. Marvim Web API

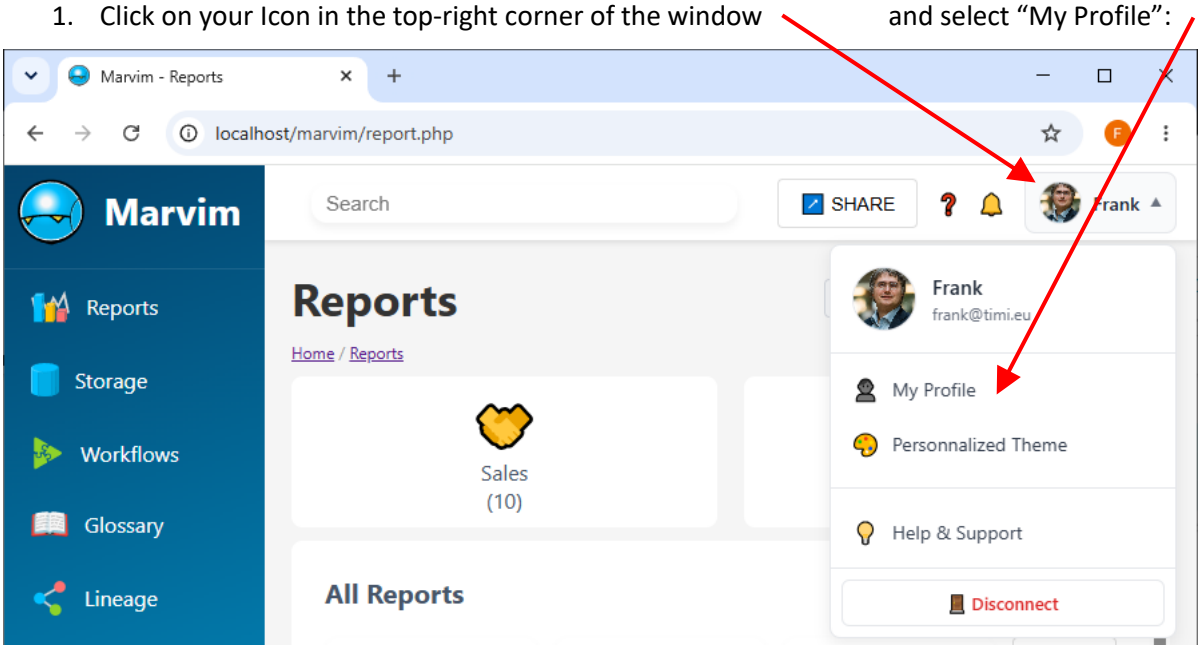
If you are using Anatella as your ETL, you don't need to read this section: i.e. You should just use the 4 dedicated Anatella boxes to automatically update the data inside Marvim.

There are 4 endpoints inside the Marvim API:

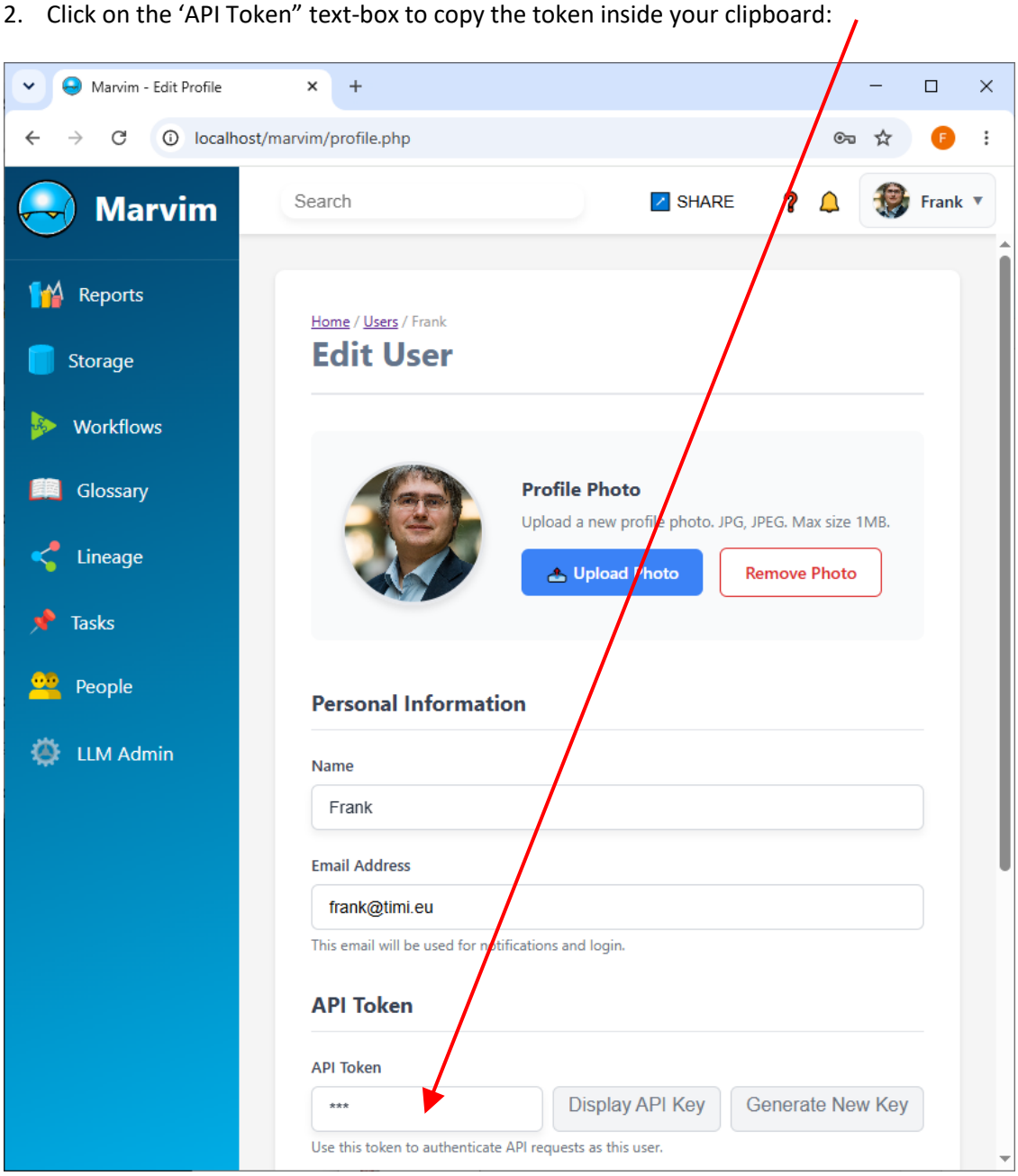
- To search the data contained inside Marvim
  <Marvim_root>/marvim/api_AdvSearch.php
- To add/update reports inside Marvim:
  <Marvim_root>/marvim/api_report.php
- To add/update Storage inside Marvim:
  <Marvim_root>/marvim/api_storage.php
- To add/update Workflows inside Marvim:
  <Marvim_root>/marvim/api_workflow.php

To use all these API, you'll need a "API Token". To get this "API Token":

1. Click on your Icon in the top-right corner of the window and select "My Profile":



2. Click on the 'API Token" text-box to copy the token inside your clipboard:



3. All REST calls to the Marvim API must contain in their HTTP header the following header:

```
Authorization: Bearer <your API Token>
```

5.1. Advanced search

Search Reports, Storage tables, Workflows, Storage columns, Glossary entries and your own Tasks in a single call, with per-category and per-field control over what gets matched.

```
GET  <marvim_root>/marvim/api_AdvSearch.php
```

5.1.1. Purpose

`api_AdvSearch.php` is Marvim's **advanced search** endpoint. One request can search across several kinds of catalog objects — Reports, Storage tables, Workflows, Storage columns, Glossary entries, and the caller's own Tasks — while controlling:

- **which object types** are searched, and
- **which text field(s)** are matched — object name, short description, or both.

If the search text is empty, or a requested category has no matching text field selected, that category is skipped — the endpoint never errors on a missing filter, it just returns fewer results.

5.1.2. Request Parameters

Read from `$_REQUEST` — send them in the query string or as form-encoded POST fields, whichever suits the client.

A flag counts as **on** only when its value is the exact string `"1"`. Anything else — `"true"`, `"yes"`, `0`, or omitted — is **off**.

PARAMETER	REQUIRED	DESCRIPTION
q	required	The search text, matched as a `LIKE '%q%'` substring. Empty or missing after trimming → the endpoint returns `[]` immediately.
report	optional	Include **Reports**
storage	optional	Include **Storage tables**
workflow	optional	Include **Workflows**
glossary	optional	Include **Glossary** entries (excludes any flagged for deletion).
tasks	optional	Include the caller's own **Tasks**. Always scoped to the authenticated user.
columns	optional	Include **Storage columns**. Only takes effect together with `storage=1` — see below.
name	optional	Match `q` against the object's **name**.
shortDescription	optional	Match `q` against the object's **short description**.

5.1.3. Combining Flags

- `report` / `storage` / `workflow` pick which asset categories are searched and are OR'ed together — set several to search multiple categories in one call.
- `name` / `shortDescription` pick which text field(s) are matched, and apply to every category being searched. They're OR'ed too: setting both returns a hit on either field.
- If **none** of `report`, `storage`, `workflow` is set, the Assets/Columns search is skipped entirely, no matter what `q` is.
- If **neither** `name` nor `shortDescription` is set, Assets, Columns and Glossary all skip — none can build a text clause.
- **Tasks** only have a name column: a Tasks search runs only when `tasks=1` *and* `name=1`. `shortDescription=1` has no effect on Tasks.

- **Columns** run only when `storage=1` *and* `columns=1` *and* a text field is set. `columns=1` alone, without `storage=1`, returns nothing.
- Every category is capped at **30 results**, ordered by descending popularity (Tasks: descending rating). There is no pagination beyond that cap.

5.1.4. Access Control

Scoping rules:

- Non-superadmin users only see Assets and Storage columns belonging to a department they hold rights to. Superadmins see all of them.
- Tasks are always filtered to `assignedToUserId = <caller>` — no user, superadmin or not, sees another user's tasks through this endpoint.
- Glossary entries are global, not department-scoped.

5.1.5. Response Format

Always a JSON array. Every element shares one shape, whatever category it came from.

RESULT OBJECT SHAPE

```
{ "id": 42, "type": "Report" | "Storage" | "Workflow" | "Column" | "Glossary" |
"Task", "name": "…", "shortDescription": "…", "icon": "ressources/…", "url": "…" }
```

FIELD	NOTES
id	Internal id of the matched row — from `Assets`, `Columns`, `Glossary` or `Tasks`, depending on `type`.
type	One of `Report`, `Storage`, `Workflow`, `Column`, `Glossary`, `Task`.
name	Display name of the object.
shortDescription	Always `""` for `Task` results — Tasks have no description field.
icon	Relative SVG path for a UI to render. **Storage** results can carry either `ressources/file.svg` (category 100–119) or `ressources/database.svg` (category 120–199) — `type` alone doesn't tell you which; use the returned value as-is.
url	Relative link to open the object. **Column** results link to the parent Storage asset's table page, since a column has no page of its own.

5.1.6. Ordering & Empty Cases

Results are **not** ranked globally across categories — they're emitted section by section, in a fixed order, each internally sorted by descending popularity (or rating, for Tasks):

1. Assets — Reports, Storage, Workflows combined
2. Columns (if requested)
3. Glossary entries (if requested)
4. Tasks (if requested)

Returns `[]` when: `q` is empty or missing · no rows match in any requested category · no category flag is set at all. Authentication failures are handled upstream and are out of scope here.

5.1.7. Usage Examples

Assume an authenticated caller and Marvim reachable at `https://marvim.example.com/`.

5.1.7.1. Example 1 — Reports & Workflows, by name

```
q=sales&report=1&workflow=1&name=1
```

Matches Reports and Workflows whose name contains “sales”. Storage, Columns, Glossary and Tasks stay out of it.

```
REQUEST
GET /api_AdvSearch.php?q=sales&report=1&workflow=1&name=1
200 OK
[ {"id":12,"type":"Report","name":"Monthly Sales Summary",
"shortDescription":"Aggregated sales figures by region",
"icon":"ressources/reports.svg","url":"oneReport.php?idasset=12"},
{"id":87,"type":"Workflow","name":"Sales Data Refresh", "shortDescription":"Nightly
ETL feeding the sales datamart",
"icon":"ressources/anatella.svg","url":"oneWorkflow.php?idasset=87"} ]
```

5.1.7.2. Example 2 — Storage tables and their columns

```
q=customer&storage=1&columns=1&name=1&shortDescription=1
```

Matches Storage assets by name or description, plus any column across all visible tables — not only the matched table.

```
200 OK
[ {"id":34,"type":"Storage","name":"Customer Master", "shortDescription":"Golden
record of all customers",
"icon":"ressources/database.svg","url":"table.php?idasset=34"},
{"id":201,"type":"Column","name":"customer_id", "shortDescription":"Unique customer
identifier", "icon":"ressources/columns.svg","url":"table.php?idasset=34"} ]
```

5.1.7.3. Example 3 — Everything at once

```
q=invoice&report=1&storage=1&workflow=1&glossary=1&tasks=1c&olumns=1&name=1&shortDescripti
on=1
```

Every category, both text fields — equivalent to ticking every “search in” checkbox in an advanced-search UI.

5.1.7.4. Example 4 — Only my Tasks

```
q=review&tasks=1&name=1
```

`shortDescription=1` would be a no-op here, since Tasks have no description column.

```
200 OK
[ {"id":5,"type":"Task","name":"Review Q3 invoice reconciliation",
"shortDescription":"","icon":"ressources/tasks.svg","url":"oneTask.php?idasset=5"} ]
```

5.1.7.5. Example 5 — Glossary only

```
q=KPI&glossary=1&name=1&shortDescription=1
```

Returns Glossary entries — not flagged for deletion — whose name or definition mentions “KPI”.

5.1.7.6. Example 6 — cURL

```
curl -G "https://marvim.example.com/api_AdvSearch.php" \ --data-urlencode
"q=dashboard" \ --data-urlencode "report=1" \ --data-urlencode "workflow=1" \ --data-
urlencode "name=1" \ --data-urlencode "shortDescription=1" \ -H "Authorization: Bearer
<token>"
```

5.2. “Reports” Meta-data Injection

This endpoint “Bulk create and update” Reports and their KPIs from a single JSON payload

```
POST  <marvim_root>/marvim/api_report.php
```

`Request body: JSONResponse: application/jsonOnly POST is accepted – any other method → 405`

5.2.1. Purpose

`api_report.php` is Marvim's bulk **report ingestion** endpoint. A single call can create or update any number of Reports at once, along with the KPIs attached to each one.

It is designed to be called repeatedly — e.g. by a scheduled job that re-exports a catalog of reports from an external tool — so most of its behavior is about what happens on the *second* and later calls for a report that already exists: which fields get overwritten, and what happens to KPIs that were present before but are missing from this call's payload.

Nothing is written to the database until **every** entry in the payload has passed structural validation, department-existence checks and rights checks. A single bad entry anywhere in `data` fails the whole batch — none of it is applied, not just the offending entry.

5.2.2. Request Body

A single JSON object, sent as the raw POST body.

SHAPE

```
{ "mode": "create" | "update" | "add", "data": [ { …report entry… }, … ] }
```

FIELD	REQUIRED	DESCRIPTION
mode	required	One of `"create"`, `"update"`, `"add"`. Governs what happens to a report's existing KPIs that aren't listed in this call — see §3. Anything else → `400`.
data	required	Array of report entries to upsert, processed in order. Must be an array (can be empty).

Each entry in *`data[]`*

FIELD	REQUIRED	DESCRIPTION
category	required	Integer **0–99** (the Report category range). Out of range → `400`.
name	required	Non-empty report name. Combined with **`server`**, this is the report's identity — see §4.
server	required	Non-empty name of the Reporting server the report lives on. Matched **case-insensitively** against existing servers of type `Reporting`; auto-created if no match is found.
department	required	Department **name** (not an id), matched **case-sensitively** against an existing department. Unknown name → `400` — departments are never auto-created.
shortDescription	optional	Free text. When omitted on an *update* to an existing report, the stored value is left untouched.
longDescription	optional	Free text. Same left-untouched-when-absent rule as above.
schema	optional	Free text. Same left-untouched-when-absent rule as above.
kpi	optional	Array of KPI objects (below). **Absent is not the same as unchanged** — it's treated as an empty list, so every existing KPI on that report becomes subject to the removal/soft-delete rule for the active **`mode`**.

Each entry in `kpi[]`

FIELD	REQUIRED	DESCRIPTION
`name`	`required`	Non-empty. Identifies the KPI within its report — resubmitting the same name updates that KPI rather than creating a duplicate.
`description`	`optional`	Maps to `KPI.shortDescription`. Omitted → left untouched on an existing KPI, stored as `NULL` on a new one.

5.2.3. Mode Semantics

All three modes create and update the report itself identically. They differ only in what happens to a KPI that existed on the report before this call but is **not** present in this call's `kpi` list.

MODE	UNLISTED EXISTING KPI
`create`	**Deleted outright** from the KPI table.
`update`	**Kept**, but its `shortDescription` is prefixed with `"Deleted on yyyyMMdd "` — a soft delete. Calling again on an already-tagged KPI does not re-stamp it with a new date.
`add`	**Left completely untouched** and not counted as removed. This mode only ever adds — it never drops or tags anything.

In every mode, a KPI name that *is* present in the payload and already exists gets its `description` updated (when supplied); one that doesn't exist yet is inserted.

5.2.4. Identity & Auto-Creation

- A report is considered to **already exist** when **`name`** + the resolved server id both match an existing Asset. **`category`** and **`department`** are *not* part of that identity — resubmitting the same name+server with a different category or department simply overwrites those fields on the existing Asset.
- The same report **`name`** on two different servers is treated as two independent reports.
- **Servers are auto-created, departments are not.** A **`server`** name with no case-insensitive match among existing `Reporting`-type servers is silently created as a new, non-alias Reporting server owned by the caller. A **`department`** name with no match instead fails the whole batch with `400`.
- Server matching is scoped to `serverType = 'Reporting'`, so a name already used by a server of another type (e.g. a Files server) is not mistaken for it — a new Reporting server with that name is created alongside it.

Typo risk: because unknown servers are auto-created rather than rejected, a misspelled **`server`** value doesn't surface as an error — it quietly produces a new, effectively duplicate server. There's no equivalent safety net here that the department check provides.

5.2.5. Access Rules

Scoping rules:

- A superadmin caller may write to any department.
- A non-superadmin caller needs a rights value **≥ 4** in the target department (per `userDepartmentRights`); otherwise the whole batch is rejected with `403` before anything is written.
- This check runs per entry, for every entry, before any writes — one under-privileged department anywhere in the batch blocks the entire call.

5.2.6. Response

On success: HTTP `200` with one result object per entry in `data`, in the same order.

SUCCESS SHAPE

```
{ "status": "ok", "results": [ { "name": "…", "assetId": 123, "action": "created" |
"updated", "kpiAdded": 2, "kpiRemoved": 1 }, … ] }
```

FIELD	NOTES
name	Echo of the entry's **name**.
assetId	The Asset id — freshly inserted, or the pre-existing one that was matched.
action	`"created"` or `"updated"`, per the identity match in §4.
kpiAdded	Count of KPI names in this call's **kpi** list that did not already exist on the report and were inserted.
kpiRemoved	Count of pre-existing KPIs deleted (**create**) or soft-deleted (**update**). **Always `0` in add mode**, by design.

5.2.7. Errors

Every error short-circuits the whole request as `{"error": "…"}` with no partial `results`. Authentication failures are handled upstream and are out of scope here.

STATUS	MESSAGE	CAUSE
405	`Only POST is supported.`	Any method other than POST.
400	`Invalid or missing JSON body.`	Body isn't valid JSON, or doesn't decode to an array/object.
400	`mode must be "create", "update" or "add".`	**mode** missing or not one of the three values.
400	`"data" must be an array.`	**data** missing or not an array.
400	`data[i] is missing field "…".`	One of **category**/**name**/**server**/**department** missing (or empty, for name/server) on entry *i*.
400	`data[i].kpi must be an array.`	**kpi** present but not an array.
400	`data[i].kpi[j] must be an object with a "name" field.`	A KPI entry has no non-empty **name**.
400	`data[i].category must be a Report category (0-99).`	**category** outside the 0–99 range.
400	`Unknown department "…" (data[i]).`	No department with that exact (case-sensitive) name.
403	`No creator rights in department "…" (data[i]).`	Caller isn't superadmin and holds rights < 4 in that department.

5.2.8. Usage Examples

Assume an authenticated caller and Marvim reachable at `https://marvim.example.com/`.

5.2.8.1. Example 1 — Create a new report with two KPIs

`mode=create`

POST /API_REPORT.PHP

```
{ "mode": "create", "data": [{ "category": 1, "name": "myTestFrank", "server":
"Azure", "department": "Sales", "shortDescription": "simple basic report",
"longDescription": "this is a long description", "kpi":
[{"name":"aa","description":"xxx"}, {"name":"bb","description":"yyy"}] }] }
```

200 OK — FIRST CALL, "AZURE" SERVER DIDN'T EXIST YET

```
{ "status":"ok", "results": [{"name":"myTestFrank","assetId":501,"action":"created",
"kpiAdded":2,"kpiRemoved":0}] }
```

5.2.8.2. Example 2 — Re-sync the same report, dropping a KPI — mode comparison

Same `name` + `server` as above, resubmitted with only KPI `"aa"`. `"bb"` is now unlisted — this is where the three modes diverge.

`mode=create`

```
{"name":"myTestFrank","assetId":501,"action":"updated", "kpiAdded":0,"kpiRemoved":1}
// "bb" was deleted
```

`mode=update`

```
{"name":"myTestFrank","assetId":501,"action":"updated", "kpiAdded":0,"kpiRemoved":1}
// "bb" kept, description → "Deleted on 20260911 yyy"
```

`mode=add`

```
{"name":"myTestFrank","assetId":501,"action":"updated", "kpiAdded":0,"kpiRemoved":0}
// "bb" untouched, no new KPI supplied
```

5.2.8.3. Example 3 — Bulk: two reports, one call

```
{ "mode": "add", "data": [ {"category":1,"name":"Sales
Dashboard","server":"Azure","department":"Sales"}, {"category":40,"name":"HR
Attrition","server":"PowerBI-Prod","department":"HR", "kpi":
[{"name":"attritionRate"}]} ] }
```

If `"PowerBI-Prod"` has no matching Reporting server yet, it's created automatically and owned by the calling user — no separate provisioning call is needed.

5.2.8.4. Example 4 — Error: unknown department

```
{"mode":"create","data":[{"category":1,"name":"X",
"server":"Azure","department":"sales"}]} // lowercase "sales" ≠ "Sales"
{ "error": "Unknown department \"sales\" (data[0])." }
```

Department matching is case-sensitive — unlike `server`, which is matched case-insensitively.

5.2.8.5. Example 5 — cURL

```
curl -X POST "https://marvim.example.com/api_report.php" \ -H "Authorization: Bearer
<token>" \ -H "Content-Type: application/json" \ -d
'{"mode":"update","data":[{"category":1,"name":"myTestFrank",
"server":"Azure","department":"Sales","kpi":[{"name":"aa"}]}]}'
```

5.3. “Storage” Meta-data Injection

This endpoint “Bulk create and update” Storage Assets (files, databases, applications, APIs) and their Columns from a single JSON payload. The endpoint also allows to “Bulk update” the 2 most important governance KPI for the columns: the columns completeness and the columns cleanliness.

POST <marvim_root>/marvim/api_storage.php

`Request body: JSONResponse: application/jsonOnly POST is accepted – any other method → 405`

5.3.1. Purpose

`api_storage.php` is Marvim's bulk **storage ingestion** endpoint. A single call can create or update any number of Storage Assets — files, databases, applications, or APIs, depending on category — along with the Columns attached to each one.

Like `api_report.php`, it's meant to be called repeatedly by a crawler or scheduled export job, so most of what it does is define what happens to a table's existing columns when a later call doesn't mention them anymore.

Nothing is written to the database until **every** entry in the payload has passed structural validation, department-existence checks and rights checks. A single bad entry anywhere in `data` fails the whole batch — none of it is applied, not just the offending entry.

5.3.2. Request Body

A single JSON object, sent as the raw POST body.

SHAPE

```
{ "mode": "create" | "update" | "add", "data": [ { …table entry… }, … ] }
```

FIELD	REQUIRED	DESCRIPTION
mode	required	One of `"create"`, `"update"`, `"add"`. Governs what happens to a table's existing columns that aren't listed in this call — see §3. Anything else → `400`.
data	required	Array of table entries to upsert, processed in order. Must be an array (can be empty).

Each entry in `data[]`

FIELD	REQUIRED	DESCRIPTION
category	required	Integer **100–199** (the Storage category range). Also determines the server's type — see §4. Out of range → `400`.
schema	required	The table's schema/path (e.g. a folder path, a DB schema name). Part of the table's identity, alongside **`name`** and **`server`**.
name	required	The table/file name. **Only checked for presence, not emptiness** — unlike `api_report.php`, an empty string here is not rejected.
server	required	Name of the server the table lives on. Matched **case-insensitively** against existing servers of the type implied by **`category`**; auto-created if no match is found. Also only checked for presence, not emptiness.
department	required	Department **name** (not an id), matched **case-sensitively** against an existing department. Unknown name → `400` — departments are never auto-created.
columns	required	Array of column objects (below). **Must be present** — unlike `kpi` on `api_report.php`, omitting it entirely is a `400`, not an implicit empty list. It *can* be sent as `[]`, with the same effect: every existing column becomes subject to the mode's removal rule.
shortDescription	optional	Free text. When omitted on an update to an existing table, the stored value is left untouched.
longDescription	optional	Free text. Same left-untouched-when-absent rule as above.

Each entry in `columns[]`

FIELD	REQUIRED	DESCRIPTION
name	required	Non-empty. Identifies the column within its table — resubmitting the same name updates that column rather than creating a duplicate.
shortDescription	optional	Omitted → left untouched on an existing column, `NULL` on a new one.
completeness	optional	Numeric. Rejected with `400` if present but non-numeric.
cleanliness	optional	Numeric. Rejected with `400` if present but non-numeric.

On an existing column, an update only actually runs if at least one of **shortDescription**, **completeness** or **cleanliness** is present in the payload. Sending just `{"name":"age"}` for a column that already exists touches nothing on it — including its `dateUpdated`.

5.3.3. Mode Semantics

Identical logic to `api_report.php`'s KPI handling, applied to Columns instead. All three modes create and update the table itself identically; they differ only in what happens to a column that existed before this call but is **not** present in this call's `columns` list.

MODE	UNLISTED EXISTING COLUMN
create	**Deleted outright** from the Columns table.
update	**Kept**, but its `shortDescription` is prefixed with `"Deleted on yyyyMMdd "` — a soft delete. Calling again on an already-tagged column does not re-stamp it with a new date.
add	**Left completely untouched** and not counted as removed. This mode only ever adds — it never drops or tags anything.

5.3.4. Identity & Auto-Creation

- A table is considered to **already exist** when **name** + the resolved server id + **schema** *all three* match an existing Asset — one more component than `api_report.php`'s name+server match. **category** and **department** are not part of the identity and are simply overwritten on a match.
- The same **name**+**schema** pair on two different servers — or the same name+server with two different schemas — is treated as two independent tables.
- **A table's server is looked up by name AND type together**, and the type is derived from **category**, not sent explicitly:

CATEGORY RANGE	SERVER TYPE
100–119	Files
120–139	Data Bases
140–159	Applications
160–199	API's

- The name+type lookup (and any auto-creation) is **case-insensitive on both parts**. This means the same **server** name can legitimately resolve to *different* server records depending on the category of the entry using it — e.g. `"THUNDERBOLT"` used with a Files category and again with a Data Bases category resolves to two distinct servers, auto-creating the second the first time it's referenced.
- A server auto-created here is a real, physical server (not an alias) owned by the calling user.

Typo / mismatch risk: as with `api_report.php`, an unrecognized **server** name is silently auto-created rather than rejected — and here a category that implies the wrong server type has the same silent effect, quietly spinning up a same-named server of a different type instead of erroring.

5.3.5. Access Rules

Scoping rules:

- A superadmin caller may write to any department.
- A non-superadmin caller needs a rights value **≥ 4** in the target department (per `userDepartmentRights`); otherwise the whole batch is rejected with `403` before anything is written.
- This check runs per entry, for every entry, before any writes — one under-privileged department anywhere in the batch blocks the entire call.

5.3.6. Response

On success: HTTP `200` with one result object per entry in `data`, in the same order.

SUCCESS SHAPE

```
{ "status": "ok", "results": [ { "name": "…", "assetId": 123, "action": "created" | "updated", "columnsAdded": 3, "columnsRemoved": 1 }, … ] }
```

FIELD	NOTES
`name`	Echo of the entry's **`name`**.
`assetId`	The Asset id — freshly inserted, or the pre-existing one that was matched.
`action`	`"created"` or `"updated"`, per the identity match in §4.
`columnsAdded`	Count of column names in this call's **`columns`** list that did not already exist on the table and were inserted.
`columnsRemoved`	Count of pre-existing columns deleted (**`create`**) or soft-deleted (**`update`**). **Always `0` in `add` mode**, by design.

5.3.7. Errors

Every error short-circuits the whole request as `{"error": "…"}` with no partial `results`. Authentication failures are handled upstream and are out of scope here.

STATUS	MESSAGE	CAUSE
`405`	`Only POST is supported.`	Any method other than POST.
`400`	`Invalid or missing JSON body.`	Body isn't valid JSON, or doesn't decode to an array/object.
`400`	`mode must be "create", "update" or "add".`	**`mode`** missing or not one of the three values.
`400`	`"data" must be an array.`	**`data`** missing or not an array.
`400`	`data[i] is missing field "…".`	One of **`category`**/**`schema`**/**`name`**/**`server`**/**`department`**/**`columns`** missing on entry *i*.
`400`	`data[i].columns must be an array.`	**`columns`** present but not an array.
`400`	`data[i].columns[j] must be an object with a "name" field.`	A column entry has no non-empty **`name`**.
`400`	`data[i].columns[j].completeness must be numeric.`	**`completeness`** present but non-numeric.
`400`	`data[i].columns[j].cleanliness must be numeric.`	**`cleanliness`** present but non-numeric.
`400`	`data[i].category must be a Storage category (100-199).`	**`category`** outside the 100–199 range.
`400`	`Unknown department "…" (data[i]).`	No department with that exact (case-sensitive) name.

403	No creator rights in department "…" (data[i]).	Caller isn't superadmin and holds rights < 4 in that department.

5.3.8. Usage Examples

Assume an authenticated caller and Marvim reachable at https://marvim.example.com/.

5.3.8.1. Example 1 — Create a new file-based table with columns

`mode=create`

```
POST /API_STORAGE.PHP
{ "mode": "create", "data": [{ "category": 102, // 100-119 -> server type "Files"
"schema": "F:/", "name": "census-income.zip", "server": "THUNDERBOLT", "department":
"Sales", "columns": [ {"name":"age", "completeness":0.98}, {"name":"class of worker",
"shortDescription":"employment category"} ] }] }
```

```
200 OK – FIRST CALL, "THUNDERBOLT" (FILES) DIDN'T EXIST YET
{ "status":"ok", "results": [{"name":"census-
income.zip","assetId":812,"action":"created", "columnsAdded":2,"columnsRemoved":0}] }
```

5.3.8.2. Example 2 — Re-sync the same table, dropping a column — mode comparison

Same `name`+`server`+`schema` as above, resubmitted with only `"age"`. `"class of worker"` is now unlisted.

`mode=create`

```
{"name":"census-income.zip","assetId":812,"action":"updated",
"columnsAdded":0,"columnsRemoved":1} // "class of worker" deleted
```

`mode=update`

```
{"name":"census-income.zip","assetId":812,"action":"updated",
"columnsAdded":0,"columnsRemoved":1} // kept, description → "Deleted on 20260911
employment category"
```

`mode=add`

```
{"name":"census-income.zip","assetId":812,"action":"updated",
"columnsAdded":0,"columnsRemoved":0} // "class of worker" untouched
```

5.3.8.3. Example 3 — Same server name, two categories — two servers

```
{ "mode": "add", "data": [ {"category":105, "schema":"F:/exports/",
"name":"leads.csv", "server":"THUNDERBOLT", "department":"Sales", "columns": []},
{"category":125, "schema":"dbo", "name":"Leads", "server":"THUNDERBOLT",
"department":"Sales", "columns": []} ] }
```

Category 105 resolves to server type `Files`, category 125 to `Data Bases` — so `"THUNDERBOLT"` is looked up (and, if needed, created) once per type. If only a Files "THUNDERBOLT" existed before this call, the second entry auto-creates a *second*, distinct "THUNDERBOLT" server of type Data Bases.

5.3.8.4. Example 4 — Error: columns field omitted

```
{"mode":"create","data":[{"category":110,"schema":"F:/",
"name":"data.csv","server":"THUNDERBOLT","department":"Sales"}]}
{ "error": "data[0] is missing field \"columns\"." }
```

Unlike `api_report.php`'s `kpi`, `columns` can't be left out — send `"columns": []` to mean "no columns".

5.3.8.5. Example 5 — cURL

```
curl -X POST "https://marvim.example.com/api_storage.php" \ -H "Authorization: Bearer
<token>" \ -H "Content-Type: application/json" \ -d
'{"mode":"update","data":[{"category":102,"schema":"F:/", "name":"census-
income.zip","server":"THUNDERBOLT","department":"Sales",
"columns":[{"name":"age","cleanliness":0.9}]}]}'
```

5.4. "Workflows" Meta-data Injection

This endpoint "Bulk create and update" Workflow-Assets — things like Anatella pipelines. For each workflow, you also tell Marvim which existing Assets are its inputs and which ones are its outputs. You can optionally send to Marvim the workflow's pipeline file.

This is the most complex Marvim's ingestion endpoints: it validates in two passes, resolves every input/output reference against Assets created elsewhere. It only ever links to Assets — it never creates them.

POST `<marvim_root>/marvim/api_workflow.php`

`Request body: JSONResponse: application/jsonOnly POST is accepted — any other method → 405Only 2 modes: create / update — no "add"`

5.4.1. Purpose

`api_workflow.php` is Marvim's bulk **workflow ingestion** endpoint. A single call can create or update any number of Workflow Assets — e.g. Anatella pipelines — and declare which Assets flow into each one and which flow out, via rows in the `workflowIO` join table.

It is the most involved of the three ingestion endpoints (`api_report.php`, `api_storage.php`, and this one) for three reasons:

- it never creates the Assets it links — every input/output must already exist, resolved by name+server+schema against rows typically created via `api_storage.php`;
- it optionally accepts and stores the pipeline's own source file (base64-encoded), writing it to disk and recording it in a separate metadata table.

Like the other two ingestion endpoints, nothing is written until the whole batch clears validation — but here that validation includes resolving *every* input/output reference to a real Asset id first. See §7.

5.4.2. Request Body

A single JSON object, sent as the raw POST body.

SHAPE

```
{ "mode": "create" | "update", "data": [ { …workflow entry… }, … ] }
```

FIELD	REQUIRED	DESCRIPTION

mode	required	One of `"create"` or `"update"` only — **there is no `"add"` mode here**, unlike the other two ingestion endpoints. Anything else → `400`.
data	required	Array of workflow entries to upsert, processed in order. Must be an array (can be empty).

Each entry in `data[]`

FIELD	REQUIRED	DESCRIPTION
category	required	Integer **≥ 200** (the Workflow category range — open-ended, unlike Storage's capped 100–199).
schema	required	The workflow's own schema/path. Part of its identity, alongside **name** and **server**.
name	required	The workflow's name (e.g. a pipeline file name). Only checked for presence, not emptiness.
server	required	Name of the server the workflow itself runs on. Auto-created as a `Workflow`-type server if no match exists — see §4. This is *not* the same server namespace used to resolve **IO** entries.
department	required	Department **name** (not an id), matched **case-sensitively**. Unknown name → `400` — never auto-created.
IO	required	Array of input/output link objects (below). Must be present — can be `[]`.
shortDescription	optional	Free text. Left untouched in the DB when omitted on an update.
longDescription	optional	Free text. Same left-untouched-when-absent rule.
file	optional	Base64-encoded content of the workflow's `.anatella` pipeline file. See §6.

Each entry in `IO[]`

FIELD	REQUIRED	DESCRIPTION
name	required	Non-empty. Name of the **existing** Asset to link.
schema	required	Schema/path of that existing Asset. Only checked for presence, not emptiness.
server	required	Non-empty. Name of the server that Asset lives on. Must already exist — see §5.
direction	required	Must loosely equal `1` or `0`. `1` = this Asset is an **input** to the workflow; `0` = it's an **output**. Anything else → `400`.

Diagram: Inputs (direction=1, e.g. `census-income.zip`) → Workflow (`csv.anatella`) → Outputs (direction=0, e.g. `census-income-clean.csv`).

This is NOT a creation endpoint for IO Assets. Every Asset referenced in **IO** must already exist — resolved by **name+server+schema** against rows created elsewhere (typically via `api_storage.php`). This endpoint only ever *links* to Assets; a reference that doesn't resolve fails the whole batch rather than creating a placeholder.

5.4.3. Mode Semantics

Only two modes, and — unlike `api_report.php` / `api_storage.php` — the "kept" behavior has no soft-delete tagging, because a `workflowIO` row has nothing to tag: it's a bare join-table row (`idWorkflow`, `idIO`, `isInput`), no description field of its own.

MODE	UNLISTED EXISTING IO LINK
create	**Deleted outright** from `workflowIO`.
update	**Left exactly as is** — no tagging, no marker, literally untouched.

An IO link's identity within a workflow is the pair `(resolved Asset id, direction)` — the same Asset can be linked as *both* an input and an output of the same workflow; that's two distinct links, not a conflict.

5.4.4. Identity & the Workflow's Own Server

- A workflow is considered to **already exist** when **`name`** + the resolved server id + **`schema`** all match an existing Asset — the same three-part identity as `api_storage.php`. **`category`** and **`department`** aren't part of it and are simply overwritten on a match.
- The workflow's own **`server`** is looked up (and auto-created if missing) among servers of type `Workflow` specifically, keyed case-insensitively by name. A server auto-created here is real and physical (not an alias), owned by the calling user.
- This is a **separate namespace** from the one used to resolve **`IO`** server references (next section) — a name can exist as a `Workflow` server and, independently, as a `Files` or `Data Bases` server with the same name, without collision.

5.4.5. Resolving IO Links

Turning an **`IO`** entry into a concrete `workflowIO` row means resolving **`name`**+**`server`**+**`schema`** to an existing Asset id. This is the most involved step in the endpoint.

1. All servers are preloaded, split into two caches: one for `Workflow`-type servers (used for §4), and one for **everything else**, keyed by lowercased name → a *list* of ids. Excluding Workflow-type servers here means a workflow server sharing a name with a Storage server can't shadow it — and Storage Assets never live on a Workflow server anyway.
2. A name can map to more than one id in that second cache — the same physical machine is often registered once per role (e.g. `"THUNDERBOLT"` as both a `Files` server and a `Data Bases` server). **`io.server`** not matching *any* id → `400 Unknown server "…"`.
3. For each candidate server id (in the order the servers were originally loaded), the endpoint looks for an Asset with that exact **`name`** + **`schema`**. The **first match wins**; the rest are never tried.
4. No candidate matches → `400`, naming the endpoint to register the Asset with first (`api_storage.php`).

5.4.6. Pipeline File Upload

An entry may optionally carry the workflow's own pipeline source as **`file`** — the base64-encoded bytes of an `.anatella` file.

Validated up front: must be a non-empty string, and must decode as valid base64 (strict mode — malformed input is rejected outright rather than silently truncated) or the whole batch fails with `400`.

The success response never confirms the file was stored— a `200` with no error means it worked.

A directory-creation or write failure at this stage returns `500` (`Could not create directory "…"` / `Could not write pipeline file for data[i]`) — the only two `500`s this endpoint can produce, and both happen mid-batch, after some entries may already have been written.

5.4.7. Validation Order

Everything below runs to completion, across the *entire* batch, before a single row is written. A failure anywhere aborts the whole request with nothing applied — with one exception, noted below.

1. **Structural pass** (no database access): every entry's required fields, every **`IO`** sub-object, the category range, and any **`file`** base64 payload are checked and decoded.
2. **Resolution pass** (read-only database connection): for every entry, in order — department exists and the caller has rights ≥ 4 there (unless superadmin); then every **`IO`** entry's server and Asset are resolved to concrete ids, per §5.
3. **Write pass** (read-write connection): only now does anything get created or modified — the workflow's own server if missing, the workflow Asset itself, the pipeline file and `WorkflowMeta` row if supplied, and the `workflowIO` rows.

The one place this can partially apply: a `500` from a file-system failure in §6 happens *during* the write pass, after earlier entries in the same batch may already have been committed. Every `400`/`403`, by contrast, is caught before the write pass starts, so those always leave the database untouched.

5.4.8. Access Rules

Scoping rules:

- A superadmin caller may write to any department.
- A non-superadmin caller needs a rights value **≥ 4** in the workflow's target department; otherwise the whole batch is rejected with `403` before anything is written.

5.4.9. Response

On success: HTTP `200` with one result object per entry in `data`, in the same order.

SUCCESS SHAPE

```
{ "status": "ok", "results": [ { "name": "…", "assetId": 123, "action": "created" |
"updated", "ioAdded": 2, "ioRemoved": 1 }, … ] }
```

FIELD	NOTES
`name`	Echo of the entry's **`name`**.
`assetId`	The workflow's Asset id — freshly inserted, or the pre-existing one that was matched.
`action`	`"created"` or `"updated"`, per the identity match in §4.
`ioAdded`	Count of (Asset, direction) pairs in this call's **`IO`** list that didn't already exist and were inserted.
`ioRemoved`	Count of pre-existing links deleted. **Always `0` in `update` mode** — nothing is ever removed there.

There is no field reporting whether **`file`** was supplied or stored — see §6.

5.4.10. Errors

Every error short-circuits the request as `{"error": "…"}` with no partial `results` — except the two `500`s, which can follow partial writes (§7). Authentication failures are handled upstream and are out of scope here.

STATUS	MESSAGE	CAUSE
`405`	`Only POST is supported.`	Any method other than POST.
`400`	`Invalid or missing JSON body.`	Body isn't valid JSON, or doesn't decode to an array/object.
`400`	`mode must be "create" or "update".`	**`mode`** missing or not one of those two values.
`400`	`"data" must be an array.`	**`data`** missing or not an array.

400	data[i] is missing field "…".	One of **category/schema/name/server/department/IO** missing on entry *i*.
400	data[i].IO must be an array.	**IO** present but not an array.
400	data[i].IO[j] must be an object.	An IO entry isn't a JSON object.
400	data[i].IO[j] is missing field "schema".	**io.schema** absent.
400	data[i].IO[j] is missing field "name".	**io.name** absent or empty.
400	data[i].IO[j] is missing field "server".	**io.server** absent or empty.
400	data[i].IO[j].direction must be 1 (input) or 0 (output).	**io.direction** absent or not loosely 0/1.
400	data[i].category must be a Workflow category (>=200).	**category** below 200.
400	data[i].file must be a non-empty base64-encoded string.	**file** present but not a non-empty string.
400	data[i].file is not valid base64.	**file** fails strict base64 decoding.
400	Unknown department "…" (data[i]).	No department with that exact (case-sensitive) name.
403	No creator rights in department "…" (data[i]).	Caller isn't superadmin and holds rights < 4 there.
400	Unknown server "…" referenced by data[i].IO[j].	**io.server** matches no non-Workflow server.
400	data[i].IO[j] references an unknown Asset (…). Register it first, e.g. via api_storage.php.	No Asset matches **io.name**+**io.server**+**io.schema** under any candidate server id.
500	Could not create directory "…" (data[i]).	The lineage/anatella directory doesn't exist and can't be created.
500	Could not write pipeline file for data[i].	Writing the decoded **file** bytes to disk failed.

5.4.11. Usage Examples

Assume an authenticated caller, Marvim reachable at https://marvim.example.com/, and that census-income.zip and census-income-clean.csv were already registered on THUNDERBOLT via api_storage.php.

5.4.11.1. Example 1 — Create a workflow with one input and one output

mode=create

```
POST /API_WORKFLOW.PHP
{ "mode": "create", "data": [{ "category": 200, "schema":
"F:/TIMi/marvim/graphsToSendTomarvim/", "name": "csv.anatella", "server":
"THUNDERBOLT", "department": "Sales", "IO": [ {"name":"census-income.zip",
"schema":"F:/", "server":"THUNDERBOLT", "direction":1}, {"name":"census-income-
clean.csv", "schema":"F:/", "server":"THUNDERBOLT", "direction":0} ] }] }

200 OK
{ "status":"ok", "results": [{"name":"csv.anatella","assetId":930,"action":"created",
"ioAdded":2,"ioRemoved":0}] }
```

5.4.11.2. Example 2 — Re-sync, dropping the output — create vs. update

Same workflow, resubmitted with only the input link. The output link (`census-income-clean.csv`, direction 0) is now unlisted.

`mode=create`

```
{"name":"csv.anatella","assetId":930,"action":"updated", "ioAdded":0,"ioRemoved":1} // output link deleted
```

`mode=update`

```
{"name":"csv.anatella","assetId":930,"action":"updated", "ioAdded":0,"ioRemoved":0} // output link left exactly as is
```

5.4.11.3. Example 3 — Same Asset as both input and output

Legitimate — a workflow that reads and rewrites the same table in place, expressed as two distinct links.

```
"IO": [ {"name":"customers.parquet", "schema":"dbo", "server":"THUNDERBOLT", "direction":1}, {"name":"customers.parquet", "schema":"dbo", "server":"THUNDERBOLT", "direction":0} ]
```

5.4.11.4. Example 4 — Attach the pipeline file

BASH — BUILD THE PAYLOAD

```
FILE_B64=$(base64 -w0 csv.anatella) curl -X POST "https://marvim.example.com/api_workflow.php" \ -H "Authorization: Bearer <token>" \ -H "Content-Type: application/json" \ -d "{\"mode\":\"update\",\"data\":[{\"category\":200,\"schema\":\"F:/TIMi/marvim/graphsToSendTomarvim/\", \"name\":\"csv.anatella\",\"server\":\"THUNDERBOLT\",\"department\":\"Sales\", \"IO\":[],\"file\":\"$FILE_B64\"}]}"
```

Sending `"IO": []` here under `mode=update` leaves existing links untouched (§3) while still replacing the stored pipeline file (§6).

5.4.11.5. Example 5 — Error: IO Asset not registered yet

```
{"mode":"create","data":[{"category":200,"schema":"F:/pipelines/", "name":"new.anatella","server":"THUNDERBOLT","department":"Sales", "IO":[{"name":"not-yet-registered.csv","schema":"F:/", "server":"THUNDERBOLT","direction":1}]}]}
{ "error": "data[0].IO[0] references an unknown Asset (name=\"not-yet-registered.csv\", server=\"THUNDERBOLT\", schema=\"F:/\"). Register it first, e.g. via api_storage.php." }
```