Ref. Ares(2020)4112626 - 05/08/2020

FUDIPO: Future Directions of Production Planning and Optimized Energy and Process Industries

Project type:	Innovation action		
Start date of project:	01/10/2016	Duration:	48 months

Deliverable 6.3 – A web-based toolbox

WP n° and title:	WP6 - A web-based toolbox
Due date:	31/07/2020
Actual submission date:	Day/Month/Year
Responsible Author(s):	Andreas Roither-Voigt, David Zelenay, Aleksandar Vangjelovski
Contributor(s):	Tieto Austria GmbH
Technical Approver(s):	Andreas Roither-Voigt
Comments:	Documentation of open-source web-based toolbox
Dissemination level:	PU
Distribution list:	
Document identifier:	D6.3 – A web-based toolbox

This project is funded by the European Commission H2020 Research and Innovation program under Grant Agreement nº 723523.

REVISION TABLE/ APPROVAL STATUS

Issue	Date	Modifications	Approved by (date)
V1.0	31/07/2020	Final description of open-source web-based toolbox project on GitLab	Andreas Roither-Voigt

GLOSSARY

Abbreviation	Definition
EC	European Commission
IT	Information Technology
OT	Operational Technology
MCR	Matlab Compiler Runtime
CSS	Cascading Style Sheets
HTTP	Hypertext Preprocessor
REST	Restful API
UI	User Interface
FMU	Functional Mockup Unit
FMI	Functional Mockup Interface - Standard
OS	Operating System
SSL	Secure Sockets Layer
JSON	JavaScript Object Notation

TABLE OF CONTENTS

TABLE OF FIGURES

1. EXECUTIVE SUMMARY

This document includes the documentation of the FUDIPO web-based toolbox open source project. The web-based toolbox contains an example models (for calculating the heating value of large CHPs and Hugin model wrapper template) and a calculation graph (Node-RED) to automatically execute the model. This Toolbox serves as an example on how to integrate and deploy heterogeneous models with varying model runtime requirements in one scale able unified solution. Encapsulating models in RESTful APIs (Flask apps) and defining a input and output data structures enables the possibility to execute different models using Node-RED. Node-RED is used as a calculation graph, which means it handles the model input data selection and preparation, model execution, model output data postprocessing and storage to a database and data visualization using Node-RED's dashboard.

The Open-Source project (MIT license) can be found on https://gitlab.com/fudipoh2020/fudipo-toolbox

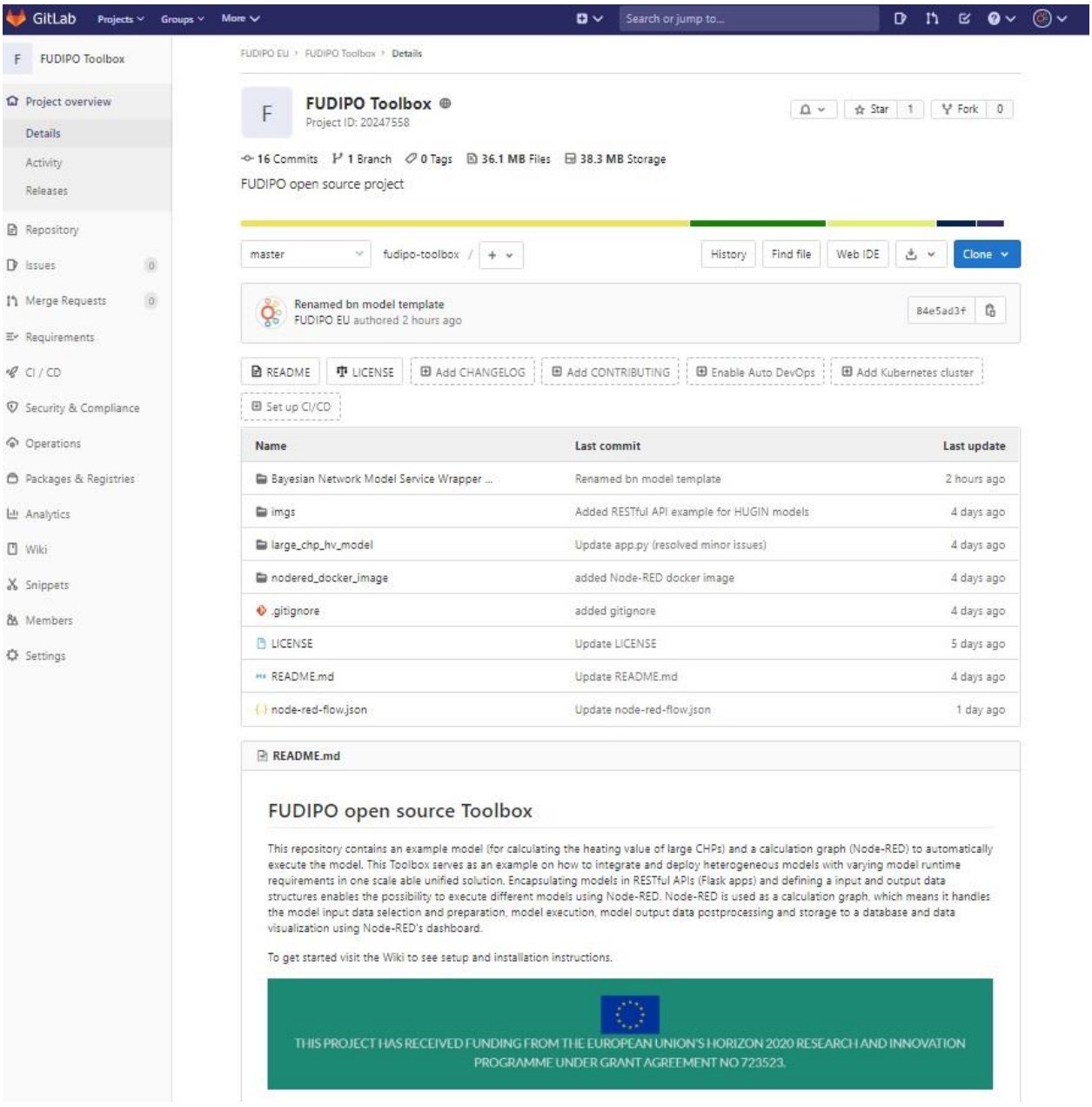

Figure 1, Screenshot of the GitLab repository

2. PRODUCT META-INFORMATION

Product name: FUDIPO platform / FUDIPO generic toolbox

Version number: 1.0

Source code and Docker container build instructions have been released on GitLab as an Open-Source project for open usage and community activity: https://gitlab.com/fudipoh2020/fudipo-toolbox

Released as commercial solution which includes service and support:

Please contact Tieto Austria GmbH for further information. Contact person: Mr. Richard Reisinger, richard.reisinger@tieto.com, Handelskai 94-96, Millennium Tower, 33rd floor, 1200 Vienna, Austria

3. FEATURES

The FUDIPO platform/toolbox provides the following main features:

- Real-time and near-real-time execution of process models for prediction and optimization in process industry plant operation
- Data-integration of input data for prediction models
- Graphical user interface for connecting data and models in an easy to use graphical programming and configuration environment
- Web-service wrapping templates and tools to run process models developed with many different supported modelling environments
- FMU – Functional Mockup Unit (FMI standard conforms dynamic models) integration framework for real-time utilization for predictions and combined usage with model-predictive-control
- Model services are used in the FUDIPO platform to call models using REST-API services which can be deployed to webservers and called easily from any software system that supports http protocol (which is practically every programming language and modern software development environment)
- Web-based graphical user-interface for editing the calculation steps and model execution structures and model result visualization (dashboards).

4. SETUP RUNTIME ENVIRONMENT

This section will cover installing and building the runtime environments required for the web toolbox. The Node-RED and the model's images can be downloaded and build with Docker in order to reduce installation complexity. Also, other plugins and configurations are already preconfigured for these images. It is only needed to have Docker installed: https://www.docker.com/products/docker-desktop

4.1 BUILDING MODEL RUNTIME DOCKER IMAGE

To be able to run the model the first step is to build the Docker image from source. To do this open the folder containing the Dockerfile of the large_chp_hv_model in a terminal. To build the docker container execute the command: `docker build -t large_chp_hv_model .` Once the build process has finished the image can be run. Please use the run command from "Creating a custom Docker network" if you want to execute the model from Node-RED, if the model is used alone without Node-RED run the docker image with the following command: `docker run -it -p 5000:5000 large_chp_hv_model` For more information and additional arguments look at the Docker documentation for building or the run command. Now the swagger UI of the API should be accessible on http://localhost:5000/api/doc.

As an addition to "large_chp_hv_model" there is also an Bayesian Network model service wrapper template included. This is an ASP.NET C# project as example on how to integrate Hugin Bayesian network models into the FUDIPO Platform using the C# API from Hugin. Please note that this project does not contain the model itself. A Hugin model (a .net file) and the Hugin API is required to successfully run this API. This project acts as an example on how to integrate such models into RESTful API using C#.

For further references please consult the API documentation from Hugin.

4.2 BUILDING NODE-RED DOCKER IMAGE

Similar to running the model image is also running the Node-Red Docker image. Before running the image Node-RED configuration can be customized by editing the "settings.js" file that can be found in "nodered_docker_image" directory. One common change is adding authentication to Node-Red for which mode information can be found on: https://nodered.org/docs/user-guide/runtime/securing-node-red. First the folder containing the Dockerfile of the Node-Red docker image need to be open in a terminal. To build the custom version of Node-RED's docker image execute the command: `docker build --tag=node-red-fudipo .` In order to run Node-RED use the run command from "Creating a custom Docker network" if Node-RED will be used to execute the large_chp_hv_model (running in Docker). If not and only Node-RED as a standalone version is required (without models

running in Docker) this command can be used to run the Node-RED image: `docker run -it -p 1880:1880 node-red-fudipo`

4.3 RUNNING A NODE-RED DOCKER CONTAINER (WITH DOCKER NETWORKING)

To execute the model from Node-RED both the model container and the Node-RED container need to be in the same Docker network. To create a new Docker network execute the following command: `docker network create --subnet=172.18.0.0/16 fudipo-net` To assign an IP address and a network to a container execute the run command of Docker with the following arguments: `docker run -p 5000:5000 -it --net fudipo-net --ip 172.18.0.3 large_chp_hv_model` This step should be done also for the Node-RED image by run command of Docker with the following arguments: `docker run -p 1880:1880 -it --net fudipo-net --ip 172.18.0.2 node-red-fudipo` If the Node-RED container is also in the same Docker network (fudipo-net) the API is reachable on the following url: http://172.18.0.3:5000/api/hello from Node-RED. Now Node-RED web interface should be accessible http://localhost:1880/

5. NODE-RED FLOW INSTALLATION AND MAINTENANCE

5.1 IMPORTING THE FLOW

Once node-red is up and running, the flow for the application need to be imported. The flow (node-red-flow.json) is stored in the root location of this repository.

- From the main configuration menu on the right to choose "Import"

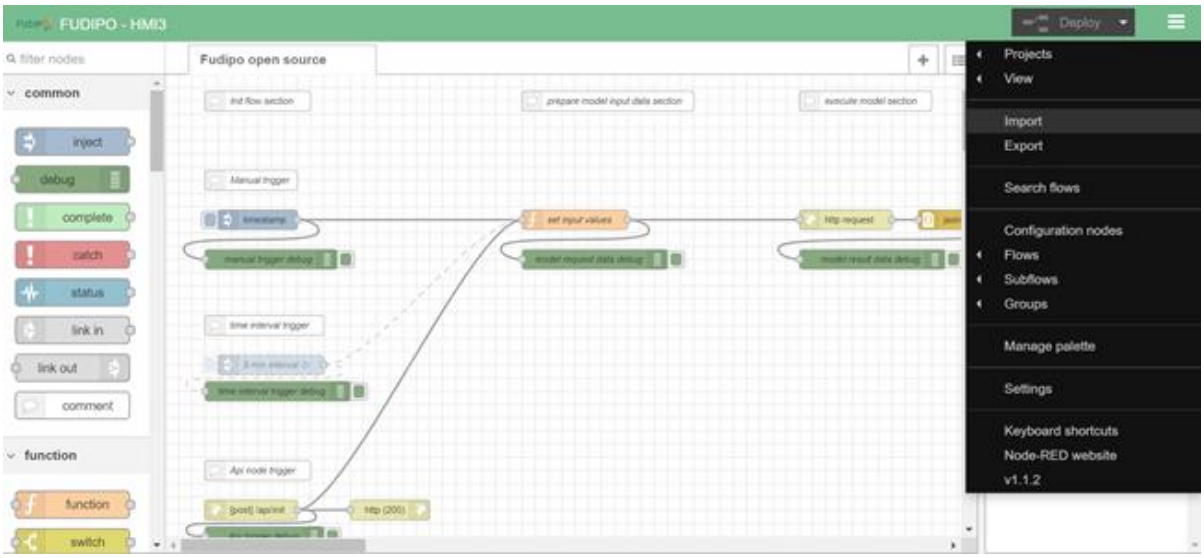

Figure 2, Node-RED's hamburger menu

- In the "Import nodes" dialog a new flow can be import from file or directly using the flow content in json format.

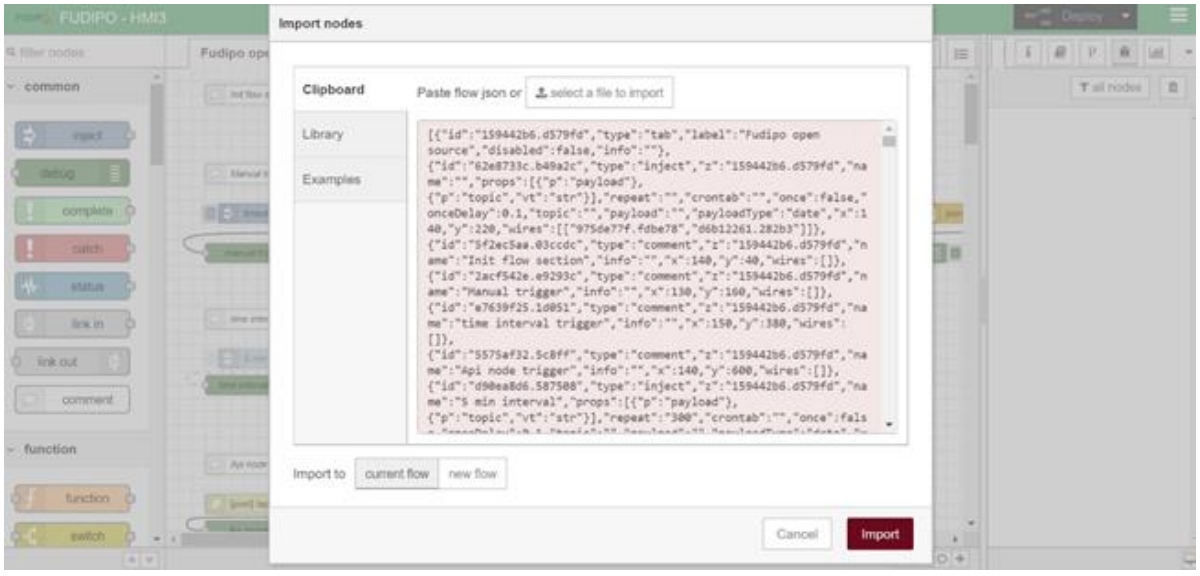

Figure 3, Node-RED's Import dialog window

5.2 FLOW CONFIGURATION

The flow is split in multiple sections in order to provide better usability and maintenance.

5.2.1 INITIALIZATION FLOW SECTION

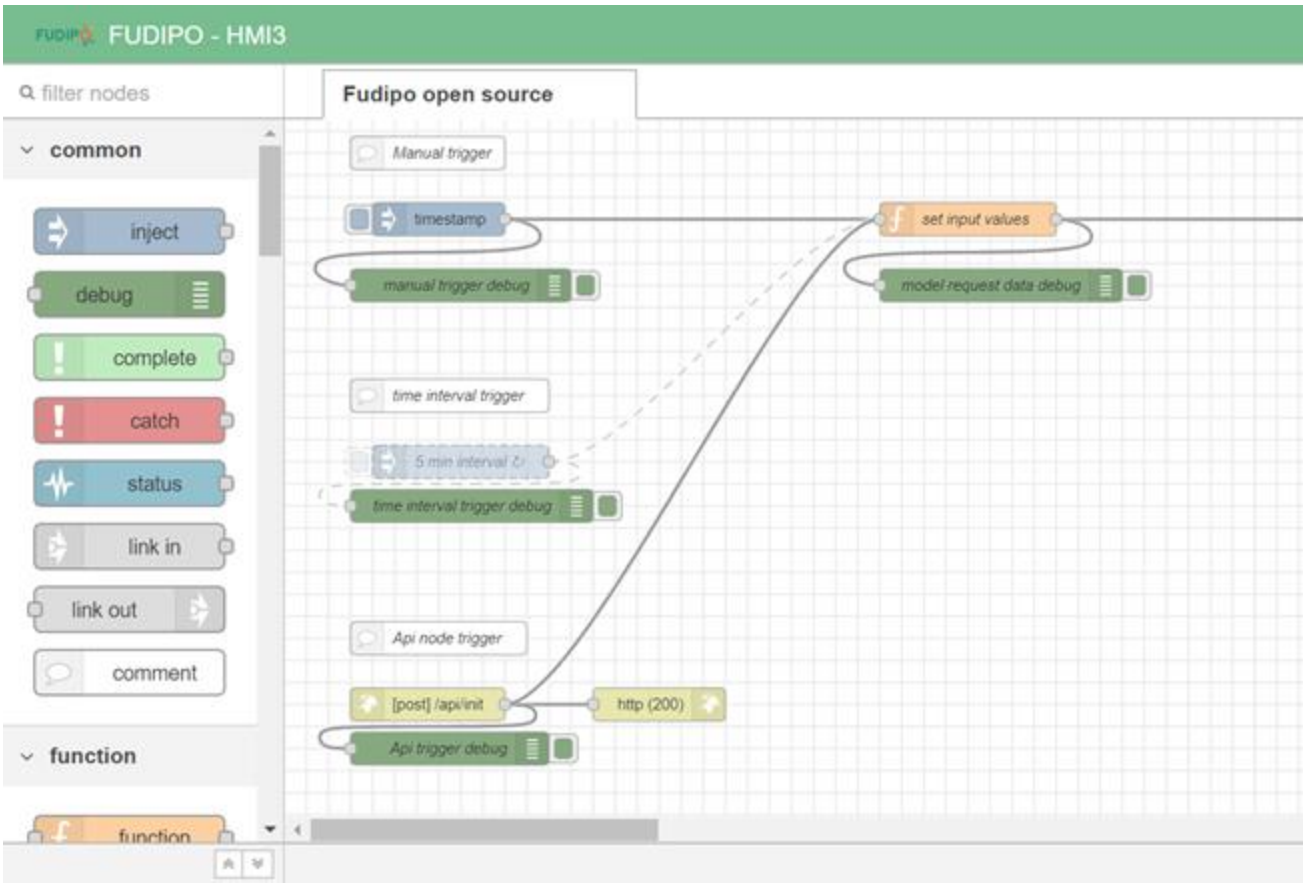

Figure 4, Node-RED's calculation graph

This section is used to start the flow and 3 diverse ways of trigger are implemented.

1. Manual initialization - only when inject node is started
2. Time interval initialization - defined time interval config in the node properties (need to be enabled first)
3. Node-red as API - HTTP post method node started when request to defined URL is sent. Used to start the flow from external source.

5.2.2 PREPARE MODEL INPUT DATA SECTION

In the function node the logic for required to call the model is implemented. In msg.url need to be set the URL for the model. Parameter msg.headers holds all headers tags to perform the request and the body of the request is stored in msg.payload. All the logic for preparing the request can be done in this function also.

5.2.3 EXECUTE MODEL SECTION

HTTP request node is node used for executing diverse types of request. The type can be defined in the node itself or as input parameter from the msg object. After execution of the model the response data is formatted to json object using json node. Adding debug nodes between steps is good practice in order to flow the data.

5.2.4 FORMAT OUTPUT SECTION

This section is used to extract important data revived from the model and to pass it to the output. In our case for the model will be important to extract the heating value from the array msg.payload["results"]["HV"]. The last item from this array is the value that need to be present on the dashboard. The code for extracting is shown on the image below.

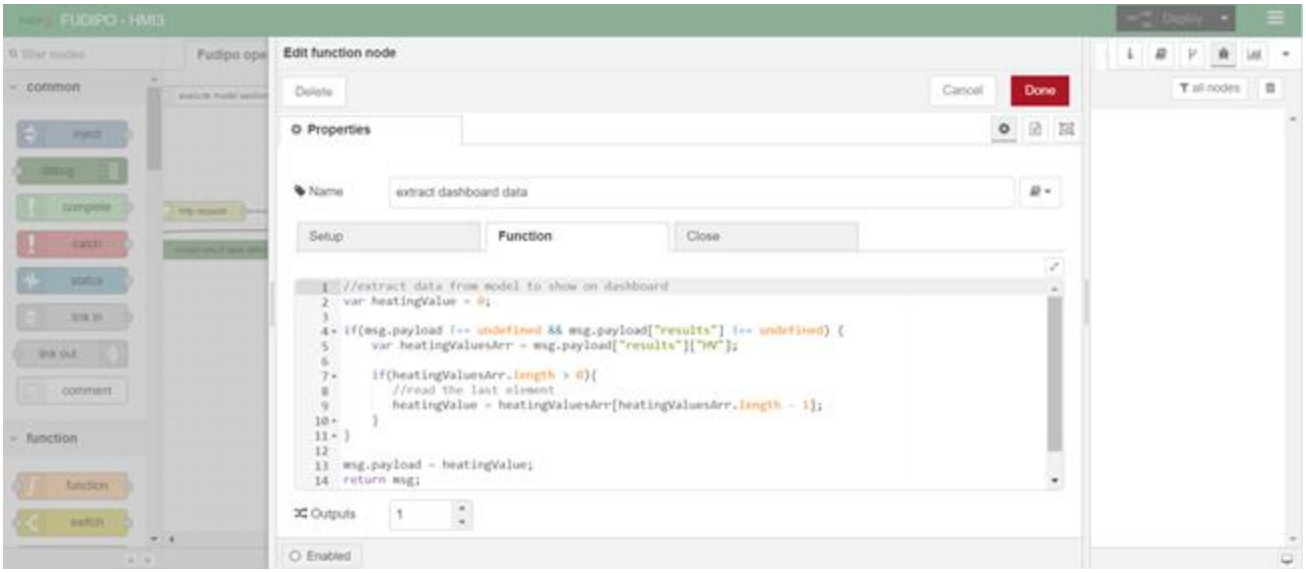

Figure 5, Node-RED's JavaScript function editor

5.2.5 GENERATE DASHBOARD SECTION

This section requires additional plugin node-red-dashboard in Node-RED to be installed. This is already done for the current image deployed on repository. In the left panel nodes for dashboard are added and can be used for creating custom dashboard. An example is shown using Gauge node. Configuration of the node is shown on the image below.

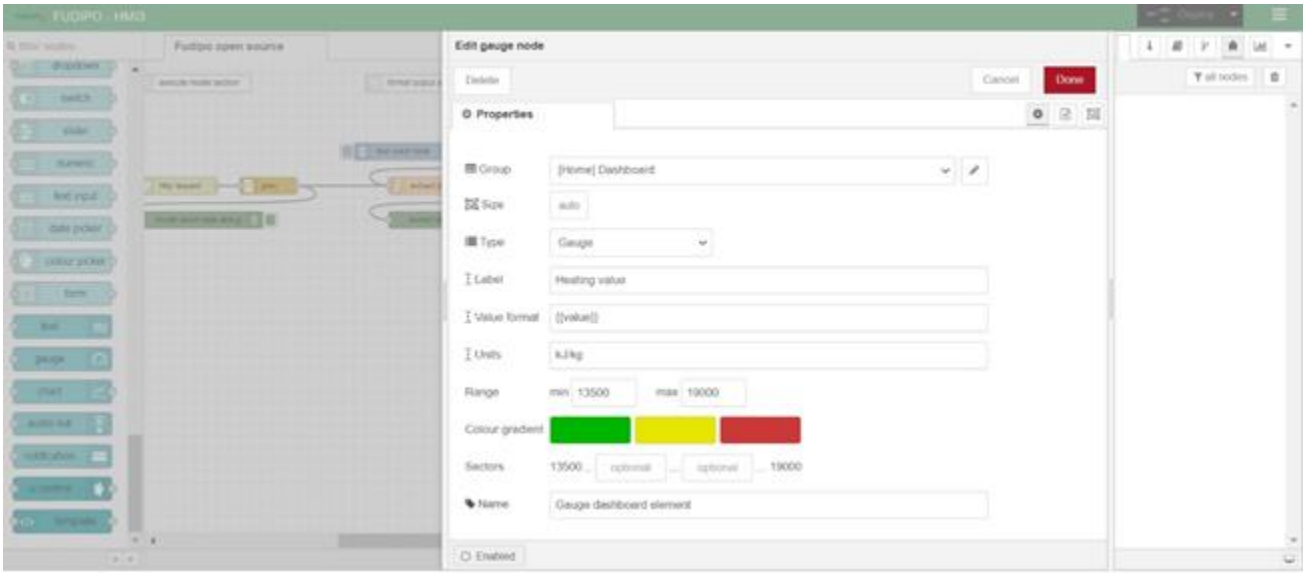

Figure 6, Node-RED's dashboard gauge node configuration

After all the nodes are in place deploy need to be done (also after every change). The calculation graph can be run using the single trigger inject node and in the debug section all the data from debug nodes will be shown. After the calculation graph is finished, preview of the dashboard can be accessed at http://localhost:1880/ui.

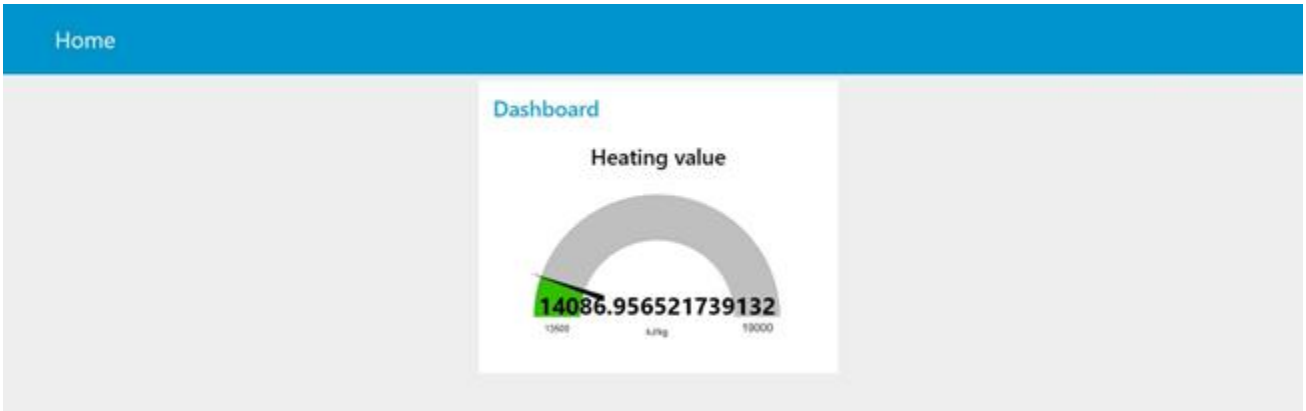

Figure 7, Node-RED's dashboard

6. TROUBLESHOOTING AND INSTRUCTIONS ON HOW TO SOLVE PROBLEMS

This section should act as a guideline on how to troubleshoot and solve unexpected problems in the toolbox. Even though the platform is designed in a way to be very versatile and stable, there is always the possibility that errors, like missing/faulty input data, crashed models, … occur. This section covers common issues and how to fix them and additionally how to detect and fix other issues.

One major difficulty is how to handle missing or faulty input data. There are many things to consider, e.g. what happens if there is no data at all, what happens when the data is faulty / incorrect or what happens if there is data missing. Models require input data from various sources (files, historian database, SQL Database, …) to ensure that the model runs correctly it is important to verify and validate the input data for completeness and correctness. Then it depends on the model itself and the use case on how to proceed with missing or faulty data. Sometimes it is better to just take the last value if there is a missing datapoint, in other scenarios it does not make sense to execute the model if there is not enough data. Nevertheless, the input data must always be checked either by the model itself or by the model service to prevent unnecessary errors and incorrect model results.

If the model itself crashes at some point because of faulty input data or any other reason, it is highly recommended that the model restarts automatically. Otherwise one faulty model execution could lead to a big loss of model result data. By deploying the model services to IIS, running them as a Windows Service or running them as a Docker container it can be ensured, that the models automatically restart and are always running, even if the model itself crashes. Sometimes faulty input data can set the model to a faulty state in which the model gets stuck. A reset of the model is then required.

Another common issue time zone synchronization. As a best practice it is recommended to always use UTC, since conversion from one time zone to another can be rather complicated and a source of failure. When using one toolbox instance for multiple time zones it is highly recommended to stick to one time zone (usually UTC) and convert to local time in the user interface. Additionally, daylight saving time needs to be considered when using only the local time zone. It needs to be taken into consideration what happens if there are multiple model results for the same timestamp because the clocks are being set back.

A big help when it comes to troubleshooting and identifying errors are log files. Unfortunately, if a Python Flask app runs in IIS the console output (stdout, stderr) are not logged to a file. Therefore, it is important and very helpful to manually implement logging functionality in the model services. Thus, allowing to access and view the console output of the app, while running it in IIS. This is especially useful if there are errors, that only occur when a model service is deployed to IIS. Alternatively, there is the possibility to stop the model service in IIS and run the model service directly from the command line interface to see its output and errors. One advantage of Docker as a model runtime

environment is, that Docker automatically logs the console output, and no logging functionality needs to be implemented manually.

6.1 COMPLETE ITERATIVE STEP-BY-STEP TROUBLESHOOTING

This section covers the troubleshooting process step by step if there occurred any unexpected issue. The first thing to do is to check Node-RED's debug window. It might contain useful error messages, like "Error wrong input", "Cannot connect to database",… If there is not a helpful error message each step in the calculation graph should be tested independently. This includes database connection / database webservice, model input parsing and validation, model execution, model results parsing and lastly storage of model results to database.

The first step is to verify that the data retrieval for the model input is working. Check if the data is retrievable and returns valid results. If the data retrieval does not work as expected, try to find out why it is not working.

Following checklist should be executed in case of malfunction of model execution:

1. Connection between the database and the server
2. Data retrieval web-service running
3. Database up and running

The next step to debug is the model input parsing and data preparation:

1. Data provided for model OK
2. Too much data provided for the specific model
3. Data verification failed
4. Syntax or semantic error in the data preparation

If there are no issues with the data retrieval or data preparation the next thing to check is the model service:

1. Model service up and running
2. Model service reachable from the server where Node-RED is running
3. Model crashed and needs to be restarted

One can try to execute the model from outside of Node-RED using the Swagger UI, if possible, to test if the model is working with static inputs as expected. If this is working try to execute the model from Node-RED with static inputs, where the results are known, and compare the model results. Afterwards try to execute the model with data from the database to see if there are faulty inputs that cause the model to crash.

The next step is to check the model results parsing. Since the data structure of the model output is very unlikely to change suddenly it is also unlikely that the model results parsing fails suddenly. Make sure that the model output is prepared as expected for storing the model results in a database.

The last step to check in the calculation graph is the storage of the model results into a database. Make sure that the data is mapped correctly and format conversions, like from an ISO datetime string to datetime of SQL, are handled properly. Consider special edge cases like different time zones for timestamps, daylight saving time or incorrect model results.

6.2 SPECIFIC COMPONENT-FOCUSED TROUBLESHOOTING

In the following sub-sections specific solutions for quick fixes of known typical issues are provided. This troubleshooting guide has been developed from experience of five demonstrator installations of the FUDIPO platform/toolbox.

6.2.1 DEBUGGING IN NODE-RED

The debug system of Node-RED is very useful to easily and quickly identify and fix errors. However, messages with more than 1000 characters are not fully displayed in the debug window by default. This can make it challenging, when the model requires a big dataset and the messages between nodes exceeds said limit. There are two ways to resolve this issue. The character limit of debug messages can either be changed in settings file or if looking at long messages is only needed occasionally the message can be written to a file which can then be opened in any text editor.

7. SYSTEM REQUIREMENTS

The minimum system requirements to run the Toolbox are:

- OS: Windows Server 2012 R2 / Windows Server 2019 / Windows 10 version 1809 (64 Bit) / Ubuntu 20.04 LTS
- CPU: dual core @ 2.0 GHz
- RAM: 16 GB
- Disk space: 25 GB

Recommended system requirements:

- OS: Windows Server 2012 R2 / Windows Server 2019 / Windows 10 version 2004 (64 Bit) / Ubuntu 20.04 LTS
- CPU: quad core @ 2.5 GHz
- RAM: 32 GB
- Disk space: 50 GB

8. REFERENCES

There are no sources in the current document.