

4.2 Developer Manual

4.2.1 Structure

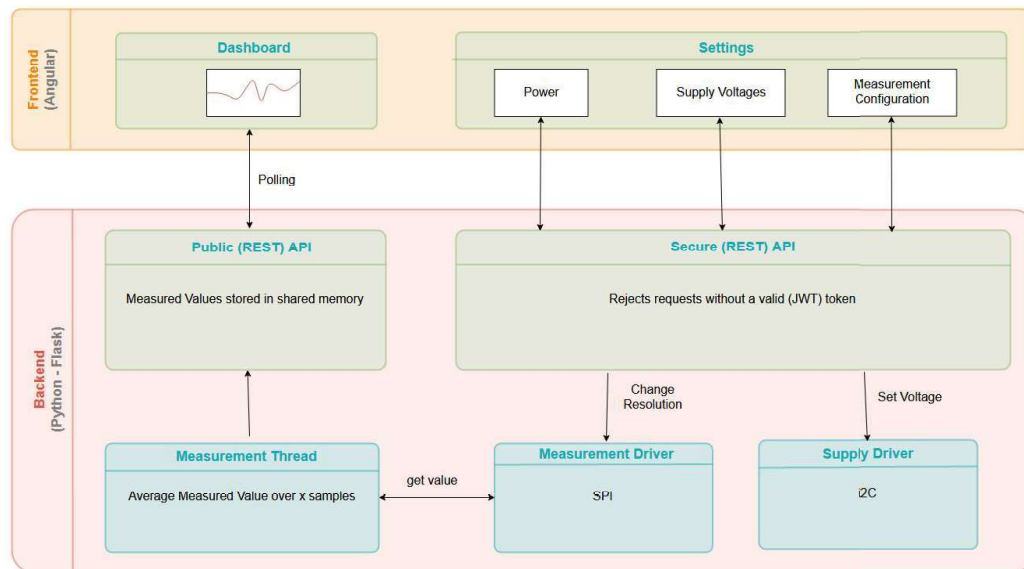


Figure 4.11: Structure of the software

4.2.1.1 Frontend

The frontend is written in Typescript² and uses the Angular³ framework. Typescript is a programming language developed by Microsoft which adds static typing to the Javascript language. A Typescript file is compiled to JavaScript before it is loaded by the browser, thus it is possible to use JavaScript syntax in a Typescript file and any developer that is familiar with JavaScript will be able to make changes to the frontend.

The main part of the frontend is the Dashboard which shows different voltage and current measurements in various graphs. If needed, custom transform functions can be applied to each measurement before the value is displayed.

In order to retrieve newly measured values, the frontend polls the backend in a predefined time interval.

With this setup it is also possible to connect any device to the DEMONSTRATOR, since only a browser that is capable of running JavaScript is needed.

The second part of the frontend consists of different configuration views. These can be used to either control the seven different voltage supplies built into the DEMONSTRATOR, change the resolution of the measurement devices, create new charts or to simply turn off the DEMONSTRATOR.

These functions call "secure" routes of the API, which can only be called if the

²<https://www.typescriptlang.org>

³<https://angular.io/>

correct bearer token is submitted as part of the request.

To obtain these tokens, the `api/login` route needs to be executed with the correct pin code. These restrictions make sure that nobody can make changes to the DEMONSTRATOR without knowing the pin code and thus the DEMONSTRATOR can be used in public areas without permanent supervision.

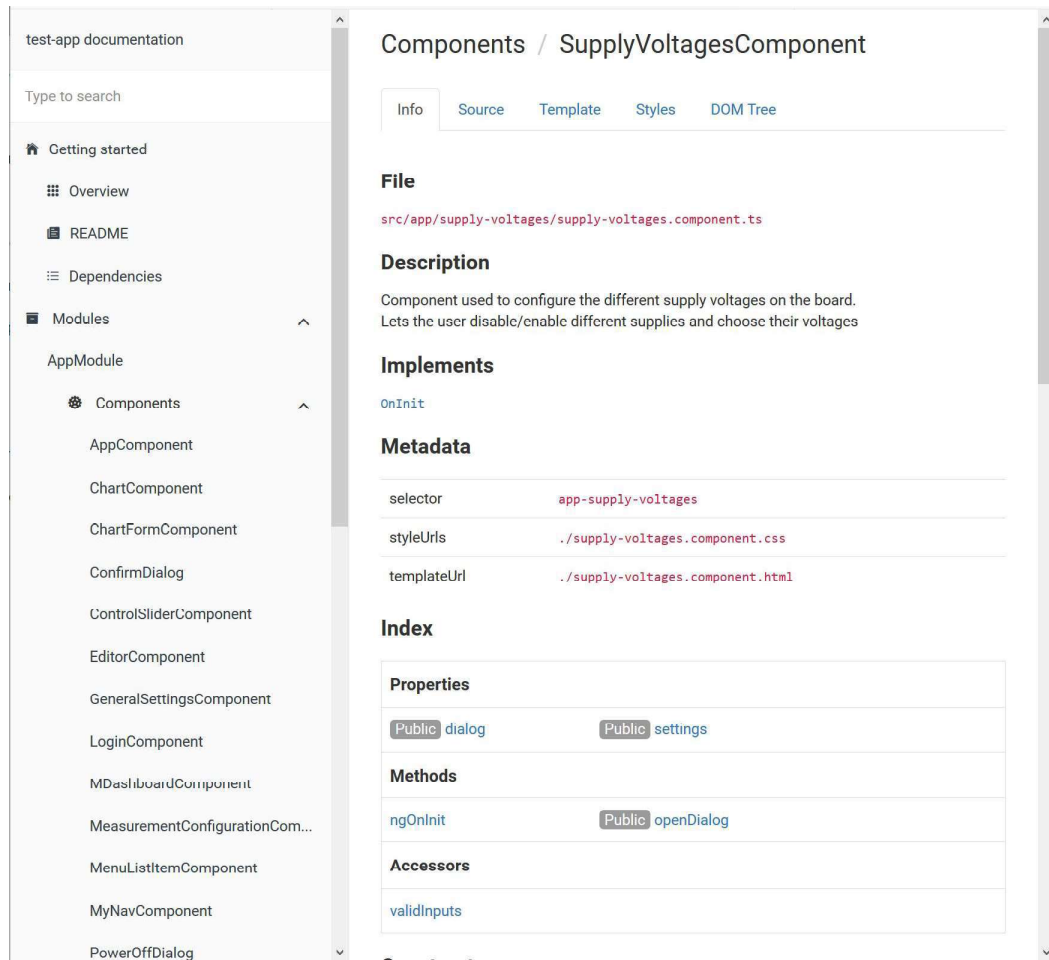
In the following section we list all relevant files in the "front" folder and explain what they are needed for.

4.2.1.2 Frontend - Files

- **angular.json:** Contains configuration information for Angular.
- **dist:** Contains frontend builds that can be deployed.
- **documentation:** Contains the software documentation in HTML format.
- **e2e:** Contains end-to-end tests.
- **node_modules:** Contains all dependencies that are imported.
- **package.json:** Contains all dependencies that are used by the application.
- **src**
 - **favicon.ico:** Icon that is displayed in the browser tab
 - **styles.css:** General style files. Applied to the whole application.
 - **app**
 - * **chart-form:** Component to edit/create charts.
 - * **editor:** Component to edit the whole application configuration.
 - * **general-settings:** Component to change general settings (brightness etc.).
 - * **login:** Login component to enter pincode.
 - * **m-dashboard:** Dashboard component containing all charts.
 - * **measurement-config:** Component to configure the measurement devices
 - * **navigation:** Component for sidebar and top actionbar
 - * **supply-voltages:** Component to edit the supply voltages
 - * **videostream:** Component to show a html videostream from backend
 - * **app-routing.module.ts:** Routing module. Contains all routes (\simeq URLs) that are accessible in this application.
 - * **app.component.[css/html/ts]:** Root of the application. All views are shown in this component.
 - * **app.module.ts:** Root module. Manages all imports, bootstraps the application.
 - * **_guards:** Contains a service that checks if a user is allowed to view a given resource.
 - * **_helpers**
 - **error.interceptor.ts:** Intercepts all requests. If a request returns 501 (Unauthorized) this interceptor displays the login component.

- **jwt.interceptor.ts:** Intercepts all requests. Adds the bearer token to the request header to authenticate user.
- * **_models:** Contains models for different datatypes.
- * **_services**
 - **app.config.ts:** Loads the conf.json file from the backend and stores it for use in the GUI.
 - **authentication.service.ts:** Handles the login process.
 - **MeasurementUpdateService.ts:** Service that polls in a given time interval for new measurement data.
- **assets**
 - * **hosts.json:** Contains all IP Addresses where the application should look for a backend server.
 - * **conf.json:** Contains the fallback configuration of the application. Only gets loaded if no backend is found.

For further information about the source code, the documentation of the application can either be found locally (`pulp-demonstrator\Server\front\documentation\index.html`) or online.



The screenshot displays the Angular documentation interface for the `SupplyVoltagesComponent`. On the left, a sidebar titled "test-app documentation" contains a search bar and a navigation menu with sections: "Getting started", "Overview", "README", "Dependencies", "Modules", and "Components". The "Components" section is expanded, showing a list of components including `AppComponent`, `ChartComponent`, `ChartFormComponent`, `ConfirmDialog`, `ControlSliderComponent`, `EditorComponent`, `GeneralSettingsComponent`, `LoginComponent`, `MDashboardComponent`, `MeasurementConfigurationCom...`, `MenuItemComponent`, `MyNavComponent`, and `PowerOffDialog`.

The main content area is titled "Components / SupplyVoltagesComponent" and features tabs for "Info", "Source", "Template", "Styles", and "DOM Tree". The "Info" tab is active, showing the following details:

- File:** `src/app/supply-voltages/supply-voltages.component.ts`
- Description:** Component used to configure the different supply voltages on the board. Lets the user disable/enable different supplies and choose their voltages
- Implements:** `OnInit`
- Metadata:**

selector	<code>app-supply-voltages</code>
styleUrls	<code>./supply-voltages.component.css</code>
templateUrl	<code>./supply-voltages.component.html</code>
- Index:**

Properties	
<code>Public</code> <code>dialog</code>	<code>Public</code> <code>settings</code>
Methods	
<code>ngOnInit</code>	<code>Public</code> <code>openDialog</code>
Accessors	
<code>validInputs</code>	

Figure 4.12: Documentation of the Angular code

4.2.1.3 Backend

The backend is written entirely in Python. It uses the Flask⁴ package to create a local REST server and Flask-Security⁵ to take care of the authentication process. The backend itself consists of a measurement thread that is running on a given interval that is configurable in the python code. With each cycle, the thread collects N-samples from all measurement devices configured in the `devices.py` file. After collecting the samples, the thread averages its values and stores it in the global `voltageValues` object. Whenever the route `/api/voltageUpdate` is requested, the server responds with the current `voltageValues` object.

Further, the backend provides some secure API routes, that are only executed if the request contains a valid JWT token. This token can only be obtained from the backend by calling the `/api/login` route with the right credentials.

4.2.1.4 Backend - Files

- **export:** Contains all exported graphs.
- **requirements.txt:** Contains all python dependencies.
- **run.py:** Starter script for the backend server.
- **logs:** Folder containing all log files.
- **flask_app**
 - **server.py:** Main File. Contains all API routes and polling thread.
 - **conf.json:** Stores the frontend configurations. (Charts, poll interval, ...)
 - **config.py:** Stores the backend configurations. (Port, PIN Code, Debug Level, ...)
 - **Devices.py:** Contains all devices that are installed on the PCB.
 - **http_codes.py:** Helper file containing most used HTTP status codes.
 - **VarSupply.py:** Interface modeling a Var-Supply.
 - **Driver.**
 - * **ADS8885.py:** Driver to communicate with ADS8885 chip using SPI.
 - * **LTC2631.py:** Driver to communicate with LTC2631 chip using I2C.
 - * **PCAL6416.py:** Driver to communicate with PCAL6416 chip using I2C.
- **html:** Contains the .html documentation of the source code.

For further information about the source code, the documentation of the backend can either be found locally (`pulp-demonstrator\Server\backend\html`) or online.

⁴<http://flask.pocoo.org/>

⁵<https://flask-security.readthedocs.io/>

file:///C:/Users/renez/OneDrive/Dokumente/ETH/FS19/g/

Index

Super-module

- `flask_app`

Functions

- `add_claims_to_access_token`
- `crossdomain`
- `getConfigObject`
- `getSystemInfos`
- `init`
- `loadConfig`
- `login`
- `logout`
- `main`
- `setVoltageOnDevice`
- `shutdown`
- `storeConfig`
- `storeConfigObject`
- `storeCsv`
- `storeImage`
- `updateAllDevices`
- `updateChartconfig`
- `updateDevs`
- `updateGeneralSettings`
- `voltageUpdate`

Classes

- `PollingThread`
 - `run`

Module `flask_app.server`

Entry point for the server application.

[SOURCE CODE](#)

Functions

```
def add_claims_to_access_token(identity)
```

[SOURCE CODE](#)

```
def crossdomain(origin=None, methods=None, headers=None,
                max_age=21600, attach_to_all=True,
                automatic_options=True)
```

Decorator function that allows crossdomain requests. Courtesy of <https://blog.skyred.fi/articles/better-crossdomain-snippet-for-flask.html>

[SOURCE CODE](#)

```
def getConfigObject()
```

[SOURCE CODE](#)

```
def getSystemInfos()
```

API Path `/api/getSystemInfos`

Request Type: `GET`

[SOURCE CODE](#)

```
def init()
```

Figure 4.13: Documentation of the backend python code

4.2.2 Setting up a new Raspberry Pi

Download Raspbian Stretch Lite⁶ and flash it to the SD card.

Boot the RASPBERRY PI and connect it to a network via wireless or an ethernet cable. It is also recommended to enable a SSH connection using the `raspi-config` command.

Now clone the DEMONSTRATOR repository⁷ to the home folder by typing

```
1 cd ~
2 git clone https://gitlab.ethz.ch/zrene/pulp-demonstrator.git
```

Move to the Server folder and execute the `install.sh` script as root.

```
1 cd ~/pulp-demonstrator/Server
2 sudo ./install.sh
```

This will set up all necessary software and configures the device.

Sometimes the script has to be run twice to finish successfully.

After running the installation script, change the password of the RASPBERRY PI.

Deploy the frontend:

```
1 cd ~/pulp-demonstrator/Server/skripts
2 sudo ./deploy.sh
```

If you plan on connecting the device to a WIFI, connect now and check what IP address your device has.

To be able to connect any device to the GUI of the DEMONSTRATOR, make sure to add the IP adress to the hosts file located at `/var/www/html/assets/hosts.json` file.

4.2.3 Enabling/Disabling the Hotspot

To enable or disable the hotspot, run the `disable_hotspot.sh` or the `enable_hotspot.sh` as root located in the `server/skripts` folder.

⁶<https://www.raspberrypi.org/downloads/raspbian/>

⁷<https://gitlab.ethz.ch/zrene/pulp-demonstrator.git>

4.2.4 Setting up the frontend to develop locally

To make changes to the frontend, it is recommended to clone the repository locally on your PC, since compiling it on the RASPBERRY PI can take quite a while. Install the Node Package Manager (NPM)⁸ to be able to install the needed frontend dependencies.

Now install the ANGULAR CLI as global dependency by executing:

```
1 npm i -g @angular/cli
```

Move to the frontend folder (`~/pulp-demonstrator/Server/front`) and install all dependencies that are needed to compile the GUI by typing:

```
1 npm install
```

The command also downloads the icon fonts and can take a while to complete. After the command is finished, execute

```
1 ng server --open
```

to start the development server locally and open the GUI in your browser. You can now start programming. Changes made to any file stored in the `src` folder will be compiled and shown as long as `ng serve` is running.

4.2.5 Setting up the backend to develop locally

To develop the backend, python3 and pip needs to be installed on your system. To install the required dependencies execute the following:

```
1 cd ./pulp-demonstrator/Server/backend
2 sudo pip3 install -r requirements.txt
```

If you plan on developing on the RASPBERRY PI make sure that you have run the `install.sh` script located in the `script` folder.

After setting up the dependencies, the backend server can be started by typing:

```
1 sudo python3 run.py
```

This will start a local (flask) server on port 8080. Verify that the server is really running by typing:

```
1 curl localhost:8080/api/loadconfig
```

or just opening the page in any browser.

If a JSON response is displayed, the backend server is up and running.

⁸<https://www.npmjs.com/>

4.2.6 Adding a new view to the GUI

To create a new view in the GUI, navigate to the "front" folder and create a new component using the ANGULAR CLI:

```
1 cd ~/pulp-demonstrator/Server/front
2 ng generate c <componentName>
```

This will create 4 new files inside a `src/<componentName>` folder:

- `<componentName>.component.html`
HTML file that contains the structure and GUI elements.
- `<componentName>.component.css`
CSS file that can be used to style the component. CSS style rules stored in this file will only be applied on the corresponding component.
- `<componentName>.component.ts`
Main part of the component. All Typescript (or Javascript) code that belongs to this component is stored here.
- `<componentName>.component.spec.ts`
File that stores end-to-end tests for the component

It will also create an entry in the `src/app/app.module.ts` file which contains all components.

To display the component as an item in the navigation menu, the new component has to be registered.

To do this we need to add a route to the GUI, so we can access the component using a URL.

Edit the `app-routing.module.ts` file located at `front/src/app`.

Import your component by appending the following line to the import statements:

```
1 import { <componentName> } from './<componentName>/<componentName>.
  component';
```

And register the component as a route by appending the following to the routes array:

```
1 { path: '<componentName>', component:<componentName> }
```

Note that the path can be chosen freely.

If you want to restrict the view so only users that are logged in can see it, change the route like follows:

```
1 { path: '<componentName>', component:<componentName>, canActivate: [
  AuthGuard] }
```

Now we can access the component using a URL. But we also need to add a link in the navigation menu.

To do this, navigate to the navigation component stored at `src/app/my-nav` and open the `my-nav.component.ts` file.

In this file add your component to the `NavItem` array, by appending the following:

```
1 {  
2     displayName: '<Name in Menu>',  
3     iconName: '<Name of a Material Icon>',  
4     route: '<route to your component>',  
5     loginRequired: <true|false>,  
6 }
```

You should now see your component in the navigation menu

4.2.7 Deploying a new version of the GUI

To deploy a new version of the GUI the Angular source code has to be compiled and then deployed to the RASPBERRY PI.

Run the following code inside the `front` folder:

```
1 ng build —prod
```

Angular will now build the newest version of the GUI and export it into `front/dist/test-app`. To deploy this folder to the RASPBERRY PI, first zip it by executing:

```
1 cd front/dist  
2 zip -r build_<buildVersion>.zip test-app
```

Now commit the zip and pull it from the RASPBERRY PI or just transfer it using a USB stick.

To deploy the zip run the `deploy.sh` script and delete the chromium cache:

```
1 # Deploy the file  
2 cd ~/pulp-demonstrator/Server/skripts  
3 sudo ./deploy.sh  
4 # Clean up chromium cache  
5 cd ~/.cache  
6 rm -rf chromium
```

The deploy script takes the zip file from the `front/dist` folder, which was last modified. If you pull more than one .zip file, the script could deploy an old version. To avoid that, just rename the file you want to deploy, so it is the file last modified.

4.2.8 Changing the IP of the Demonstrator

To change the IP address of the DEMONSTRATOR, edit the file `Server/skripts/enable_hotspot.sh` and change the value of `BASE_IP` to an IP of your choice.

Then run `disable_hotspot.sh` followed by `enable_hotspot.sh`.

4.2.9 Trouble shoot

4.2.9.1 Frontending

- **Can't connect to frontend (connection refused)**
Make sure the Angular development server is running (`ng serve`) and the port you are using is correct (4200).
If it is not working on the RASPBERRY PI, make sure you got the right IP address and are on the same network.
- **Frontend stays blank**
Make sure that the `ng serve` command isn't logging any errors.
Open the browser console by pressing `Ctrl + Shift + i` and check if any error messages appear.
This problem is often the result of a syntax error in one of the HTML file.
- **The frontend loads, but no data is displayed.**
Make sure that the backend server is up and running.
Make sure the IP-address of the RASPBERRY PI is stored in the `hosts.json` file located in the `assets` folder.
- **"Cors policy no 'access-control-allow-origin' header is present on the requested resource", appears as error in the Browser console.**
Make sure to add the `@crossdomain(origin=*)` annotation to your backend function.
- **I deployed a new GUI but can't see any changes.**
Make sure that your `build.zip` was the last file modified, you ran the `./deploy.sh` script and deleted the chromium cache.

4.2.9.2 Backend

- **How can i restart the server without rebooting the Raspberry Pi?**

```
1 # get process id of server task
2 ps -A | grep python
3 sudo kill <processId>
4 sudo python3 /home/pi/pulp-demonstrator/Server/backend/run.py &
```

- **Where are the log files located?**
The log files are stored in `pulp-demonstrator/Server/backend/logs`.

- **Where can I change the log level?**

Open the `config.py` file located at `pulp-demonstrator/Server/backend/flask_app` and change the value of the `LOGGING_LEVEL` variable.

- **How can I show the log output on the console?**

Open the `config.py` file located at `pulp-demonstrator/Server/backend/flask_app` and change the value of the `LOG_TO_CONSOLE` variable.

4.2.10 Calibrating the measurements

To calibrate the measurements as described in chapter 7, use the `calibrate.py` file located in the `skripts` folder.

Connect a stable power supply to the channel which you want to calibrate and start the script by typing:

```
1 sudo python3 calibrate.py -d "devId" -o "outputFileName.csv"
```

where `devId` is the ID of your measurement device (e.g. `vm1,vm2,cm3,cm4`).

The script will ask for the current resolution range and then continuously ask for the current input.

Every time an input value is entered, the skript will log the specified value and the measured value of the ADS8885 chip in the outputfile.

If you think you have enough values, exit the skript by typing `end` or pressing `ctrl+c`.

Now import the `.csv` file into the software of your choice to fit a linear approximation of type $a \cdot x + b$.

Once you have obtained the value for `a` and `b` update the file `flask_app/Devices.py` and change the values of the `ResoltuionScaler` for the calibrated device.

4.2.11 Using a different PCB

To use a different PCB the following changes have to be made:

1. Create a driver for all the chips on the PCB and store them in the driver folder.
2. Update the `VarSupply.py` file located at `backend/flask_app`.
3. Update the `Devices.py` file, replace all old devices with the new ones from your PCB.
4. If needed (structure of `Devices.py` changed) modify the `_setVoltageOnDevice()` function located in the `server.py` file, to fit your need.
5. Update the `conf.json` file to register your new devices in the frontend.

4.2.12 Adding public functions to help transforming data

To create a custom dataset that acts on different measurements, additional functionality may be needed.

It is possible, to declare public functions in the frontend, which can later be used to transform data. In this section, we will show how to register a new function that can be used in the GUI.

Suppose we want to create a new custom dataset, that is always 1 if `vm1` exceeds a given threshold and is 0 otherwise.

To be able to do this, we want to create a custom function

`stepDetect(sourceId, threshold, measuredData)` which returns 1 if the voltage from `sourceId` exceeds the `threshold` or 0 otherwise. This function should later be used in the GUI to create a custom dataset.

To achieve this, we have to create a new **public** function in the frontend, that can be used in any Chart.

Open the file

`/front/src/app/m-dashboard/chart/chart.component.ts` and create a new function by inserting the following snippet under the import statements:

```
1 export function stepDetect(sourceId:string, threshold: number,
2   measuredData: Record<string, DataEntry> ) : number {
3   // get DataEntry which contains measurement that belongs to the
4   source id
5   let entry : DataEntry = measuredData[sourceId];
6   // scale value to si unit
7   let scaledValue : number = scaleToSiUnit(entry);
8   // perform step detection and return 1 or 0
9   return scaledValue >= threshold;
10 }
```

Now, after compiling and deploying the application, this function can be used inside the Chart Editor.

It is also possible to create a buffer variable in the `chart.component.ts` file. This enables various functions (e.g. convolutions, AC-coupling) to be implemented.

The screenshot shows a configuration window for a function named 'StepDetection'. The window has a title bar with the name 'StepDetection' and a close button. Inside, there are several input fields and checkboxes. The 'Name' field is set to 'StepDetection'. The 'Unit' field is empty. The 'Axis' field is set to 'laxis'. There are three checkboxes: 'Auto Label', 'Auto Range', and 'Neg. Range', all of which are unchecked. The 'Measurement Device' field is set to 'vm1'. The 'Color' field is set to a blue color. The 'Apply function' field is checked, and the function code 'stepDetect('vm1', 2.5, data);' is entered. A 'Delete dataset' button is located at the bottom right of the window.

Figure 4.14: Example of the edge detection function used in the GUI

4.2.13 Change sampling rate

To change the sampling rate for the frontend, navigate to Settings-Editor and scroll down to the general properties. Change the value of the `timerInterval` to any poll interval (ms) and press the save button.

4.2.14 Frontend configuration file - `conf.json`

The `conf.json` file contains all information that is displayed in the frontend. It consists of four main parts:

1. **Charts:** Contains all information about the charts that are displayed.
2. **Deployment:** Contains all information about the backend. (Protocol used, path to api, hostname etc.)
3. **General:** Contains general information (e.g. Poll interval, custom options for all charts).
4. **Supplies:** Contains all voltage supplies that can be used on the PCB.

You can find a full documentation of the file in the appendix (—TODO—) or online at (—TODO—)