**API endpoint**

Staging API: <https://api-staging.gismolab-sgmp.cwfang.me/api/>

**Logging in**

POST to /api/user/login

Use this to obtain administrator access:

{

"email": "chihweif@gmail.com",

"password": "Aa!9FRTW7Mg6"

}

Sample response:

{

"accesstoken": "xxxxxxxx",

"house\_description": "House B",

"house\_id": 1, …

}

In subsequent requests, append this in header to authenticate:

Authorization = Bearer <accesstoken>

**Getting a list of devices**

Currently we have 2 houses, house\_id=1 is House B and house\_id=10 is House A.

POST to /api/device/list

{

"house\_id": 1

}

Sample response:

{

"devices": [

{

"description": "Sonnen Controller",

"device\_id": 1,

"name": "sonnen",

"type": "sonnen"

},

{

"description": "eGauge Controller",

"device\_id": 2,

"name": "egauge",

"type": "egauge"

},

{

"description": "Powerflex acc\_id=03, acs\_id=02",

"device\_id": 6,

"name": "pf\_02",

"type": "powerflex"

}

],

"status": "ok"

}

**Reading raw data of a specific device**

This can be useful to explore the fields of a specific device.

POST to /api/data/read

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "device",

"house\_id": 1,

"device\_name": "egauge"

}

Where the start\_time and end\_time are Unix timestamps in milliseconds. I recommend only reading 20 seconds of data, otherwise the response can be huge.

Sample response:

{

"data": [

{

"data": {

"A.Battery": 2.853,

"A.EV": -4.452,

"A.GridPower": 0.237,

"A.L1\_Frequency": 60.002,

"A.L1\_Voltage": 118.109,

"A.L2\_Frequency": 60.002,

"A.L2\_Voltage": 118.106,

"A.Load": 3.681,

"A.Solar": 1.844,

"A.SubPanel": -2.616,

"ts": "2021-11-08 12:11:19"

},

"timestamp": 1636402274453

}

],

"status": "ok"

}

Now you can see that the device “egauge” has fields named “A.Battery”, “A.EV”, “A.GridPower” and so on.

**Reading analytics data**

POST to /api/data/read

We can now write some formulas to read the data we want. For example, to read “A.Battery” field of egauge:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "egauge.A.Battery",

"house\_id": 1

}

We can average it over a specific time range. To get hourly average:  
{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "egauge.A.Battery",

"house\_id": 1,

"average": 3600000

}

We can also aggregate the results:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "egauge.A.Battery",

"house\_id": 1,

"agg\_function": "max"

}

Currently the supported agg\_functions are min, max, avg and count. Count can be useful to read events which are sparse data.

**Continuous Aggregation**

Continuous aggregations are jobs defined in the backend database that will be executed periodically to aggregate the historical data. Currently we make the jobs to execute each hour to calculate hourly average. By this we can accelerate the lookup speed of historical data. We define the aggregations through analytics API.

POST to /api/analytics/list

{

"house\_id": 1

}

Sample response:

{

"analytics": [

{

"analytics\_id": 8,

"continuous\_aggregation": true,

"description": "Sonnen battery current",

"formula": "sonnen.status.Pac\_total\_W/1000",

"name": "battery"

},

{

"analytics\_id": 9,

"continuous\_aggregation": true,

"description": "Sonnen battery charging current",

"formula": "-neg(sonnen.status.Pac\_total\_W/1000)",

"name": "battery\_charging"

},

{

"analytics\_id": 10,

"continuous\_aggregation": true,

"description": "Sonnen battery discharging current",

"formula": "pos(sonnen.status.Pac\_total\_W/1000)",

"name": "battery\_discharging"

}, …

],

"status": "ok"

}

To read a specific analytic item:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "analytics.solar",

"house\_id": 1

}

Notice that some of the analytic items have continuous\_aggregation=true. This means that when the fields within the formula are read through /api/data/read, it will default to the hourly average! To force reading raw data, specify “fine”: true in the request. For example:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "egauge.A.Solar",

"fine": true,

"house\_id": 1

}

You can see without “fine”: true the API will only return hourly average because it applied the optimization automatically and read data from the aggregation.

I would suggest reading raw data with a timespan not larger than 1 day to avoid causing excessive stress on the database.

**Advanced Formula Usage**

We can also combine multiple fields:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "egauge.A.Battery + egauge.A.EV",

"fine": true,

"house\_id": 1

}

If some of the fields have aggregations and some don’t have, make sure to specify “fine”: true in the request. Otherwise the API will automatically apply optimizations to the fields that have continuous aggregation and cause a timestamp mismatch between fields.

The analytics engine supports arithmetic operators (+ - \* / ^) and some built-in functions: sin, cos, tan, abs, exp, pos, neg, avg. Pos will replace all negative values with zero, and neg will replace all positive values with zero. Avg takes a second parameter as the interval and will average the time series over the given interval.

For example, to get the discharge power from sonnen:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "pos(sonnen.status.Pac\_total\_W)",

"fine": true,

"house\_id": 1

}

To get the charging power:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "-neg(sonnen.status.Pac\_total\_W)",

"fine": true,

"house\_id": 1

}

**Reading events**

Right now we have two types of useful events: EV\_START\_CHARGING and EV\_END\_CHARGING. To get all events, POST to /api/data/read

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "events.::EV\_END\_CHARGING",

"house\_id": 1

}

The specification of the event name is “<device\_name>::<event\_type>”. If the device name is empty then the API retrieves all events of the specified type.

You can even specify the field of the event:  
{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "events.::EV\_END\_CHARGING.duration",

"house\_id": 1

}

To get the average charging duration (you can also use min, max and count!):

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "events.::EV\_END\_CHARGING.duration",

"agg\_function": "avg",

"house\_id": 1

}

To get the average power during each charge:

{

"start\_time": {{start\_time}},

"end\_time": {{end\_time}},

"type": "analytics",

"formula": "(events.::EV\_END\_CHARGING.energy/1000) / (events.::EV\_END\_CHARGING.duration/1000/3600)",

"house\_id": 1

}