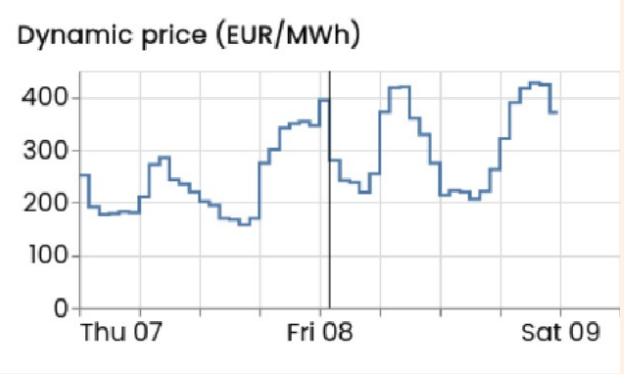
Using FlexMeasures to build a climate tech startup, in 15 minutes

Seita

LF ENERGY

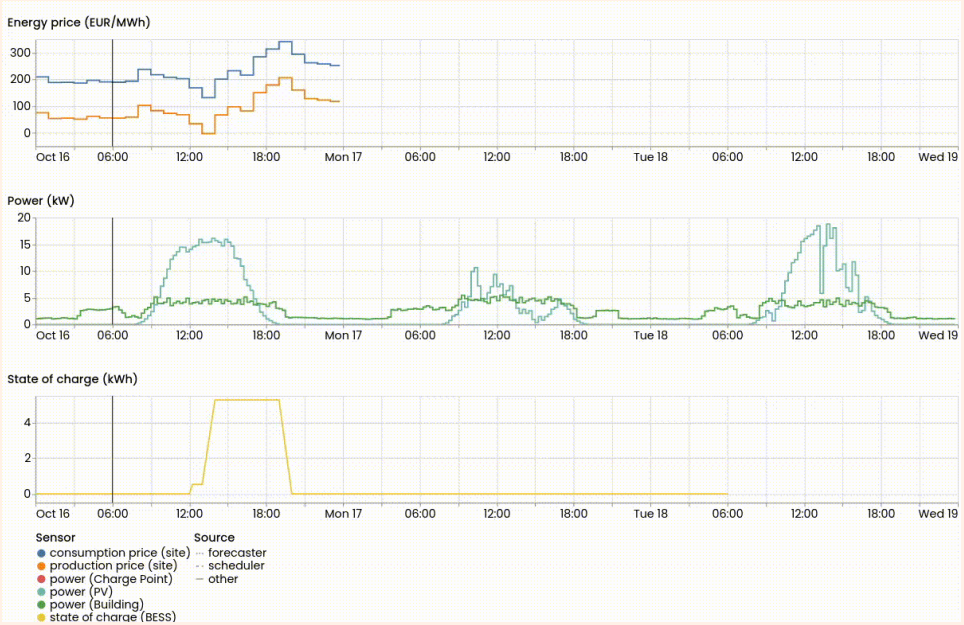
FlexMeasures

Behind the meter & more



LF ENERGY FlexMeasures

Real-time EMS



Proven in:

- Vehicle-to-grid (V2G)
- Water sanitation
- Smart heating

LFENERGY FlexMeasures

Objective: Add smart scheduling to your electrified heating operation.

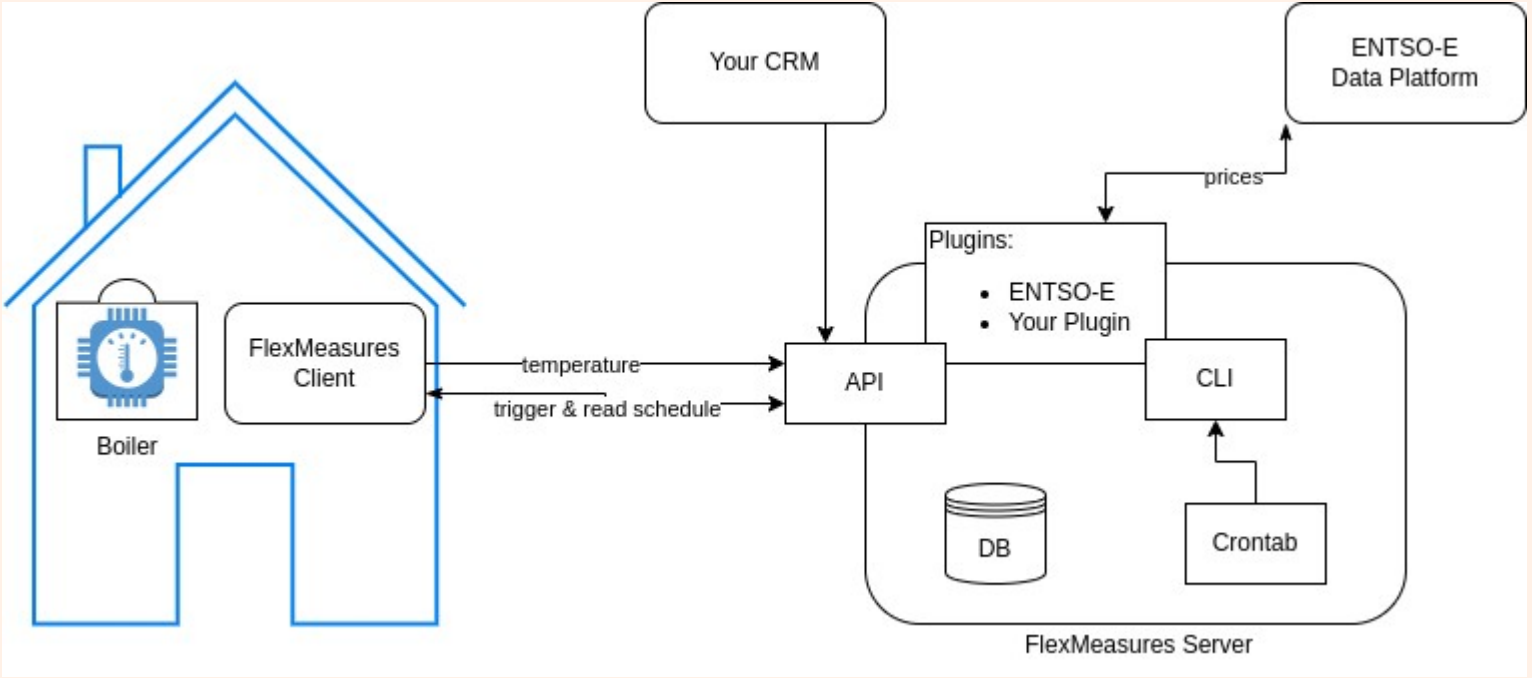
1. Create customer account
2. Send temperature readings
3. Pull energy prices
4. Compute a heating schedule
5. Configure the data dashboard
6. Report on costs

We'll use these interfaces:

- Plugin
- Client
- CLI
- API

LF ENERGY FlexMeasures

Objective: Add smart scheduling to your electrified heating operation.



LF ENERGY
FlexMeasures

Create your plugin

```
(venv) victor@pop-os:~/Work/Seita$ cookiecutter https://github.com/FlexMeasures/flexmeasures-plugin-template
You've downloaded /home/victor/.cookiecutters/flexmeasures-plugin-template before. Is it okay
to delete and re-download it? [y/n] (y): y
  [1/11] plugin_name (Your plugin name, e.g. 'MyPlugin'. Spaces possible. Prepending
'flexmeasures-' is a nice convention for integration plugins.): smart-boiler-plugin
  [2/11] plugin_slug (smart-boiler-plugin):
  [3/11] module_name (smart_boiler_plugin):
  [4/11] description (): FlexMeasures plugin to create your own Smart Heating StartUp
  [5/11] author_name ():
  [6/11] author_email ():
  [7/11] plugin_url ():
  [8/11] minimal_flexmeasures_version (0.14.0): 0.18.0
  [9/11] api_blueprint (y): y
  [10/11] ui_blueprint (y): n
  [11/11] cli_blueprint (y): n
```

LF ENERGY FlexMeasures

Create your plugin

```python
__version__ = "Unknown version"

"""
The __init__ for the smart-boiler-plugin FlexMeasures plugin.
FlexMeasures registers the BluePrint objects it finds in here.
"""

from importlib_metadata import version, PackageNotFoundError
from flask import Blueprint

# API
smart_boiler_plugin_api_bp: Blueprint = Blueprint(
    "smart-boiler-plugin API", __name__, url_prefix="/smart-boiler-plugin/api"
)
from smart_boiler_plugin.api import customer_crud
```

LF ENERGY FlexMeasures

New customer account via the API

```python
from flexmeasures import Account, User, Asset, Sensor, UserRole, AssetType
from flask import current_app, request

from flexmeasures.data.services.users import create_user
from flask_security import auth_required, current_user, auth_token_required
from flask_security.recoverable import send_reset_password_instructions
from flask_login import login_required
from flask_json import as_json

from .. import smart_boiler_plugin_api_bp
from utils import get_random_string



@smart_boiler_plugin_api_bp.route("/boiler-customer", methods=["POST"])
@auth_token_required
def create_boiler_customer():
    data = request.get_json()
    user = create_user(
        username=data["user_name"],
        email=data["user_email"],
        password=get_random_string(),
        user_roles = "account-admin",
        account_name=data["name"]
    )
customer = user.account
customer.consultancy_account_id = current_user.account.id
send_reset_password_instructions(user)
```

```python
asset_type = AssetType.query.filter_by(name="battery").one_or_none()
if asset_type is None:
    asset_type = AssetType(name="battery")
    current_app.db.session.add(asset_type)
    current_app.db.session.flush()

boiler = Asset(owner=customer, name="my-boiler", generic_asset_type_id=asset_type.id)
temperature = Sensor(generic_asset=boiler, name="temperature", unit="°C",
                     event_resolution="PT15M")
fill_rate = Sensor(generic_asset=boiler, name="fill-rate", unit="kW", event_resolution="PT15M",
                   attributes={"consumption_is_positive" : True})
demand = Sensor(generic_asset=boiler, name="demand", unit="kWh", event_resolution="PT15M")
cost = Sensor(generic_asset=boiler, name="cost", unit="EUR", event_resolution="PT15M")


current_app.db.session.add_all([customer, user, boiler, temperature, fill_rate, demand, cost])
current_app.db.session.commit()

return dict(
    boiler=boiler.id,
    fill_rate=fill_rate.id,
    temperature=temperature.id,
    demand=demand.id,
    cost=cost.id
)
```

FlexMeasures

New customer account via the API

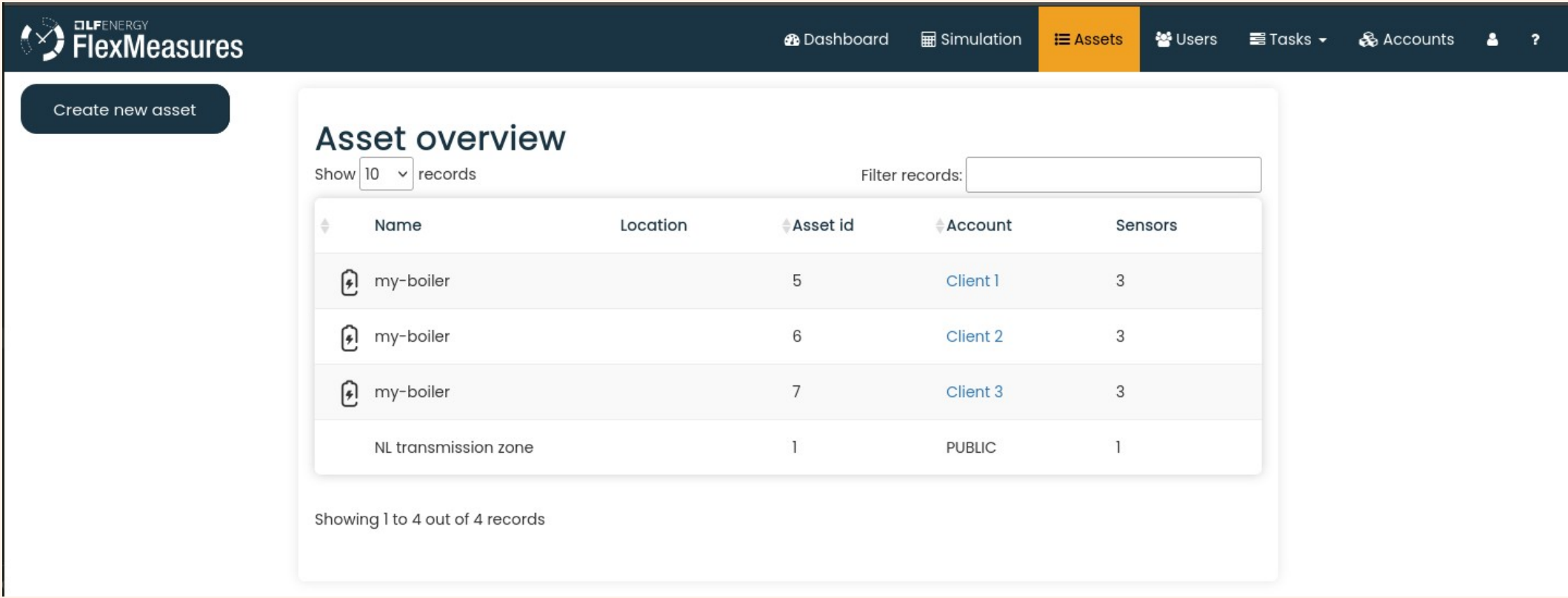
```
# we treat the boiler as a heat battery

asset_type = AssetType.query.filter_by(
  name="battery"
).one_or_none()

boiler = Asset(
    owner=customer,
    name="my-boiler",
    generic_asset_type_id=asset_type.id
)
```

```
# sensors with unit, resolution etc.
fill_rate = Sensor(
    generic_asset=boiler,
    name="fill-rate",
    unit="kW",
    event_resolution="PT15M",
    attributes={"consumption_is_positive" : True}
)
...
temperature = Sensor(
    generic_asset=boiler,
    name="temperature",
    unit="°C",
    event_resolution="PT15M"
)
```

Essentially, these are your business objects. Next, we'll create your business rules :)

FlexMeasures

See the assets in the UI



LFENERGY FlexMeasures

Local measurements: send temperature

Task:

Feed local measurements (here: temperature) regularly into your FlexMeasures server.

We use the **FlexMeasures client**.

```python
import pytz
from client import FlexMeasuresClient

client = FlexMeasuresClient(email="email@email.com", password="pw")

my_temperature_reading = 67.1  # in Fahrenheit, which is 19.5C
my_temperature_sensor_id = 3
now = pytz.timezone("Europe/Amsterdam").localize(datetime.now())


await client.post_measurements(
    sensor_id=my_temperature_sensor_id,
    start=now,
    duration="PT15M", # iso timedelta
    values=[my_temperature_reading],
    unit="degF",
)
```

LF ENERGY FlexMeasures

Data from 3rd parties: prices

Task:

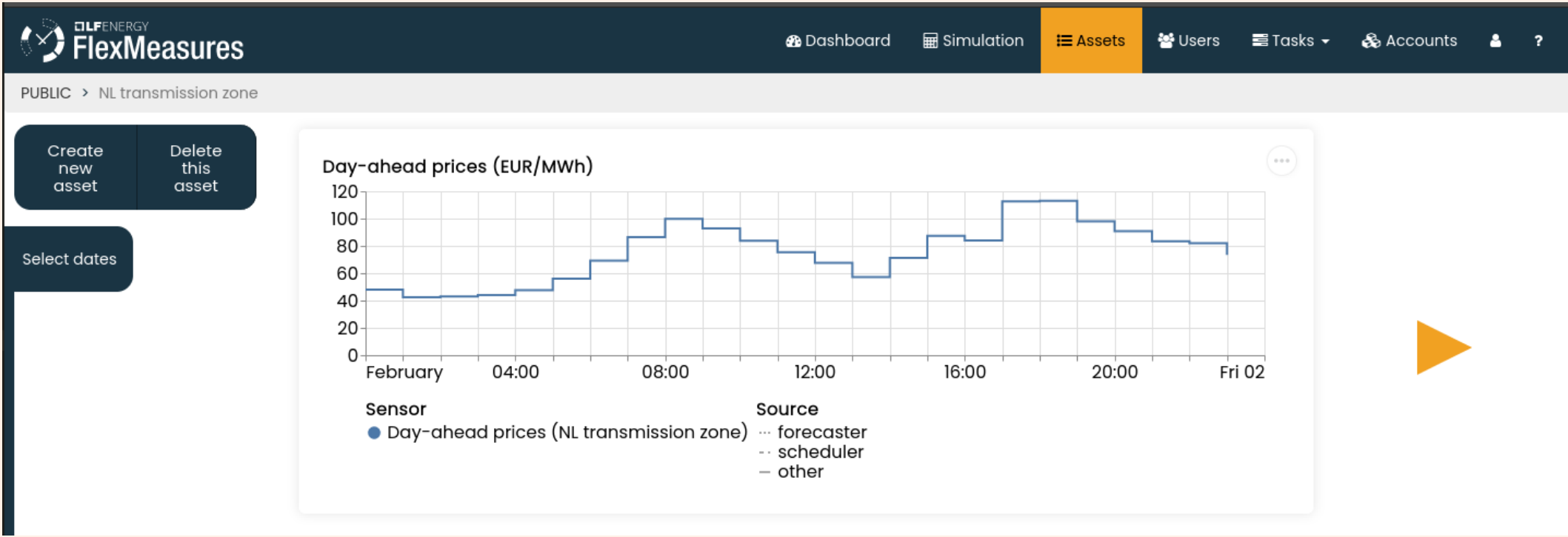
Get the latest price data from third-party-APIs (could become a daily cron job).

We use an existing **FlexMeasures plugin**.

```
$ gh repo clone SeitaBV/flexmeasures-entsoe
$ echo '
FLEXMEASURES_PLUGINS=[
    "~/smart-boiler/flexmeasures-entsoe",
    "~/smart-boiler/smart-boiler-plugin"
]
ENTSOE_COUNTRY_CODE = "NL"
ENTSOE_COUNTRY_TIMEZONE = "Europe/Amsterdam"
ENTSOE_DERIVED_DATA_SOURCE = "FlexMeasures ENTSO-E"
ENTSOE_AUTH_TOKEN = "get-yours-from-ENTSOE"
' >> ~/.flexmeasures.cfg

$ flexmeasures entsoe import-day-ahead-prices
$ # by default, this imports today's wholesale prices
```

LF ENERGY FlexMeasures

Data from 3rd parties: prices



LFENERGY FlexMeasures

Compute heating schedule

Task:

Ask FlexMeasures to compute a schedule for the boiler.

We are using **the API**.

```
token = requests.post(f"{FM_URL}/api/requestAuthToken",
            json={
                "email" : "admin@admin.com",
                "password" : "admin"
            }
)["auth_token"]
```

LF ENERGY FlexMeasures

Compute heating schedule

Next to the schedule timing, we can pass FlexMeasures a detailed “flex-model” and “flex-context”.

This tells FlexMeasures about the situation.

```
DAY_AHEAD_PRICE_SENSOR = 1


schedule_specs = {
    "start": "2024-02-04T00:00:00+01:00",
    "duration": "P1D",  # plan 24h ahead
    "flex-model": {
        "soc-at-start": 3.75,  # known energy content at beginning
        "soc-unit": "kWh",
        "soc-min": 3.7,
        "soc-max": 5.1,
        "soc-targets": [{"datetime": "2024-02-04T07:00:00+01:00", "value": 4.8}], # make it warm
when it counts
        "consumption-capacity": "1.5kW",
        "production-capacity": "0kW",  # we only consume
        "storage-efficiency": "99.95%",  # over 24H, this is around 95% (.99.95^{1/(24*4)} ~= .95)
    },
    "flex-context": {
        "consumption-price-sensor": DAY_AHEAD_PRICE_SENSOR,
        "production-price-sensor": DAY_AHEAD_PRICE_SENSOR
    }
}:w
```

LF ENERGY FlexMeasures

Compute heating schedule

We tell FlexMeasures to queue a scheduling job.

And later, we ask for the computed values.

```
import requests

schedule_id = requests.post(
  f"{FM_URL}/api/v3_0/sensors/{FILL_RATE_SENSOR}/schedules/trigger",
  headers={"Authorization": token, "Content-Type": "application/json"},
  json=schedule_specs,
)["schedule"]

schedule = requests.get(
  f"{FM_URL}/api/v3_0/sensors/{FILL_RATE_SENSOR}/schedules/{schedule_id}",
    headers={"Authorization": token, "Content-Type": "application/json"}
)["values"]
```

LF ENERGY FlexMeasures

Data dashboard

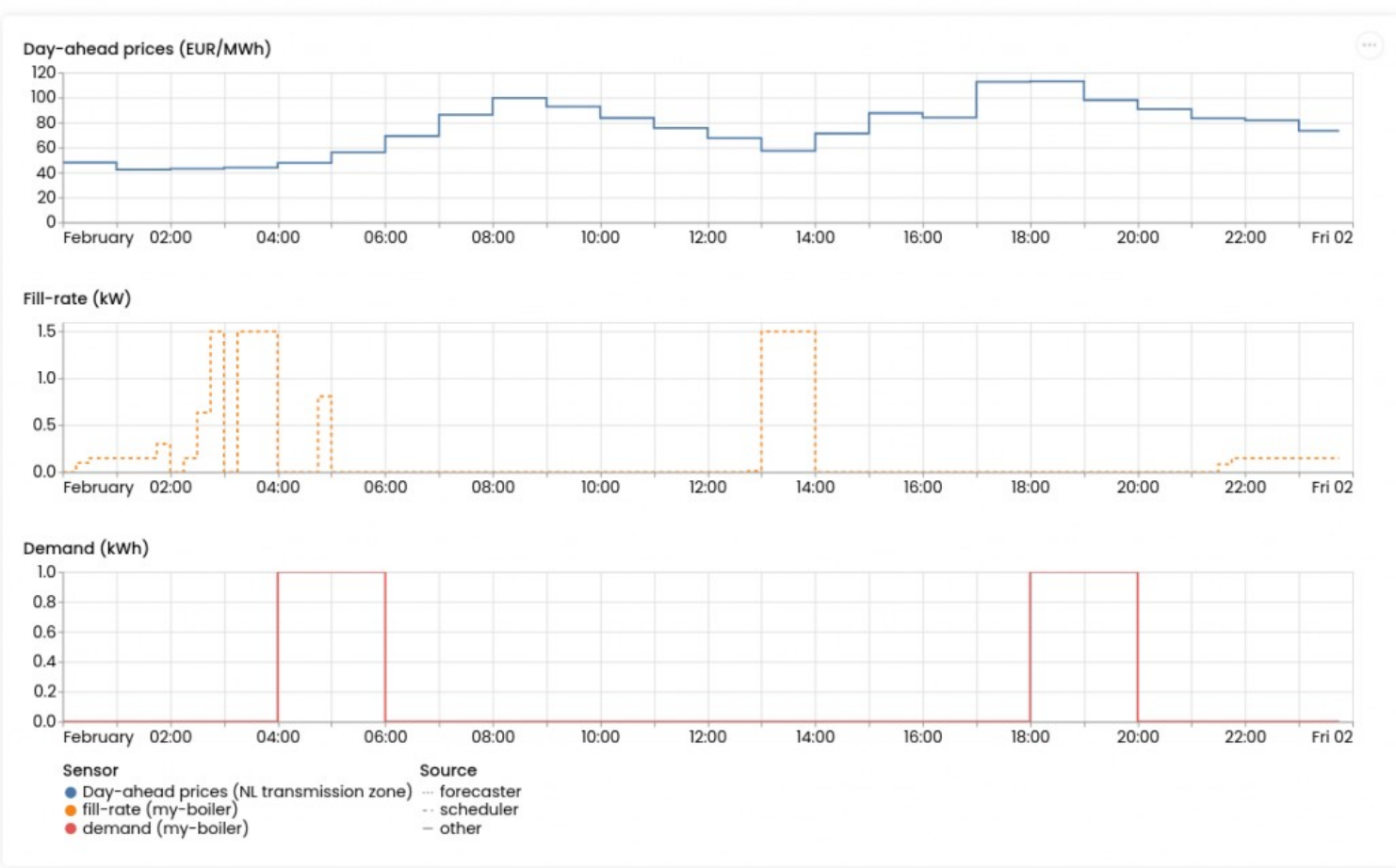
We tell FlexMeasures what time series we want to see on the boiler's asset page.

```
day_ahead_price = Sensor.query.filter_by(
        name="Day-ahead prices"
).one_or_none()

boiler.attributes["sensors_to_show"] = [
    day_ahead_price.id,
    fill_rate.id,
    demand.id
]
```

LF ENERGY FlexMeasures

Data dashboard

LF ENERGY FlexMeasures



Add to your own UI: GET {FM_URL}/api/v3_0/assets/{BOILER_ID}/chart

Report on heating costs

Task:

Run a daily report on the energy costs.
We use the **reporting** feature here.

We configure what the inputs are (fill rate) and where prices are (our day-ahead prices).

```
// costs-parameters.json

{
    "input" : [{"sensor" : FILL_RATE_SENSOR}],
    "output" : [{"sensor" : COSTS_SENSOR}]
}


// reporter-costs-config.json

{
    "consumption_price_sensor": DAY_AHEAD_PRICE_SENSOR,
    "loss_is_positive" : false
}
```

LF ENERGY FlexMeasures

Report on heating costs

This could be another daily cron job, which will make sure we report on energy costs.

```
$ flexmeasures add report\
    --reporter ProfitOrLossReporter\
    --config reporter-costs-config.json\
    --parameters costs-parameters.json\
    --start-offset DB,-1D --end-offset DB;
```

LF ENERGY FlexMeasures

Thank you

- http://flexmeasures.io
- nicolas@seita.nl

Roadmap

- Sector coupling ((EV) batteries + heat storage)
- Partnerships (e.g. ESCos, OEMs, industry, HEMS)
- Support open standards (e.g. S2) & grow open source community

S2 Standard

Seita