Project 9

Guido Dino Ballabio

Aarhus University

3 December, 2018

Section 1

An API for Sensors' Data

Web Service

- Abstract and Connectionless Communication →

Web Service

- Abstract and Connectionless Communication →
- Client-Server Architecture →

Web Service

- Abstract and Connectionless Communication →
- Client-Server Architecture →
- HTTP

REST

- Read-Only $\rightarrow$ Stateless

REST

- Read-Only $\rightarrow$ Stateless
- Only representation $\rightarrow$ Simplification

REST

- Read-Only $\rightarrow$ Stateless
- Only representation $\rightarrow$ Simplification
- One *collection of data* for each sensor

REST

- Read-Only $\rightarrow$ Stateless
- Only representation $\rightarrow$ Simplification
- One *collection of data* for each sensor
- One *resource* for each sensor data-point

Ideal API

Only GET request:

`domain-name/api/sensor/timestamp`

Ideal API

Only GET request:

`domain-name/api/sensor/timestamp`

with self-explaining response:

```
{
  "sensor": "Thermometer-0",
  data: [
          {
            "timestamp": "2018-11-11 21:11:11",
            "temperature": 20
          }
        ]
}
```

Section 2

Implementation

Data Retrivial

- FTP server

Data Retrivial

- FTP server
- Polling for update and download

Data Retrivial

- FTP server
- Polling for update and download
- Preprocessing with pandas (from csv)

- Database creation

- One *table* per collection
- One row per data-point

Flask

- Simple web server
- Per-url function

Flask

- Simple web server
- Per-url function

Problem

- Low-level
- Lacking default security

Libraries

SQLAlchemy Introspection + Flask = "sandman"

Libraries

SQLAlchemy Introspection + Flask = "sandman"

Unreliable

Lackluster support for timestamps as text and bugs

Libraries

SQLAlchemy Introspection + Flask = "sandman"

Unreliable

Lackluster support for timestamps as text and bugs

Enter Datasette

Developed especially for publicing read-only data from database

Bonus Implements advanced queries and even SQL quries.
No security concerns given the read-only property.

Example

`GET http://127.0.0.1:8001/db-a23fb6b/CO2/2018-11-04+00:00:00.000000.json`

note: .json is just appended to just one of the possible formats


```
{
  "database": "db",
  "table": "CO2",
  "rows": [
            ["2018-11-04 00:00:00.000000", 527, ..., 0]
          ],
  "columns":  [
                "Timestamp",
                "Centrale kraftv\u00e6rker DK1",
                ...,
                "Solceller DK2"
              ],
  "primary_keys": ["Timestamp"],
  "primary_key_values": ["2018-11-04 00:00:00.000000"],
  "query_ms": 0.45609474182128906
}
```

Section 3

API Reference

Brief API Doumentation

```
GET /[database-id/]table-name/resource-id
```

Brief API Doumentation

`GET /[database-id/]table-name/resource-id`

`GET /[database-id/]table-name/YY-MM-DD+h:m:s.ms.json`

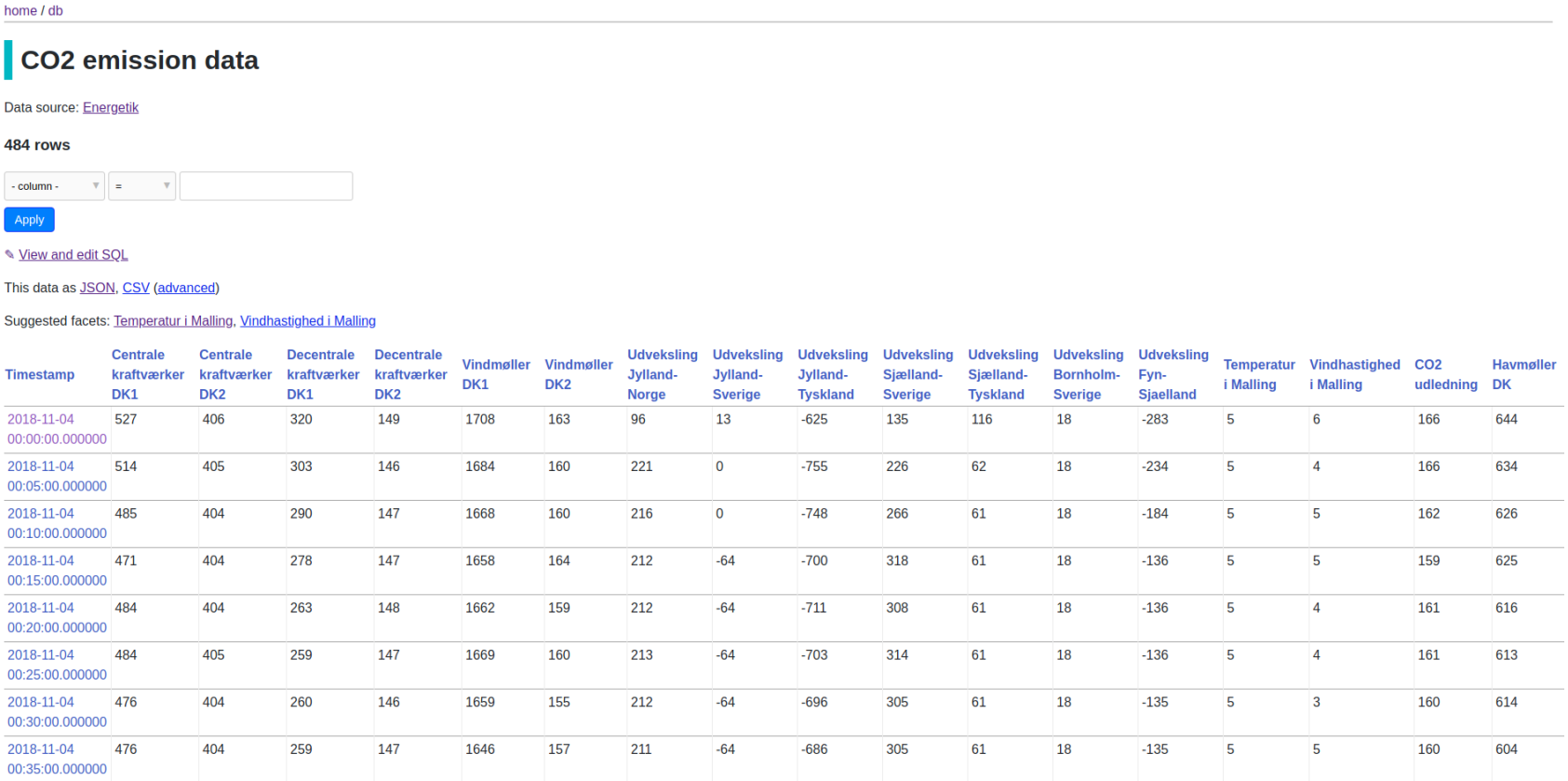
with query string parameters for quick search

`?attribute1=name&attribute2=140`

Bonus

All sort of advanced search options (SQL, facets, text search) are freely inspectable through a web page and thorugh: `/-/inspect`

Example Screenshot



home / db

CO2 emission data

Data source: Energetik

484 rows

- column - = Apply

View and edit SQL

This data as JSON, CSV (advanced)

Suggested facets: Temperatur i Malling, Vindhastighed i Malling

Timestamp	Centrale kraftværker DK1	Centrale kraftværker DK2	Decentrale kraftværker DK1	Decentrale kraftværker DK2	Vindmøller DK1	Vindmøller DK2	Udveksling Jylland-Norge	Udveksling Jylland-Sverige	Udveksling Jylland-Tyskland	Udveksling Sjælland-Sverige	Udveksling Sjælland-Tyskland	Udveksling Bornholm-Sverige	Udveksling Fyn-Sjaelland	Temperatur i Malling	Vindhastighed i Malling	CO2 udledning	Havmøller DK
2018-11-04 00:00:00.000000	527	406	320	149	1708	163	96	13	-625	135	116	18	-283	5	6	166	644
2018-11-04 00:05:00.000000	514	405	303	146	1684	160	221	0	-755	226	62	18	-234	5	4	166	634
2018-11-04 00:10:00.000000	485	404	290	147	1668	160	216	0	-748	266	61	18	-184	5	5	162	626
2018-11-04 00:15:00.000000	471	404	278	147	1658	164	212	-64	-700	318	61	18	-136	5	5	159	625
2018-11-04 00:20:00.000000	484	404	263	148	1662	159	212	-64	-711	308	61	18	-136	5	4	161	616
2018-11-04 00:25:00.000000	484	405	259	147	1669	160	213	-64	-703	314	61	18	-136	5	4	161	613
2018-11-04 00:30:00.000000	476	404	260	146	1659	155	212	-64	-696	305	61	18	-135	5	3	160	614
2018-11-04 00:35:00.000000	476	404	259	147	1646	157	211	-64	-686	305	61	18	-135	5	5	160	604

Figure 1: screenshot

Section 4

Conclusion

Success

*In this way all the **cold database** can be easly made accessible to the public in a programmtic way*

Surprise

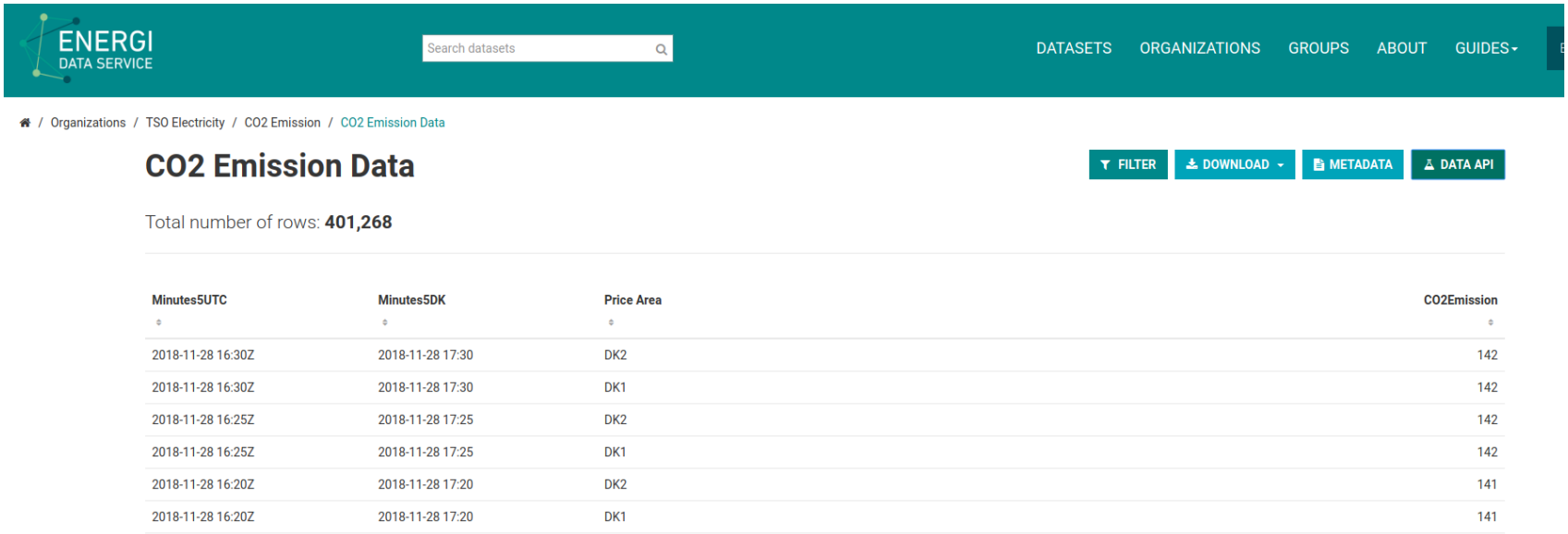


Figure 2: energinet