CONNECTED URBAN TWINS

Das Energiewende-Dashboard

Ein Erfolgsbeispiel interkommunaler Kooperation

Partnerstädte:

Hamburg

Stadt Leipzig

Landeshauptstadt München

Gefördert durch:

Bundesministerium für Wohnen, Stadtentwicklung und Bauwesen

KFW

CUT in a nutshell

- Connected Urban Twins (CUT) – Urbane Datenplattformen und Digitale Zwillinge für integrierte Stadtentwicklung
- Smart-City-Modellprojekten des Bundesministeriums für Wohnen, Stadtentwicklung und Bauwesen (BMWSB)
- Hamburg, München, Leipzig
- Entwicklung Digitaler Zwillinge
- Open Source Paradigma

3

Partnerstädte im Kooperationsprojekt

5 Jahre

Projektlaufzeit: Januar 2021 bis Dezember 2025

3D-Stadtmodelle von Hamburg, Leipzig und München

Als Sebastian in CUT angefangen hat…

Anforderungen zum Projektstart:

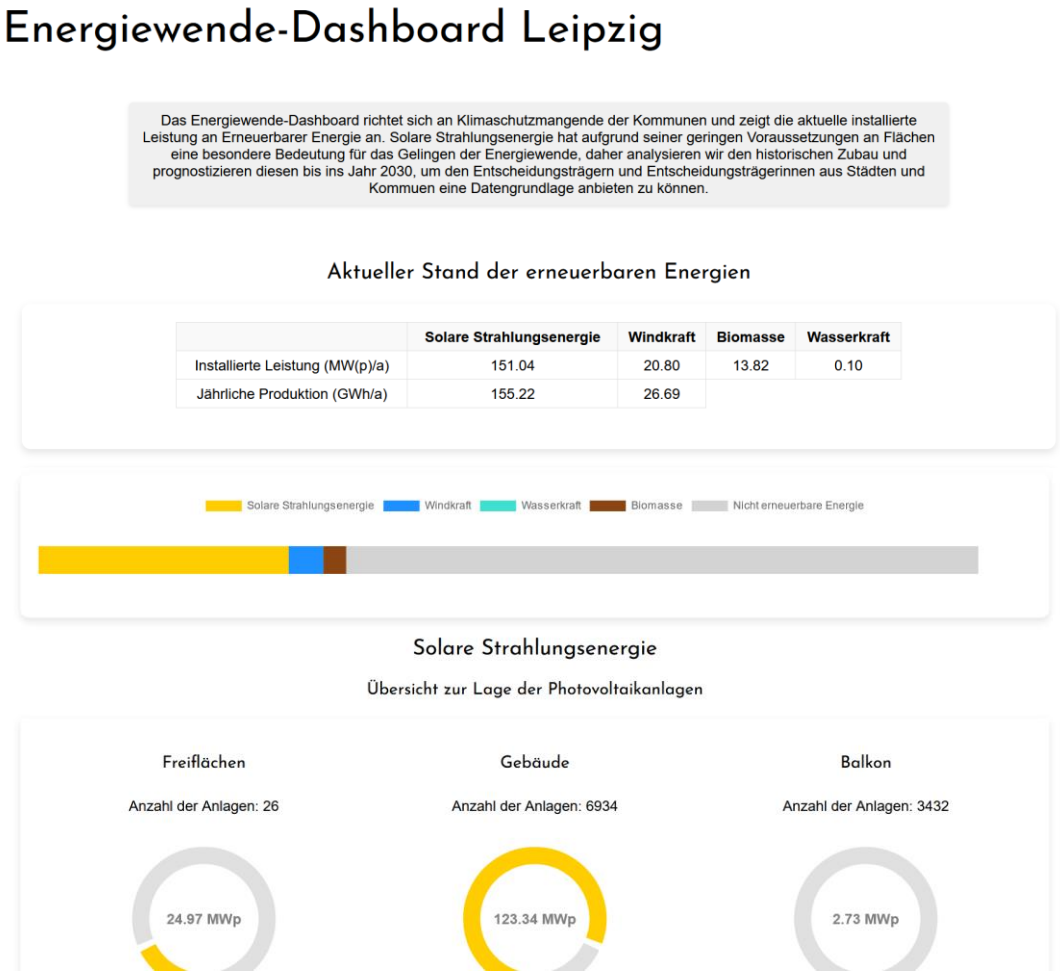
- Wir bauen replikationsfähige Softwareprodukte.
- Alles Open Source.
- Zum Projektende sollen die entwickelten Lösungen produktionsreif sein.
- Wir wollen „Was-wäre-wenn“-Szenarien und Digitale Zwillinge.

Energiewende-Dashboard

Ziel
Darstellung des aktuellen Ausbaus erneuerbarer Energien im Stadtgebiet, um die Fachämter des Stadtplanungsamtes, Referat Nachhaltigkeit und Klimaschutz, Amt für Umweltschutz bei der Planung der Energiewende zu unterstützen (mit Bezug zu Kommunaler Wärmeplanung, energetischen Quartierskonzepten etc.).

Zielgruppe
- Klimaschutzmanagende
- Stadtplanende

Datenquelle
Öffentlich und frei verfügbare Daten des Marktstammdatenregisters der Bundesnetzagentur.



https://leipzig.energiewende-dashboard.platforming.io/

Architektur

Die Entwicklungscommunity setzt auf Microservices und CI/CD.

Frontend	Frontend (React.js)
Backend	Backend in Python
Data Storage	PostgreSQL
Data Analysis Jobs	KPI Berechnung (Python)
Data Inbound Service	Datenintegration (Python, HTTPS-Webrequests)
MaStR	Externe API des Marktstammdatenregisters der Bundesnetzagentur

Ein Community-Produkt

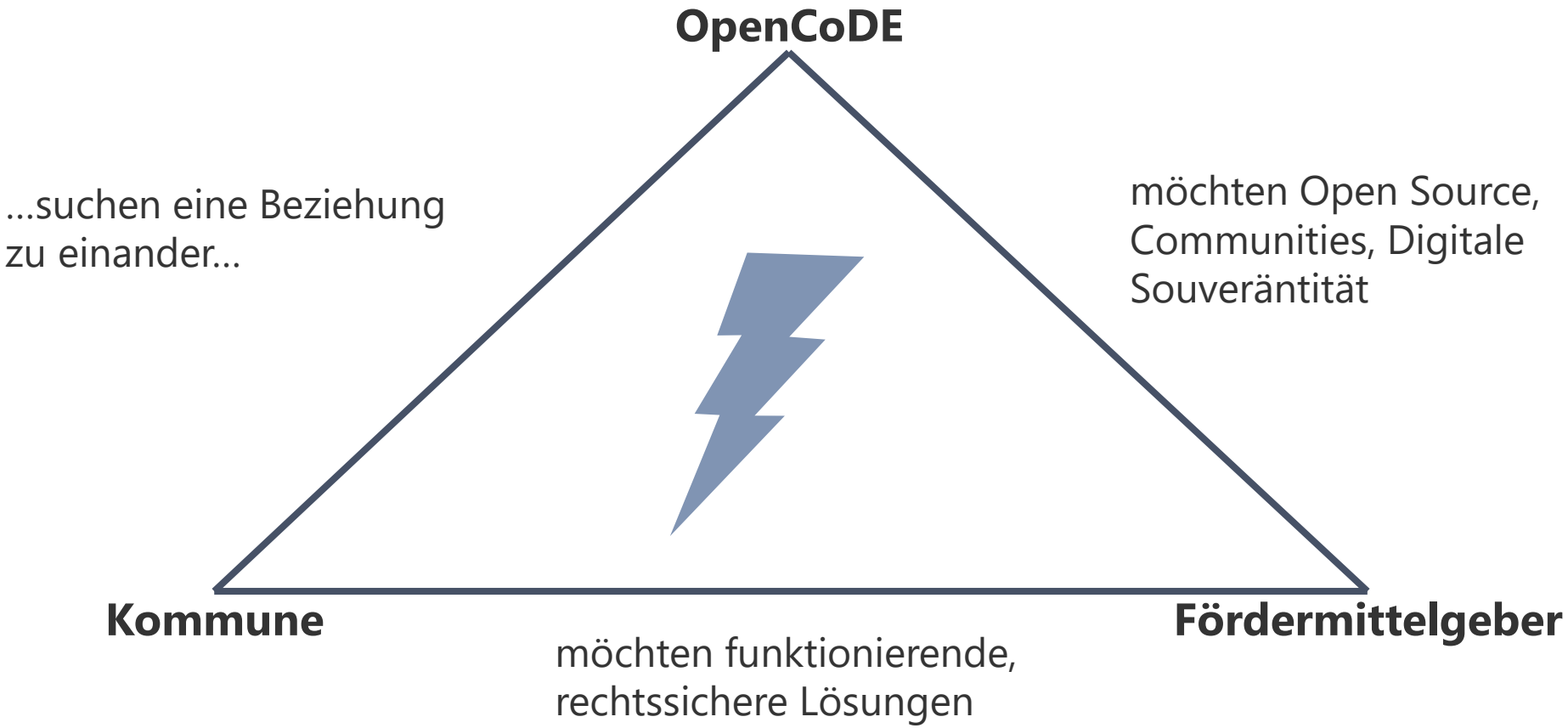
Entwicklung
Die Wartung und Weiterentwicklung des Energiewende-Dashboards wird durch die MPSC Community der Arbeits- und Entwicklungsgemeinschaft Energiedatenplattformen sichergestellt.

Meilensteine 2025
- Betriebskonzept für dauerhafte Nutzung
- Weiterentwicklung mit Fokus auf Fragestellungen aus Klimaschutz und Stadtplanung
- Veröffentlichung auf GitLab und openCode

https://www.smart-city-dialog.de/raum/113/beitraege/11456

„Hätte ich damals gewusst, was ich heute weiß, hätte ich es nie getan.“

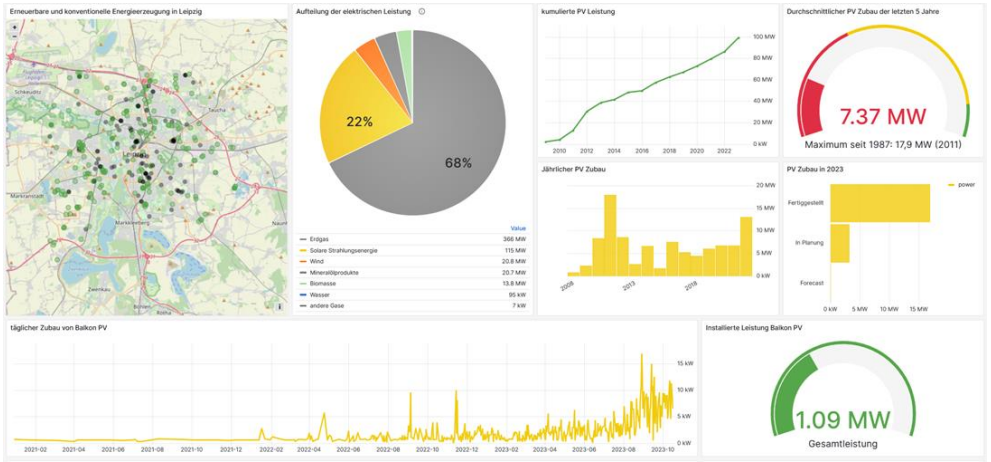
Unser Ökosystem



Erste Schritte, nächste Schritte

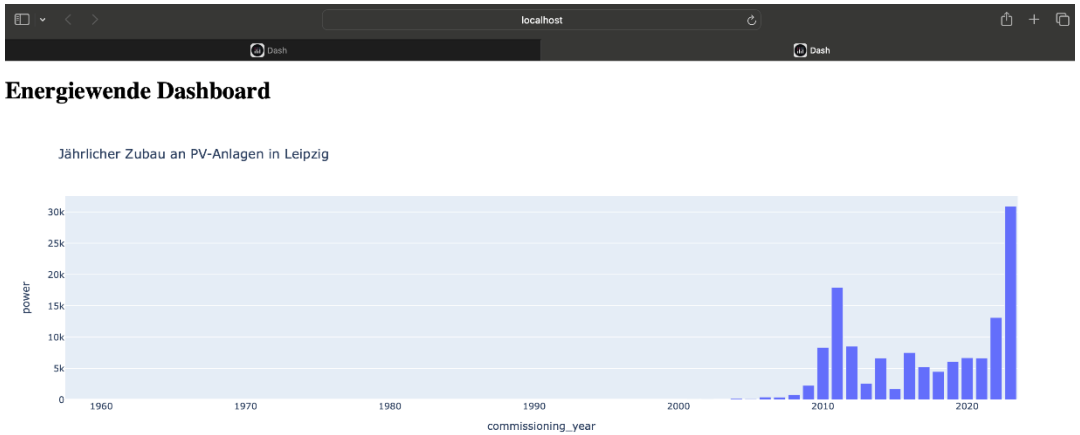
Entwicklung es Energiewende-Dashboards Teil 1

Juni 2023: PoC und erste Analysen in grafana



#Proof of Concept
#statische Daten
#100% Open Source
#1 Entwickler

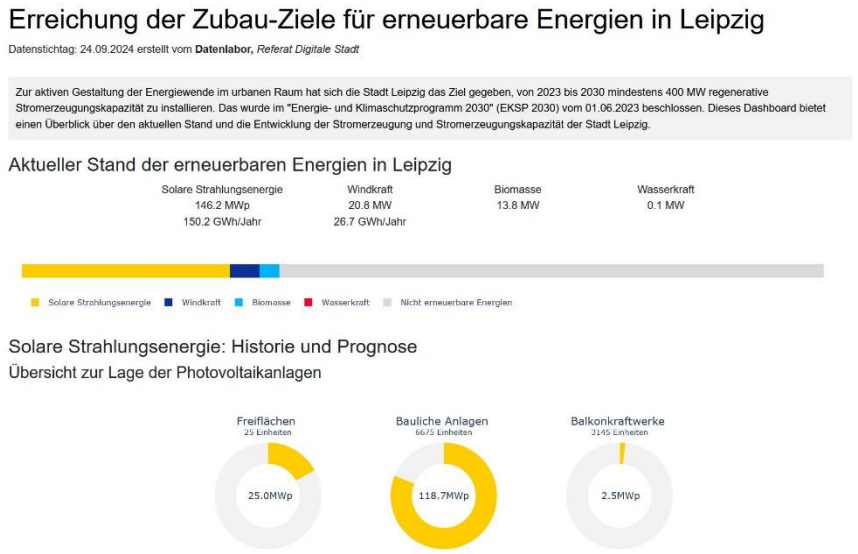
Dezember 2023: erster PyDash PoC



#Replikationsreifes Softwareprodukt
#Technologischer Durchstich
#Datenaktualisierung

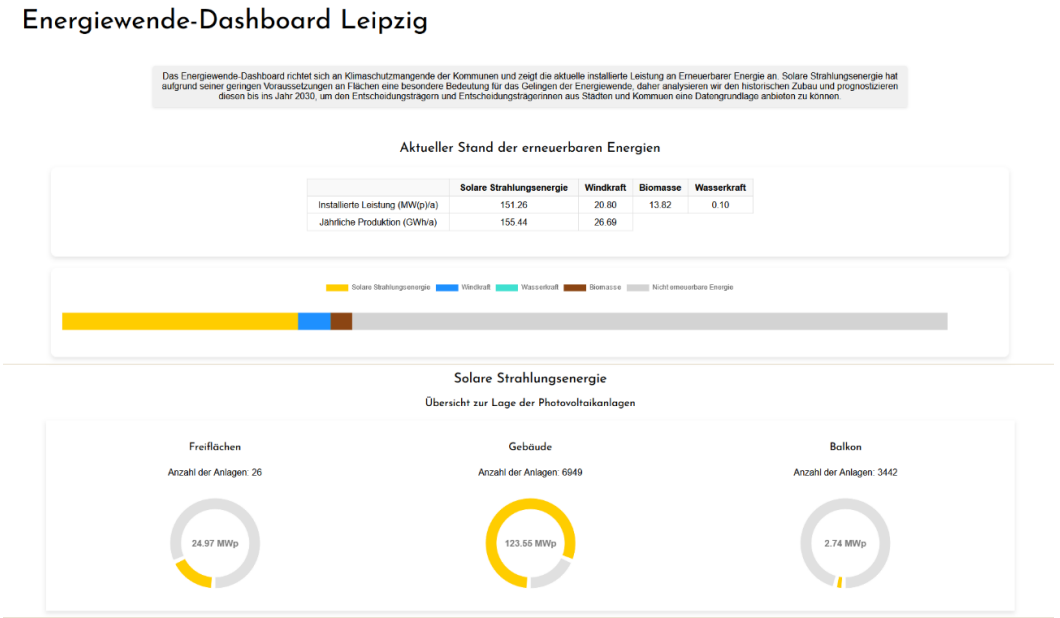
Entwicklung es Energiewende-Dashboards Teil 2

Juni 2024: erste PyDash Variante, dockerisiert, ins Internet exposed



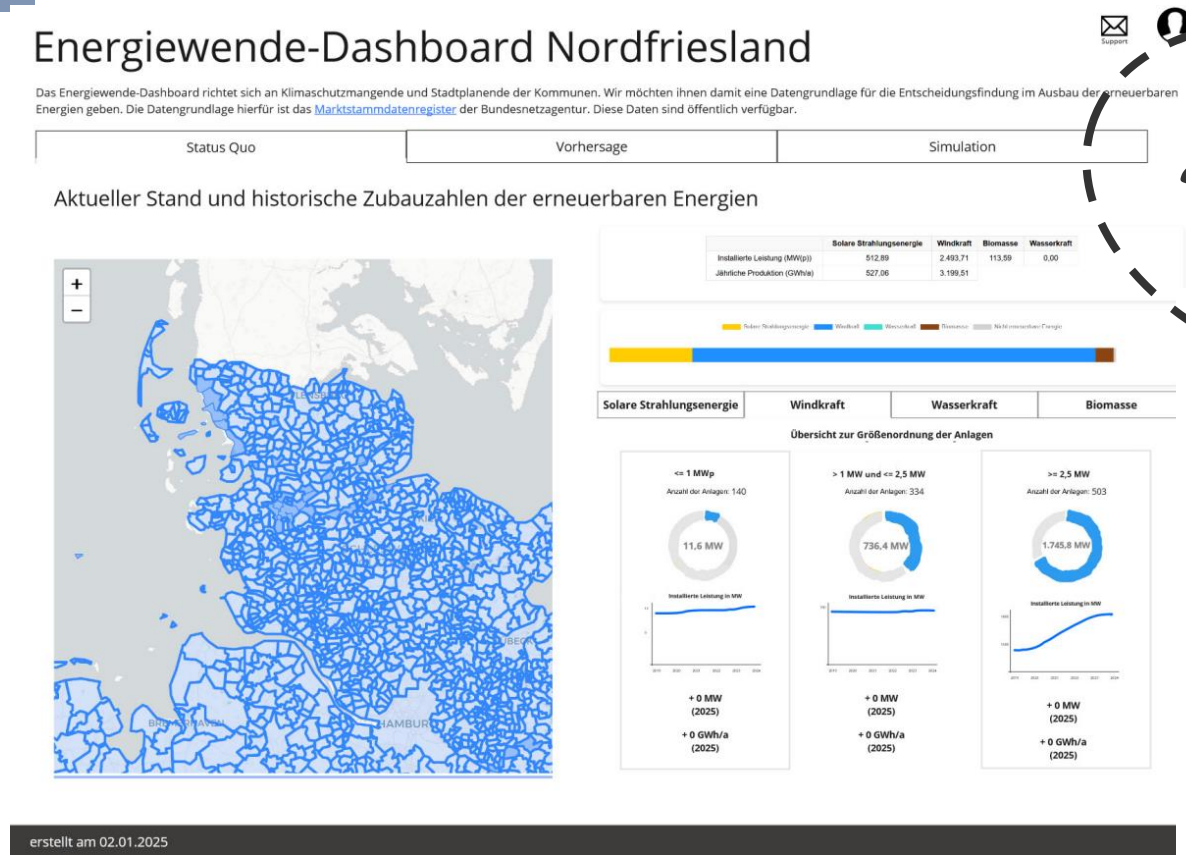
#AgileEntwicklung
#TDD
#OOP
#2 Entwickler*innen

Dezember 2024: React-Version, Microservices, Python-Backend, Tests

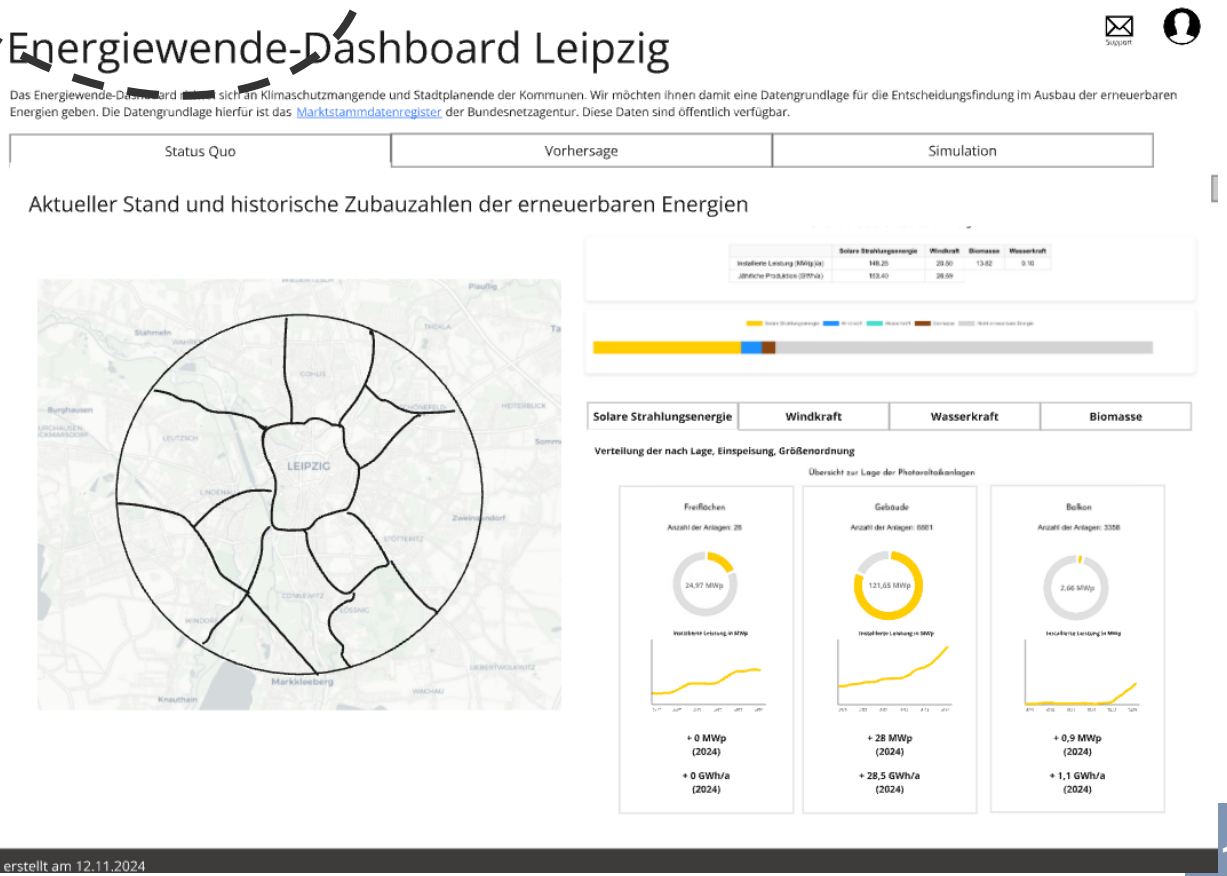


#Frontend
#Backend
#Data Jobs
#Datenbank

Entwürfe für die nächste Version



DRAFT



Komplexe Probleme erfordern selbstorganisierte Teams!

CUT

- Trotz organisatorischer **Herausforderungen** aus dem Ökosystem:
  Das Dashboard ist nur so gut wie es ist, weil es in einer Community entwickelt wurde.

- Die Community zieht die guten Leute an, weil sie einen Raum zum Lernen und für Weiterentwicklung bietet.

- Die Community zwingt einen zu professionellen Arbeitsweisen und bringt Methoden mit sich, die man nutzen muss, damit man überhaupt eine sinnvolle Lösung entwickeln kann.

- Software-Qualität und moderne IT (Microservices, Kubernetes, TDD, …) sind kein Selbstzweck. Sie stellen sicher, dass die Lösung langfristig funktionieren kann

- Jedes Feedback verbessert die Lösung. Replikation generiert Feedback.

Wie arbeitet unsere Community zusammen?

#Nützlichkeit #Verfügbarkeit #NoExcuses

```python
pv_only = MaStReg.query('source == "Solare Strahlungsenergie"')
print('Number of photovoltaic assets (all status) in Leipzig: ' + str(pv_only.shape[0]))
print(pv_only.groupby('status').agg({'status':'count', 'power':{'sum', 'mean', 'median'}}))

# filter only for assets which are "In Betrieb", because we want to work with the assets which are producing now
pv_only_active = pv_only.query('status == "In Betrieb"')
pv_annual_addition = pv_only_active.groupby('commissioning_year').agg({'power':'sum'})

pv_annual_addition['cum_sum'] = pv_annual_addition['power'].cumsum()
pv_daily_addition = pv_only_active.groupby('commissioning_date').agg({'power':'sum'})

```

Kein Spaghetti-Code

```python
pv_annual_addition['returns'] = pv_annual_addition['power'].pct_change()
# pv_annual_addition['returns'].hist()
print('--- key statistics on annual pv addtion returns ---')
print(pv_annual_addition['returns'].agg([np.mean, np.median]))
print('25%-quantile ' + str(pv_annual_addition['returns'].quantile(0.25)))
print('25%-quantile ' + str(pv_annual_addition['returns'].quantile(0.75)))

print('all time HIGH annual addition: ' + str(pv_annual_addition['power'].max()))
print('all time LOW annual addition: ' + str(pv_annual_addition['power'].min()))
print('annual addition of the last 5 years: ' + str(pv_annual_addition['power'].tail(5)))
print('5 year mean: ' + str(np.mean(pv_annual_addition.loc[2018:2022, 'power'])))

five_yr_mean = np.mean(pv_annual_addition.loc[2018:2022, 'returns'])
print('5 yr mean (returns): ' + str(five_yr_mean))
print('5 yr median (returns): ' + str(np.median(pv_annual_addition.loc[2018:2022, 'returns'])))
```

Objekte helfen

- Prinzipien aus der OOP helfen dabei die Komplexität zu managen
- Anstatt langer, umständlicher Abfrage-Konstrukte oder Modelle kann man die Aufgaben in Klassenmethoden kapseln
- Vorteile:
  - einheitliche Schnittstellen
  - gut zu testen

```python
### Asset Class
import pandas as pd

from datetime import datetime

class Asset:
    """defines a class for general energy asset"""

    def __init__(self, max_power, status, commissioning_date):···

    # Add a decorated max_power() method returning _power
    @property
    def max_power(self):···

    # Add a setter max_power() method
    @max_power.setter
    def max_power(self, new_max_power):···

    # Add a decorated status() method returning _status
    @property
    def status(self):···

    # Add a setter status() method
    @status.setter
    def status(self, new_status):···

    # Add a decorated is_active() attribute
    @property
    def is_active(self):···
    @property
```

Kein Weg führt an Tests vorbei

- Automatisches Testen sichert langfristige die Funktionalität und Qualität des Codes
- Anfangs dauert es länger, aber im Laufe der Zeit gibt es euch Geschwindigkeit

```
============================= test session starts =============================
platform win32 -- Python 3.11.1, pytest-8.1.1, pluggy-1.4.0
rootdir: C:\Users\BoehmSe03\_Analyse_Tools\Code\energiewende-dashboard
plugins: dash-2.16.1
collected 40 items

tests\asset_modelling\test_asset_modelling.py ....................................  [ 97%]
tests\test_main.py x                                                             [100%]

============================== warnings summary ===============================
tests/asset_modelling/test_asset_modelling.py::test_AssetPortfolio_annual_growth_path_nonempty_portfolio
  C:\Users\BoehmSe03\_Analyse_Tools\Code\energiewende-dashboard\tests\asset_modelling\test_asset_modelling.py:519: FutureWarning: The 'kind' keyword in Da
taFrame.resample is deprecated and will be removed in a future version. Explicitly cast the index to the desired type instead
    expected_result = helper_df.resample('YE', on = 'commissioning_date', kind = 'period').sum()

tests/asset_modelling/test_asset_modelling.py::test_AssetPortfolio_annual_growth_path_nonempty_portfolio
  C:\Users\BoehmSe03\_Analyse_Tools\Code\energiewende-dashboard\src\asset_modelling\asset_modelling.py:251: FutureWarning: The 'kind' keyword in DataFrame
.resample is deprecated and will be removed in a future version. Explicitly cast the index to the desired type instead
    df = df.resample('YE', on = "commissioning_date", kind='period').sum()

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
=================== 39 passed, 1 xfailed, 2 warnings in 4.56s ====================
```

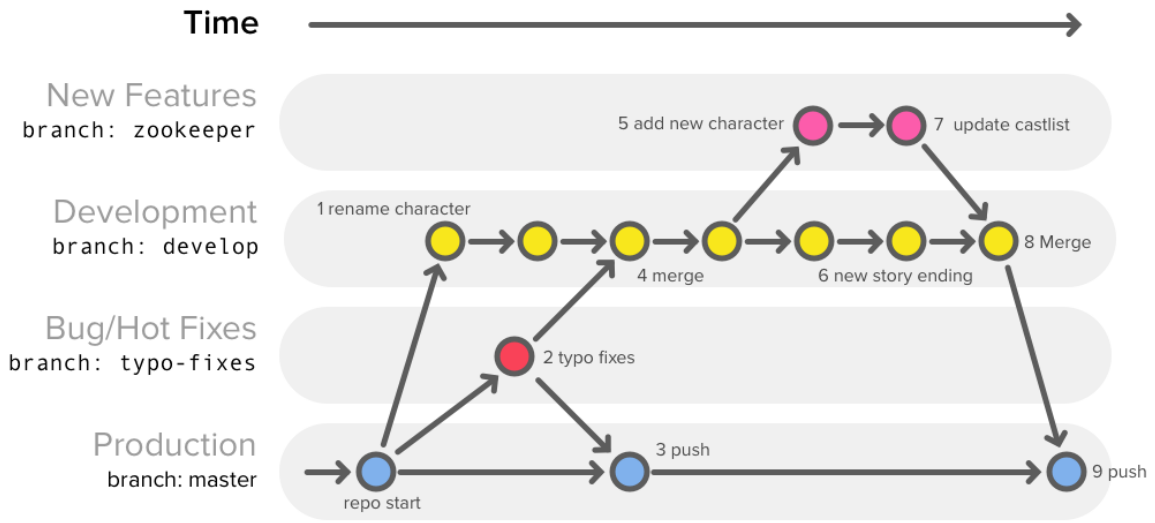
Wie wir Tests in die Entwicklung einbinden

- Wir nutzen TDD
- TDD … test driven development
- Zuerst Testfälle definieren
- Dann diese implementieren
- Dann Code entwickeln
- Nach jeder Änderung die Tests laufen lassen

```python
def test_Asset_init_error_with_negative_max_power():
    """negative max_power input raises a ValueError"""

    # define test inputs
    invalid_max_power = -1
    valid_status = 'In Betrieb'
    valid_commissioning_date = datetime(2024, 2, 28)

    # define expected error message
    expected_error_msg = "Power can not be negative for this asset type."

    # if Asset raises a ValueError then store exeption as exception_info
    with pytest.raises(ValueError) as exception_info:
        test_asset = Asset(invalid_max_power, valid_status, valid_commissioning_date)

    # check if ValueError contains correct message
    assert exception_info.match(expected_error_msg)
```

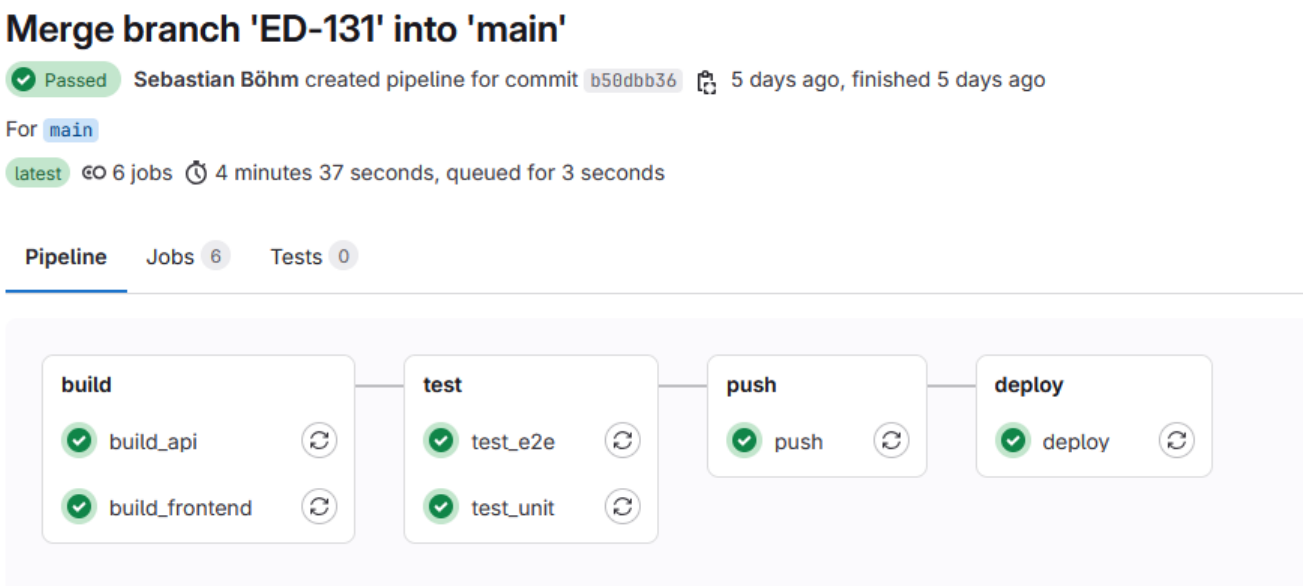
„Keine Versionskontrolle – kein Mitleid“

- Eine Versionskontrolle ermöglicht das kollaborative Arbeiten an einem Projekt
- Alle Änderungen werden nachvollziehbar verwaltet
- Durch das aufteilen der Entwicklung auf verschiedene Branches wird sichergestellt, dass nur funktionstüchtige Software ausgeliefert wird



Continuous Integration und Continuous Deployment

- CI / CD stellt sicher, dass die Software kontinuierlich und standardisiert ausgeliefert wird
- In einer Pipeline werden die einzelnen Schritte definiert, die für die Auslieferung notwendig sind
- Zum Beispiel sollte es immer auch einen Test-Schritt geben



Containerisierung erleichtert die Nutzung

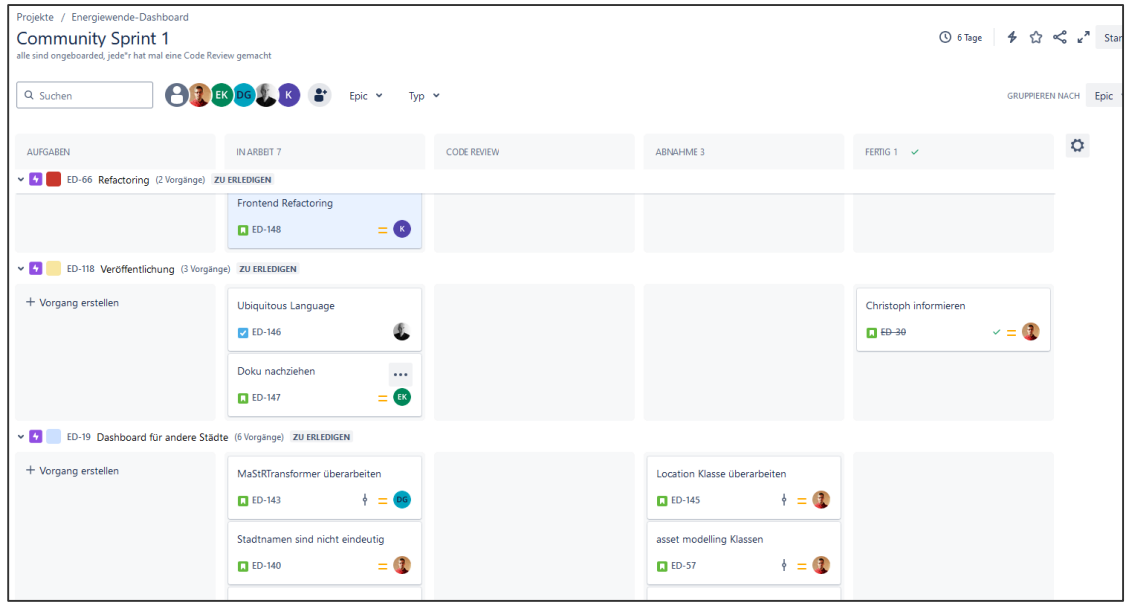
- Container stellen sicher, dass der Code überall läuft – unabhängig von Betriebssystem, lokal installierten Paketen usw.
- Außerdem sind sie eine zusätzliche Sicherheitsschicht im Rechenzentrum

docker-compose.yml 1.15 KiB

```yaml
services:
  backend:
    build: .
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/energiewende_db
      - PAGE_SIZE=25000 # size of buckets for data download
    ports:
      - "8080:8000"
    healthcheck:
      test: ["CMD-SHELL", "curl http://localhost:8000/db_health"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    depends_on:
      db:
        condition: service_healthy
    links:
      - db

  frontend:
    build:
      context: frontend
      args:
        REACT_APP_API_BASE_URI: http://localhost:8080
    ports:
      - "5001:80"
    depends_on:
      backend:
        condition: service_healthy
    links:
      - db


  db:
    image: postgres:13
    environment:
```

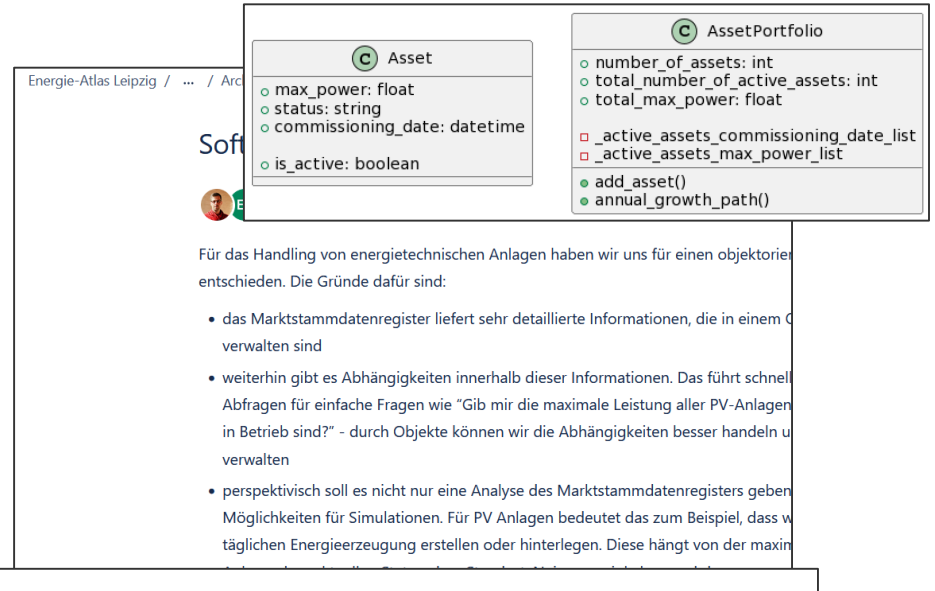
Ein fokussierter Arbeitsprozess

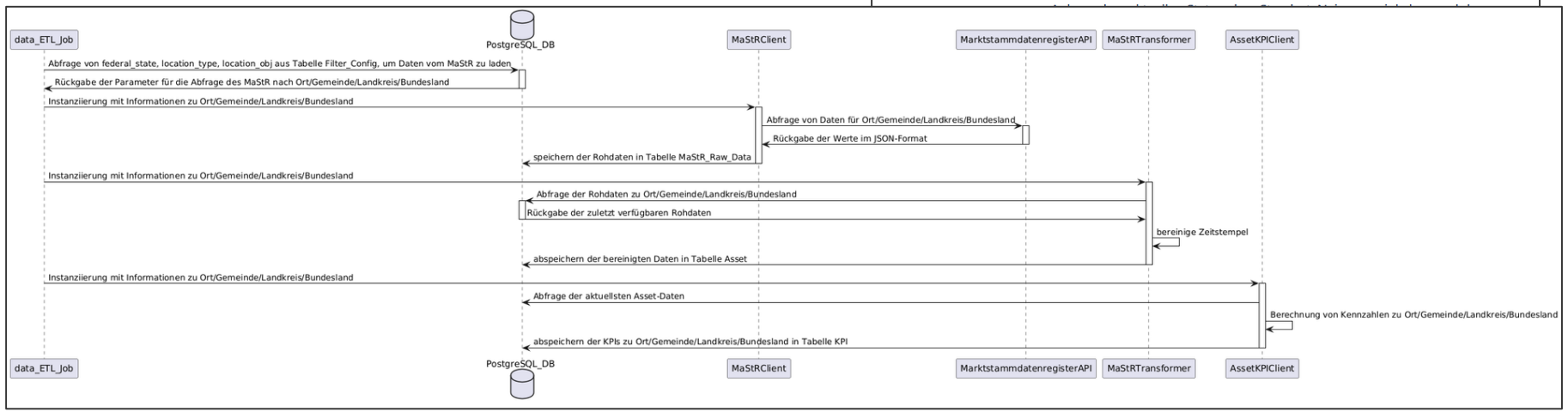
- Wir arbeiten agil in 4 Wochen Sprints
- Das Planning findet gemeinsam statt
- Wir nutzen die üblichen Kommunikationstools
- Die Arbeit ist weitestgehend asynchron organisiert (guter Guide: https://handbook.gitlab.com/handbook/company/culture/all-remote/asynchronous/)



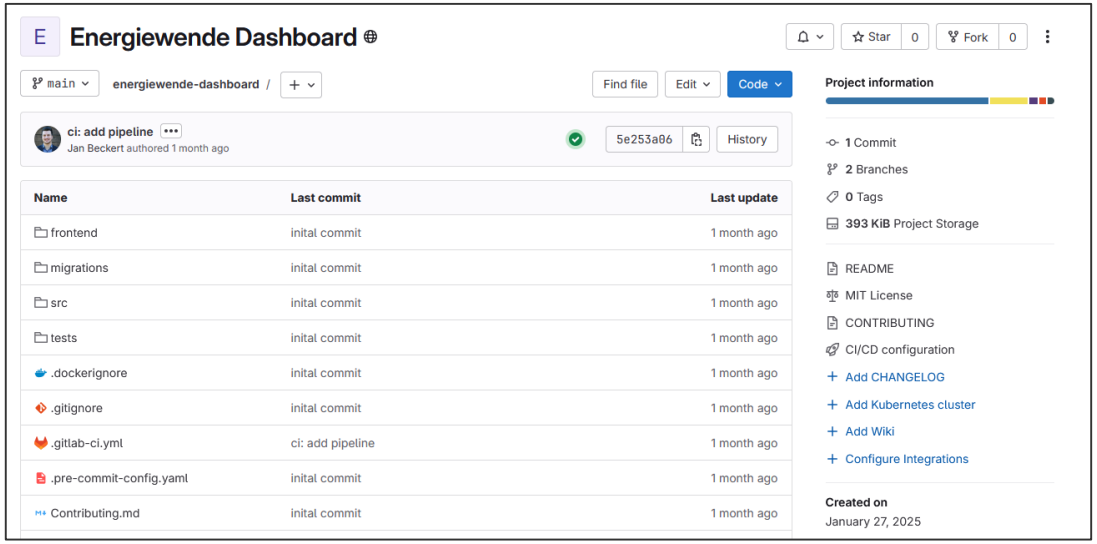
Eine schlanke Doku erleichtert das Leben

- Wir wollen unser Wissen teilen, um langfristig unseren Code weiterentwickeln zu können
- Wir dokumentieren unsere Entwicklungsschritte direkt am Ticket
- Architektur-Entscheidungen werden als ADR (Architectural Decision Record) festgehalten

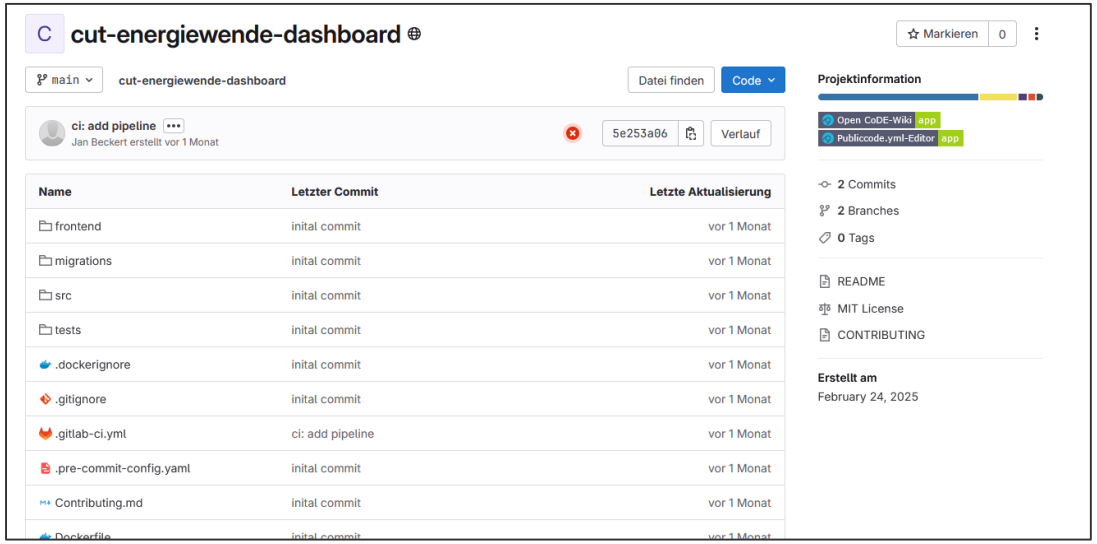




Verfügbar auf GitLab und OpenCoDE



https://gitlab.com/entwicklungscommunity-energiedaten/energiewende-dashboard



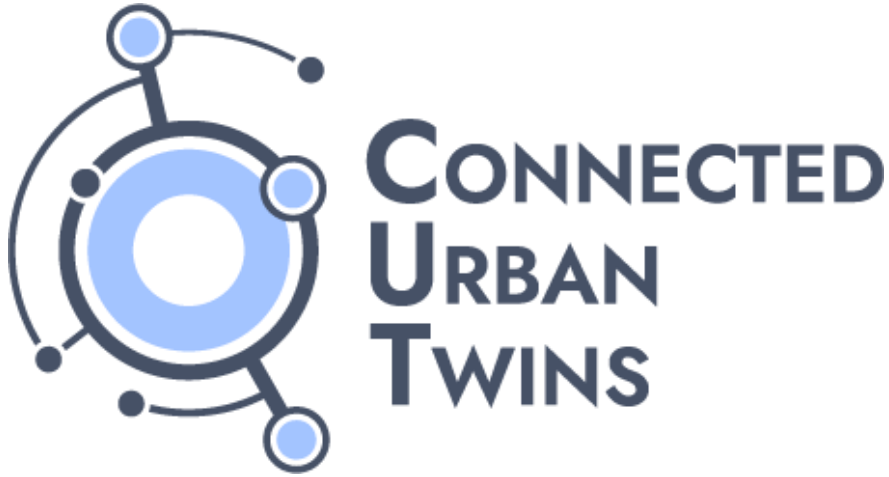
https://gitlab.opencode.de/connected-urban-twins/cut-energiewende-dashboard

Unsere Community lebt weiter

- Fördermittel enden, aber gute Produkte nicht
- Die Entwicklungscommunity lebt weiter und arbeitet an ihrer Vision die passenden Werkzeuge für die Energiewende und eine nachhaltige Zukunft zu gestalten
- Du willst mitmachen?

  Dann sprich uns einfach an ☺

CUT



Vielen Dank!

Partnerstädte:

Hamburg

Stadt Leipzig

Landeshauptstadt München

Gefördert durch:

Bundesministerium für Wohnen, Stadtentwicklung und Bauwesen

KFW