

C02

Präsentiert von:

Patrick Ade

21adpa1bif@hft-stuttgart.de

Emre Gezer

21geem1bif@hft-stuttgart.de

Aaron Mele

21meaa1bif@hft-stuttgart.de

Morten Stetter

22stmo1bif@hft-stuttgart.de

Im Juni 2025

Gruppen-Projekt

In Pervasive-Computing 2025

bei Professor Knauth

Agenda

- Die CO2-Ampel
- Unser Projekt
 - Ziel
 - Aufbau
 - Challenges
 - Demo

Die CO2-Ampel – Hardware & Funktion

Hardware Details:

- ESP8266 (NodeMCU V3)
- CO2-Sensor (Sensirion SCD 30)
- Neopixel LED Ring

Funktion:

- LED-Ring zeigt visualisiert den CO2 Gehalt in der Luft
- Bei über 2200 ppm blinken alle LEDs rot

Ziel:

- CO2 Gehalt darstellen damit ersichtlich ist wann gelüftet werden sollte



Die CO2-Ampel – Ausgangszustand

Sendet Daten in JSON Format:

```
{ "metadata":{  
  "mac-address":"08:3A:8D:E3:DF:AF",  
  "time":"2025-04-10 11:33:21+01"  
},  
  "co2":528,  
  "temp":26.0,  
  "rh":24.2  
}
```

Ursprünglicher Datenzugriff:

1. Zugriff im selben WLAN über `http://sensorID.local` - (z.B. <http://esp8266AA.local>)

Dafür wird ein öffentlich verfügbarer MQTT-Broker verwendet (test.mosquitto.org)

2. Access Point Modus der CO2-Ampel

Die CO2-Ampel wird zum Access Point falls keine WIFI-Verbindung in das WLAN möglich ist

Durch Verbindung mit Access Point ist Zugriff auf gesendete Daten möglich.

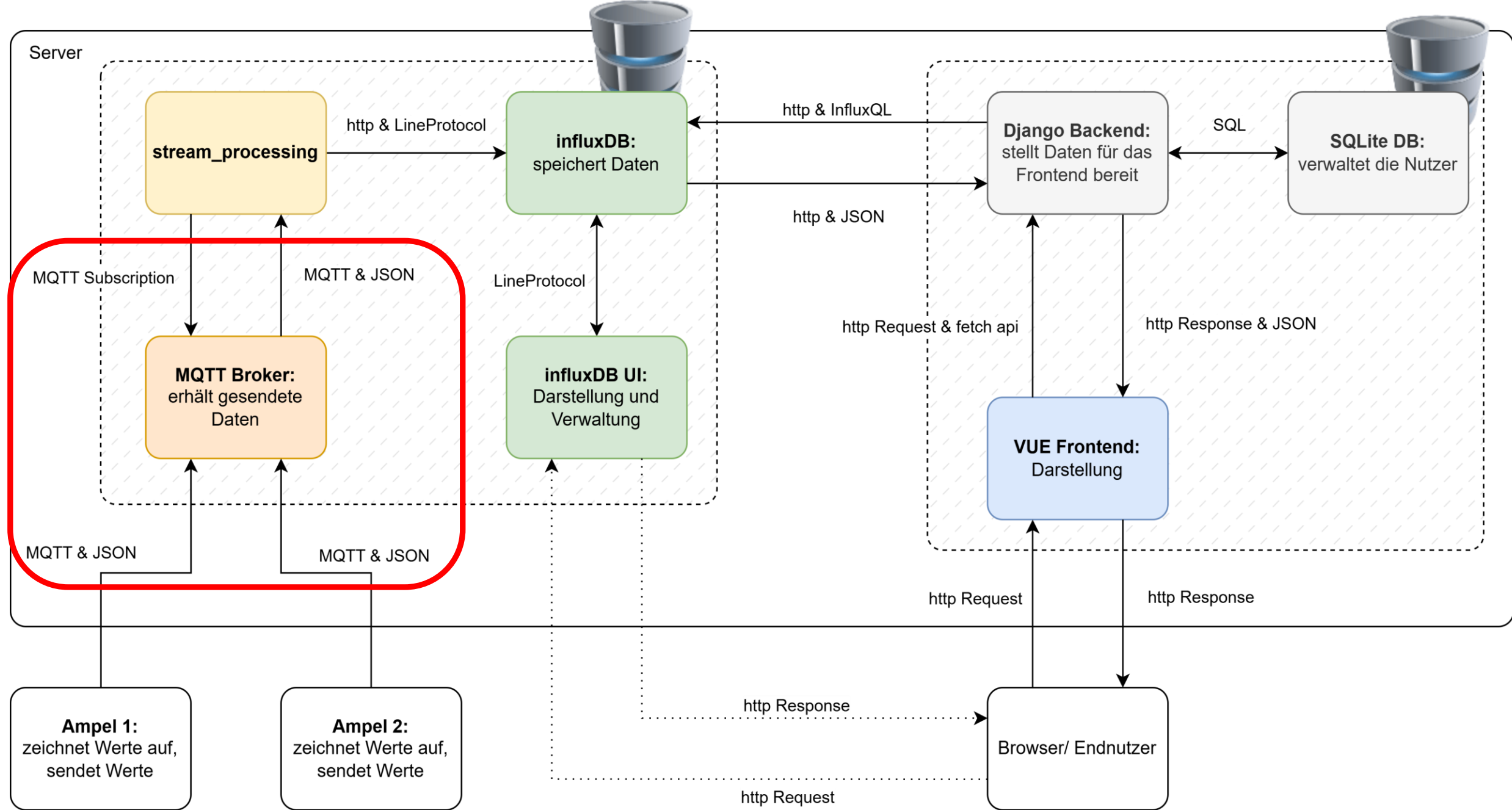
Vom Ausgangszustand zum Ziel



- Öffentlicher MQTT-Broker (test.mosquitto.org)
- Zugriff auf einzelne Ampeln



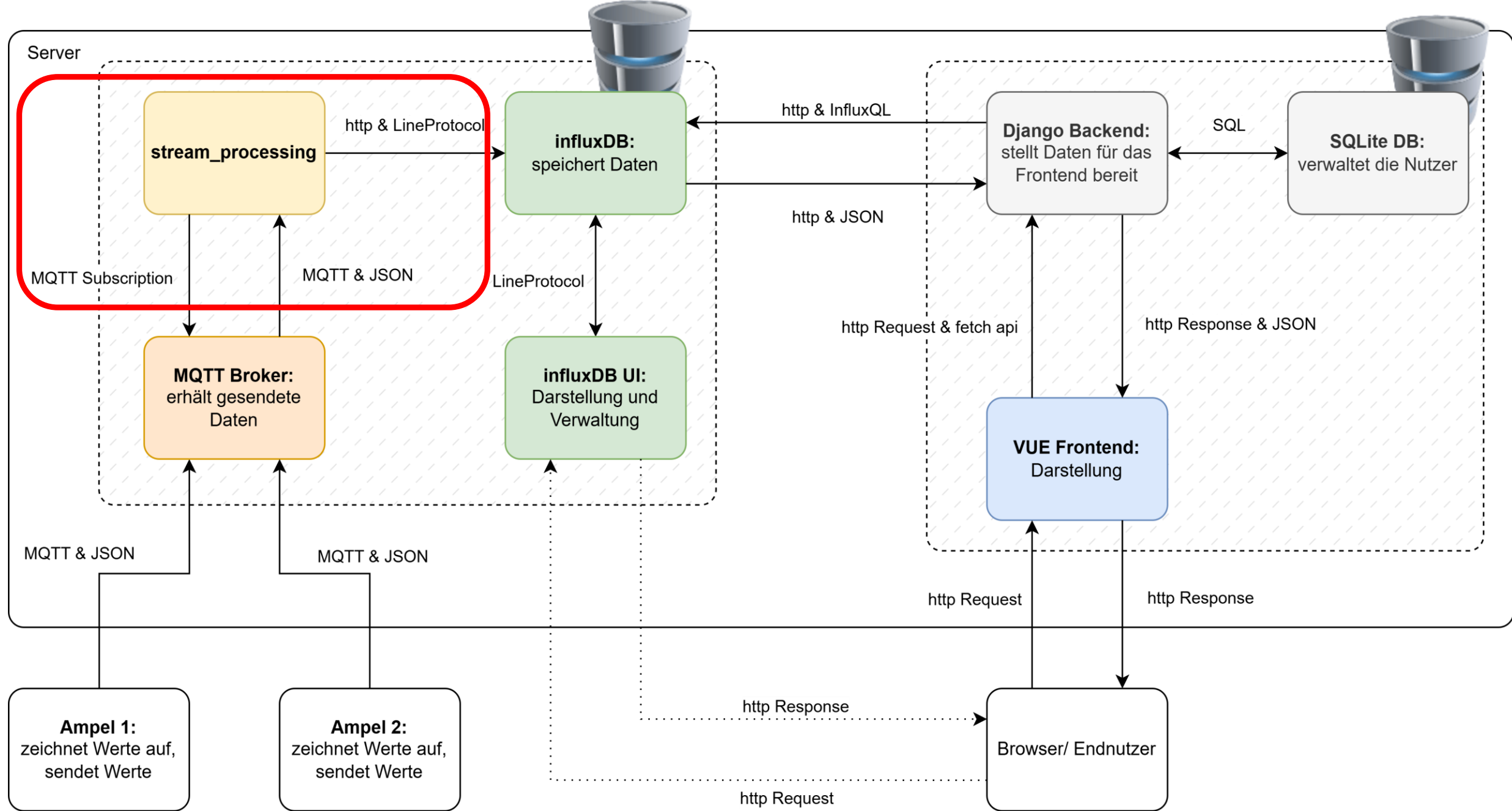
- Dedizierter Server
- Alle Daten aggregiert und gespeichert



MQTT-Broker – Eclipse Mosquitto

- **MQTT (Message Queuing Telemetry Transport)**
 - Protokoll für Nachrichtenübertragung
 - geringe Bandbreite & Energieverbrauch
 - **Clients** (Geräte) kommunizieren über einen **Server** (Broker) – Client/Server
- **Pub/Sub-System**
 - **Publisher**: sendet Nachrichten zu einem „Topic“
 - **Subscriber**: abonniert ein Topic, um relevante Nachrichten zu erhalten
 - **Broker**: verwaltet Topics und leitet Nachrichten weiter
- Open Source Software wie **Eclipse Mosquitto** dient als **Broker**





Stream-Processing

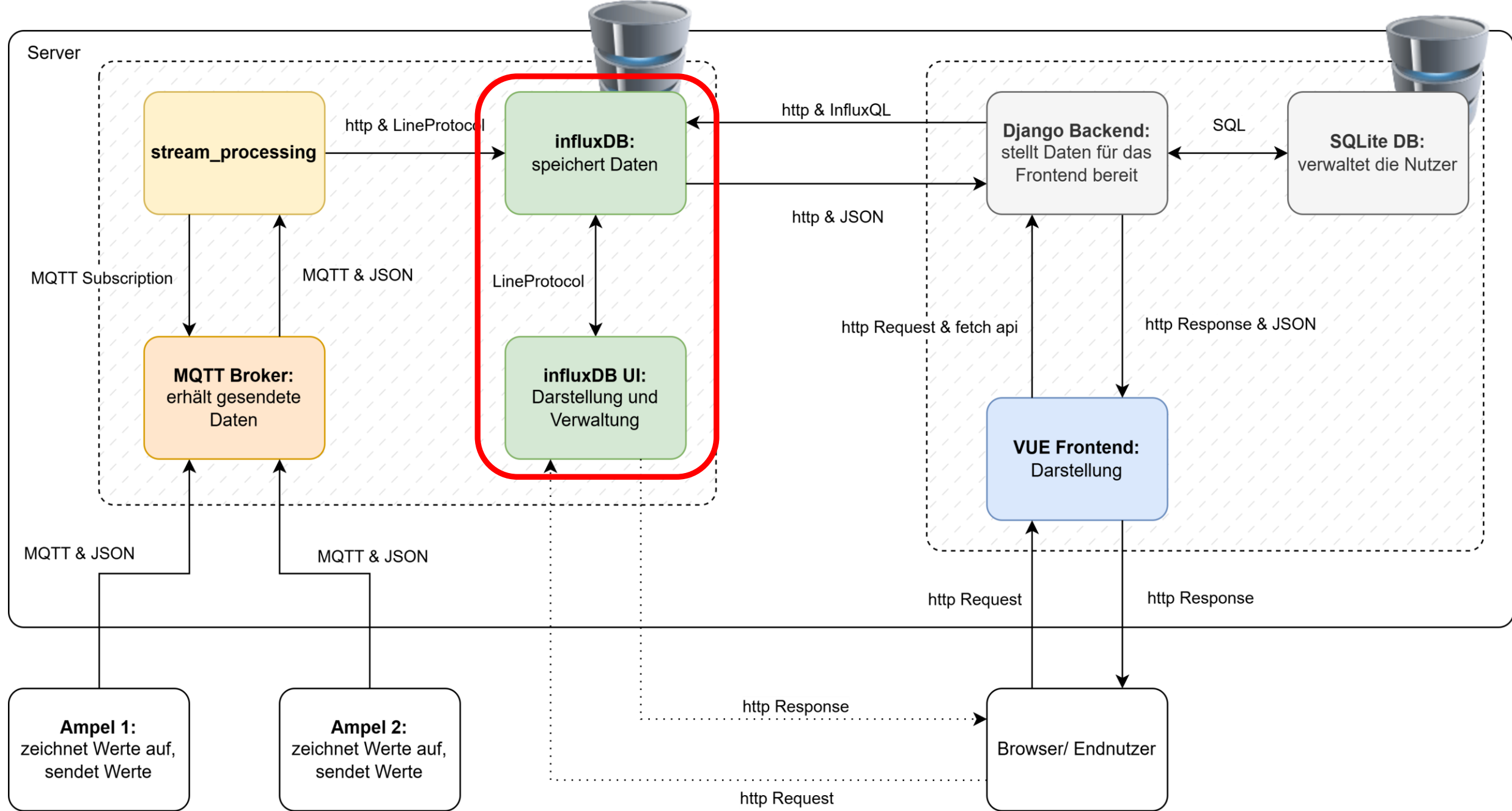
- Subscribed auf das Topic beim MQTT-Broker
- **Transformiert** JSON-formatierte Daten zu DB verträglichen Daten
- Ordnet MAC-Adressen automatisch den zugehörigen HFT-Räumen zu
- Möglichkeit zur Datenmanipulation bevor sie in die DB geschrieben werden (zB. Filtern, Join)

```
{  
  "AC:0B:FB:DA:9A:DB": "1/210",  
  "08:3A:8D:E3:DF:AF": "2/101"  
}
```

JSON to LineProtocol

```
def write_to_influxDB(self, msg: dict, metadata: dict):  
    try:  
        print(msg)  
        self.influx_writer.write_point(  
            measurement=self.MEASUREMENT_NAME,  
            tags={  
                self.TAG_ROOM: self.mac_to_room[metadata["mac-address"]],  
                self.TAG_MAC: metadata["mac-address"],  
            },  
            fields={  
                self.FIELD_CO2: msg["co2"],  
                self.FIELD_TEMP: msg["temp"],  
                self.FIELD_HUMIDITY: msg["rh"],  
            },  
            timestamp=metadata["time"], # fix  
        )  
    .  
    .  
    .
```

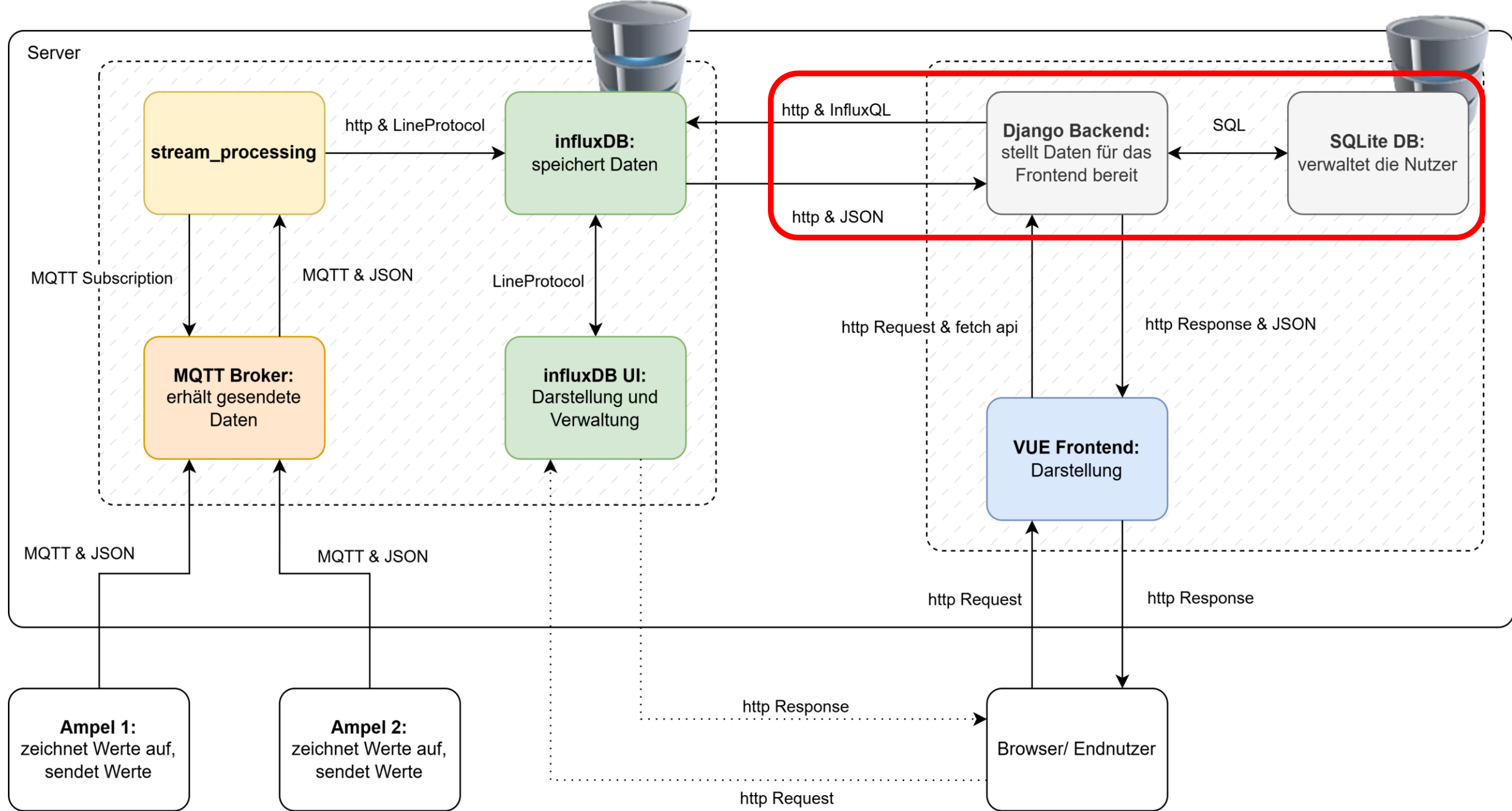
sensor_data,room=1/210,mac=AA:BB Co2=528,temperature=35.0 1465839830100400200
Measurement,——tag_set——— —field_set——— —timestamp———



InfluxDB

Merkmal	InfluxDB (<i>time-series-DB</i>)	SQL-Datenbanken (z. B. <i>MySQL</i> , <i>PostgreSQL</i>)
Datenmodell	Optimiert für time-series (Messwerte mit Timestamps)	Relationales Modell mit Tabellen, Zeilen und Spalten
Leistung bei Zeitreihen	Sehr hohe Performance bei Zeit-basierten Abfragen	Nicht speziell auf time-series optimiert
Einsatzbereiche	Monitoring, IoT, Metriken, Sensoren	Allgemeine Datenhaltung, Transaktionen, Business-Anwendungen

- Bringt eigene Benutzeroberfläche mit



Django Rest API : Endpunkte

```
urlpatterns = [  
    path("api/set_csrf_token/", views.set_csrf_token, name="set_csrf_token"),  
    path("api/login", views.login_view, name="login"),  
    path("api/logout", views.logout_view, name="logout"),  
    path("api/user", views.user, name="user"),  
    path("api/register", views.register, name="register"),  
    :  
    path("api/room_data_range", views.room_data_range, name="room_data_range"),  
    :  
    :  
]
```

Django Rest API: Funktionalität (1)

```
@require_http_methods(["GET"])
def room_data_range(request):
    try:
        room = request.GET.get("room")
        start = request.GET.get("start", "-30d")
        stop = request.GET.get("stop", "now()")

        if not room:
            return JsonResponse(
                {"error": "Missing 'room' parameter"},
                status=400
            )

    load_dotenv()
    client = InfluxDBHelper(
        url=os.getenv("INFLUXDB_URL"),
        token=os.getenv("INFLUXDB_TOKEN"),
        org=os.getenv("INFLUXDB_ORG"),
        bucket=os.getenv("INFLUXDB_BUCKET"),
    )

    tables = client.get_room_data_in_range(room, start, stop)
```

„http://URL/api/
room_data_range?
room=1/210
&start=-7d
&stop=now()”

Influx Query Language

```
def get_room_data_in_range(self, room_id: str, start: str = "-30d", stop: str =  
    "now()"):  
  
    query = f'''  
        from(bucket: "{self.bucket}")  
        > range(start: {start}, stop: {stop})  
        > filter(fn: (r) => r["_measurement"] == "sensor_data")  
        > filter(fn: (r) => r["room"] == "{room_id}")  
        > filter(fn: (r) => r["_field"] == "co2" or r["_field"] == "humidity" or  
r["_field"] == "temperature")  
    '''  
  
    return self.query_api.query(org=self.org, query=query)
```

|>

Django Rest API : Funktionalität (2)

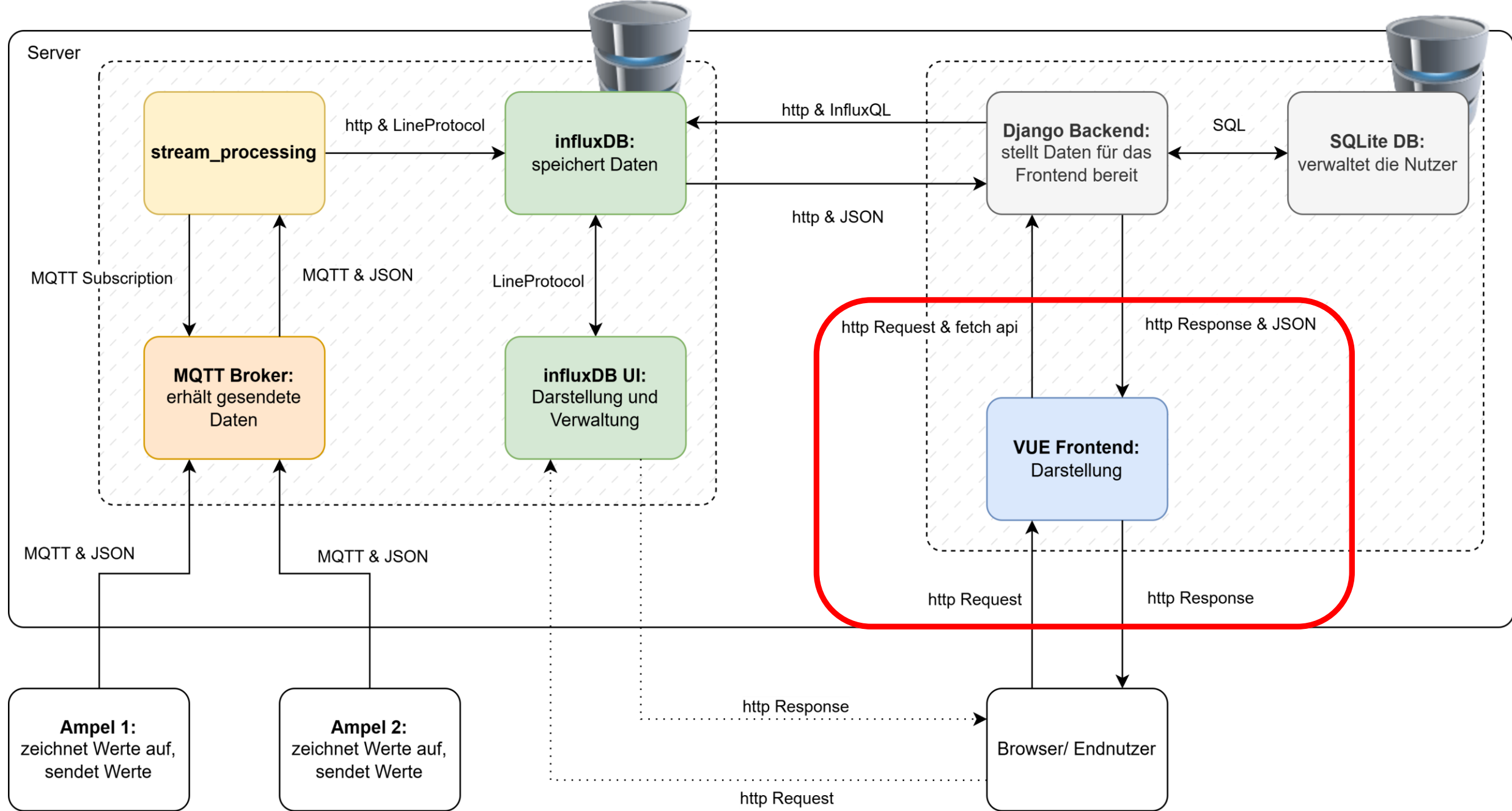
```
data_by_time = {}

for table in tables:
    for record in table.records:
        timestamp = str(record.get_time())
        field = record.get_field()
        value = record.get_value()

        if timestamp not in data_by_time:
            data_by_time[timestamp] = {}
        data_by_time[timestamp][field] = value

return JsonResponse({
    "room": room,
    "data": data_by_time,
    status=200})
except json.JSONDecodeError:
    return JsonResponse(
        {"success": False, "message": "Invalid JSON"},
        status=400
    )
```

```
{
  "room": "1/210",
  "data": {
    "2025-06-04 09:08:00+00:00": {
      "co2": 528,
      "humidity": 24.2,
      "temperature": 27.0
    },
    "2025-06-04 09:15:00+00:00": {
      "co2": 528,
      "humidity": 24.2,
      "temperature": 28.0
    }
  }
}
```



Vue: Struktur und Funktion

```
src:
|
+---components:
|   |   NavBar.vue
|
+---router:
|   |   index.ts
|
+---views:
|   |   ChartView.vue
|
+---App.vue
```

```
index.ts :

import ChartView from '../views/ChartView.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/charts',
      name: 'charts',
      component: ChartView
    },
    .
    .
    other views
  ]
})
```

Vue: Aufruf der REST API

```
onMounted(async () => {  
  try {  
    const url =  
      `http://localhost:8000/api/room_data_range?room=${encodeURIComponent(room)}&start=-7d&stop=now()`  
    const response = await fetch(url)  
  
    if (!response.ok) {  
      throw new Error(`HTTP-Fehler: ${response.status}`)  
    }  
  
    const json = await response.json()  
    const entries = Object.entries(json.data).sort(  
      ([a], [b]) => new Date(a).getTime() - new Date(b).getTime()  
    )  
    .  
    .  
    .  
    Data processing  
  }  
})
```



Data Explorer

d



Graph

CUSTOMIZE

Local

SAVE AS



Query 1 (0.07s)

+

View Raw Data

CSV



Past 3h

SCRIPT EDITOR

SUBMIT

FROM

Search buckets

co2-test

_monitoring

_tasks

+ Create Bucket

Filter

_measurement

Search _measurement tag va

✓ sensor_data

Filter

_field

Search _field tag values

✓ co2

humidity

temperature

Filter

room

Search room tag values

✓ 1/218

2/101

WINDOW PERIOD

CUSTOM

AUTO

auto (1m)

Fill missing values

AGGREGATE FUNCTION

CUSTOM

AUTO

mean

median

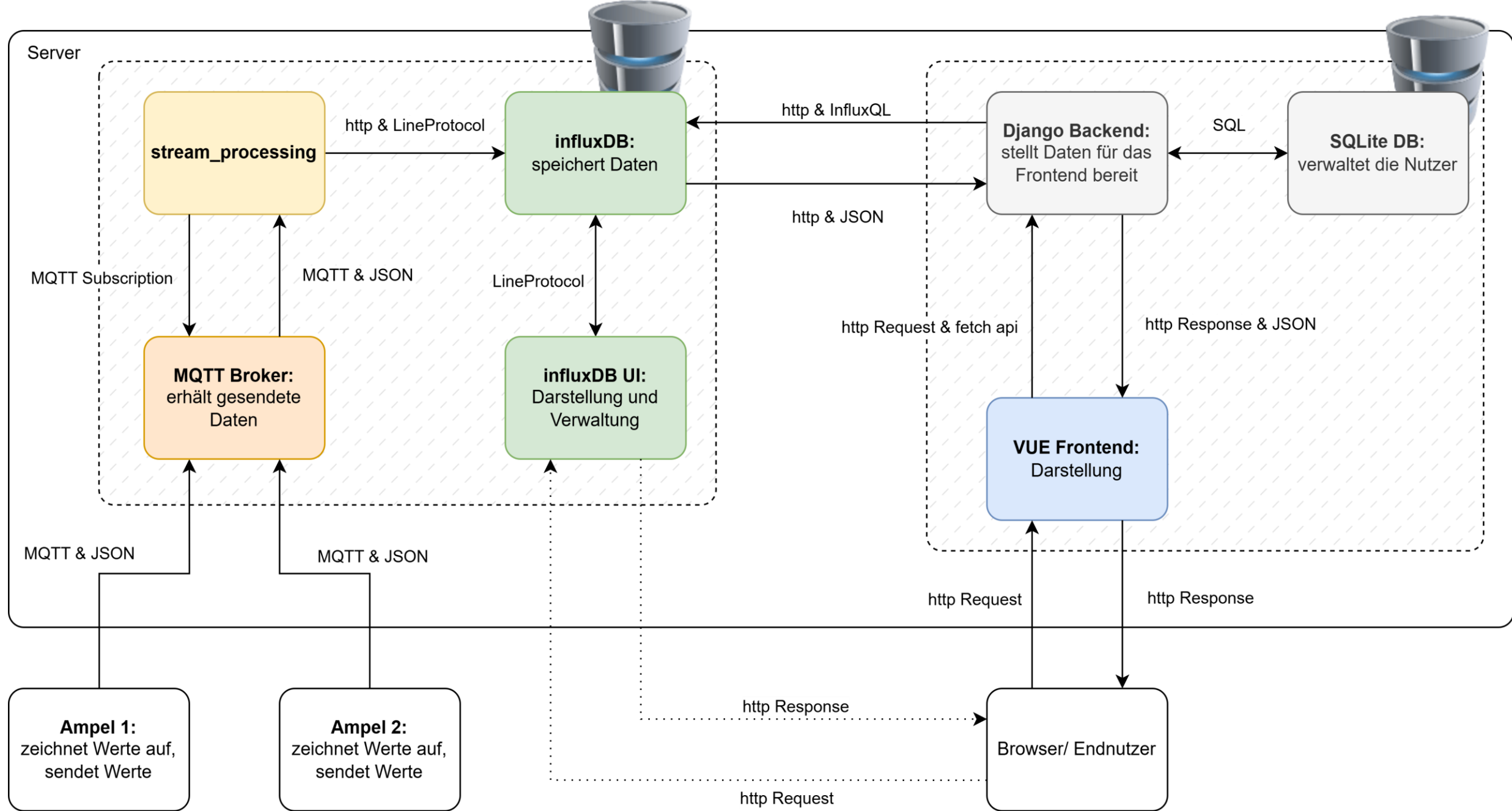
last

Offene Themenpunkte

- Frontend
 - Ampel für Sensor Activity (Sendet oder ist offline)
 - CSV Generierung einbinden / im Backend fixen
- Deployment und SSH

Quellen

- <https://mosquitto.org/>
- <https://transfer.hft-stuttgart.de/gitlab/co2ampel/ampel-documentation>



Änderungen im Backend

- Django Ninja
- Swagger
- JWT Token

NinjaAPI 1.0.0 OAS 3.1
[/api/openapi.json](#)

user

POST /api/login Login View

GET /api/user User View

POST /api/register Register View

sensor

GET /api/room_data_range Room Data Range

GET /api/get_rooms Get Rooms

Frontend

- Neue Funktion
- Anzeige des Sende-Status