Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

DATAINSAMLING OCH VISUALISERING AV MQTT-data

Elina Hansson, Konstantin Karpov

2023:04

Datum för godkännande: 23.05.2023

Handledare: Björn-Erik Zetterman

EXAMENSARBETE

Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Elina Hansson, Konstantin Karpov
Arbetets namn:	Datainsamling och visualisering av MQTT-data
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	Abra Automation

Abstrakt

I det här examensarbetet har vi genomfört en insamling och överföring av data från ett befintlig PLC-system, skapat databasstrukturer för lagring av data samt implementerat mjukvara för visualisering av datan. Syftet var att kunna utvärdera systemets effektivitet samt användarnas beteende. Vi strävade efter att utveckla en lösning som skulle kunna tillämpas på andra system som arbetar med MQTT-data, utan att vara beroende av ett specifikt system eller en specifik tjänst.

För att uppnå detta mål har vi utvecklat en mikrotjänstarkitektur som består av en MQTT-broker, en MQTT-klient, en databas, ett API och en webbserver för visualiseringen.

Nyckelord (sökord)
MQTT, docker, bokeh, Node-RED

Högskolans serienummer:	**ISSN:**	**Språk:**	**Sidantal:**
2023:04	1458-1531	Svenska	59 sidor

Inlämningsdatum:	**Presentationsdatum:**	**Datum för godkännande:**
03.05.2023	16.05.2023	23.05.2023

DEGREE THESIS

Åland University of Applied Sciences

Degree Programme:	Information Technology
Author:	Elina Hansson, Konstantin Karpov
Title:	Data collection and visualization of MQTT data
Academic Supervisor:	Björn-Erik Zetterman
Commissioned by:	Abra Automation

Abstract

In this thesis, we conducted data collection from a PLC in an existing system, created database structures for data storage and created a visualization of the data. The aim was to evaluate the efficiency of the system and analyze user behavior. We aimed to develop a solution that could be applied to other systems working with MQTT data, without being dependent on a specific system or service.

To achieve this goal, we developed a microservices architecture consisting of an MQTT broker, an MQTT client, a database, an API, and a web server for visualization.

Keywords
MQTT, docker, bokeh, Node-RED

Serial number:	**ISSN:**	**Language:**	**Number of pages:**
2023:04	1458-1531	Swedish	59 pages

Handed in:	**Date of presentation:**	**Approved:**
03.05.2023	16.05.2023	23.05.2023

INNEHÅLLSFÖRTECKNING

1. INLEDNING

I detta kapitel beskriver vi syftet med projektet, vilken metod vi har använt oss av för att genomföra det, samt eventuella avgränsningar vi har gjort.

1.1 Syfte

Syftet med detta projekt var att analysera det befintliga systemets, en sopsugsstations (se kapitel 2.1), prestanda och effektivitet i verkliga händelser genom att samla och visualisera data från en existerande PLC-miljö. Dessutom ville vi kunna analysera användarnas beteende för att optimera systemet. Projektet har fokuserat på data från en sopsugsstation, men målet är att implementationen ska vara anpassningsbar för liknande användningsfall utan att vara beroende av specifika plattformar såsom molntjänster. Uppdragsgivaren är Abra Automation, ett företag baserat på Åland som specialiserar sig inom industriell automation.

1.2 Metod

Då det inte fanns exakta krav på genomförandet från uppdragsgivaren så kan vår valda arbetsmetod bäst beskrivas som “Evolutionary Prototyping” (Wikipedia contributors, 2022). Detta innebar att vi skapade en robust prototyp och sedan förbättrade den successivt allteftersom vi upptäckte behovet av ytterligare förbättringar under projektets gång (Hansen, 2022), (Björnvik, 2021).

Vi började projektet med ett uppstartsmöte med uppdragsgivaren, varefter vi kunde sammanställa en kravspecifikation baserat på vad som gicks igenom på mötet. Under projektets gång har vi fått hjälp av uppdragsgivaren med frågor som dykt upp angående PLC och Node-RED, men resterande arbete har skötts på egen hand med hjälp av regelbundna tillfällen med vår handledare.

Uppdragsgivaren gav oss möjligheten att lära oss grunderna i PLC-programmering (se kapitel 2.7) genom att erbjuda en kurs under projektets gång.

1.3 Avgränsningar

Från början hade vi tänkt att vi inom projektet skulle hinna med:

- Datainsamling
- Visualisering av data
- Analysering av data
- Skapa kod för att själva kunna korrigera PLC:n

Tidigt i projektet blev det dock uppenbart att vi hade åtagit oss en uppgift som var för omfattande, med hänsyn till den begränsade tidsramen för genomförandet. Dessutom var vi relativt nya på många av de ramverk och program som krävdes för att genomföra projektet. Som en konsekvens av detta beslutade vi att **begränsa vårt fokus till datainsamling, infrastruktur för databehandling och visualisering** av den erhållna datan.

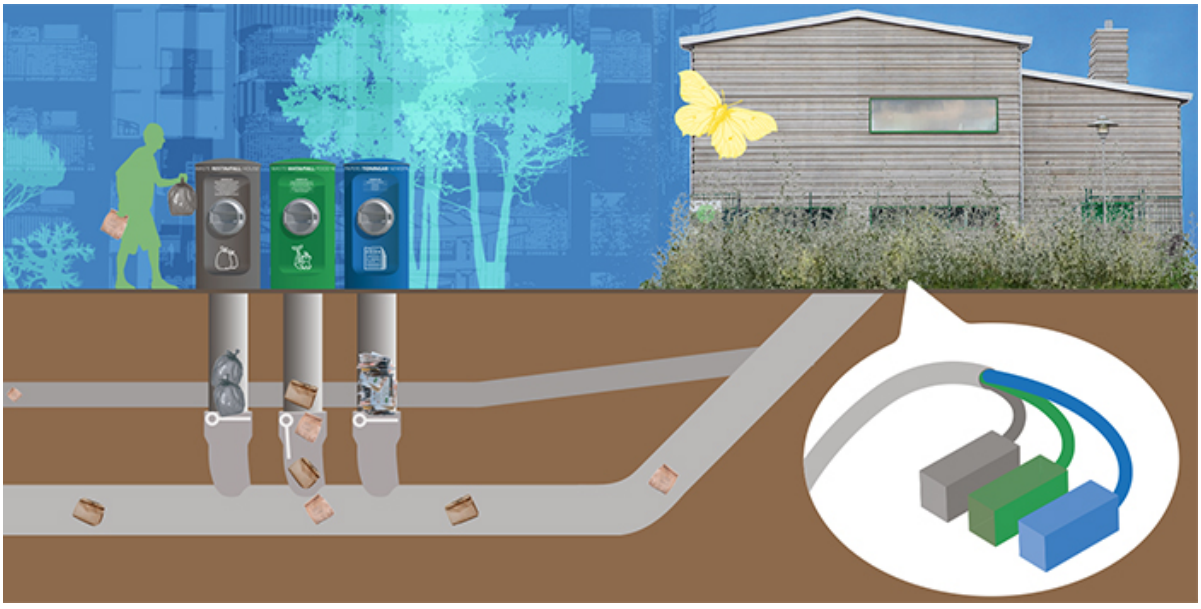
Det var även en önskan från uppdragsgivaren att datan skulle presenteras i en mobilapplikation. Dock kom vi överens om att detta inte skulle inkluderas i det nuvarande projektet redan i samband med kravspecifikationen.

2. TEORI

I detta kapitel förklarar vi de definitioner som används i vårt projekt.

2.1 Sopsug

En sopsug är ett system för avfallshantering som används i större bostadsområden, kontorskomplex och andra platser där stora mängder avfall genereras (*Så fungerar sopsugen*, 2020). Se figur 1.



Figur 1.Sopsug (Nu öppnar hallonbergens sopsug, n.d.)

En sopsug fungerar genom att avfallet sugs upp av en kraftfull fläkt genom en behållare som kallas inmatningshuvud. Detta skapar ett undertryck som drar in avfallet genom rörsystemet och transporterar det genom en luftkanal till en central anläggning för sortering och behandling. Luftkanalen är designad för att minimera friktion och maximera luftflödet för att möjliggöra höga hastigheter och effektiv transport av avfallet (*Så fungerar sopsugen*, 2020)

När avfallet når den centrala anläggningen sorteras det och behandlas på olika sätt beroende på avfallstyp och lokal lagstiftning. Det återvinningsbara materialet separeras från icke-återvinningsbart material, till exempel genom mekanisk eller kemisk separation, medan

det icke-återvinningsbara materialet behandlas vidare, till exempel genom förbränning (*Så fungerar sopsugen*, 2020).

2.2 Mikrotjänstarkitektur

Vi har bestämt att utveckla en lösning som bygger på mikrotjänstarkitektur, där varje mikrotjänst är ansvarig för en specifik uppgift inom systemet. Varje mikrotjänst är självständig och kan utvecklas, distribueras och skalas separat. Detta gör att nya funktioner och förbättringar kan implementeras snabbt och enkelt utan att påverka andra delar av systemet. Dessutom minskar användningen av mikrotjänster risken för att ett fel i en tjänst påverkar hela systemet. Nackdelarna med mikrotjänster inkluderar svårigheter med informationsöverföring mellan tjänsterna, komplexitet i testning och distribution (Wikipedia contributors, 2023d).

2.3 IoT

IoT står för "Internet of Things" och är en teknik som möjliggör anslutning av enheter och sensorer till internet för att samla in, överföra och analysera data. Med IoT-teknologi kan enheter och sensorer kopplas samman i ett nätverk för att samla in och utbyta data. En viktig del av IoT är molntjänster som möjliggör hantering och analys av den stora mängd data som samlas in från enheterna. Molntjänster ger också möjlighet att distribuera programvaruuppdateringar och hantera enheter på distans.

Exempel på IoT-applikationer inkluderar fjärrövervakning av medicinska tillstånd, smarta hemlösningar, fjärrstyrning av fordon och övervakning av energiförbrukning (AWS, n.d.).

IoT kan också benämnas som IIoT för att specifikt betona dess användning inom industriella tillämpningar (Wikipedia contributors, 2023b).

2.4 AWS

AWS (Amazon Web Services) är en molnplattform som tillhandahåller en mängd olika tjänster som möjliggör lagring, databashantering, beräkning, nätverk, AI och IoT-lösningar i molnet. De uppräknade tjänsterna kan sammanfattas som “Public Cloud Services” vilket

innebär att tjänsterna är tillgängliga över internet och ger kunderna flexibilitet och skalbarhet för att skapa och driva IT-infrastrukturer på en global skala. AWS har också en global infrastruktur med hög tillgänglighet och som garanterar hög driftsäkerhet och säkerhet för kundernas data (*Cloud Computing Services*, n.d.).

2.4.1 IoT Core

AWS IoT Core är en molntjänst från Amazon, som möjliggör insamling och bearbetning av data från IoT[1]-enheter. Tjänsten stödjer flera protokoll och gör det enkelt att hantera stora mängder data från flera enheter. Tjänsten AWS IoT Core inkluderar även regler för att vidarebefordra data till andra AWS-tjänster och stöder integration med andra AWS-tjänster som Lambda, Kinesis och S3 för att bygga skalbara IoT-lösningar (*AWS IoT Core*, n.d.).

2.4.2 Elastic Container Service

AWS Elastic Container Service (ECS) är en tjänst som gör det enkelt att köra och skalera Docker-containrar (se kapitel 2.5) i molnet. Med AWS ECS kan användaren köra applikationer utan att behöva hantera containrarnas underliggande infrastruktur, vilket sparar tid och minskar komplexiteten (AWS ECS, n.d.).

2.4.3 Amazon RDS

Amazon RDS (Relational Database Service) är en molnbaserad databastjänst från Amazon Web Services (AWS) som hjälper användaren att installera och driva relationella databaser (se kapitel 2.14.1) i molnet. Amazon RDS ger utvecklaren verktyg att skapa, hantera och skala upp eller ner relationella databaser som MySQL, PostgreSQL, Oracle, SQL Server och MariaDB. Tjänsten tar hand om uppgifter som databashantering, säkerhet, säkerhetskopiering och återställning (*Amazon RDS*, n.d.).

2.4.4 Amazon EC2

En AWS EC2-instans är en virtuell maskin i molnet som kan skapas och konfigureras enligt behov. Instanser kan startas eller stoppas på begäran, och användare betalar endast för den tid

[1] Internet of Things

som instansen faktiskt använder. Användare kan också välja vilken typ av instans som passar deras behov bäst, beroende på prestanda och lagringskapacitet (*AWS EC2 Instance*, n.d.).

AWS EC2-instanser kan också konfigureras med olika säkerhetsfunktioner och nätverksinställningar för att skydda applikationer och data från obehörig åtkomst. Genom att använda EC2 kan användare enkelt skala upp eller ner sina resurser efter behov utan att behöva hantera fysisk maskinvara eller serverunderhåll (*AWS EC2 Instance*, n.d.).

2.5 Docker

Docker är en plattform som används för att skapa och köra applikationer i isolerade miljöer, kallade containrar. Genom att använda containrar blir applikationerna mer portabla och enklare att flytta från en miljö till en annan. Docker automatiserar också processerna för att bygga, distribuera och köra containrar vilket förenklar hanteringen av applikationer (Docker Overview, 2023).

Docker är också en säkrare och mer resurseffektiv miljö att köra applikationer i, eftersom varje container är isolerad från andra containrar och från värdmaskinen. Detta minskar risken för konflikter mellan applikationer, och ger en mer förutsägbar miljö för applikationerna att köras i (*Docker Overview*, 2023).

2.5.1 Docker-Image

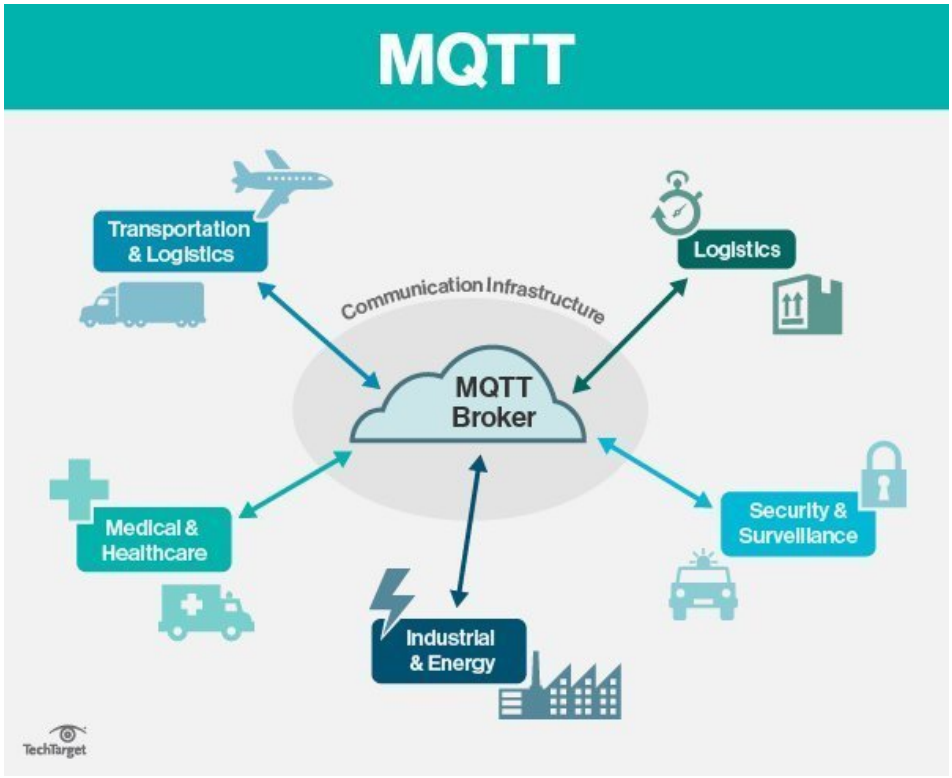
En Docker-image är en färdigpaketerad miljö som innehåller allt som behövs för att köra en applikation, inklusive koden, runtime, systemverktyg, bibliotek och inställningar. Det är en lösning för att snabbt och enkelt distribuera en applikation i en enhetlig och portabel miljö över olika system och miljöer.

Docker-images kan skapas från en Dockerfile, som innehåller en beskrivning av hur applikationen ska paketeras, vilket gör det möjligt att återskapa samma miljö på andra system. Docker-images kan delas och distribueras via ett centralt repository som Docker Hub (Dock, n.d.).

2.6 MQTT

MQTT (Message Queuing Telemetry Transport) är ett meddelandeprotokoll som möjliggör tillförlitlig och effektiv kommunikation mellan olika enheter och plattformar. MQTT kan även fungera på nätverk med låg bandbredd och hög fördröjning. De egenskaperna, i samband med att det stöds av ett brett utbud av programvarubibliotek och verktyg, gör det till ett populärt val för IoT- och M2M-applikationer (*MQTT Version 3.1.1*, n.d.; Wikipedia contributors, 2023c).

MQTT använder en publikations-prenumerations-modell (eng publish-subscribe model) för meddelandeöverföring, där enheter kan publicera meddelanden till ett visst ämne (topic) och andra enheter kan prenumerera på samma ämne för att få meddelanden. Mellan enheterna finns en MQTT-broker som säkerställer att all kommunikation skickas till rätt ställe (Wikipedia contributors, 2023c). Se figur 2.



Figur 2.Visualisering av MQTT-kommunikation (Bernstein et al., 2021).

2.6.1 MQTT-Broker

En MQTT-broker är en server eller ett program som agerar mellanhand mellan MQTT-klienter som är anslutna till den. Broker-tjänsten tar emot meddelanden från klienterna, sparar dem och vidarebefordrar dem till alla andra klienter som har prenumererat på samma ämne. MQTT-brokern är ansvarig för att hantera alla meddelanden och data som skickas mellan klienterna, och garanterar att meddelandena levereras till rätt mottagare. Brokern kan också hantera frågor om tillgänglighet och säkerhet, till exempel autentisering och auktorisering av klienter (Wikipedia contributors, 2023c).

2.6.1.1 Eclipse Mosquitto

Eclipse Mosquitto är en open source MQTT-broker skapad av Eclipse Foundation. Det är en lättviktig MQTT-broker som har inbyggt stöd för säkerhet och som kan köras på olika plattformar (*Eclipse Mosquitto*, 2018).

2.6.2 MQTT-Klient

En MQTT-klient är en enhet eller programvara som använder MQTT-protokollet för att ansluta till en MQTT-broker och skicka eller ta emot meddelanden. Klienten kan vara en sensor, en mobilapp, en webbapplikation eller en annan typ av enhet eller programvara som har implementerat MQTT-protokollet. Beroende på applikationens behov kan MQTT-klienter välja olika nivåer av pålitlighet när de skickar meddelanden. MQTT-klienter kan ansluta till MQTT-brokern med hjälp av olika protokoll, t.ex. TCP/IP, SSL/TLS eller WebSockets, och kan autentisera sig med hjälp av användarnamn och lösenord eller andra autentiseringsmetoder som stöds av MQTT-brokern (Wikipedia contributors, 2023c).

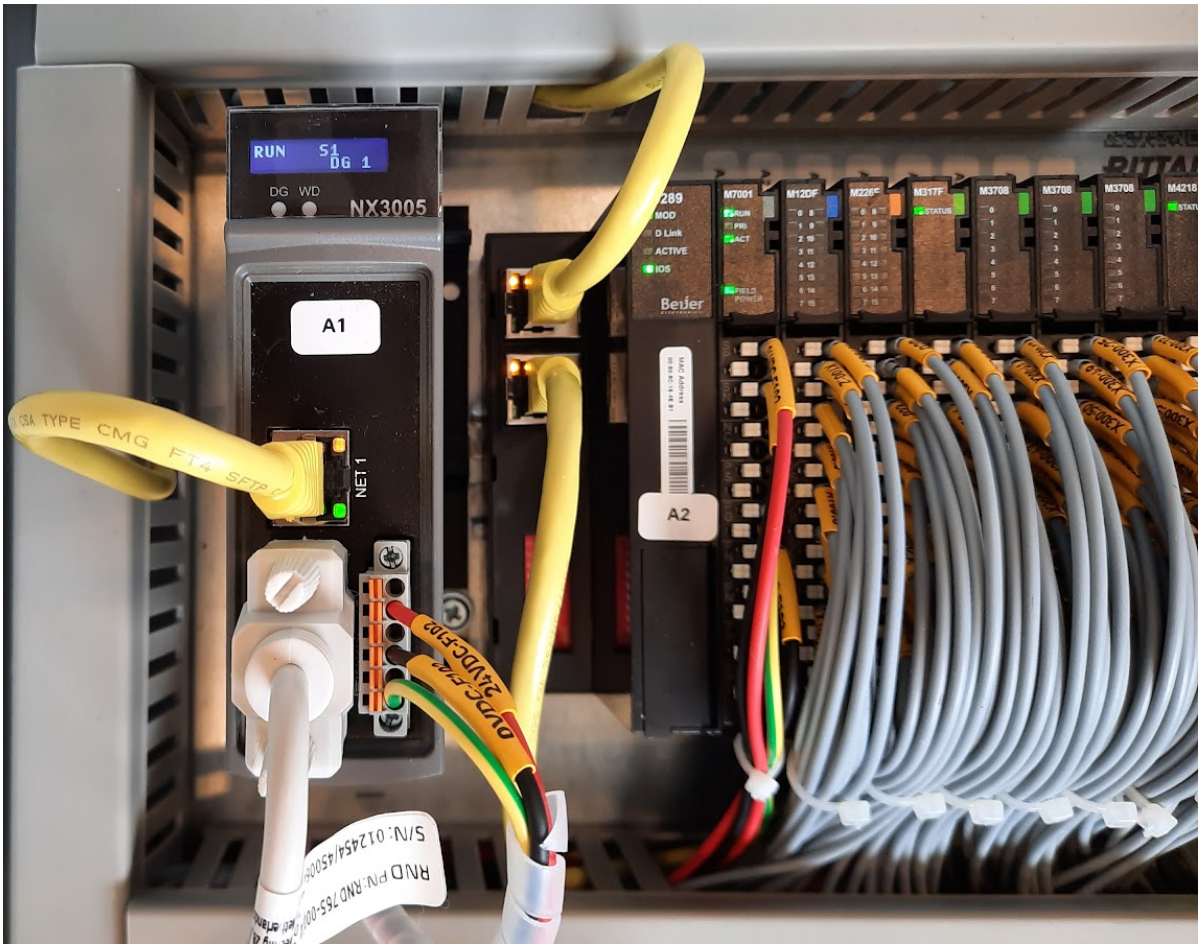
2.6.2.1 Eclipse Paho

Eclipse Paho är ett open source-bibliotek med inbyggd funktionalitet för en MQTT-klient. Det stöder flera plattformar och programmeringsspråk och kan hantera olika nivåer av säkerhet (*Eclipse Paho*, 2013).

2.7 PLC

Programmable Logic Controller (PLC) är en dator som används för att styra automatiserade processer i industrianläggningar och maskiner. PLC:er är utformade för att vara pålitliga och tåliga i miljöer med hög elektrisk störning, vibrationer, höga och låga temperaturer och andra tuffa förhållanden. Driftsäkerhet är viktigt för de tillämpningar PLC:er används för (*What Is a Programmable Logic Controller (PLC)?*, n.d.).

En PLC består av en processor, minne, in- och utgångar och ett programmerbart gränssnitt. En PLC kan hantera flera digitala och analoga ingångar och utgångar, olika kommunikationsgränssnitt (modbus, ethernet), och reagera på förändringar av indata i realtid så att utdata genast ändras. Ett exempel på en PLC visas i figur 3.



Figur 3. PLC i ett elskåp

PLC:er används i en mängd olika industriella applikationer, inklusive tillverkning, elproduktion, byggnadsautomation, vatten- och avloppsrening. PLC:er möjliggör automatisering av processer och system, vilket ökar effektiviteten, minskar kostnaderna och förbättrar säkerheten (Wikipedia contributors, 2023f).

2.7.1 HMI-gränssnitt

Ett HMI-gränssnitt (Human Machine Interface) är en teknologi som används för att skapa en kommunikationsplattform mellan en människa och en maskin. HMI-gränssnitt kan innefatta en mängd olika teknologier, som till exempel skärmar, knappar, röstkommandon, geststyrning eller tangentbord. Genom HMI-gränssnittet kan användaren ge instruktioner eller kommandon till maskinen och få feedback på resultatet av sina handlingar (*Human-Machine Interface (HMI) - Glossary*, n.d.).

HMI-gränssnitt är vanligt förekommande inom många olika branscher och användningsområden, som till exempel inom industriautomation, medicinteknik, bilteknik och datorsystem. Ett väl utformat HMI-gränssnitt kan hjälpa till att öka effektiviteten, minska fel och förbättra användarupplevelsen.

2.7.2 CODESYS

CODESYS är en mjukvara som används för att programmera automations- och styrsystem. Mjukvaran är utvecklad för att vara plattformsoberoende och stödjer en mängd olika operativsystem och hårdvaruplattformar. I CODESYS kan användaren skapa, testa och implementera programvarulösningar för industriella applikationer. Mjukvaran inkluderar en integrerad utvecklingsmiljö (IDE), en online-debugger och en inbyggd simulering av systemet (*CODESYS Group*, 2023).

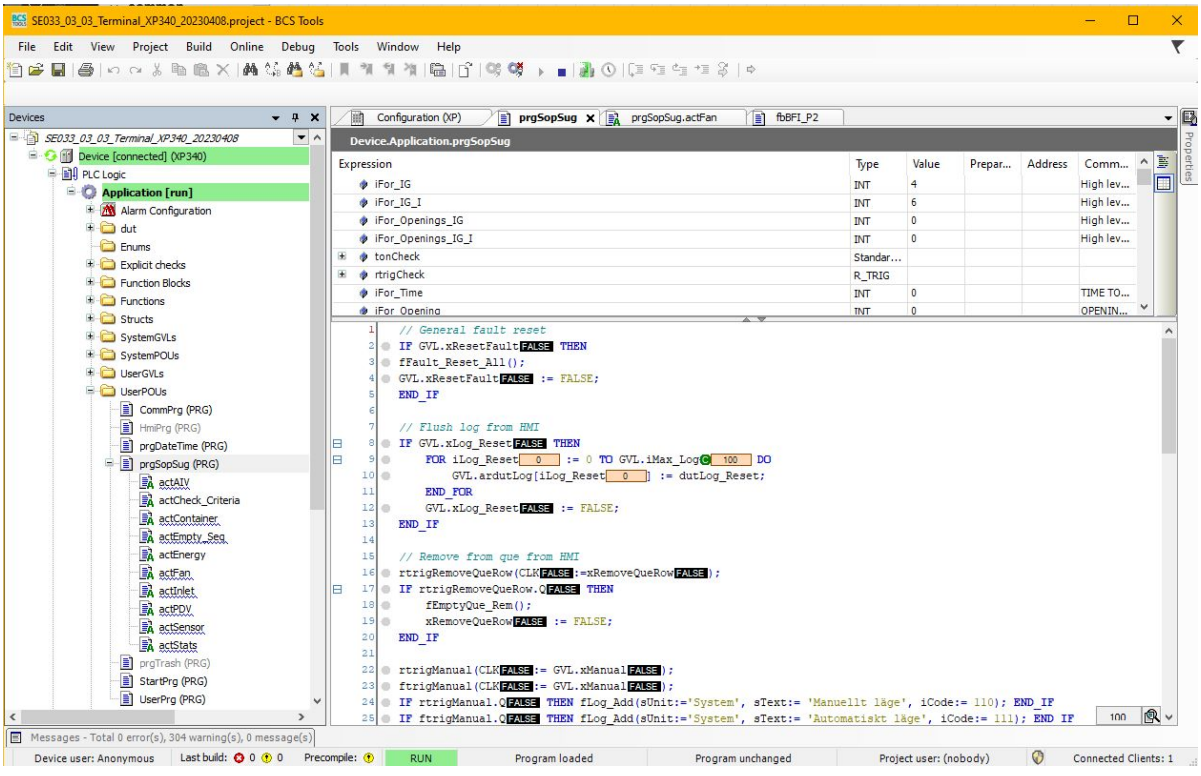
En av fördelarna med CODESYS är att den stödjer en mängd olika programmeringsspråk, inklusive IEC 61131-3, C++, C# och Python. Detta gör det möjligt för utvecklare att använda det språk som är bäst lämpat för den specifika applikationen.

CODESYS är också väl anpassad för molnintegration och IIoT (Industrial Internet of Things), vilket gör det möjligt att övervaka och styra automations- och styrsystem från fjärrplatser (*CODESYS Group*, 2023).

2.7.3 BCS Tools

BCS Tools är en PLC-programmeringsmjukvara som används för att konfigurera PLC-program, skapa HMI-gränssnitt samt att konfigurera och övervaka nätverkskommunikationen mellan enheter. Företaget Beijer Electronics utvecklade det baserat på CODESYS-plattformen, men med specifikationer som är mera anpassade efter Beijer Electronics egna PLC-enheter och HMI-lösningar (*BCS Tools - Beijer Electronics*, n.d.).

Figuren 4 är en skärmdump av BCS Tools från projektet.



Figur 4. BCS Tools-gränssnitt

2.8 API

Ett API är en uppsättning regler och protokoll som tillåter olika programvarusystem att kommunicera och interagera med varandra. API:er fungerar som en bro mellan olika applikationer och tillåter dem att utbyta data och funktionalitet. Ett API definierar olika metoder eller funktioner som kan användas för att interagera med en specifik applikation eller tjänst. Det kan inkludera metoder för att hämta, skicka, uppdatera eller ta bort data från en applikation eller för att utföra specifika funktioner.

API:er är utformade med hjälp av ett gränssnitt eller en specifikation som beskriver hur de olika anropen och svar ska struktureras. Det kan inkludera information om vilka parametrar som krävs, hur data ska formateras och hur felmeddelanden ska hanteras. För att använda ett API måste utvecklare vanligtvis få en API-nyckel eller en auktoriseringsmekanism för att autentisera och identifiera sig själva vid anrop. Detta hjälper till att skydda API:et och kontrollera vilka resurser som är tillgängliga för olika användare (Wikipedia contributors, 2023g).

2.8.1 REST-API

Ett REST-API är en typ av webbtjänst som använder sig av det arkitekturmönster som kallas "Representational State Transfer" (REST) för att möjliggöra data- och funktionsöverföring mellan olika applikationer och system via internet. REST-API:er utformas för att vara skalbara, flexibla och lättförståeliga (What Is RESTful API?, n.d.).

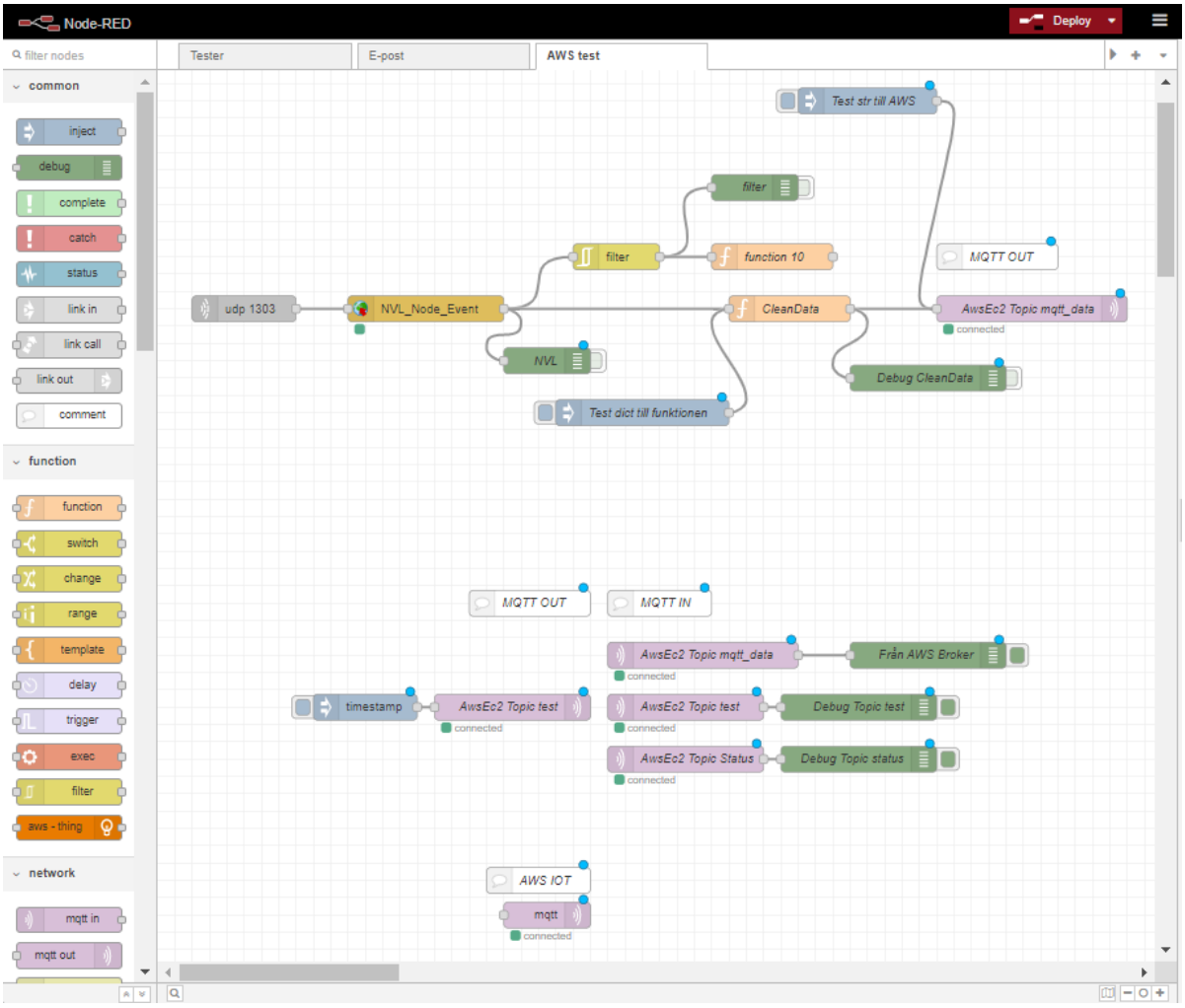
I ett REST-API används HTTP-protokollet för att hantera dataöverföring, och vanligtvis används HTTP-metoderna GET, POST, PUT och DELETE för att utföra olika operationer på data. Varje anrop till ett REST-API är oberoende av andra anrop och innehåller all nödvändig information för att utföra den specifika åtgärden. Dataöverföringen sker vanligtvis i JSON- eller XML-format, och URI-adressering används för att identifiera dataresurser (What Is a REST API?, n.d.).

2.9 Node-RED

Node-RED är en open source-plattform som ger användarna möjlighet att skapa programvarufunktioner och automatisera arbetsflöden genom att enkelt koppla samman hårdvara, API:er och onlinetjänster, till exempel GitHub, Gmail, AWS IoT. Verktyget fungerar som en redigerare som körs i en webbläsare och gör det möjligt att skapa och redigera flöden mellan olika enheter, samt att implementera dem direkt. Node-RED bygger på Node.js och är utvecklat för att fungera på enheter med låg prestanda och begränsad anslutning till internet. Plattformen har ett användargränssnitt som gör det lätt att skapa visuella flödesscheman och funktioner med hjälp av noder (OpenJS Foundation, n.d.).

Plattformen Node-RED är flexibel och stöder ett stort antal hårdvaruplattformar och anslutningsteknologier, vilket gör den till en bra lösning för IoT- och M2M-projekt (Machine-to-Machine). Dessutom finns det ett stort antal tillgängliga bibliotek med fördefinierade funktioner och moduler som gör det lätt att skapa avancerade funktioner och arbetsflöden (OpenJS Foundation, n.d.).

Figuren 5 är en skärmdump av ett Node-RED-flöde som visar användningen av MQTT-in- och ut-noder, Funktion-noder och Debug-noder.

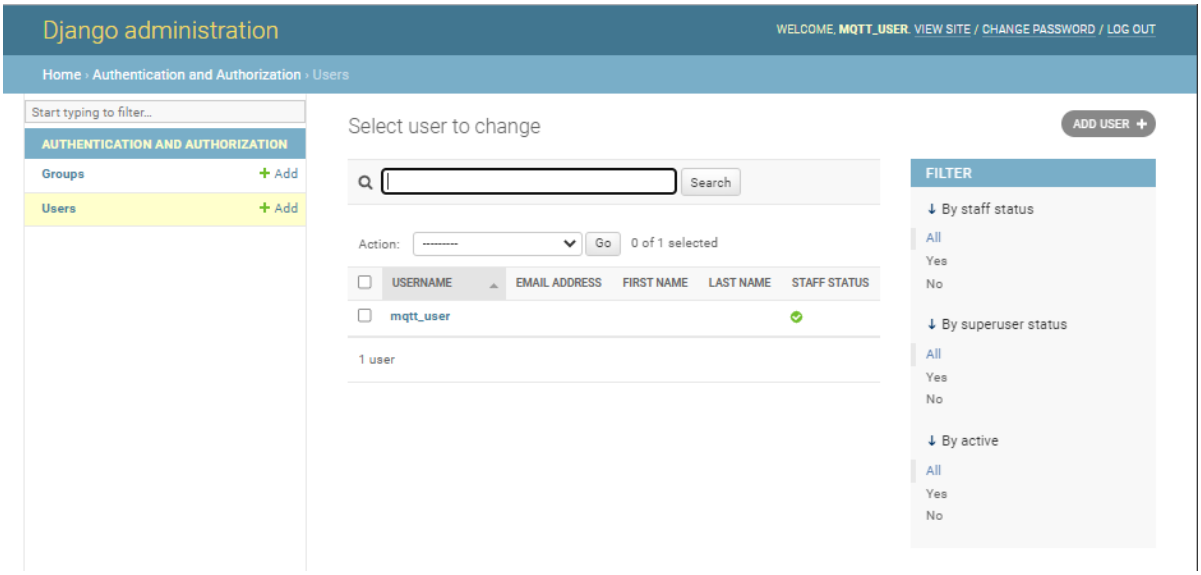


Figur 5. Node-RED-gränssnitt

2.10 Django

Django är ett högnivå- och Python-baserat webbramverk som underlättar utvecklingen av säkra och skalbara webbapplikationer. Det tillåter snabb och enkel utveckling av webbapplikationer med inbyggda funktioner för databasåtkomst, URL-routning, användarautentisering, templating (visningsmallar) och mycket mer. Ramverket är open source och följer principerna för modell-visnings-kontroller arkitektur (MVC) (*Django Introduction*, n.d.).

I figur 6 visas Django-administrationssidan, som erbjuder ett gränssnitt där administratörer kan utföra en mängd olika åtgärder som att lägga till, ta bort och uppdatera användare eller för att ändra behörigheter för varje användare. Dessutom kan administratörerna enkelt övervaka vilka användare som är aktiva och vilka som inte är det.



Figur 6. Konfiguration av användare i Django-administration

2.10.1 Django REST Framework

Django REST (Representational State Transfer) Framework är ett flexibelt verktyg för att skapa webb-API:er. Det bygger på Django-ramverket och erbjuder en enkel, snabb och skalbar lösning för att utveckla API:er (*Django Introduction*, n.d.).

Det finns inbyggda verktyg för att hantera autentisering, serialisering och vyhantering, vilket gör det möjligt att skapa API:er med minimal kod. Dessutom erbjuder det stöd för vanliga HTTP-metoder som GET, POST, PUT och DELETE, och kan användas för att skapa både JSON- och HTML-baserade API:er (*Django Introduction*, n.d.).

REST-framework ger också möjlighet att hantera autentisering, hantera versioner av API:et, och skapa dokumentation för API:et. Det är även utformat för att integreras med andra

verktyg och plattformar som Django, React, AngularJS och andra JavaScript-ramverk (*Django Introduction*, n.d.).

2.11 pytest

pytest är ett Python-baserat testramverk som används för att skriva, organisera och köra automatiserade tester, det vill säga tester som utförs av datorprogramvara istället för manuellt av människor. pytest stöder ett brett spektrum av testtyper inklusive enhetstester, funktionella tester och acceptanstester (*Pytest: Helps You Write Better Programs — Pytest Documentation*, n.d.).

2.12 Bokeh

Bokeh är en open source, Python-baserad programvara och ett bibliotek för att skapa interaktiva, webbaserade diagram, plottar och visualiseringar. Det används för att skapa interaktiva dashboards, dataapplikationer och analytiska verktyg för att utforska och presentera data på ett lättförståeligt sätt. Bokeh stöder många vanliga diagramtyper som linje- och stapeldiagram, men kan också skapa mer avancerade och anpassade visualiseringar som till exempel geografiska kartor och interaktiva 3D-diagram (Van de Ven, n.d.).

2.12.3 Bokeh Server

Bokeh Server är en webbapplikation som används för att skapa interaktiva datavisualiseringar med hjälp av Python och Bokeh-biblioteket. Bokeh Server möjliggör att skapa interaktiva visualiseringar som kan delas och användas av olika användare med internet som utgångspunkt för distribution (Van de Ven, n.d.).

Bokeh Server fungerar genom att skapa en server som kan hantera flera användare och ge dem tillgång till en interaktiv visualisering via en webbläsare. Servern kan kommunicera med backend-datakällor och utföra olika data-transformationer och analyser. Användare kan sedan manipulera visualiseringen genom att interagera med knappar, reglage, diagram och andra element i webbgränssnittet (Van de Ven, n.d.).

Bokeh Server är utformad för att fungera på alla plattformar, inklusive desktop, mobil och webb. Dessutom finns det stöd för integration med andra webbtekniker som Flask, Django och Jupyter Notebook (Van de Ven, n.d.).

2.13 Pandas

Pandas är ett Python-baserat open source-programvarubibliotek som används för datahantering och dataanalys. Pandas erbjuder verktyg för att manipulera och analysera stora mängder data och är utformat för att hantera olika typer av data, inklusive strukturerad, halvstrukturerad och ostrukturerad data. Det kan hantera data från olika källor, inklusive CSV-filer, Excel-ark, databaser och webb-API:er (*Pandas*, n.d.).

Pandas tillhandahåller två primära datatyper, Series och DataFrame, för att hantera data. Series är en etiketterad, endimensionell array som kan innehålla data av olika typer. Data Frame är en etiketterad, tvådimensionell datastruktur som består av rader och kolumner.

2.14 Databasdefinitioner

Här har vi beskrivit definitioner på databasrelaterade begrepp.

2.14.1 Relationsdatabas

En relationsdatabas är en typ av databas som organiserar och lagrar data i tabeller med relationer mellan dem. Varje tabell representerar en entitet, till exempel en person eller en produkt, och varje rad i tabellen representerar en specifik instans av den entiteten. Relationerna mellan tabellerna etableras genom att använda nycklar, vanligtvis primärnycklar och främmande nycklar. Detta tillåter effektiv lagring, uppdatering och hämtning av data genom att använda strukturerade frågor med hjälp av språket SQL (Structured Query Language) (Wikipedia contributors, n.d.).

2.14.2 Fakta- och dimensionstabeller

En faktatabell är en central tabell inom datalagerhantering, vilket är en teknik för att hantera och analysera stora mängder data. Faktabellen innehåller vanligtvis numerisk information eller faktiska händelser (till exempel försäljning, kunder, produktionshastighet) som är relevanta för en organisations verksamhet (Wikipedia contributors, 2023a).

En faktatabell är vanligtvis kopplad till flera dimensionstabeller, som används för att beskriva faktaenheterna i faktabellen (t.ex. tid, plats, produkttyp). Genom att kombinera informationen i faktabellen med informationen i dimensionstabellerna kan användare av datalagret utföra avancerade analyser och rapporteringar.

2.14.3 Entity-relationship diagram

Entity-relationship (ER) modellen är en datamodelleringsteknik som används för att beskriva relationer mellan olika dataenheter i en databas. ER-modellen består av två huvudelement: Entiteter och Relationer. Entiteter representerar objekt eller företeelser som är av intresse för systemet, medan relationer representerar sambandet mellan entiteterna. Ett diagram som använder sig av den här tekniken kallas Entity-relationship diagram (ERD) (Wikipedia contributors, 2023e).

3. KRAV

Den här kravställningen har tagits fram tillsammans med uppdragsgivaren av projektet. Kraven har prioriterats enligt hög, medium och låg prioritet.

3.1 Hög prioritet

1. Data kan skickas från befintlig PLC till Node-RED och vidare till databas
2. Designa och skapa databas för den data som samlas in i krav 1
3. Filtrera data innan den skickas till databasen
4. Hämta data från databasen och visualisera med ett interaktivt gränssnitt

3.2 Medium prioritet

5. Analysera den insamlade datan och se var man kan effektivisera sopsugsstationen

3.3 Låg prioritet

6. Styra PLC baserat på analys av datainsamlingen, för att effektivisera processerna

4. VERKTYG

I detta kapitel beskriver vi de verktyg vi använt under processen att ta fram denna lösning.

4.1 Pycharm

PyCharm[2] är en integrerad utvecklingsmiljö (IDE) som är avsedd för Python och skapad av företaget JetBrains. Den integrerade utvecklingsmiljön har en mängd inbyggd funktionalitet, inklusive funktioner för datainsamling, vilket passade vårt projekt bra. En annan orsak till att vi valde PyCharm är att vi har tidigare erfarenhet av att arbeta med Jetbrains integrerade utvecklingsmiljöer.

4.2 Insomnia

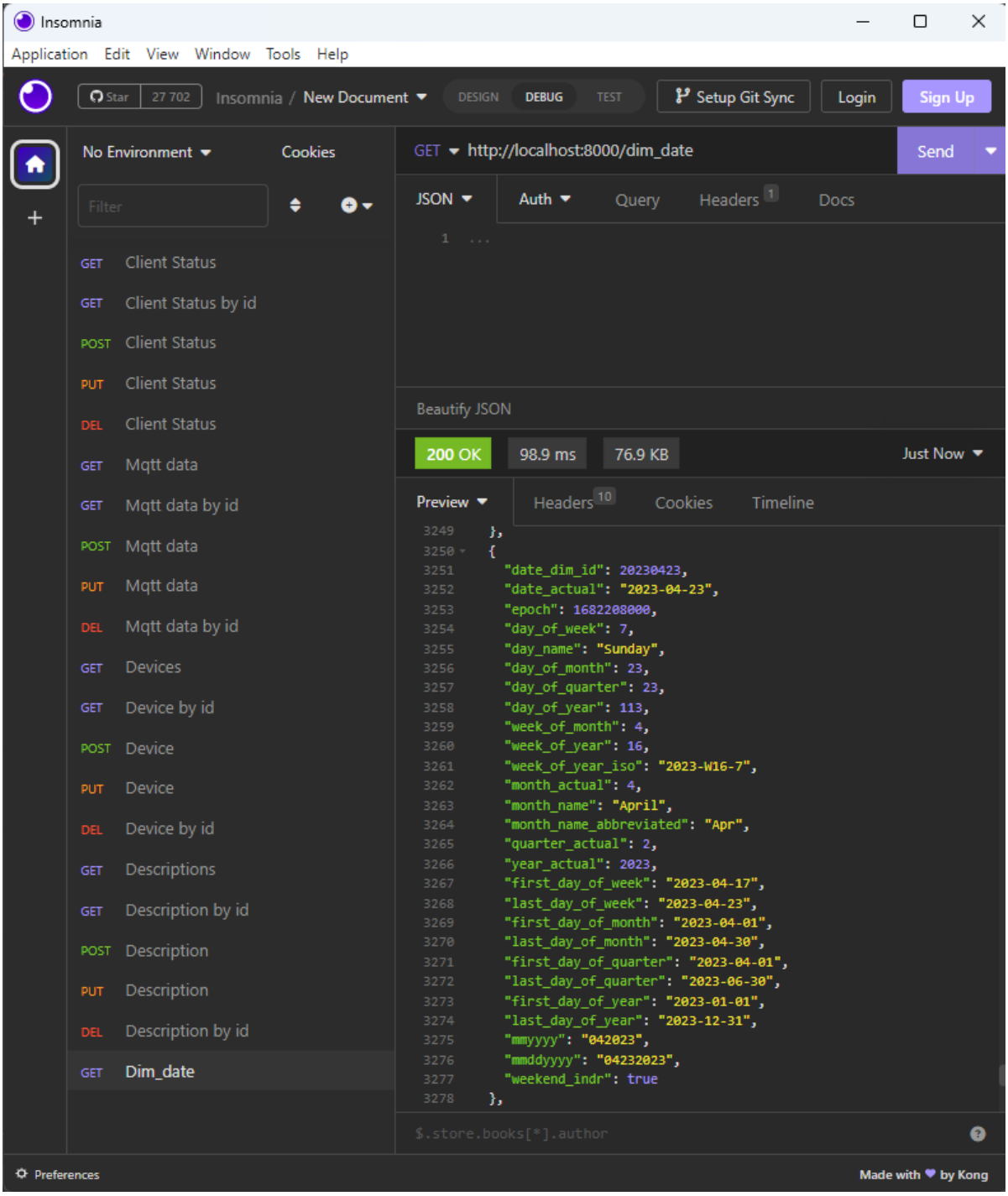
Insomnia[3] är en plattformsoberoende klient för att testa och utveckla REST-API:er. Den ger användare ett gränssnitt där man kan ansluta till och testa API:er och mikrotjänster på ett effektivt sätt. Insomnia stöder en mängd olika autentiseringsmetoder, inklusive OAuth 2.0, API-nycklar, basautentisering och JWT-token (*The Collaborative API Development Platform*, n.d.).

Vi valde att arbeta med Insomnia då vi har tidigare erfarenhet av det och det är ett verktyg som passar bra för att testa API-anslutningspunkter.

Figur 7 visar en skärmdump av Insomnias gränssnitt som används för att göra en HTTP GET-förfrågan till anslutningspunkten "dim_date". På vänster sida av gränssnittet visas alla tillgängliga anslutningspunkter som kan användas för att hämta data från API:n. API-url:en visas ovanpå gränssnittet och visar den aktuella URL som används för att hämta data.

[2] https://www.jetbrains.com/pycharm/
[3] https://insomnia.rest/

I resultatfönstret visas den detaljerade informationen för GET-förfrågan som skickades. Detta inkluderar HTTP-statuskoden (200), vilket indikerar att förfrågan var framgångsrik, tiden som det tog att få svaret, storleken på den mottagna datan och andra relevanta uppgifter.

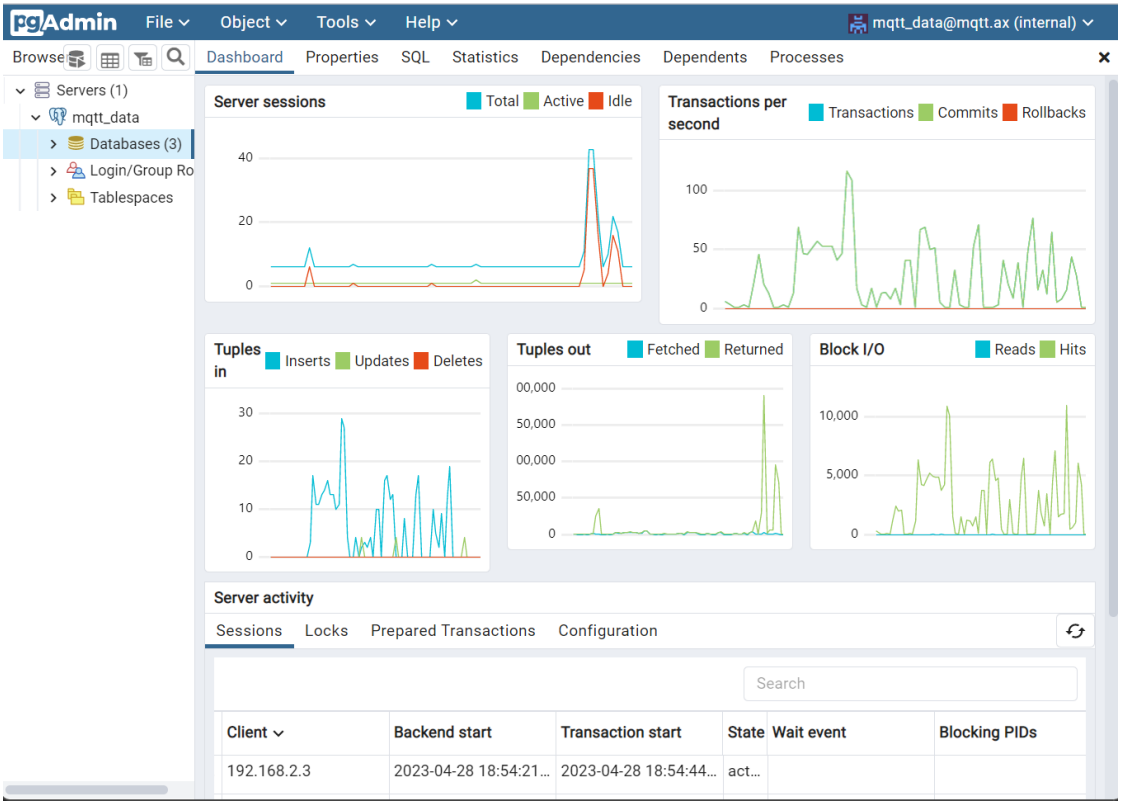


Figur 7. Insomnia-gränssnitt

4.3 pgAdmin

För PostgreSQL[4] finns det också en SQL-editor som vi valde att arbeta med för att få en bättre överblick över databasen och för att lättare kunna göra ändringar i databasen.

pgAdmin[5] är en open source-plattform för administration av PostgreSQL-databaser. Det är en omfattande verktygssamling som ger användare ett grafiskt användargränssnitt för att administrera och hantera databaser med hjälp av en mängd olika funktioner, inklusive databasdesign, datahantering och serverkonfiguration. pgAdmin stöder också PostgreSQL-databaser i molntjänster, samt SQL Server och MySQL-databaser via ett gemensamt gränssnitt. pgAdmin har en webbaserad klient som gör det möjligt att ansluta till och hantera PostgreSQL-databaser från en webbläsare alternativt som programvaror för Windows, Mac OS X och Linux (*PgAdmin - PostgreSQL Tools*, n.d.). Figur 8 visar en skärmdump av pgAdmin-gränssnittet som användes under projektet.

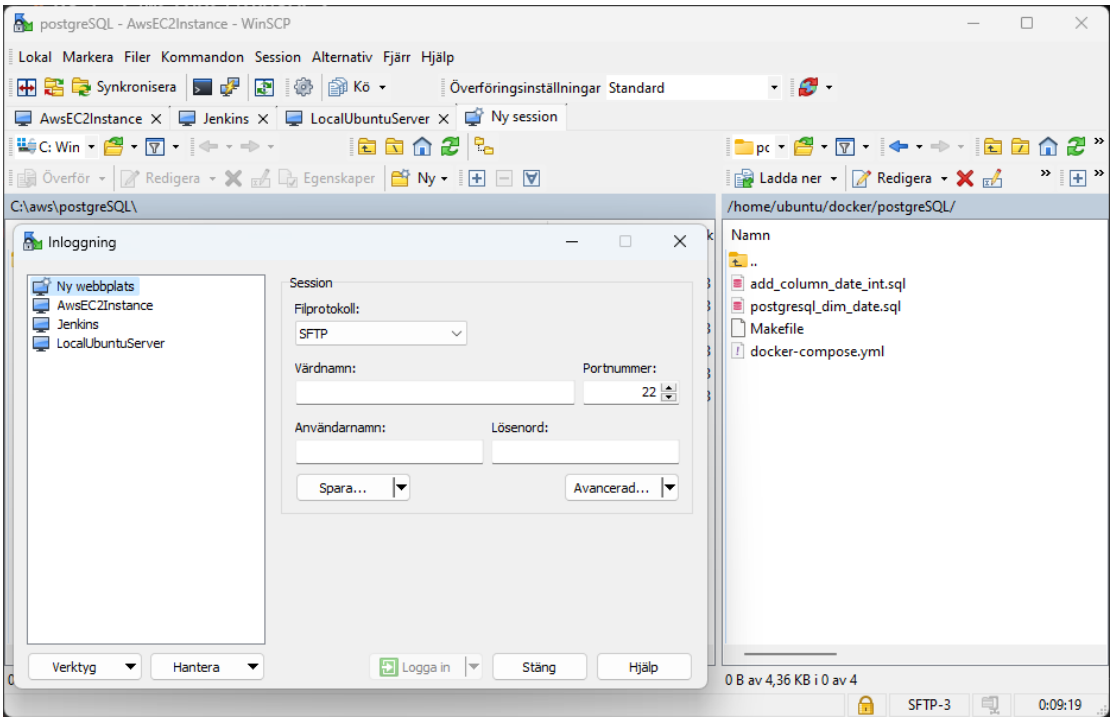


Figur 8. pgAdmin gränssnitt

[4] https://www.postgresql.org/
[5] https://www.pgadmin.org/

4.4 WinSCP

WinSCP[6] är en open source-plattform för säker överföring av filer mellan datorer som kör Microsoft Windows och Unix / Linux-operativsystem. Det är en filöverföringsklient som stöder både SFTP-(SSH File Transfer Protocol) och SCP-(Secure Copy Protocol) protokoll för överföring av filer, och det erbjuder också ett grafisk användargränssnitt för att underlätta filhantering. WinSCP är tillgängligt på flera språk och ger en mängd funktioner, inklusive en inbyggd textredigerare och sökfunktioner (*Introducing WinSCP*, n.d.). Figur 9 visar en skärmdump av WinSCP-gränssnittet som användes under projektet.



Figur 9. WinSCP-gränssnitt.

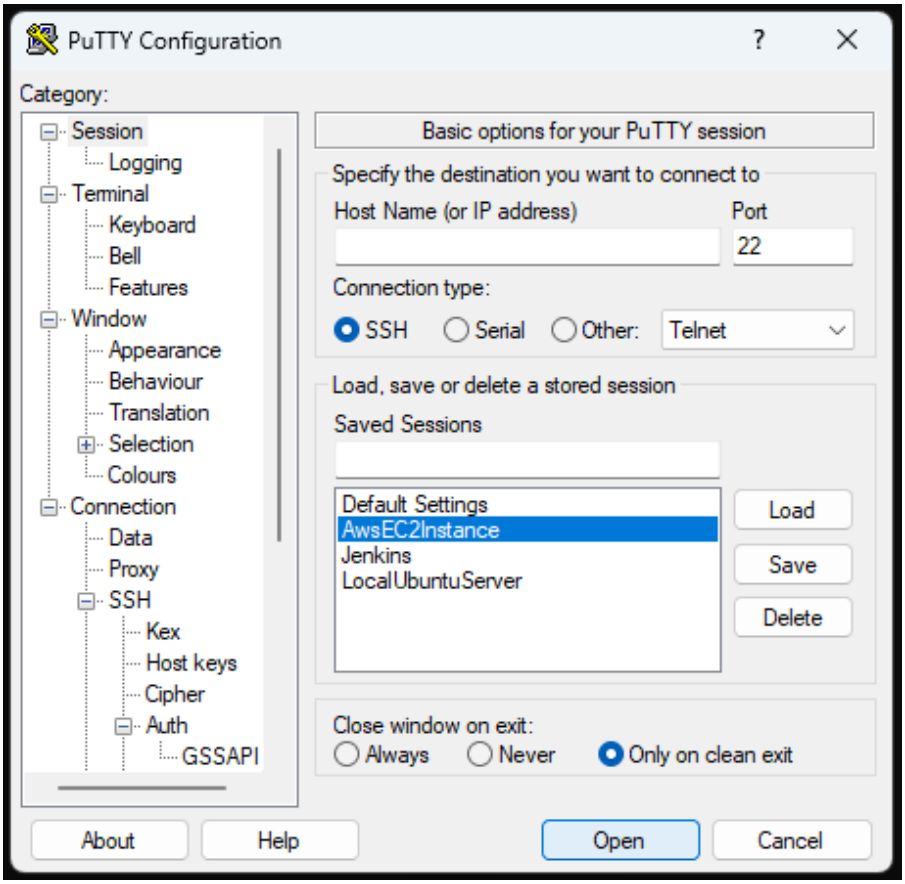
Vi valde att jobba med WinSCP då det var ett smidigt verktyg för att överföra filer från vår lokala arbetsmiljö till projektets AWS EC2-instans och den inbyggda textredigeraren var väldigt praktisk då vi behövde göra ändringar under utvecklingen.

[6] https://winscp.net/eng/index.php

4.5 Putty

Putty[7] används som en klient för fjärranslutning till andra datorer över nätverket med hjälp av olika protokoll, inklusive SSH, Telnet, rlogin och RAW. Det är ett mångsidigt verktyg som är särskilt användbart för administratörer och utvecklare som behöver fjärråtkomst till servrar och andra datorer för att hantera och felsöka programvara och system. Putty har ett enkelt användargränssnitt som gör det lätt att ansluta till en fjärrdator, samt en rad funktioner som gör det möjligt att anpassa och optimera anslutningen för olika ändamål(*Download PuTTY - a Free SSH and Telnet Client for Windows*, n.d.).

Figur 10 visar en skärmdump av Putty-gränssnittet som användes under projektet.



Figur 10. Putty-gränssnitt.

Vi använde Putty för att enkelt komma åt vår EC2-instans under projektets utveckling.

[7] https://www.putty.org/

4.6 Automation av byggprocess - Makefile

En Makefile[8] är en fil som innehåller instruktioner för att automatisera byggprocessen för ett program eller en applikation. Genom att använda Makefiler kan utvecklare skapa en serie steg som automatiserar skapandet av en färdig applikation från dess källkod.

En Makefile består av en serie regler som specificerar hur varje del av applikationen ska byggas, vilket inkluderar kompilering av källkoden, länkning av objektfiler, hantering av beroenden. Makefiler kan användas med en mängd olika programvaruprojekt och är särskilt användbara i större projekt med många filer och beroenden.

I figur 11 visas ett exempel på en Makefile från projektet som innehåller kommandot “run” för att anropa andra kommandon som skapar Docker-containrar (se kapitel 5.2) och skriver ut IP-adressen för containern “db”.

```
C: > aws > postgreSQL > M Makefile
1
2   run:
3       docker-compose up -d
4       docker inspect  -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' db
5
```

Figur 11. En Makefile från projektet

Makefilen läser av ett "make"-kommando som tar filnamnet på Makefilen som argument och kör sedan de regler som beskrivs i Makefilen.

Figur 12 visar terminalgränssnittet där vi använder kommandot "make run" och där utskriften från kommandot visas.

```
PS C:\aws\postgreSQL> make run
docker-compose up -d
[+] Running 2/2
 ✓ Container db       Started                                                    2.0s
 ✓ Container pgadmin  Started                                                    2.0s
docker inspect  -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' db
192.168.2.2
PS C:\aws\postgreSQL>
```

Figur 12. Resultatet av “make run” kommandot i terminalgränssnittet

[8] https://makefiletutorial.com/

Vi beslutade att använda en Makefile eftersom den har förenklat utvecklingsprocessen betydligt vid användning av långa kommandon och när vi har behövt köra flera kommandon i följd.

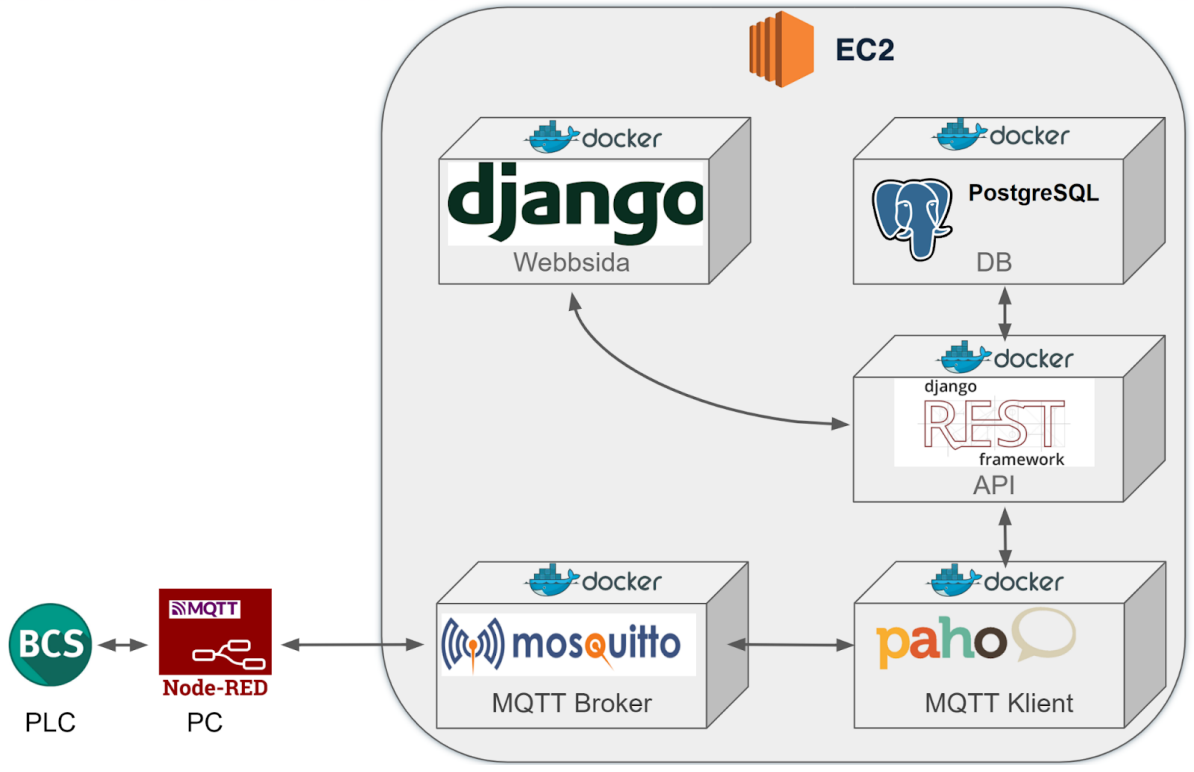
5. IMPLEMENTATION

När vi startade projektet fanns det redan en sopsugsstation som var installerad och igång, med en PLC som skickar data till Node-RED via MQTT-protokollet. Vår uppdragsgivare hjälpte oss att få tillgång till den data som vi behövde för att samla in informationen.

Vi beslöt oss för att använda en mikrotjänstarkitektur i projektet, med följande delar:

- MQTT-Broker
- MQTT-Klient
- API
- Databas
- Webbsida

Vårt mål är att alla dessa delar ska integreras i separata Docker-containrar (se kapitel 5.2) för att göra systemet mer flexibelt och lättanvänt. Figur 13 visar en grafisk representation av hur olika tjänster är kopplade till varandra.



Figur 13. Mikrotjänstarkitektur

För att säkerställa att allting fungerade, skapade vi varje tjänst i vår lokala miljö innan de överfördes till AWS för testkörning. Vi implementerade varje tjänst på AWS allt eftersom de blev klara. Då vår lokala arbetsmiljö körs på Windows och vår AWS EC2-instans körs på Linux, så har vi också emellanåt använt en lokal Linux-server för att testa att allting är rätt konfigurerat innan vi satte upp det till EC2-instansen.

5.1 AWS

Vår uppdragsgivare ville ha projektet i molnet och vi valde AWS då vi är bekanta med det sedan tidigare och vill lära oss mer om plattformen då den används på flera arbetsplatser. Vi började med att titta på AWS IoT, AWS Elastic Container Service och AWS RDS, men med de tjänsterna skulle vår lösning vara bunden till att alltid använda AWS, vilket var något vi ville undvika. Vi beslöt oss därför att använda EC2-instans för största flexibilitet.

5.2 Docker

5.2.1 Docker-nätverk

För att möjliggöra kommunikation mellan containrar och skapa ett gemensamt nätverk som de kan ansluta till, använder vi ett Docker-nätverk.
För att skapa ett Docker-nätverk med namnet "network_mqtt", använder vi följande kommando:
docker network create --driver=bridge --subnet=x.x.x.x/x --gateway=x.x.x.x network_mqtt

Detta skapar en nätverksbrygga med det angivna nätverksintervallet och gateway-adressen. Genom att använda samma nätverksnamn för flera containrar kan de ansluta till samma nätverk och kommunicera med varandra via IP-adresser. Detta kan beskrivas som ett isolerat nätverk för intern kommunikation inom Docker-containrar.

5.2.2 Dockerfile

För tre av våra tjänster behövde vi skapa egna Docker-images då det inte fanns färdiga Docker-images att använda. Det tjänster som vi skapade egna images för är:

- MQTT-klienten med hjälp av Paho-biblioteket

- API:et med hjälp av Django REST Framework
- webbsidan med hjälp av Django-ramverket

Följande befintliga Docker-images har vi använt i projektet:

- eclipse-mosquitto[9]
- postgres[10]
- dpage/pgdmin4[11]

I figur 14 använder vi en Python-baserad bas-image för att skapa en Docker-container. Efter att ha definierat och skapat lämpliga mappar i containern, kopieras applikationens filer och konfigurationsfiler in i containern. Dessutom exponeras port 8000 för att göra applikationen tillgänglig från utsidan. För att se loggmeddelanden i realtid har vi definierat en miljövariabel "PYTHONUNBUFFERED". Genom att göra detta säkerställs att utdata från applikationen visas direkt i konsolen.

```
1   FROM python
2
3   # Necessary, so Docker doesn't buffer the output and that you can see the output
4   # of your application (e.g., Django logs) in real-time.
5   ENV PYTHONUNBUFFERED 1
6
7
8
9   RUN mkdir /docker_django_api
10  RUN mkdir /docker_django_api/app
11  RUN mkdir /docker_django_api/djangoapi
12
13  WORKDIR /docker_django_api
14
15  COPY /requirements.txt /docker_django_api/requirements.txt
16
17
18
19  COPY ./app ./app
20  COPY ./djangoapi ./djangoapi
21  COPY /manage.py .
22  COPY /Makefile .
23
24  EXPOSE 8000
```

Figur 14. Dockerfile för API-tjänsten

[9] https://hub.docker.com/_/eclipse-mosquitto
[10] https://hub.docker.com/_/postgres
[11] https://hub.docker.com/r/dpage/pgadmin4

För att sedan bygga en Docker-image behövs det två kommandon:

- *docker build -t <image_name> .*
- *docker run -p <port>:<port> <image_name>*

Vi skapade ett snabbkommando med Makefile (se kapitel 4.6) som utförde uppgiften åt oss: *make docker*

5.2.3 Docker compose

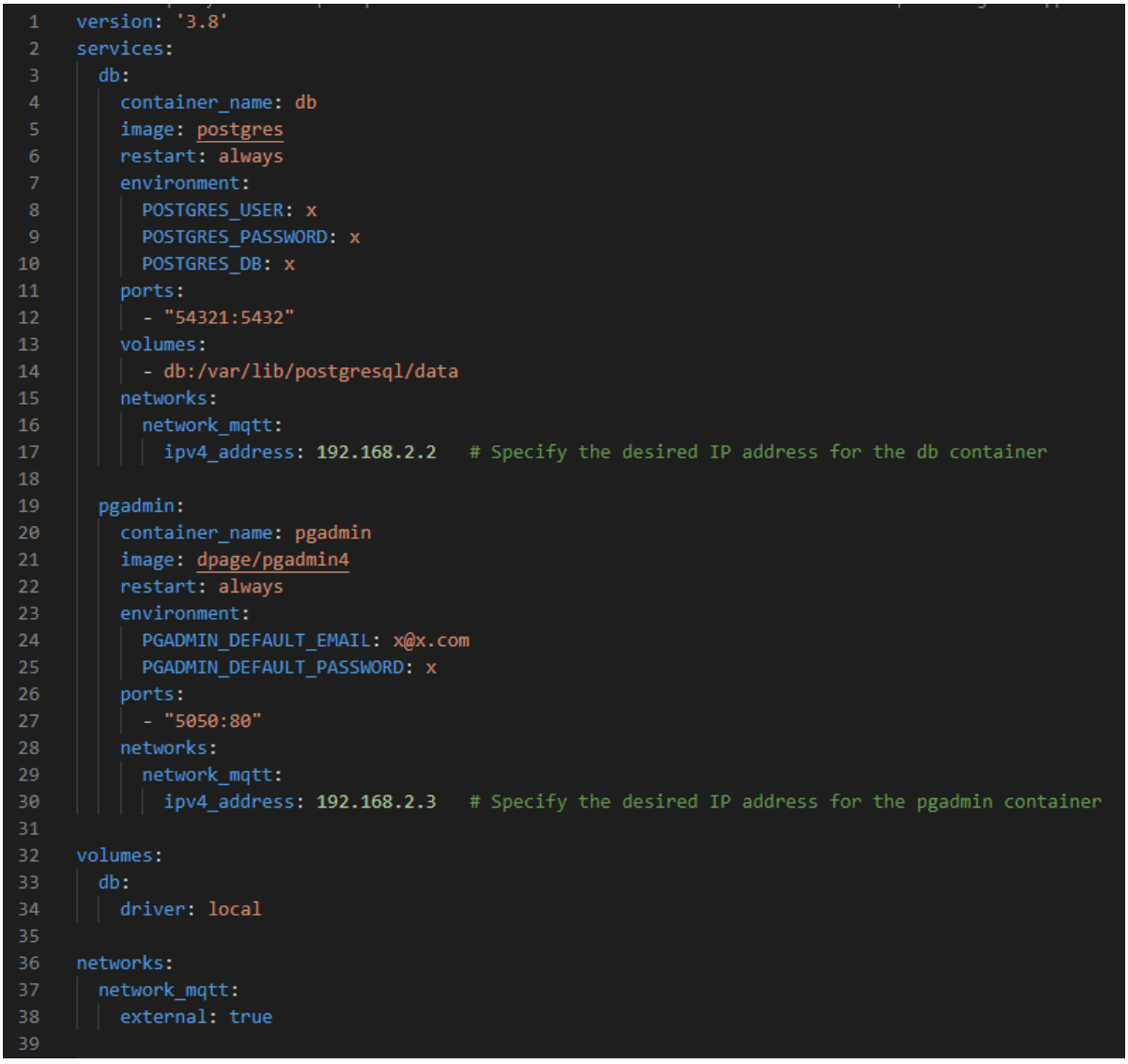
Här specificerade vi att Docker-containrarna ska kopplas till det gemensamma nätverket med en viss IP-adress. För databasen behövde vi också specificera var backup-datan ska lagras fysiskt (bestående lagring) utanför Docker-containern, då datan annars skulle gå förlorad ifall att Docker-containern stängs manuellt eller av annan orsak slutar köra.

Figur 15 visar en Docker-compose-fil som definierar två separata Docker-containers, "db" och "pgadmin", som används för att köra en PostgreSQL-databas och ett administratörsgränssnitt (pgAdmin).

För databas-containern definieras "postgres" som bas-image och miljövariabler för att konfigurera användare, lösenord och databasnamn. Port 54321 exponeras för att tillåta anslutningar till PostgreSQL från utsidan, och volymen "db" används för att lagra databasens data utanför containern.

För pgAdmin-containern används "dpage/pgadmin4" som bas-image och miljövariabler definieras för att ange e-postadress och lösenord för pgAdmin. Port 5050 exponeras för att tillåta åtkomst till pgAdmin från utsidan.

Slutligen definieras vår externa nätverksanslutning "network_mqtt" som används för att kommunicera mellan containrar och möjliggör att de kan nås via specifika IP-adresser.



```yaml
version: '3.8'
services:
  db:
    container_name: db
    image: postgres
    restart: always
    environment:
      POSTGRES_USER: x
      POSTGRES_PASSWORD: x
      POSTGRES_DB: x
    ports:
      - "54321:5432"
    volumes:
      - db:/var/lib/postgresql/data
    networks:
      network_mqtt:
        ipv4_address: 192.168.2.2   # Specify the desired IP address for the db container

  pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: x@x.com
      PGADMIN_DEFAULT_PASSWORD: x
    ports:
      - "5050:80"
    networks:
      network_mqtt:
        ipv4_address: 192.168.2.3   # Specify the desired IP address for the pgadmin container

volumes:
  db:
    driver: local

networks:
  network_mqtt:
    external: true

```

Figur 15. Docker-compose för Databas

5.3 MQTT

I det här projektet använde vi oss av en MQTT-broker och två stycken MQTT-klienter.

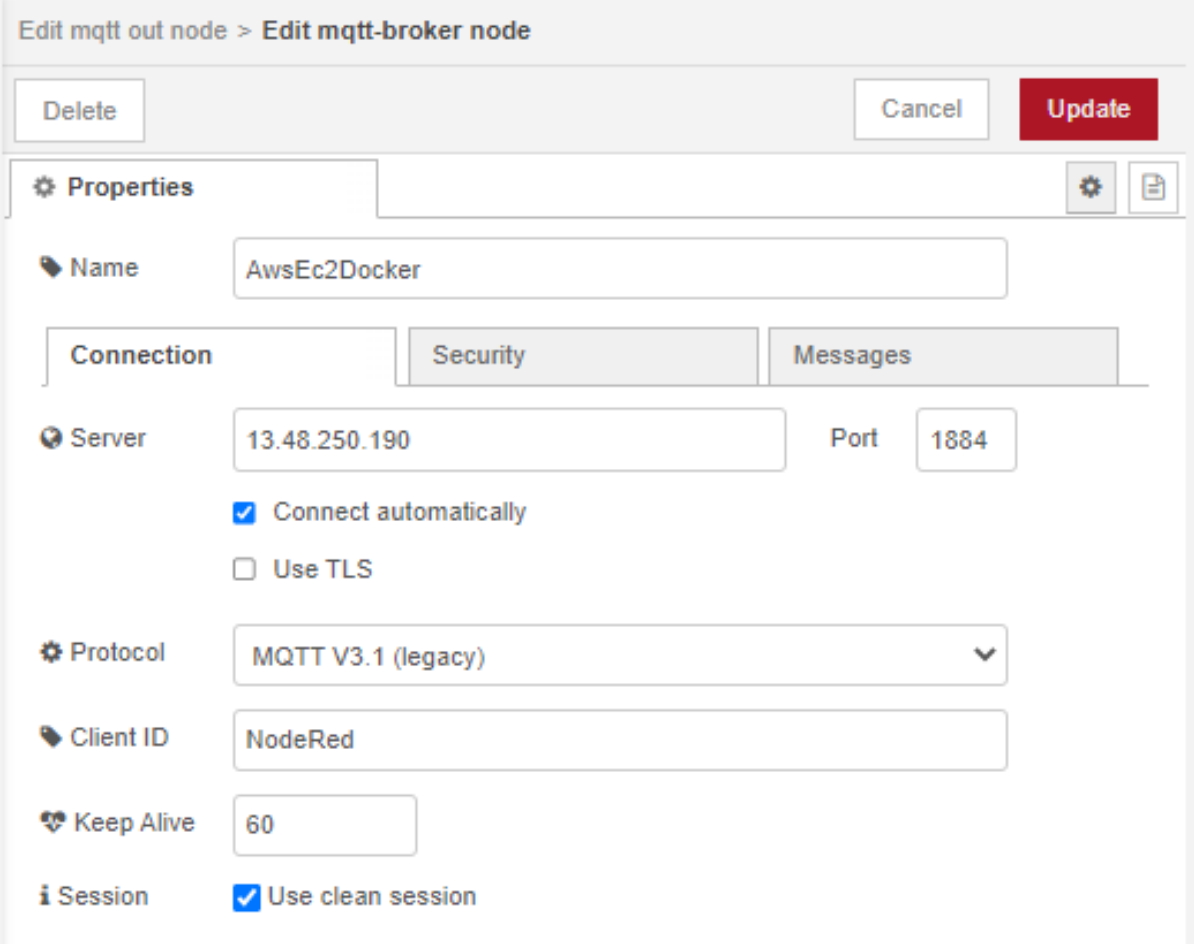
5.3.1 MQTT-broker

För att kunna vidarebefordra data från Node-RED, vilket uppfyller krav nr 1 från projektets kravspecifikationer, behövde vi skapa en MQTT-broker. Vi valde att använda oss av Eclipse Mosquitto, då det finns färdiga Docker-images som enkelt kunde integreras i vårt system och därmed sparade vi tid och undvek behovet av att installera och konfigurera Mosquitto manuellt. Genom att använda en befintlig Docker-image kunde vi smidigt lägga till den rätta konfigurationen för vårt projekt.

För att kunna testa att vår MQTT-broker fungerade så använde vi oss av Node-REDs inbyggda MQTT-klient.

5.3.2 Node-REDs MQTT-Klient

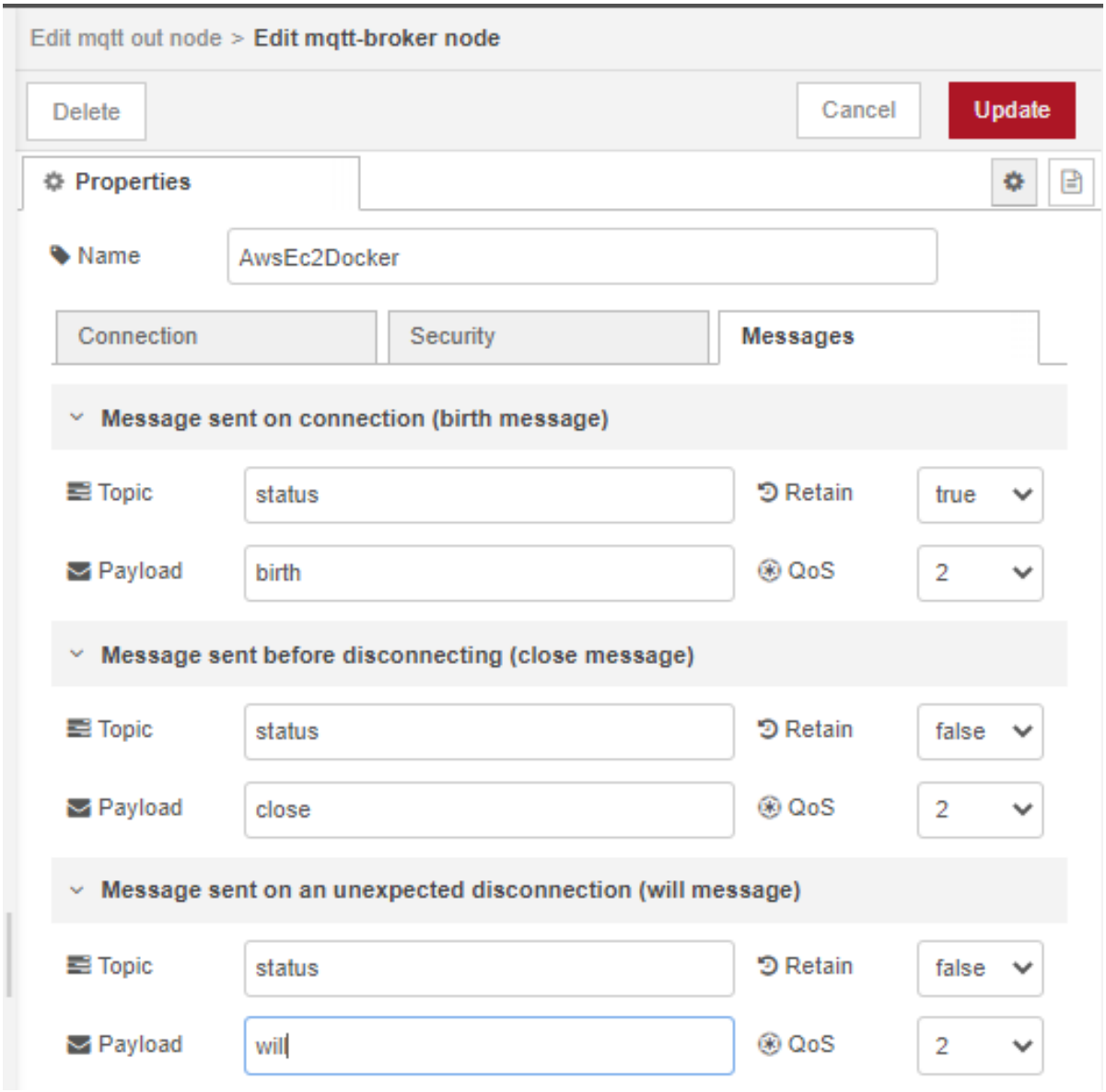
För att kunna koppla Node-REDs MQTT-klient till MQTT-brokern så använde vi konfigurationen i figur 16.



Figur 16. Inställningar av MQTT-Klient på Node-RED. Anonymiserad IP-adress.

För att kunna övervaka om en klient är ansluten eller inte, har vi gjort inställningar på Node-REDs MQTT-klient och logiken i vår MQTT-klient. Det ger oss möjlighet att hålla koll på statusen för klienten och vidta åtgärder om klienten av någon anledning inte är tillgänglig.

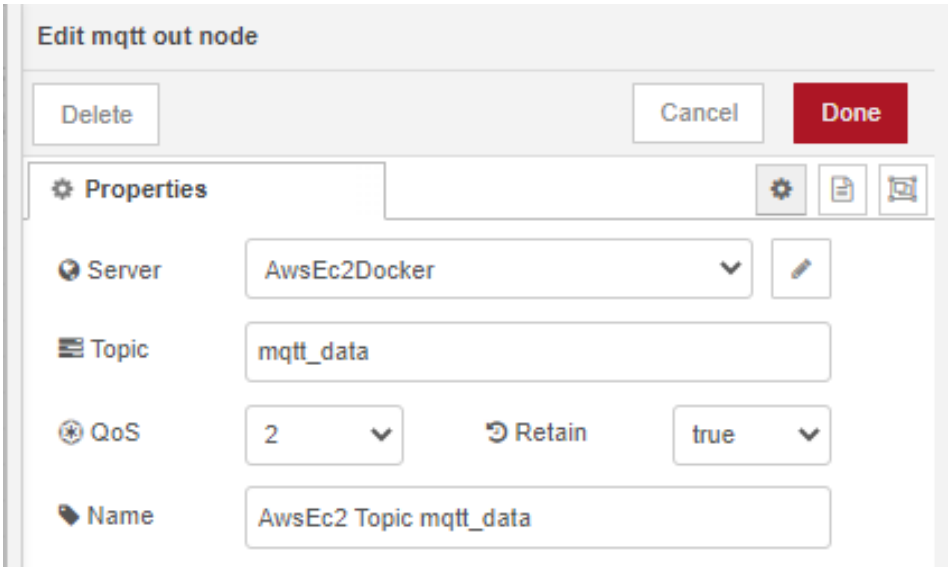
Figur 17 visar de olika inställningarna som används för att konfigurera Node-RED MQTT-klienten för övervakning och felsökning.



Figur 17. Inställningarna som används för att övervaka klientanslutning

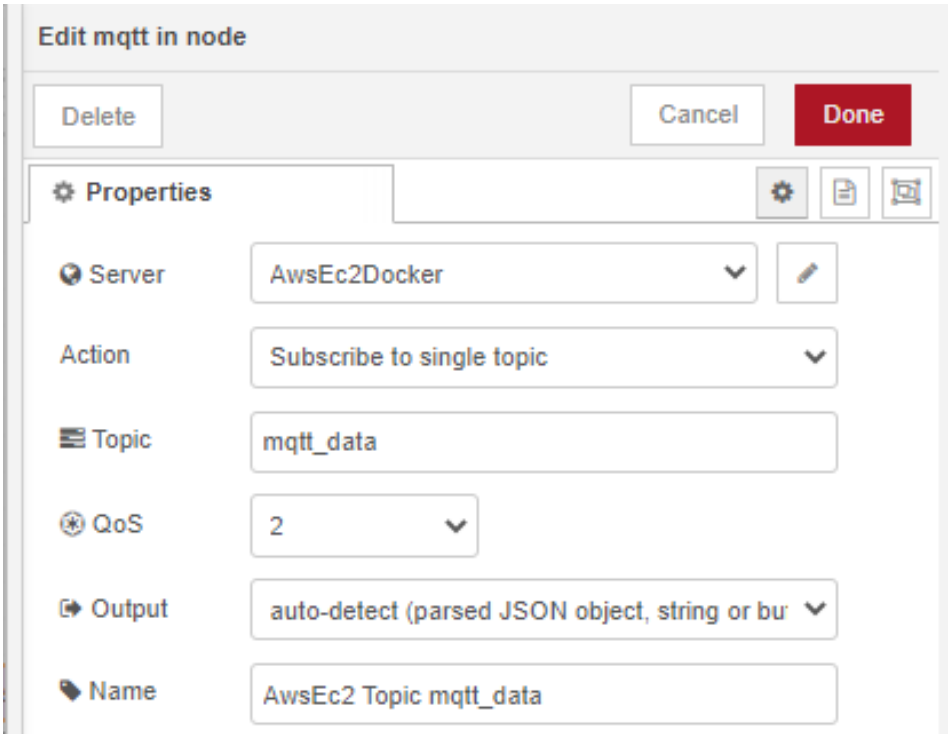
5.3.2.1 MQTT-Publikation och -prenumeration

I figur 18 kan det ses hur vi konfigurerar MQTT-klienten på Node-RED för att publicera ett MQTT-ämne till vår MQTT-broker.



Figur 18. Inställningar för publikationen av ett ämne

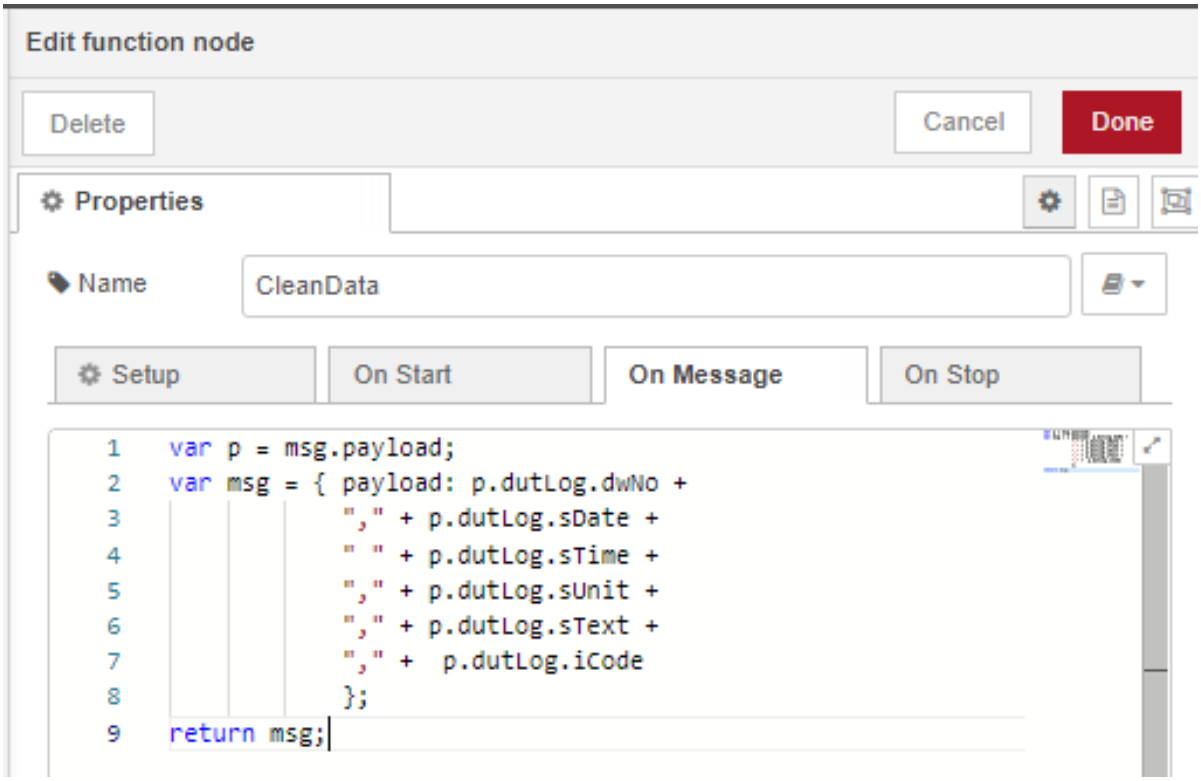
I figur 19 kan det ses hur vi konfigurerar Node-RED MQTT-klienten för att prenumerera på ett visst MQTT-ämne från vår MQTT-broker.



Figur 19. Inställningar för prenumerationen på ett ämne

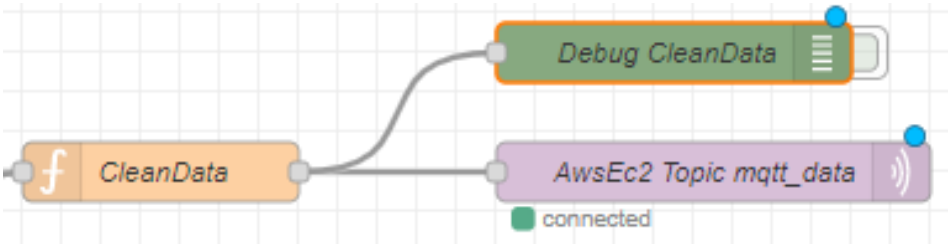
5.3.2.2 MQTT-Meddelande

Vi ville minska mängden data som skickades över MQTT, vilket uppfyller krav nr 3 från projektets kravspecifikationer. Därför skapade vi en funktion i Node-RED som konfigurerar strängen som skickas från PLC:n, så att den inte innehåller key-värdet. Strängens format kan ses i figur 20.



Figur 20. Konfigurationen av PLC-strängen

Efter att vi konfigurerat strängen så skickas den vidare både till en debug-nod, för att underlätta utvecklingen, och till MQTT-klienten för att publiceras. Se figur 21.



Figur 21. Kopplingen av funktionen till MQTT-klienten och debug-noden

5.3.3 MQTT-klient implementation

Utöver Node-REDs MQTT-klient så behövde vi också skapa en till MQTT-klient.
Först skapar vi en anslutning till vår MQTT-broker och prenumererar på två ämnen:

- mqtt_data
- status

Ämnet "mqtt_data" innehåller datasträngen som kommer från PLC:n via funktionen "CleanData" och ämnet "status" kommer från MQTT-brokern som meddelar om klientens anslutningstatus.

Efter att anslutningen gått igenom så tar klienten emot och formaterar tillbaka datasträngen till JSON-objekt innan den skickas vidare till API:et. Eftersom Eclipse Paho är gjort av samma utvecklare som Eclipse Mosquitto så valde vi att använda det för vår MQTT-klient.

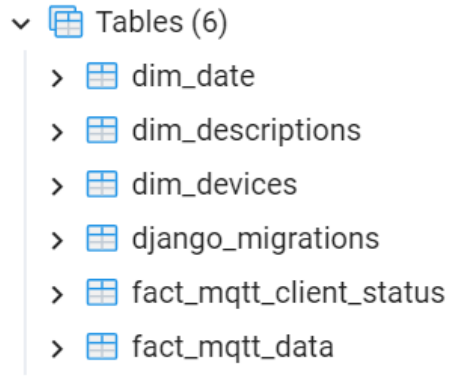
5.4 Databas

Efter att vi fått MQTT-kommunikationen att fungera så riktade vi vår uppmärksamhet mot att skapa en lämplig databaslösning, vilket uppfyller krav nr 2 från projektets kravspecifikationer. Vi valde att använda PostgreSQL eftersom vi var intresserade av att arbeta med en relationsdatabas och PostgreSQL är en välkänd, open source-relationsdatabas. En annan orsak var att PostgreSQL tillåter användningen av mera avancerade datatyper, såsom timezone-aware timestamps, vilket var något vi behövde för det här projektet.

5.4.1 Databasdesign

För att snabbt kunna börja samla in data skapade vi först en tabell som kunde ta emot all information. Därefter har vi kontinuerligt anpassat och utökat denna tabell genom att lägga till nya kolumner och designat nya tabeller när det behövdes.

Det har möjliggjort snabb datainsamling samtidigt som det har gjort det möjligt för oss att säkerställa att all kommunikation från MQTT-klienten fungerar som den ska. Dessutom har det hjälpt oss att välja de mest lämpliga datatyperna för de olika kolumnerna i vår databas och på så sätt optimera dess prestanda och funktionalitet. I figur 22 visas vilka tabeller som används i projektet.



Figur 22. Tabeller i databasen

Vi har två faktatabeller i vår databas, “fact_mqtt_data”-tabellen och ”fact_mqtt_client_status”-tabellen, som innehåller data som samlats in från MQTT-klienten. Båda tabellerna innehåller faktiska observationer och är centralt placerade i vår databas.

Datumet då observationen gjordes fungerar som en gemensam nyckel för båda faktatabellerna. Detta datum är också kopplat till dimensionstabellen “dim_date”, vilket hjälper oss att gruppera data för visualisering och analys på olika tidsskalor.

Tabellen “fact_mqtt_data” innehåller också referenser till två dimensionstabeller, nämligen “dim_devices” och “dim_descriptions”. Dessa dimensionstabeller ger ytterligare information om enheter och beskrivningar.

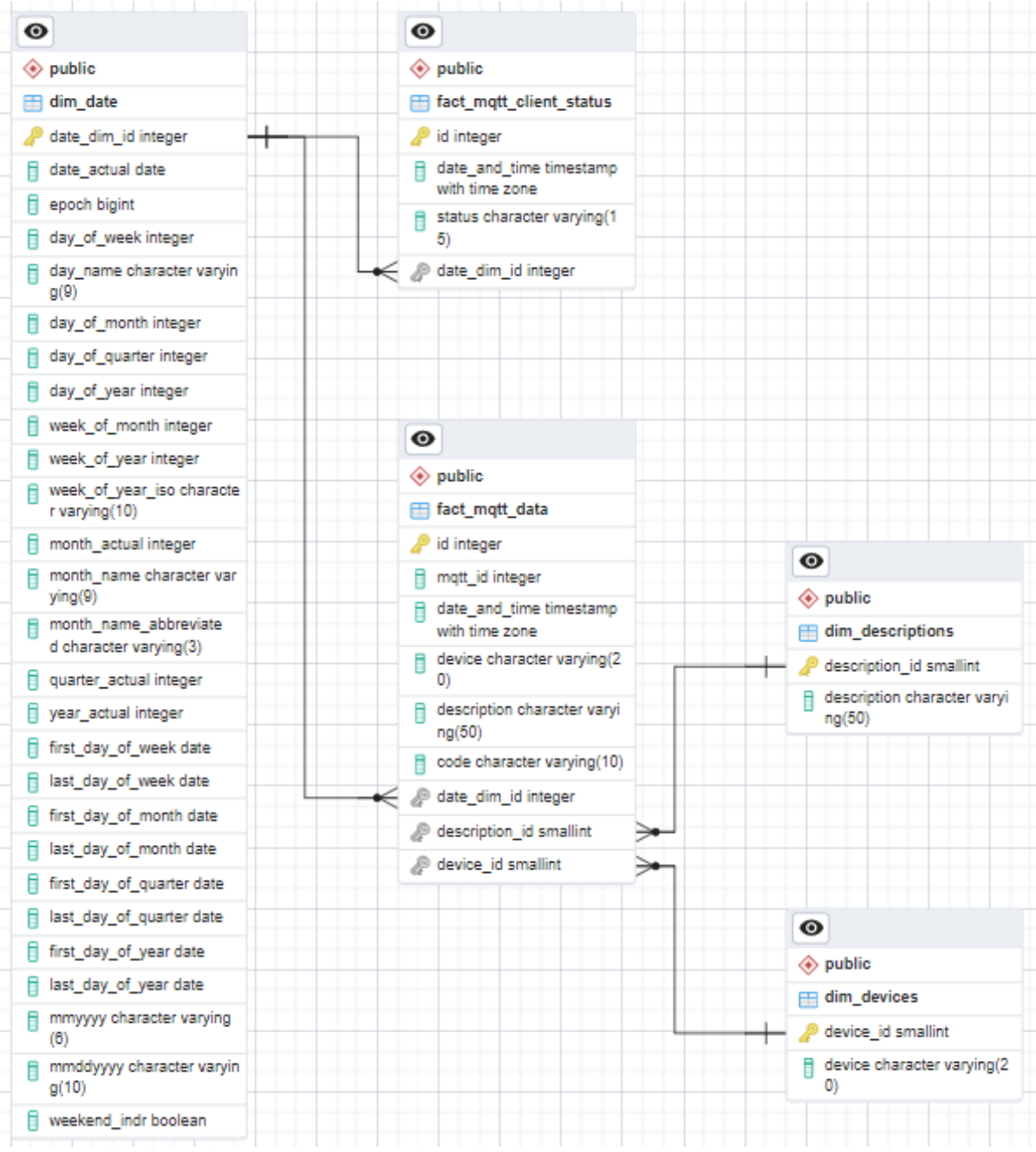
Genom att använda dimensionstabeller som referenser kan vi göra mer avancerade analyser och visualiseringar.

Vi har ytterligare en tabell i vår databas som heter “django_migrations”. Denna tabell används av ramverket Django för att upptäcka ändringar i databasstrukturen och utföra nödvändiga databasåtgärder för att anpassas efter dessa ändringar.

Varje gång vi gör en förändring i vår databasstruktur, såsom att lägga till eller ta bort en tabell eller en kolumn, skapar Django en migrationsfil som beskriver ändringen. Denna migrationsfil upptäcks sedan i “django_migrations” tabellen till dess att ändringen faktiskt

har genomförts i databasen.

Figur 23 visar det Entity Relationship Diagram (ERD) som skapades i pgAdmin för vår databas. Diagrammet representerar de olika tabellerna i vår databas och deras relationer till varandra. Varje rektangel i diagrammet representerar en tabell, och dess kolumner visas inuti rektangeln. Linjerna mellan entiteterna representerar relationerna mellan dem och indikerar vilken typ av relation som finns mellan tabellerna, till exempel en till många eller en till en relation.



Figur 23. ERD av vår databas

5.4.2 Databastjänsten

Databastjänsten i vårt projekt används enbart för att lagra data och innehåller ingen logik. Istället hanteras all logik kring datahantering av API-tjänsten som är kopplad till databasen. Detta innebär att vår databas fungerar som en passiv lagringsplats för data, medan API-tjänsten tar hand om allt som rör logiken kring hantering av datan.
Genom att klart skilja på databasens roll som en ren datalagring och API-tjänstens roll som en logisk hanterare av datan, kan vi skapa en mer flexibel och skalbar arkitektur för vårt projekt. Detta var inte ett krav från uppdragsgivaren, men genom att använda denna design kan vi enklare göra ändringar i logiken utan att det påverkar databasstrukturen och vice versa.

5.5 API

I projektet utförde vi först formatering och filtrering av data i MQTT-klienten. För att öka flexibiliteten och underlätta hanteringen av data mellan MQTT-klienten, databasen och visualiseringsverktyget, beslutade vi oss sedan för att skapa en API-tjänst.

Vi implementerade API:et som en brygga mellan MQTT-klienten och databasen. Genom att använda Django REST-ramverket kunde vi enkelt bygga API-tjänster för att hantera dataflöden. På så sätt kunde vi enkelt flytta data från MQTT-klienten till databasen och tillbaka. Vi kunde även använda samma API för att hämta data för visualisering av insamlad data.

För att säkerställa att vårt API fungerade korrekt skapade vi automatiserade tester med pytest och testade anslutningspunkterna med positiva och negativa värden. Vi använde också Insomnia-verktyget för att manuellt testa API:ets anslutningspunkter och verifiera att de olika operationerna fungerade som de skulle och hanterade data korrekt.

För att kunna arbeta med vår API-lösning på olika miljöer skapade vi två konfigurationsfiler: en för att köra API:et lokalt, där vi anslöt till vår testdatabas, och en annan för att ansluta till vår produktionsdatabas som ligger i en Docker-container på vår AWS EC2-instans.

Genom att ha separata konfigurationsfiler för varje miljö kunde vi enkelt byta mellan dessa när vi skulle utveckla och testa vår API-lösning i olika miljöer. Detta gjorde också att vi kunde säkerställa att API:et fungerade korrekt i varje miljö, oavsett var databasen var placerad.

5.6 Visualisering

I projektet valde vi att använda Bokeh som verktyg för att visualisera vår insamlade data. Bokeh är en interaktiv visualiseringsplattform för Python som är specifikt utformad för att hantera data. Vi ansåg att detta verktyg passade våra behov, då det tillät oss att skapa anpassade, interaktiva diagram och visualiseringar. Med hjälp av Bokeh-verktyget uppfyller vi krav 4 från vår kravspecifikation.

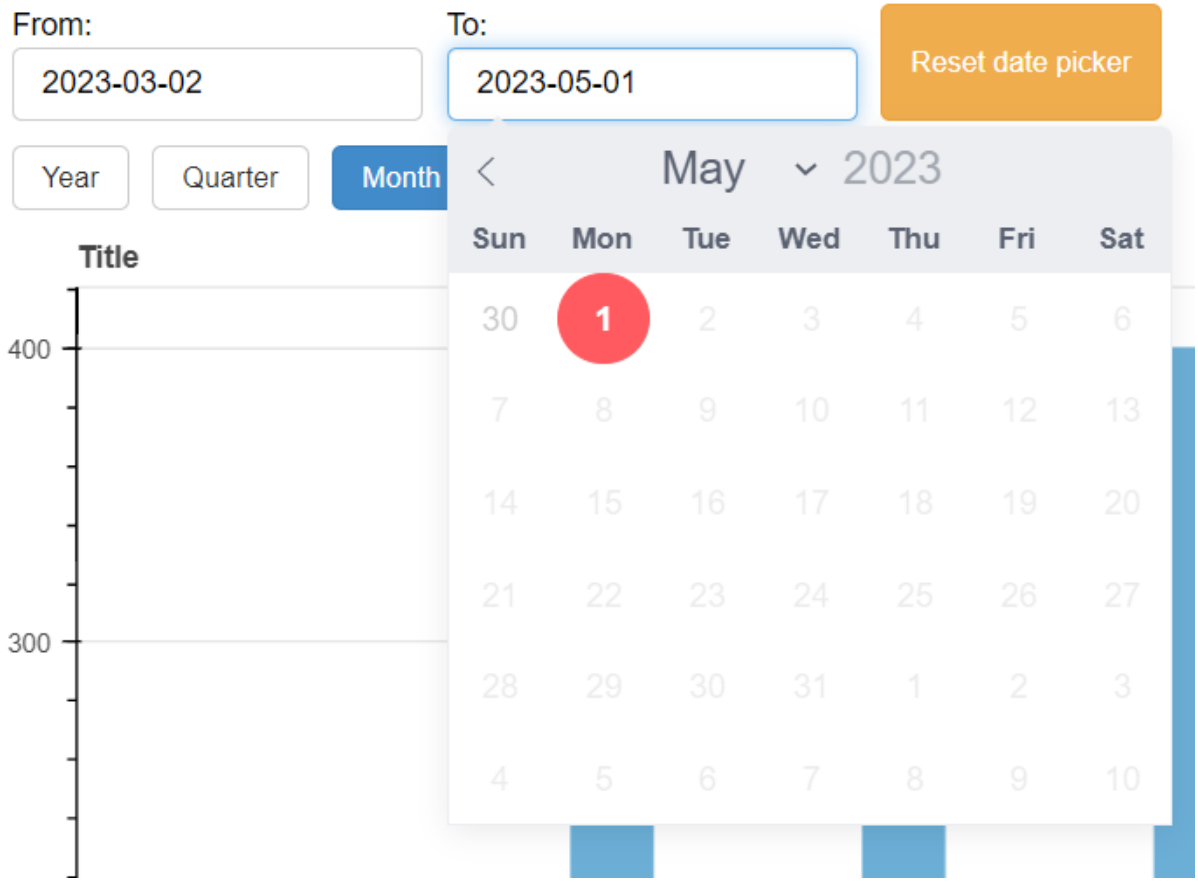
För att effektivt hantera och manipulera vår data valde vi att använda Pandas DataFrame. Genom att organisera vår data i rader och kolumner kunde vi enkelt bearbeta och anpassa den efter våra behov. Vi utnyttjade också Pandas DataFrames funktionalitet för att filtrera och gruppera vår data på ett effektivt sätt. Detta var särskilt viktigt för att kunna skapa detaljerade datavisualiseringar och för att analysera vår data på ett mer djupgående sätt.

Vi ville också ha funktionalitet för att kunna välja specifika enheter och också kunna specificera händelser. För det behövde vi använda oss av “callback-metoden”, det vill säga en funktion som uppdaterar graferna efter vad användaren väljer via interaktion.

5.6.1 Filtrering av data enligt en vald period

Vi har implementerat två Bokeh DatePickers som används för att välja en period. Den ena väljer ett från-datum och den andra ett till-datum. Vi har satt standardvärdena så att till-datumet är dagens datum och från-datumet är 60 dagar bakåt i tiden från dagens datum och har också inkluderat en reset-knapp bredvid DatePickers för att användaren enkelt ska kunna återställa valda datum till standardvärdena.

Dessutom har vi satt begränsningar för användaren. Användaren kan inte välja ett till-datum som är före från-datumet och kan inte välja ett datum som är efter dagens datum i till-datumet. Dessa begränsningar hjälper användaren att undvika felaktiga val av datum och därigenom förbättra användarupplevelsen. Se figur 24 för en visuell representation av DatePickers och reset-knappen.

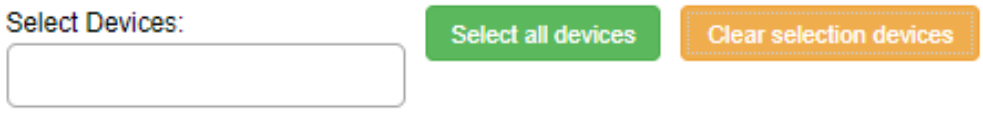


Figur 24. Representation av Bokeh DatePickers och reset-knappen

5.6.2 Filtrering av data enligt enheter och beskrivningar

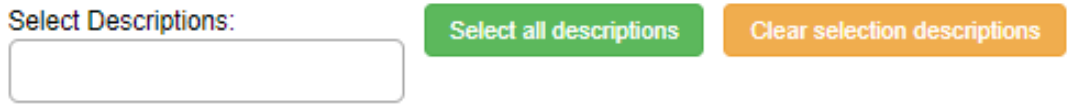
För att filtrera data använder vi också Bokeh Multichoice-widget, som möjliggör att användaren får välja flera alternativ från en lista. När användaren gör ett val uppdateras visualiseringen automatiskt för att visa enbart den data som motsvarar valda alternativ.

I figur 25 visas en Bokeh Multichoice-widget som är en dropdown-lista och används för att filtrera data enligt enheter, baserat på perioden som valdes i punkt 5.6.1. Enheterna listas automatiskt baserat på den valda perioden, och användaren kan välja en eller flera enheter från listan genom att klicka på dem eller söka efter dem i sökfältet på widgeten.



Figur 25. Filter för enheter

I figur 26 visas en Bokeh Multichoice-widget som används för att filtrera data enligt beskrivningar, baserat på perioden som valdes i punkt 5.6.1 och enheterna som valdes i filter för enheter. I det fältet kan användaren också skriva in sökord för att filtrera beskrivningarna.



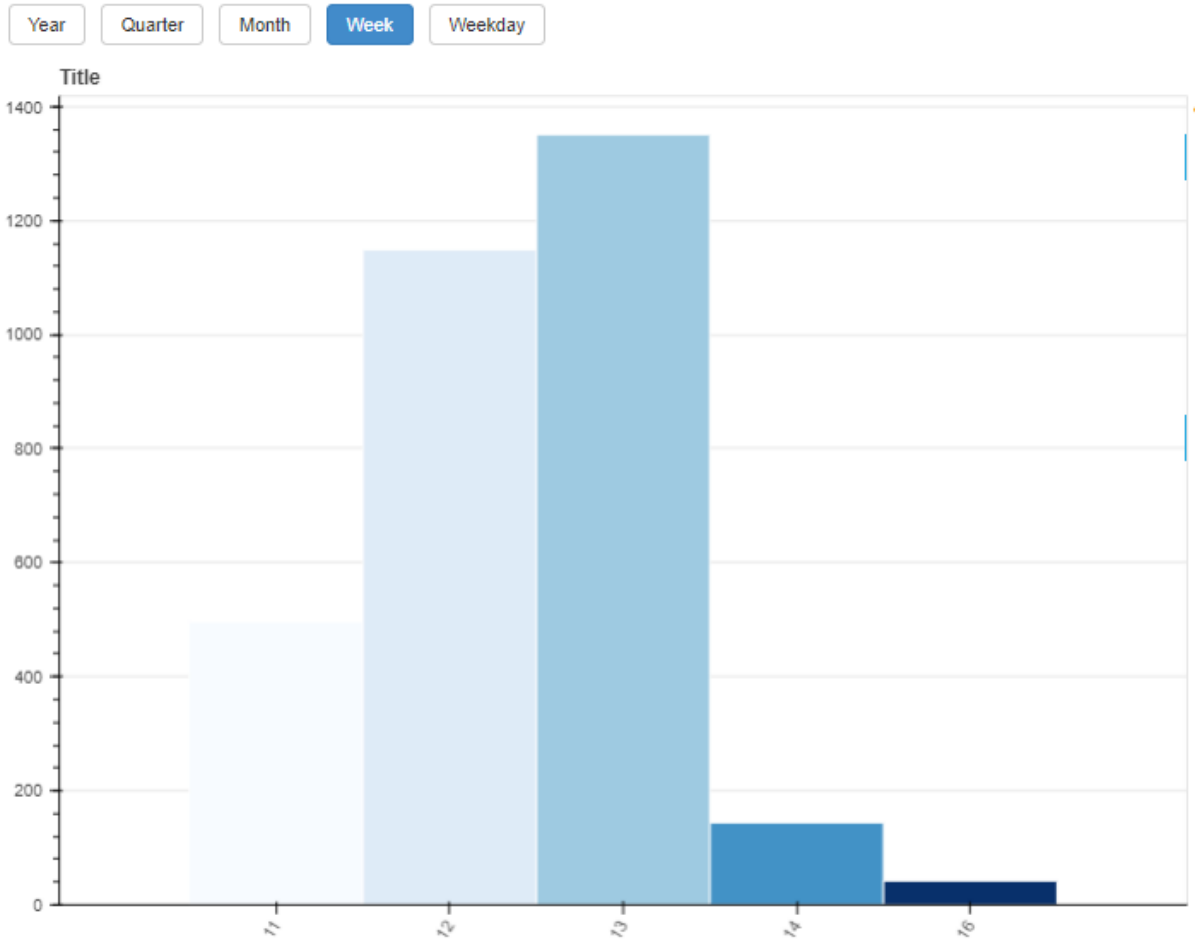
Figur 26. Filter för beskrivningar

För att göra filtreringen ännu enklare för användaren har vi inkluderat två knappar för varje Bokeh Multichoice-widget. Den ena knappen väljer alla alternativ från listan medan den andra rensar alla valda alternativ. Denna design förenklar användarens filterprocess och ger dem möjlighet att välja önskade alternativ på ett enkelt sätt.

5.6.3 Gruppering av data efter tid

Vi har utvecklat en funktionalitet som gör det möjligt för användaren att gruppera data efter år, kvartal, månad, vecka och veckodag samtidigt som antalet händelser räknas. För att åstadkomma detta har vi använt oss av tabellen "dim_date" som innehåller detaljerad information om datum. Genom att använda denna tabell kunde vi effektivt implementera denna funktionalitet.

Figur 27 visar hur användaren kan välja hur data ska grupperas i ett stolpdiagram genom att trycka på knapparna för år, kvartal, månad, vecka eller veckodag. Genom att välja en av dessa knappar uppdateras visualiseringen.

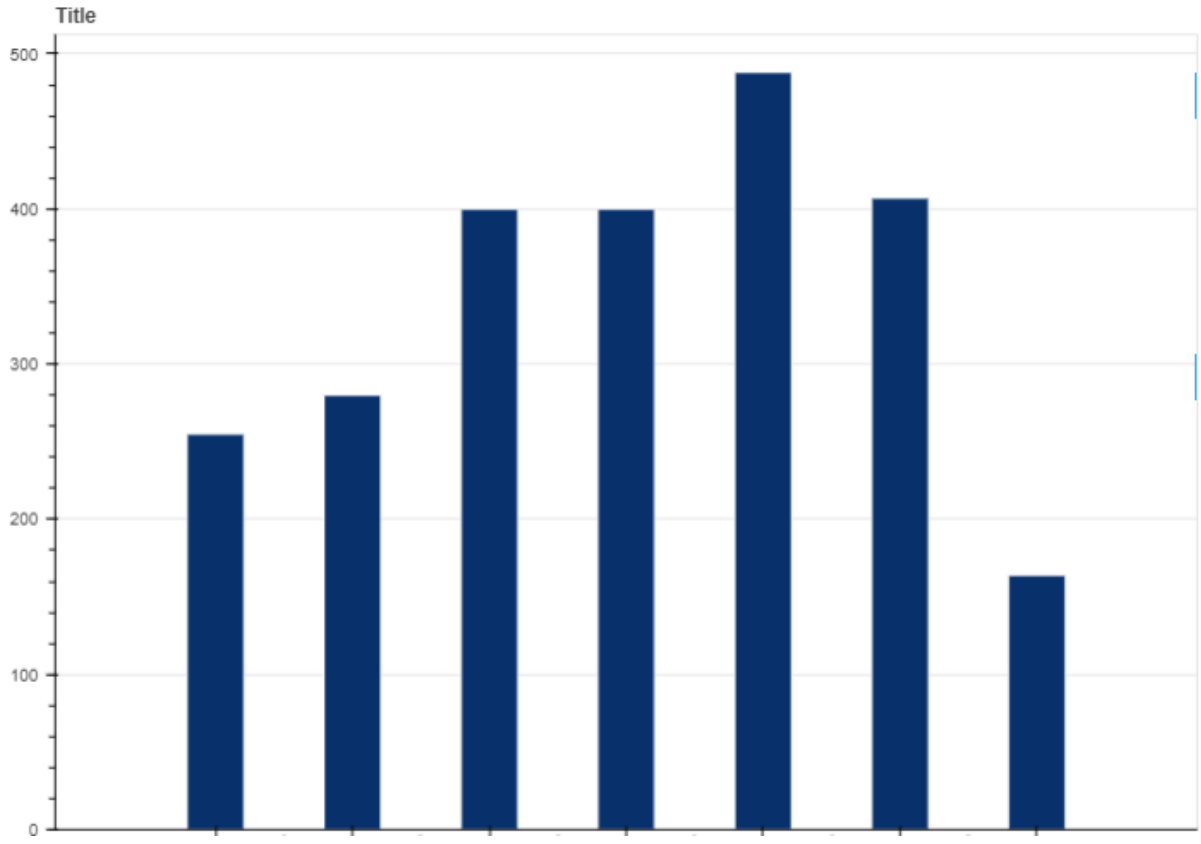


Figur 27. Stolpdiagram av gruppering av data efter vecka

5.6.4 Gruppering av data efter enhet

Vi har valt att gruppera data efter enheter för att underlätta för användare att få en överblick över hur olika enheter presterar. Genom att visualisera data på detta sätt kan användare enklare identifiera eventuella mönster eller avvikelser mellan enheterna och därmed göra mer informerade beslut. Detta är särskilt användbart när man vill analysera trender över tid eller när man behöver göra jämförelser mellan olika enheter.

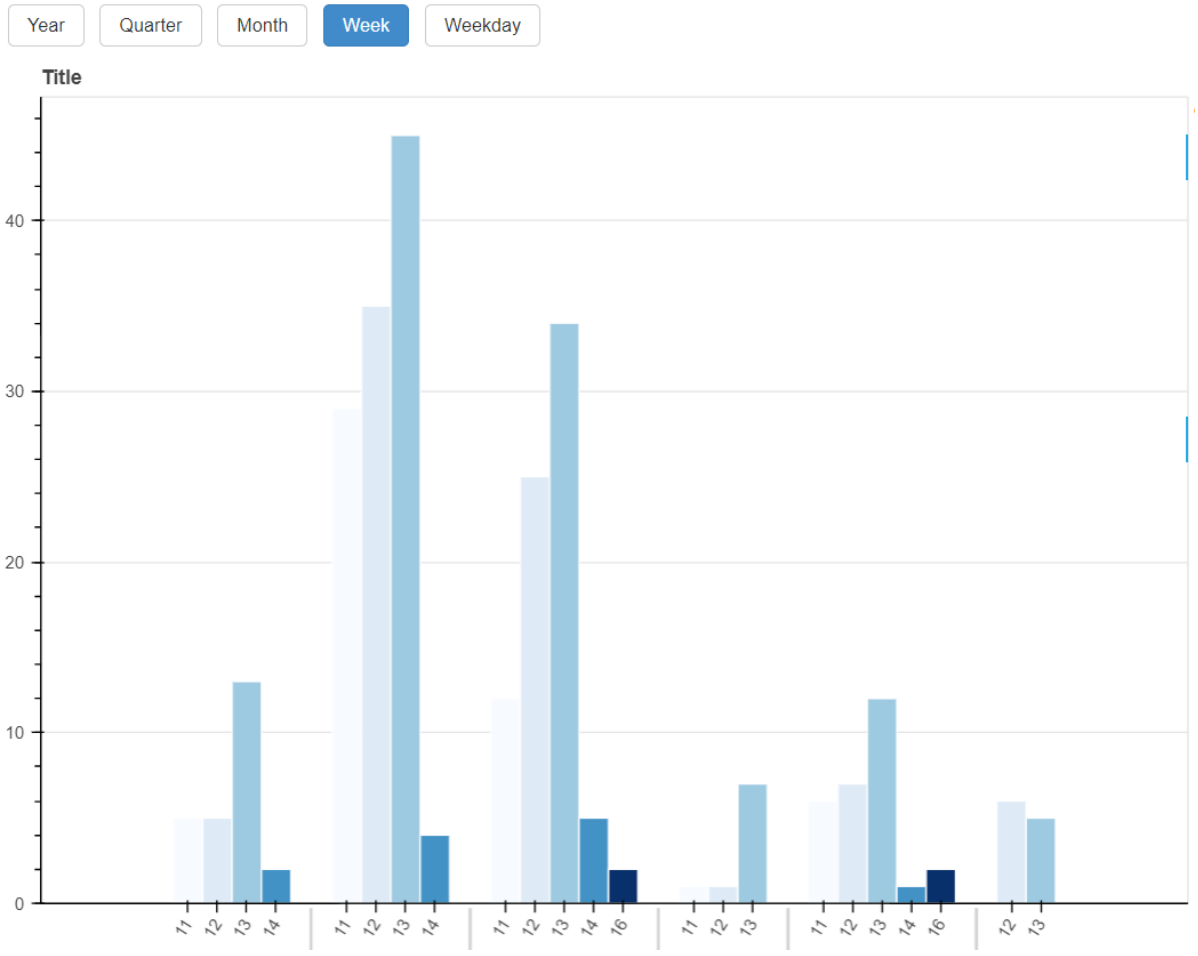
För att ge en bättre överblick över antalet händelser för varje enhet har vi skapat ett stolpdiagram i figur 28. Varje stolpe i diagrammet representerar en enhet som har valts med hjälp av Bokeh Multichoice-widgeten för enhetsfiltrering.



Figur 28. Stolpdiagram av gruppering av data efter enhet

5.6.5 Visualisering av gruppering av data efter vecka och enhet

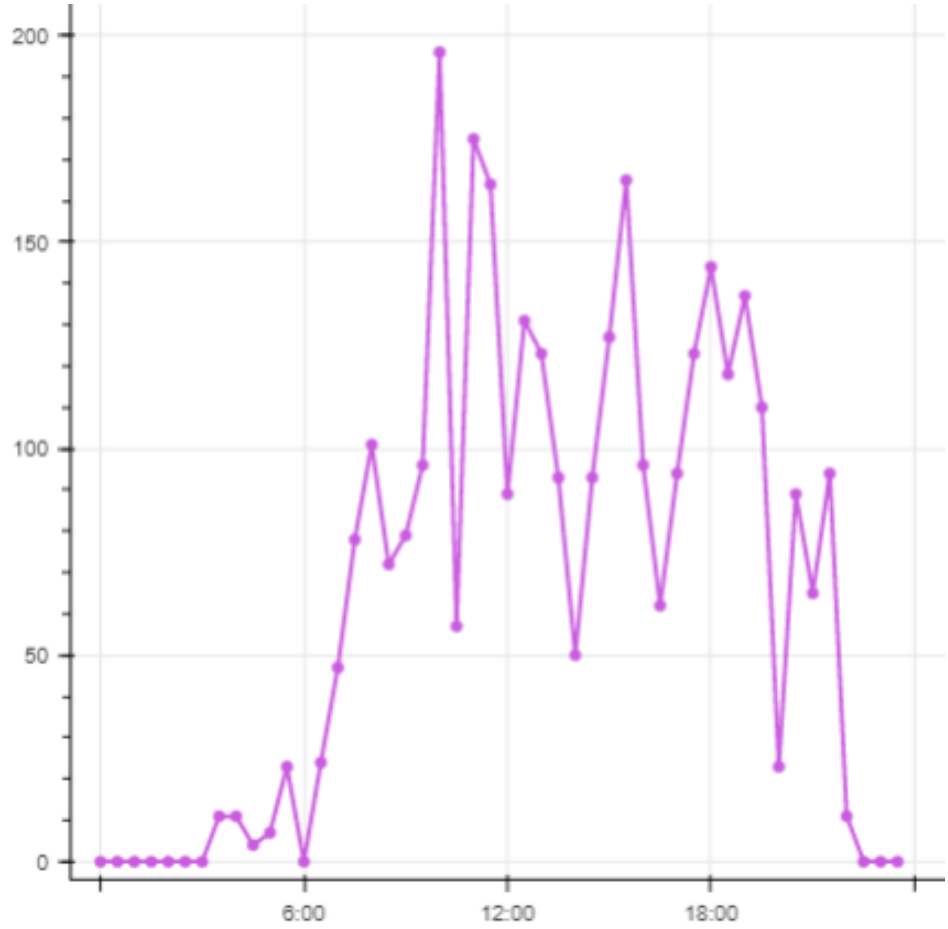
I figur 29 visas ett stolpdiagram med gruppering av data efter vecka, som visar antalet händelser för varje enhet. För att underlätta förståelsen av datan och för att framhäva viktiga trender, har vi valt att färglägga staplarna baserat på hur nära varje vecka är till det aktuella datumintervallet. De veckor som är närmast "till"-datumet får en mörkare färg, medan de äldre veckorna får en ljusare färg.



Figur 29. Stolpdiagram som visar gruppering av data efter vecka och enheter

5.6.6 Visualisering av antalet händelser över tid

I figur 30 visas ett linjediagram där x-axeln representerar tiden från 0:00 till 24:00, medan y-axeln visar antalet händelser. Tiden har grupperats i 30-minuters intervaller. Diagrammet ger en tydlig överblick över hur antalet händelser förändras över tid.

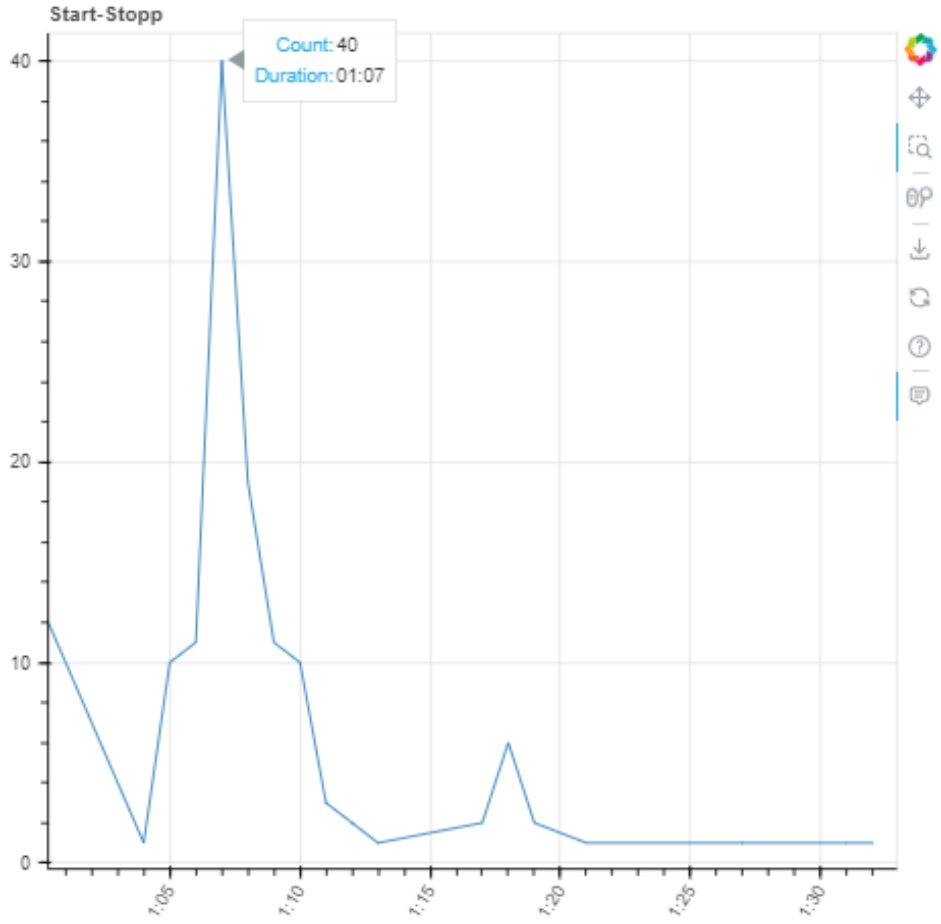


Figur 30. Linjediagram av antalet händelser över tid

5.6.7 Visualisering av tidsförlopp

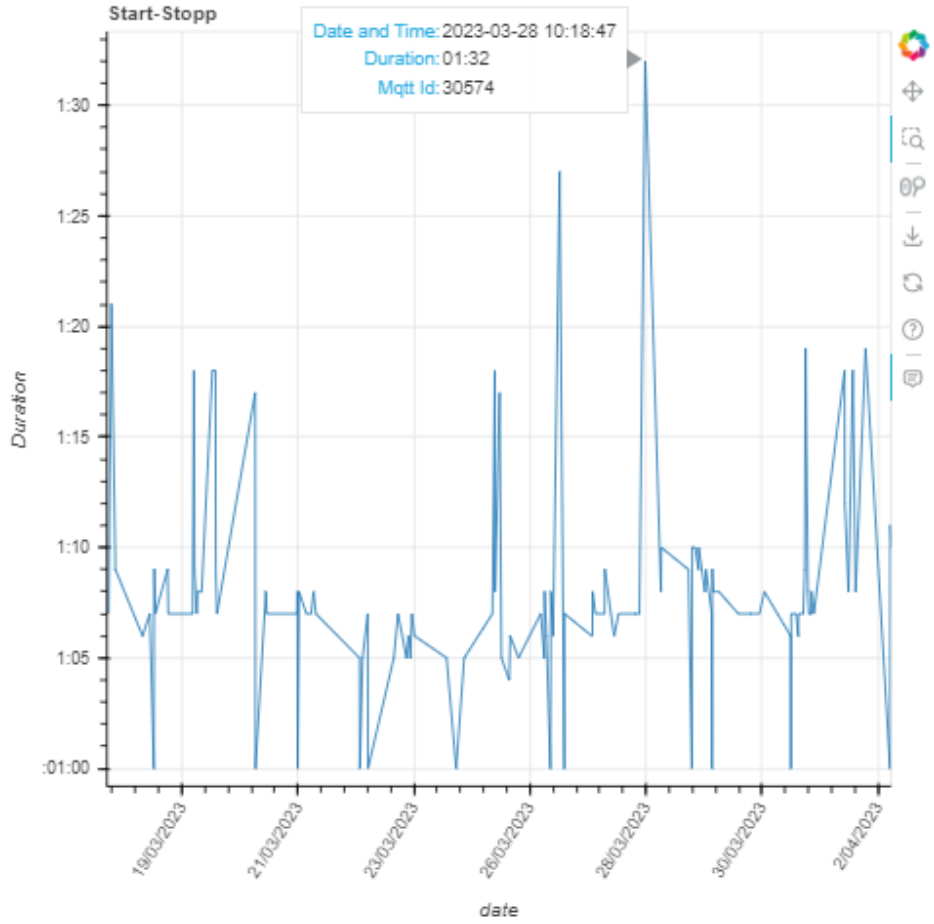
Figur 31 visar ett linjediagram som visar hur lång tid det tog för en händelse att genomföras från start till stopp och hur antalet händelser varierar över tid.

På x-axeln visas tiden i minuter och på y-axeln visas antalet händelser som har genomförts.



Figur 31. Linjediagram av tidsförloppet för händelser baserat på varaktighet i minuter och antalet händelser

Figur 32 är också ett linjediagram där x-axeln visar datumet då händelsen genomförs och y-axeln visar den totala tiden i minuter som varje händelse tog att genomföra. Genom att analysera grafen kan man identifiera händelser som tog längre tid att genomföra.



Figur 32. Linjediagram av tidsförloppet för händelser baserat på datum och varaktighet i minuter

Med hover-funktionen kan användaren se mer detaljerad information genom att föra muspekaren över en viss punkt på grafen. Dessutom finns ett verktygsfält på höger sida av diagrammet som ger användaren möjlighet att zooma, återställa vyn och spara som fil. Genom att analysera linjediagrammet kan det ses hur länge det tar för en händelse att genomföras och hur antalet genomförda händelser varierar över tid, vilket kan hjälpa till att identifiera potentiella flaskhalsar och förbättringsmöjligheter.

6. SLUTSATS

I detta kapitel redogörs för slutsatser av arbetet.

6.1 Resultat

Vårt mål var att samla och visualisera data från en sopsugsstation för lättare kunna avläsa dess prestanda och effektivitet. Detta gjordes genom att skicka datan över MQTT-protokollet, skicka det till databasen via ett API och sedan vidare till Django för att visualisera det med hjälp av Bokeh.

Implementationen blev mera komplex än vi initialt trodde och vissa delar tog mycket längre tid än vi förväntat oss. Fastän vi lyckats med att samla och visualisera data så finns det mycket funktionalitet kvar att göra innan det är en klar produkt. Den implementation vi hann färdigt med blev både flexibel och tjänsterna blev väl separerade från varandra. Vi har också uppfyllt alla våra högprioritets-krav.

En stor del av projektet har genomförts genom parprogrammering, vilket innebär att vi har arbetat tillsammans på samma kod. Vi har lärt oss många nya saker tillsammans, eftersom det fanns många nya områden som var obekanta för oss båda. Vid problem som var lätta att lösa individuellt, arbetade vi enskilt och bad om hjälp när det behövdes. Vi har också delat våra arbetserfarenheter och förklarat för varandra om det uppstod några missförstånd. Efter individuellt arbete har vi granskat och diskuterat varandras kod vid nästa tillfälle för att säkerställa kvaliteten på koden och samarbetet.

Vi har presenterat projektet åt uppdragsgivaren.

6.2 Reflektioner

Även om projektet blev mer komplext än vi förväntade oss och vi stötte på problem längs vägen, så har det ändå varit en intressant och lärorik erfarenhet. Vi har även kunnat reflektera över hur arbetet hade kunnat utföras bättre.

Förbättringen som skulle ha haft störst effekt hade varit om vi från början haft en bättre förståelse av projektet. Vi upptäckte att vi behövde ändra implementationen flera gånger eftersom vi inte helt förstod hur allt skulle fungera, speciellt med tanke på att vi själva behövde hitta lösningar för all funktionalitet. Det hade också varit bättre att noga planera varje steg innan vi började implementera för att undvika att behöva göra om kod och spara tid.

Efter en tids arbete med projektet insåg vi också att vi borde ha valt ett annat verktyg än Bokeh för att skapa layouten på webbsidan. Bokeh är framför allt lämpligt för att skapa interaktiva visualiseringar, men har begränsade möjligheter när det gäller att skapa en välstrukturerad layout. För att skapa en bra layout på webbsidan är det både enklare och mer flexibelt att använda Django. Bokeh kan sedan användas för att skapa interaktiva visualiseringar och diagram som ska visas på sidan.

För att underlätta felsökning och förståelse av koden skulle det ha varit till hjälp att skapa fler tester från början.

6.3 Framtida utveckling

Det finns mycket man kan förbättra med det här projektet, till exempel:

- Mera tester för att bättre upptäcka fel och buggar
- Bättre säkerhet för alla tjänster, till exempel lägga till rättighetsnivåer för dataåtkomst
- Mera strukturerad och tilltalande layout med hjälp av HTML och CSS
- Mera grafer och interaktivitet av graferna för att bättre kunna avläsa data
- Användning av CI/CD med hjälp av Jenkins[12] för att underlätta utvecklingen
- Vyer anpassade för mobil enhet så att slutanvändarna lättare ska kunna använda sig av projektet
- Ta emot feedback från slutanvändare och implementera förbättringar baserat på feedback
- Utveckla API:ets anslutningspunkter med mera filtrering och bättre dokumentation

[12] https://www.jenkins.io/

KÄLLFÖRTECKNING

Amazon RDS. (n.d.). AWS. https://aws.amazon.com/rds/?nc2=h_ql_prod_fs_rds

AWS. (n.d.). *What Is IoT (Internet of Things)?* AWS. https://aws.amazon.com/what-is/iot/

AWS EC2 Instance. (n.d.). Secure and Sizable Cloud Compute - AWS EC2 - Amazon Webservices. https://aws.amazon.com/ec2/

AWS ECS. (n.d.). AWS. https://aws.amazon.com/ecs/

AWS IoT Core. (n.d.). AWS. https://aws.amazon.com/iot-core/?nc2=type_a

BCS Tools - Beijer Electronics. (n.d.). Retrieved April 18, 2023, from https://www.beijerelectronics.com/en/Products/software/BCS___Tools

Bernstein, C., Brush, K., & Gillis, A. S. (2021, January 27). *What is MQTT and How Does it Work?* IoT Agenda; TechTarget. https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport

Björnvik, J. (2021). Blokus3D - *Ett klassiskt brädspel producerat med Unity och C#* [Examensarbete, Högskolan på Åland]. Theseus. https://urn.fi/URN:NBN:fi:amk-202105179021

Cloud Computing Services. (n.d.). AWS. https://aws.amazon.com/

CODESYS Group. (2023, April 27). https://www.codesys.com/

Django introduction. (n.d.). MDN Web Docs. Retrieved March 29, 2023, from https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

Docker overview. (2023, March 21). Docker Documentation. https://docs.docker.com/get-started/overview/

Dock, M. (n.d.). *Docker: Accelerated, Containerized Application Development*. Retrieved May 1, 2023, from https://www.docker.com/

Download PuTTY - a free SSH and telnet client for Windows. (n.d.). Retrieved April 24, 2023, from https://www.putty.org/

Eclipse Mosquitto. (2018, January 8). Eclipse Mosquitto. https://mosquitto.org/

Eclipse Paho. (2013, January 31). Projects.eclipse.org. https://projects.eclipse.org/projects/iot.paho

Hansen, K. (2022). *MIGRERING AV EN BEFINTLIG WEBBAPPLIKATION - från funktionsorienterad till objektorienterad kod i PHP* [Examensarbete, Högskolan på Åland]. Theseus. https://urn.fi/URN:NBN:fi:amk-2022100120715

Human-Machine Interface (HMI) - Glossary. (n.d.). Retrieved April 29, 2023, from https://csrc.nist.gov/glossary/term/human_machine_interface

Introducing WinSCP. (n.d.). Retrieved April 24, 2023, from https://winscp.net/eng/docs/introduction

MQTT Version 3.1.1. (n.d.). Retrieved May 1, 2023, from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Nu öppnar hallonbergens sopsug. (n.d.). Retrieved April 29, 2023, from https://www.savab.se/nu-oppnar-hallonbergens-sopsug/

OpenJS Foundation. (n.d.). *Node-RED*. Node-RED. Retrieved March 21, 2023, from https://nodered.org/

pandas. (n.d.). Retrieved April 29, 2023, from https://pandas.pydata.org/

PgAdmin - PostgreSQL Tools. (n.d.). PgAdmin. Retrieved April 24, 2023, from https://www.pgadmin.org/

pytest: helps you write better programs — pytest documentation. (n.d.). Retrieved April 29, 2023, from https://docs.pytest.org/en/7.3.x/

Så fungerar sopsugen. (2020, October 20). Envac. https://www.envac.se/envacs-sopsugsystem/sa-fungerar-sopsugen/

The Collaborative API Development Platform. (n.d.). Retrieved April 24, 2023, from https://insomnia.rest/

Van de Ven, B. (n.d.). *Bokeh*. Bokeh.org. Retrieved April 19, 2023, from https://bokeh.org/

What Is a Programmable Logic Controller (PLC)? (n.d.). TechTalk Blog. Retrieved April 29, 2023, from https://www.polycase.com/techtalk/electronics-tips/what-is-a-programmable-logic-controller.html

Wikipedia contributors. (n.d.). *Relationsdatabas*. Wikipedia, The Free Encyclopedia. https://sv.wikipedia.org/w/index.php?title=Relationsdatabas&oldid=49899598

Wikipedia contributors. (2022, February 9). *Software prototyping*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Software_prototyping&oldid=1070741889

Wikipedia contributors. (2023a, February 18). *Dimension (data warehouse)*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Dimension_(data_warehouse)&oldid=1140121428

Wikipedia contributors. (2023b, February 21). *Industrial internet of things*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Industrial_internet_of_things&oldid=1140771158

Wikipedia contributors. (2023c, April 13). *MQTT*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=MQTT&oldid=1149596857

Wikipedia contributors. (2023d, April 18). *Microservices*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Microservices&oldid=1150585909

Wikipedia contributors. (2023e, April 26). *Entity–relationship model*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Entity%E2%80%93relationship_model&oldid=1151859798

Wikipedia contributors. (2023f, April 26). *Programmable logic controller*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Programmable_logic_controller&oldid=1151797879

Wikipedia contributors. (2023g, May 14). *API*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=API&oldid=1154821274