Arkitekturdokument

Sagar Batra
August Gustafsson
Erik Hellberg
Fredrik Holm
Philip Norberg
Linus Roos
Johannes Tarkka
Elis Öhman
Jakob Österberg

6 juni 2023

Version 3.0

Projektidentitet

Grupp E-post: TDDD96_2023_12@groups.liu.se

Kund: Fredrik Wanhainen, Neue
E-post: fredrik.wanhainen@neue-labs.com

Handledare: Dag Harju
E-post: dagha100@student.liu.se

Kursansvarig: Kristian Sandahl
E-post: kristian.sandahl@liu.se

Projektdeltagare

Namn	**Ansvar**	**E-post**
Sagar Batra	Analysansvarig	sagba622@student.liu.se
August Gustafsson	Konfigurationsansvarig	auggu628@student.liu.se
Erik Hellberg	Testledare	erihe217@student.liu.se
Fredrik Holm	Teamledare	freho169@student.liu.se
Philip Norberg	Arkitekt	phino637@student.liu.se
Linus Roos	Dokumentansvarig	linro911@student.liu.se
Johannes Tarkka	Driftsättningsansvarig	johta890@student.liu.se
Elis Öhman	Kvalitetssamordnare	elioh505@student.liu.se
Jakob Österberg	Utvecklingsledare	jakos322@student.liu.se

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	24/01/2023	Första utkast	Alla	Alla
1.0	17/02/2023	Första version inför inlämning 1	Philip	Johannes
2.0	09/03/2023	Andra version inför inlämning 2	Philip, Elis	August
3.0	14/04/2023	Tredje version inför inlämning 3	Philip	Elis

INNEHÅLL

1 INLEDNING

1.1 Syfte

Detta dokument beskriver filosofin, beslut, begränsningar, motiveringar, viktiga delar och andra övergripande aspekter av systemet som formar designen och implementeringen. Det fungerar även som en manual för utvecklarna till systemet, samt som ett översiktsdokument för kunden av systemet.

1.2 Tänkta läsare

Dokumentet är skapat för utvecklarna av systemet och kunden av systemet som slutprodukt.

1.3 Definitioner och förkortningar

API
API står för Application Programming Interface och är ett gränssnitt som tillåter olika programvaror att kommunicera med varandra.

Amazon Web Services
Amazon Web Services (AWS) är en molnplattform som erbjuder en mängd olika molntjänster, inklusive datorresurser, databaser, lagring, nätverk, utvecklingsverktyg, säkerhet och mer.

AWS IoT Core
AWS IoT Core är en molntjänst från AWS som möjliggör anslutning och hantering av Internet of Things (IoT)-enheter.

Big Data Analytics
Big Data Analytics (BDA) är en process där stora mängder data analyseras för att upptäcka mönster, trender och insikter som kan användas för att fatta beslut.

BigQuery
En molnbaserad data warehouse-tjänst från Google Cloud som gör det möjligt att lagra, hantera och analysera stora mängder data med hjälp av SQL-frågor.

BlockBax
En low-code mjukvaruplattform för att skapa program och appar.

Flask
Flask är ett Python-ramverk för att skapa webbapplikationer.

Google Cloud Function
Google Cloud Functions är en serverless computing-tjänst som låter utvecklare köra kod i Googles molninfrastruktur utan att behöva hantera virtuella maskiner eller fysiska servrar. Funktioner kan skrivas i

flera språk och används för att hantera kortlivade uppgifter som automatisering av uppgifter och dataströmmar.

Google Looker
Google Looker är en molnbaserad plattform för affärsintelligens som tillhandahåller verktyg för att visualisera och analysera data från olika källor.

HTTP
HTTP står för Hypertext Transfer Protocol och är ett protokoll som används för att överföra data över internet.

Low-code
Low-code teknik innebär att man kan bygga mjukvara och applikationer genom att dra och släppa färdiga komponenter och använda ett visuellt gränssnitt istället för att skriva kod från grunden.

Microsoft Azure
Microsoft Azure är en molnplattform som erbjuder en rad olika molntjänster, inklusive databaser, virtualisering, lagring, nätverk, analys, maskininlärning och mer.

Neue
Ett Linköpingsbaserat teknikföretag med inriktning mot sensortjänster.

Noodl
En low-code mjukvaruplattform för att skapa program och appar.

Pub/Sub
Pub/Sub står för ”Publish/Subscribe” och är en arkitekturmodell för meddelandeutbyte mellan olika programkomponenter eller system. I Pub/Sub-modellen skickar avsändare (eng. *publisher*) meddelanden till en mellanhand (eng. *broker*) som i sin tur distribuerar meddelandena vidare till mottagare (eng. *subscriber*).

React
React är ett JavaScript-bibliotek för att skapa användargränssnitt i webbläsare och mobila appar.

Recharts
Recharts är ett open source-bibliotek som används för att skapa interaktiva och anpassningsbara diagram och grafer i React-baserade webbapplikationer.

REST API
REST API står för Representational State Transfer Application Programming Interface. Det är en arkitekturstil för att skapa webbtjänster som använder HTTP-protokollet för att överföra data.

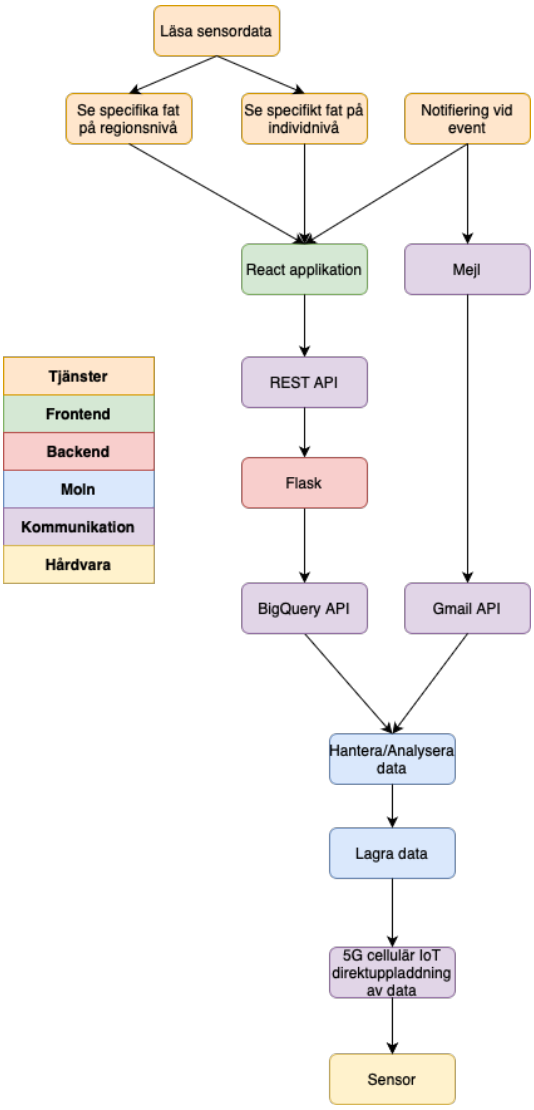
SendGrid
SendGrid är en molnbaserad mejlplattform för att hantera och skicka mejl.

SQL
Structured Query Language (SQL) är ett programspråk som används för att hantera och manipulera databaser.

2 SYSTEMÖVERSIKT

I denna sektion ges en översikt och detaljerad vy av systemet.

2.1 Övergripande vy



Figur 1: Systemvy

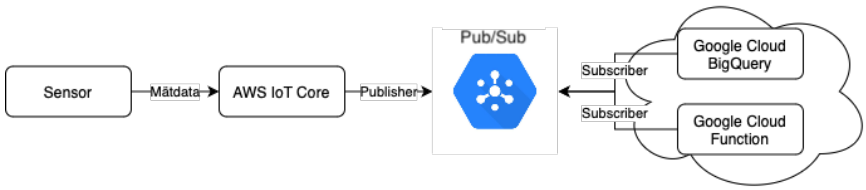
2.2 Dataflöde detaljvy

Denna sektion beskriver mätdatans väg från sensor till applikation.

Molnet är kärnan i systemet, där merparten av datan lagras och hanteras. En del av datan lagras även i back-endens buffer. Data strömmar till och från molnet på olika sätt och via olika initiativ. Dessa sätt beskrivs i underrubrikerna nedan.

2.2.1 *Till moln*

En sensor som är placerad på ölslangen och mäter flödet skickar flödesdata till AWS IoT Core. Denna data publiceras sedan av AWS i Google Cloud Pub/Sub som en topic. BigQuery och Cloud Function prenumerar på denna topic för att erhålla flödesdatan. All flödesdata lagras, hanteras och analyseras i BigQuery. Cloud Functions syfte är att hitta anomalier i flödesdatan, som till exempel för hög uppmätt temperatur. Detta är relaterat till kundkravet [1] om eventhantering. Överföringsprocessen mellan sensor och moln sker kontinuerligt med ett givet tidsintervall, givet att sensorn ej kopplats bort från ölslangen.



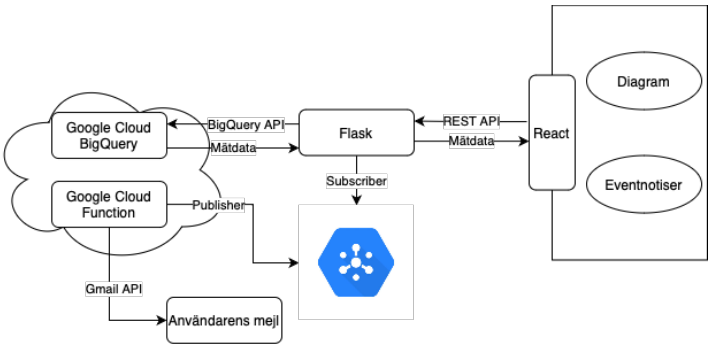
Figur 2: Dataflöde till moln

2.2.2 *Från moln*

När mätdatan går från moln till användare finns två vägar. I den ena vägen går mätdatan till applikationen, och i den andra vägen går mätdatan till användarens mejl. Den andra vägens syfte är att anomalidata ska kunna nå användaren även när denne inte använder applikationen.

Till applikation

Hämtning av mätdata från moln till applikation initieras när användaren öppnar applikationen. React skickar då REST API-anrop över HTTP-protokollet till back-enden i Flask. Flask tar emot anropet, tolkar innehållet och begär den önskade datan från molnet genom BigQuery API. Samtidigt som detta sker hämtar Flask potentiellt existerande anomalidata som Cloud Function publicerat i en Pub/Sub-koppling mellan Cloud Function och Flask. När Flask har utfört begäran från React samt hämtat potentiell anomalidata, svarar den React med denna data. React använder sedan datan för att uppdatera gränssnittet i applikationen.



Figur 3: Dataflöde från moln

Till användarens mejl

Om en anomali upptäcks i mätdatan använder Cloud Function sig av Gmail API för att varna användaren. Cloud Function förbereder ett mejl med relevant information om anomalin, till exempel den uppmätta temperaturen i det fallet då temperaturen var en anomali. Sedan skickas mejlet med hjälp av Gmails API till den anvgina mejladressen för användaren.

2.3 Front-endarkitektur

Detta är förklaring av figurerna i bilagorna.

När användaren öppnar applikationen kommer denne till välkomstskärmen. Här ges en översiktsvy av världen i form av en karta. En tabell visas också med statistik på världsnivå. T.ex totalt antal liter i världen etc.

Användaren kan sedan klicka på en notifikationsknapp för att få fram notifikationer som gäller hela världen.

Om användaren vill gå ner på mindre nivå, aka kontinentnivå, så kan denne klicka på en kontinent. Då kommer samma upplägg av vy som hos världen, fast hos kontinent. Detta ska sedan gå att klickas ner på landsnivå, county/landskapsnivå, stadsnivå och pubnivå.

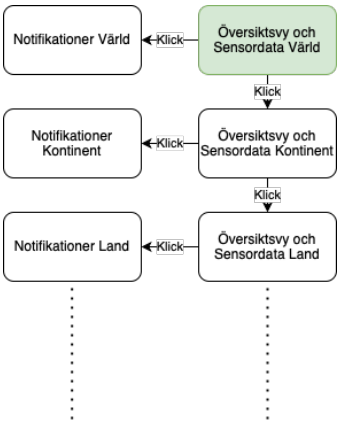
Användaren ska även kunna zooma sig runt i kartan. Alltså ska det finnas möjlighet att nå O'learys i Karlstad genom att zooma sig ner till Karlstad och sedan klicka, istället för att behöva klicka Europa->Sverige->Karlstad->O'learys.

3 MOTIVERING

I denna sektion beskrivs de motiveringar och experiment som låg bakom valen av komponenter som systemet består av. I de fallen Neue beslutat om komponenter saknas motivering.

3.1 Molntjänst

Neue tog beslutet om vilken molntjänst som skulle användas för projektet. Valen stod mellan AWS, Microsoft Azure och Google Cloud. Neue beslutade sig för en mix mellan Google Cloud och AWS.



Figur 4: Front-endarkitektur

3.1.1 *BigQuery*

Efter en dialog mellan Neue och en representant hos Google rekommenderades BigQuery som Big Data Analytics (BDA)-verktyg. Gruppen ansåg att denna rekommendation var tillräcklig, då ytterligare forskning från gruppens sida sannolikt skulle ge marginell ökad nytta jämfört med rekommendationen. Därmed blev projektets BDA-verktyg BigQuery.

3.1.2 *Looker*

Likt BigQuery var Looker en rekommendation från en representant hos Google. Gruppen valde att inte använda Looker. Detta på grund av att gruppen fick tillgång till Looker sent i utvecklingen. Gruppen hade vid detta laget redan en lösning som fungerade och var lättförståelig. Looker valdes även bort på grund av att gruppen upplevde det som komplicerat att exportera grafer från Looker till applikationen.

3.1.3 *AWS IoT Core*

Varje molntjänst tillhandahåller IoT-tjänster. Google Cloud kom dock att under sommaren 2023 stänga ner sin IoT-tjänst. Neue beslutade sig därför att använda sig av AWS IoT-tjänst istället.

3.1.4 *Google Cloud Pub/Sub*

AWS och Google Cloud tillhandahåller Pub/Sub-tjänster. Då gruppen enbart hade tillgång till Google-kontot, och gruppen skulle stå för utvecklingen, blev valet att använda sig av Google Clouds Pub/Sub tjänst istället för AWS.

3.1.5 *Google Cloud Function*

Till moln med Pub/Sub

Övervägd lösning för att hantera anomalidata till molnet var:

- Låt Cloud Function ställa schemalagda SQL-frågor till BigQuery om inkommande mätdata på jakt efter anomalier

Att låta Cloud Function ställa schemalagda SQL-frågor till BigQuery kunde möjligtvis ge upphov till fördröjningar om anomalier. Detta eftersom att anomalidatan, istället för att nå Cloud Function via Pub/Sub direkt, hade varit tvungen att ta vägen genom BigQuery först. Dock låg den största motiveringen bakom Pub/Sub-lösningen ur ett utvecklingsperspektiv. I det fallet då en sensor monterats ner, hade schemalagda förfrågningar fortsatt att skickas, helt förgäves. Utvecklarna av systemet hade då manuellt behövt stänga av de schemalagda förfrågningarna. I ett system med många sensorer hade detta blivit mycket onödigt arbete.

Med Cloud Function som är kopplad till Pub/Sub skulle ingen fördröjning ske. Samt att eftersom Cloud Function enbart skulle köras då ny mätdata landat i Pub/Sub, skulle heller ingen kod exekveras förgäves i det fallet då en sensor monterats ner.

Från moln med Pub/Sub

Lösningen för att skicka anomalidata från Cloud Function till back-enden i Flask blev ytterligare en Google Cloud Pub/Sub-koppling. Detta valdes då gruppen tidigare hade satt upp Pub/Sub-koppling och kände sig bekväma med detta.

Från moln med Gmail

Gmail valdes som mejltjänst då det var en tillhandahållen tjänst av Google, vilka vi vid dåvarande tillfälle var bekväma med att använda. För autentisering använder sig Gmail av OAuth, som gruppen tidigare hade erfarenhet av från andra av Googles tjänster, så som BigQuery API.

En alternativ lösning var SendGrid. För att använda SendGrid behövdes det dock skapas ett konto hos dem, skapa sin egna API-nyckel och konfigurera DNS. Gruppen ansåg att Gmails konfigurationssteg skulle vara smidigare, och valde den lösningen.

3.2 Back-end och front-end

Vad gäller front- och back-end-lösning, hade Neue även där gjort sin egna forskning. Gruppen fick två rekommendationer, Noodl och Blockbax, och uppmanades att välja mellan dessa. Neue var även öppna för alternativa lösningar. Efter att ha granskat Noodl och Blockbax kunde det konstateras att båda tekniker kunde uppfylla samtliga av kraven för vad som behövdes visualiseras i applikationen. Teknikerna uppfyllde också kraven om eventhantering samt integrering med Google Cloud.

Omgående fick gruppen experimentera med Noodl. Gruppen anslöt sig även till Noodls publika Discord-kanal där frågor och funderingar blev besvarade. Till en början såg det ut som att Noodl skulle bli vald som teknik, men det uppstod problematik när data skulle föras över från molnet till Noodl. Under felsökningstiden för detta beslöt sig några gruppmedlemmar att utforska ifall alternativa lösningar hade smidigare kopplingsmöjligheter. Gruppmedlemmarna lyckades snabbt med en kombination av Flask och React. De lyckades, utöver kopplingen, även visualisera en rad av data. Detta fick gruppen att överväga denna lösningskombination över Noodl.

En röstning angående teknikval föreslogs. Innan denna röstning gjordes, experimenterade några gruppmedlemmar med ännu en alternativ lösning. Den lösningen innebar att implementera back-enden med Node.js istället för Flask. Experimentet lyckades, och gruppen hade nu tre teknikval för omröstningen. Node.js valdes som alternativ då gruppen vid forskning om det populäraste sättet att skriva back-end-kod i JavaScript fann Node.js som svar.

3.2.1 *Flask och React*

Röstningen om teknikval gjordes och Flask- och React-lösningen fick majoriteten av rösterna på grund av att:

- Gruppen ansåg att inlärningskurvan för low-code var brantare än att lära sig webbutveckling i React samt programmeringsspråket JavaScript.
- Hälften av gruppmedlemmarna hade erfarenhet av Flask, medans ingen hade erfarenhet av Node.js.
- React huserade den största utvecklargemenskapen, vilket ansågs kunna underlätta för gruppen då kunskapen gällande någon front-end-lösning nästan var obefintlig.
- Gruppen ansåg att React uppfyllde de funktionella kraven, i form av ett stort utbud av bibliotek gällande att visualisera data.

4 KRAVUPPFYLLNING

I denna sektion kopplas arkitekturen till kravspecifikationen [1] och besvarar hur arkitekturen uppfyller dessa.

4.1 Funktionella krav

- Användandet av biblioteket Recharts uppfyller kraven 1, 2, 3, 4, 5, 6, 7, 9 och 13 som säger att olika typer av sensordata skall kunna visualiseras.
- Med hjälp av BigQuery uppfylls krav 10 som handlar om lagring i molnet.
- De arkitektoriska valen som innefattar Cloud Function, SendGrid och Pub/Sub-koppling till back-enden i Flask uppfyller krav 12 som handlar om notifikationer.
- Vi använder React för att skapa en SPA genom att definiera en enda komponent som hanterar allt innehåll på sidan. Med hjälp av Reacts hooks som useState och useEffect, kan vi hantera tillståndet för olika delar av applikationen och hämta data från molnet. Detta uppfyller krav 11.

4.2 Icke-funktionella krav

- Användandet av Google Cloud uppfyller krav 14.
- Utvecklandet av back-enden i Flask skriven i Python uppfyller krav 17.
- Användandet av Flask som kan använda BigQuery API uppfyller krav 18.
- Användandet av React för att skapa en webbapplikation uppfyller krav 19.

5 TILLTÄNKTA ANVÄNDARE

Användarna av systemet är Neue och de ölbolagen som väljer att köpa sensortjänsten.

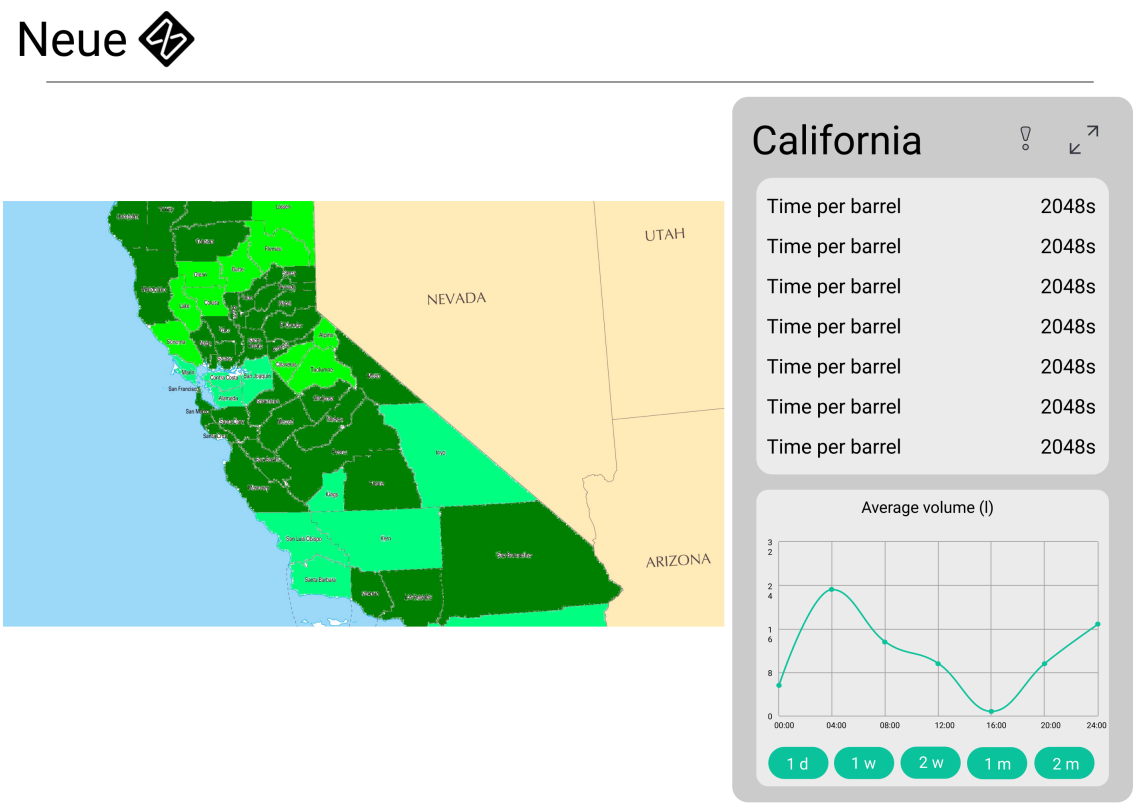
Neue är avsedda att använda molntjänsten. Där ansvarar Neue för att data strömmar felfritt från sensor till front-end, samt säker datalagring.

Ölbolaget använder sensorn samt front-enden. De monterar en sensor och får via webbapplikationen eller mejl information från fat.

6 BEGRÄNSNINGAR

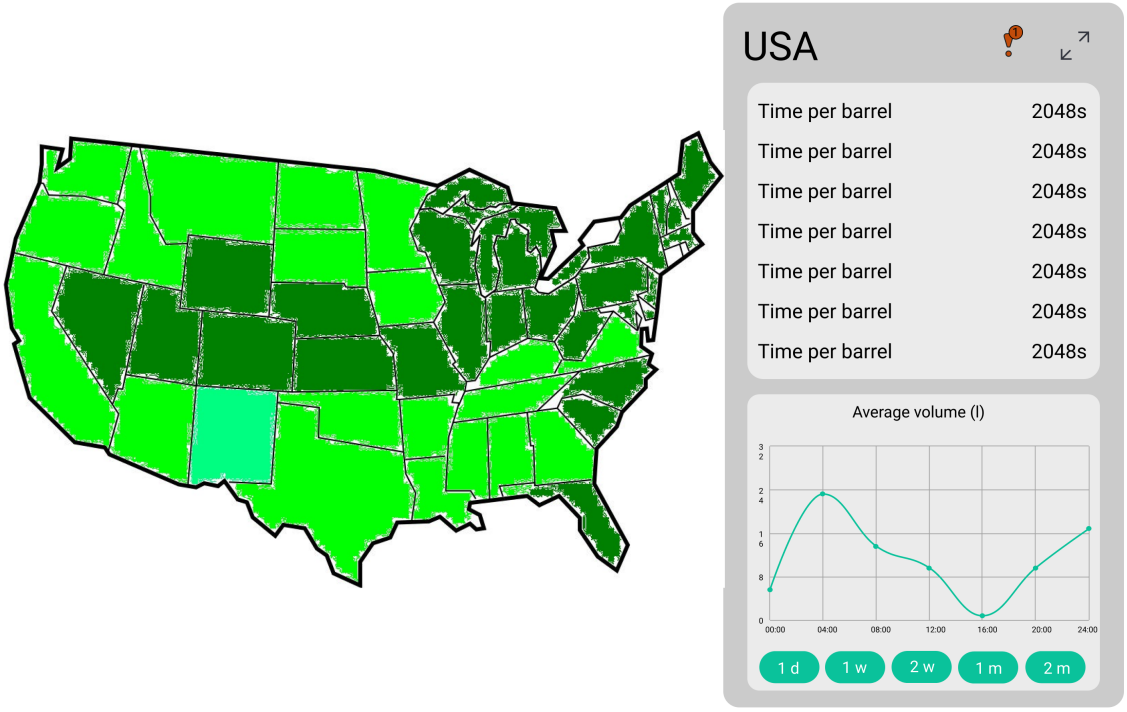
Att bli tilldelade en specifik molntjänst kan vara en begränsning. Alternativa molntjänster som AWS och Microsoft Azure skulle möjligtvis ha fördelar över Google Cloud för gruppens specifika projekt.

7 BILAGOR

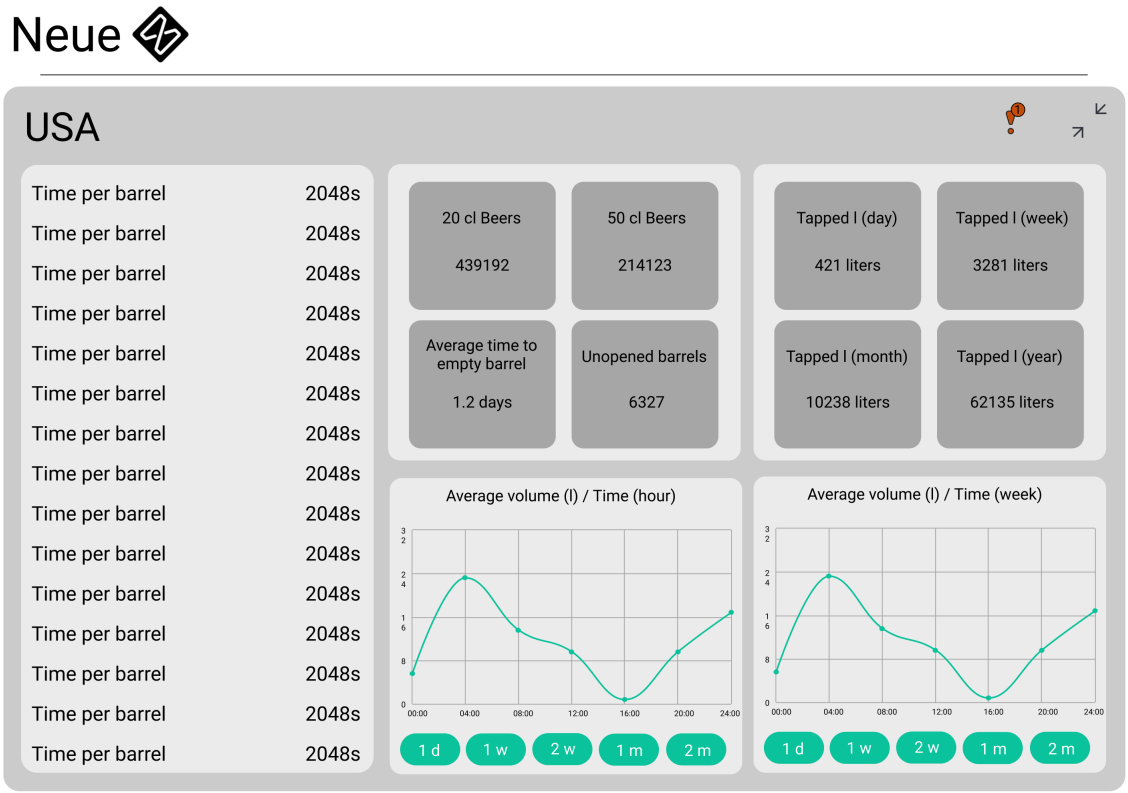


Figur 5: Front-endarkitektur

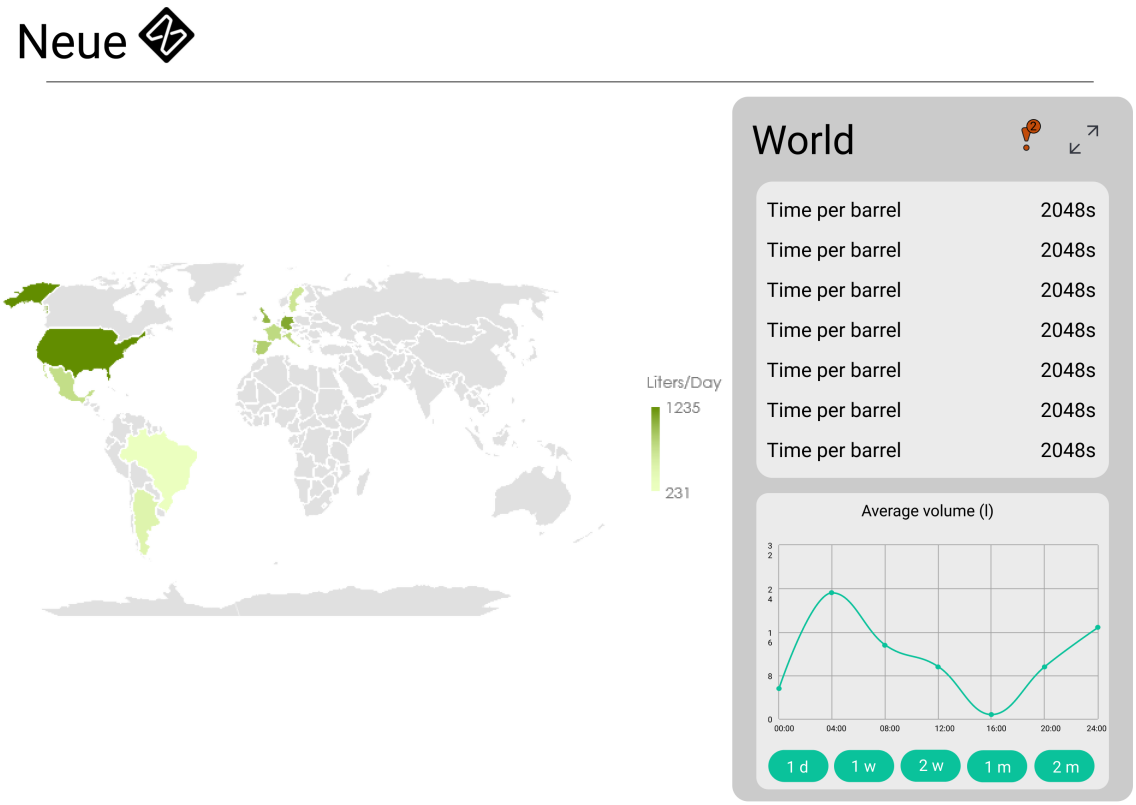
Neue



Figur 6: Front-endarkitektur



Figur 7: Front-endarkitektur



Figur 8: Front-endarkitektur

REFERENSER

[1] Sagar Batra, August Gustafsson, Erik Hellberg m. fl. ”Kravspecifikation Internet of kegs”. I: (2023).