YRKESHÖGSKOLAN
NOVIA

Utveckling av grafiska mätare i Exreport

Filip Westersund

Examensarbete för ingenjörs (YH)-examen

Utbildningsprogrammet för el- och automationsteknik

Vasa 2021

EXAMENSARBETE

Författare:	Filip Westersund
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktningsalternativ:	Informationsteknologi
Handledare:	Kaj Wikman

Titel: Utveckling av grafiska mätare i Exreport

Datum: 4.6.2021 Sidantal: 25

Abstrakt

Den globala Covid-19-pandemin orsakade nya problem inom sjukvården i Finland. Det uppstod ett behov att samla in nya sorters data, till exempel smittade personer, exponerade personer och Covid-19-relaterade dödsfall. För att kunna hantera och undersöka data som samlats in behövdes också ett sätt att rapportera informationen.

Företaget Neotide Ab hade redan skapat en grund för att visualisera den ovan nämnda typen av information i rapporteringsprogrammet Exreport, med hjälp av data från sjukhusinfektionsregistret SAI. Eftersom det fanns oanvänd information, fanns ett behov av att vidareutveckla rapporteringsmöjligheterna. Detta gällde främst den grafiska delen av rapporteringen. Syftet med det här examensarbetet var att utveckla Covid-19-mätare för den grafiska delen av Exreport.

Arbetet förklarar vilka tekniker som används för lösningen. Det nämns hur Covid-19-data hämtas, hanteras och används samt hurudana diagramtyper som är lämpliga för data som skall presenteras. Vidare berättas också hur de grafiska mätarna utvecklas antingen från grunden eller genom att tillämpa existerande kodbibliotek. Resultatet blev tre nya mätare, som presenterar relevanta Covid-19-data för användaren.

Språk: svenska Nyckelord: Covid-19, Exreport, Python, mätare, diagram

---

OPINNÄYTETYÖ

Tekijä:	Filip Westersund
Koulutus ja paikkakunta:	Sähkö- ja automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto:	Tietotekniikka
Ohjaaja:	Kaj Wikman

Nimike: Graafisten mittareiden kehittäminen Exreportissa

Päivämäärä: 4.6.2021 Sivumäärä: 25

Tiivistelmä

Maailmanlaajuinen Covid-19-pandemia aiheutti uusia ongelmia terveydenhuollossa Suomessa. Syntyi tarve kerätä uudentyyppistä tietoa, kuten esimerkiksi tartunnan saaneet henkilöt, altistuneet henkilöt ja Covid-19:n aiheuttamat kuolemat. Kaikkien kerättyjen uusien tietojen hallitsemiseksi ja tutkimiseksi tarvitaan myös tapa raportoida tietoa.

Neotide Oy oli jo luonut perustan yllä mainittujen tietojen visualisoimiselle Exreport-raportointiohjelmassa sairaalainfektiorekisterin SAI:n tietojen avulla. Koska käyttämätöntä tietoa oli, raportointimahdollisuudet täytyi kehittää. Tämä koski lähinnä raportoinnin graafista osaa. Opinnäytetyön tarkoituksena oli kehittää Covid-19-mittareita Exreportin graafiselle osalle.

Työssä selitetään mitä tekniikoita ratkaisuun käytetään. Mainitaan, miten Covid-19-tietoja haetaan, käsitellään ja käytetään sekä minkä tyyppiset kaaviot soveltuvat esitettäville tiedoille. Työ kertoo myös miten graafisia mittareita kehitetään, joko tyhjästä tai soveltamalla olemassa olevia koodikirjastoja. Tuloksena on kolme uutta mittaria, jotka esittävät olennaisia Covid-19-tietoja käyttäjille.

Kieli: ruotsi Avainsanat: Covid-19, Exreport, Python, mittari, kaavio

---

BACHELOR'S THESIS

Author:	Filip Westersund
Degree Programme:	Electrical Engineering and Automation
Specialization:	Information Technology
Supervisor:	Kaj Wikman

Title: Development of graphical meters in Exreport

Date: 4.6.2021 Number of pages: 25

Abstract

The global Covid-19 pandemic caused new challenges for the Finnish healthcare system. There were new kinds of data to collect, for example, infected persons, exposed persons, and Covid-19 related deaths. To be able to manage and examine all the new data that was collected, a way of reporting the information was also needed.

The company Neotide Oy had already created a basis for this reporting in the reporting program Exreport, using data from the hospital infection register SAI. As there was unused information, there was a need to further develop the reporting capabilities. This mainly applied to the graphical part of the report. The purpose of this thesis was to develop Covid-19 meters for the graphical part of Exreport.

The thesis explains which techniques were used for the solution. It mentions how Covid-19 data is retrieved, handled, and used, as well as what types of diagrams are suitable for the different kinds of data. It also tells how the graphical meters are developed either from scratch or from changes made to existing code libraries. The result was three new meters, which present relevant Covid-19 data to the user.

Language: Swedish Key words: Covid-19, Exreport, Python, meter, diagram

---

Innehållsförteckning

Figurförteckning

Förteckning över kodexempel

Förkortningar och begrepp

CSS	Cascading Style Sheets, är en stilmall för layout av HTML-dokument.
Exreport	Ett webbaserat rapporteringsprogram utvecklat av Neotide Ab.
HTML	HyperText Markup Language, är ett märkspråk även kallat sidbeskrivningsspråk.
JSON	JavaScript Object Notation, är ett textbaserat dataformat.
MMKR	Moniresistenttien Mikrobien Kantajien Rekisteri, eller registret för multiresistenta mikrober, är utvecklat av Neotide Ab.
MRSA	Meticillinresistenta *Staphylococcus aureus,* är en stafylokockbakterie som är resistent mot vissa antibiotika.
SAI	Sjukhusets Antibiotika- och Infektionsuppföljningssystem, utvecklat av Neotide Ab.
SQL	Structured Query Language, är ett programmeringsspråk som kan användas för att skapa och behandla relationsdatabaser.
SSMS	SQL Server Management Studio, är en integrerad miljö för att hantera olika SQL-strukturer.
SVG	Scalable Vector Graphics, är ett vektorgrafik-format för tvådimensionella bilder som stöder animationer och interaktivitet.
URI	Uniform Resource Identifier, är en databehandlingsterm.

1 Inledning

I kapitlet presenteras uppdragsgivaren för examensarbetet, Neotide Ab, samt arbetets bakgrund och dess målsättning.

1.1 Uppdragsgivare

Uppdragsgivaren för examensarbetet är IT-företaget Neotide Ab. Neotide är grundat år 1999 och hade då sin verksamhet i Helsingfors. År 2001 flyttade företaget hela verksamheten till Vasa, där det fortfarande verkar idag. Grundarna av företaget heter Johan Kullas och Patrik Simons. Företagets kundbas är sjukhus, städer och kommuner. (Neotide Ab, u.å.b)

Neotide har skapat olika program och verktyg som till exempel Exreport, SAI, LogMonitor och Santsi för att underlätta organisering av information, till exempel inom sjukvården. Exreport är ett rapporteringsprogram som kan kombinera data från olika databaser och sedan presentera en rapport enligt användarens önskemål och behov. SAI-systemet är ett sjukhusinfektionsregister som underlättar uppföljning av till exempel infektioner och antibiotikaanvändning. LogMonitor är ett system som övervakar användningen av sekretessbelagda register samt hur informationen i dem sprids. Santsi är ett webbaserat måltids- och varubeställningssystem. Fokus i examensarbetet ligger på programmet Exreport och SAI-systemet. (Neotide Ab, u.å.c)

1.2 Bakgrund

När Covid-19-pandemin slog till i Finland under våren 2020 uppstod ett behov av att utveckla rapporteringsmöjligheter för smittfall i sjukvårdsdistrikt i Finland. Det finns 21 sjukvårdsdistrikt i Finland (Social- och Hälsovårdsministeriet, 2021). Eftersom alla dessa sjukvårdsdistrikt samlar in data på olika sätt och i olika mängder blev det svårigheter att effektivt rapportera och presentera Covid-19-information.

Företaget Neotide Ab hade redan etablerat sig inom sjukvården med sitt sjukhusinfektionsregister SAI och rapporteringsprogram Exreport. Så det blev naturligt för sjukvårdsdistrikten att vända sig till Neotide för hjälp. Därför blev Exreport, i kombination med SAI-systemets Covid-19-modul, passande för ändamålet att rapportera Covid-19-

smittfall. SAI-systemet samlar in data om Covid-19-smittfall och Exreport visualiserar informationen.

På basis av insamlade data om Covid-19 fanns, förutom en vanlig tabellrapport, också en möjlighet att visa en grafisk rapport. Denna rapport visade till exempel en karta med sjukvårdsdistriktet i fråga och distriktets antal smittade per kommun. Ett annat exempel är en stapelgraf över positivt testade de senaste 30 dagarna.

Neotide planerade att utvidga och förbättra den grafiska rapporten och det resulterade i det här examensarbetet. Denna grafiska rapport finns till för att kunder lättare skall kunna visa trender och information utan att behöva gå på djupet med mycket siffror i de vanliga rapporterna. Mätarna var inte på någon kunds begäran utan utvecklades inom företaget för att sedan kunna säljas ifall behov av dessa finns.

1.3 Målsättning

Målsättningen för examensarbetet var att utveckla och presentera tre nya grafiska mätare till den grafiska rapporten i Exreport. Alla tre mätare visar olika sorters relevant data och information om Covid-19, som sjukvårdsdistrikten har samlat in. Meningen är att mätarna skall visa följande:

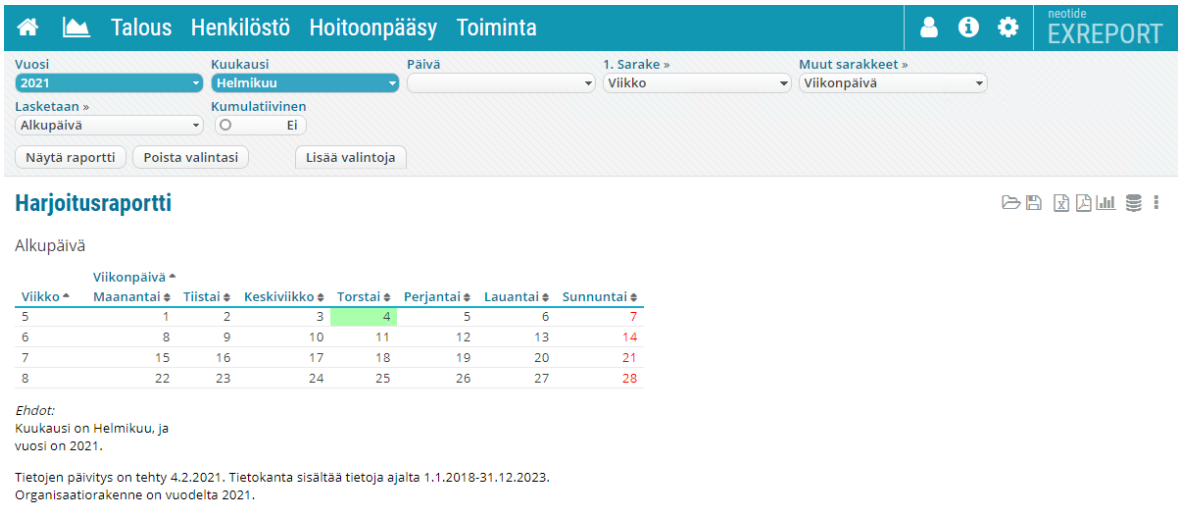
- Mätare 1: Nya smittfall, senaste 21 dagarna. Både kända och okända smittkällor skall visas och de skall vara uppdelade i eget sjukvårdsdistrikt, annat sjukvårdsdistrikt eller utomlands.
- Mätare 2: Inom valbar tidsrymd skall en listning av olika smittkedjor samt antal smittade i denna kunna visas. Genom att välja en kedja öppnas en lista över smittade som hör till kedjan.
- Mätare 3: Inom valbar tidsrymd skall det visas hur många exponerade som blivit positiva under sin karantänstid.

2 Teori och teknik

Följande kapitel presenterar och förklarar rapporteringsprogrammet Exreport, IT-systemet för sjukhusinfektioner SAI, registret för multiresistenta mikrober MMKR, programmeringsspråket Python, databashanteringsprogrammet Microsoft SQL Server Management Studio samt val av tekniker för examensarbetet.

2.1 Exreport

Neotide har skapat ett rapporteringsprogram vid namn Exreport. I programmet kan användaren skapa rapporter som baserar sig på olika sorters uppgifter som samlats in från kunden, till exempel ekonomi-, personal- eller patientuppgifter. (se Figur 1) Exreport kan öppnas genom en webbläsare. All information som finns i Exreport hämtas från programmets egen databas. Data hämtas och uppdateras från de ursprungliga källorna hos kunden med tidsbestämda intervall. (Neotide Ab, u.å.a)



Figur 1. Ett exempel på en rapport i Exreport.

2.2 SAI

IT-systemet SAI är också skapat av Neotide och bokstäverna SAI står för Sjukhusets Antibiotika- och Infektionsuppföljningssystem. Programmet är till för att kunna registrera infektionsrelaterade data. Genom att registrera användningen av antibiotika eller en infektion, kan man lättare få tillgång till aktuell information och därmed också få rätt bild av infektionsläget och antibiotikaanvändningen. Med hjälp av SAI-systemet kan sjukskötare upptäcka mönster i smittspridning och ta till åtgärder enligt det. (Neotide Ab, u.å.d)

2.3 MMKR

Moniresistenttien Mikrobien Kantajien Rekisteri, MMKR, är ett register som kan integreras med SAI-systemet. En svensk översättning för MMKR skulle grovt bli, Register för bärare av multiresistenta mikrober. Dock används den finska förkortningen som namn för programmet. Registret är till för att göra uppföljning och registrering av patienter som bär på multiresistenta mikrober. MMKR innehåller många olika bakterier och virus. Ett exempel är MRSA, som är en stafylokockbakterie och kan orsaka finnar, bölder eller andra hudinfektioner (Institutet för hälsa och välfärd, 2021). I och med pandemin har registret också utvidgats med Covid-19-viruset för att bättre kunna följa upp smittsituationer. (Neotide Ab, 2020)

Detta MMKR-register är ett webbaserat program. Eftersom det är webbaserat kan programmet användas sida vid sida med andra webbaserade program på samma server, som till exempel Exreport. MMKR-registret kan också utnyttja kopplingarna som finns i SAI-systemet genom att läsa in information om en smittbärare från andra system. På samma sätt är det också möjligt att föra över information till andra system genom detta öppna gränssnitt i MMKR. (Neotide Ab, 2020)

2.4 Python

Python är ett programmeringsspråk som anses vara lätt att lära sig. Språket är objektorienterat och idealiskt för att göra skript och applikationsutveckling. Programmeringsspråket är skapat av Guido van Rossum och namnet Python härstammar från humorserien ”Monty Python’s Flying Circus”. Python är öppen källkod, vilket betyder att det är helt gratis att ladda ner Python och att använda det i sina egna applikationer. Dessutom är det möjligt att köra Python både på Windows och Unix-liknande varianter, såsom Linux eller MacOS. (Python Software Foundation, 2021)

Ifall man skriver mycket kod kan det löna sig att spjälka upp koden i så kallade moduler för att organisera och strukturera sitt program. I en modul kan det finnas definitioner, klasser och variabler (se Figur 2). Modulen kan sparas med ett namn och därefter importeras, varpå det går att återanvända modulens innehåll (se Figur 3).

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Figur 2. En modul som kan räkna och skriva ut Fibonacci serien (Python Software Foundation, 2021).

```
>>> import fibo
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Figur 3. Exempel när modul "fibo" importeras och används (Python Software Foundation, 2021).

2.5 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio, SSMS, är en integrerad miljö för att hantera olika SQL-strukturer (Microsoft, 2021). Structured Query Language, SQL, är ett programmeringsspråk som kan användas för att skapa och behandla relationsdatabaser. Genom att använda enkla kommandon, som till exempel: CREATE och INSERT, så kan man skapa en tabell i databasen och lägga in data i tabellen. Vidare finns också kommandona UPDATE och DELETE, vilka uppdaterar eller tar bort data. På liknande sätt kan man välja bestämda data ur en tabell och visa dem som en resultattabell genom kommandot SELECT (se Figur 4). (Stephens, 2008)

```
SELECT FirstName, LastName, Clue,
       Street, City, State, Zip
FROM People
WHERE NOT Clue IS NULL
ORDER BY Clue, LastName, FirstName
```

Figur 4. Exempel på ett SELECT-kommando (Stephens, 2008).

Genom SSMS kan man konfigurera, upprätthålla och utveckla allting som ingår i en SQL-server (Microsoft, 2021). Ett viktigt verktyg i SSMS är ”Object Explorer” eller objektutforskaren på svenska. Med hjälp av objektutforskarens hierarkiska struktur är det enkelt att hitta och hantera olika sorters objekt i varje SQL-server som är i användning. (Microsoft, 2021)

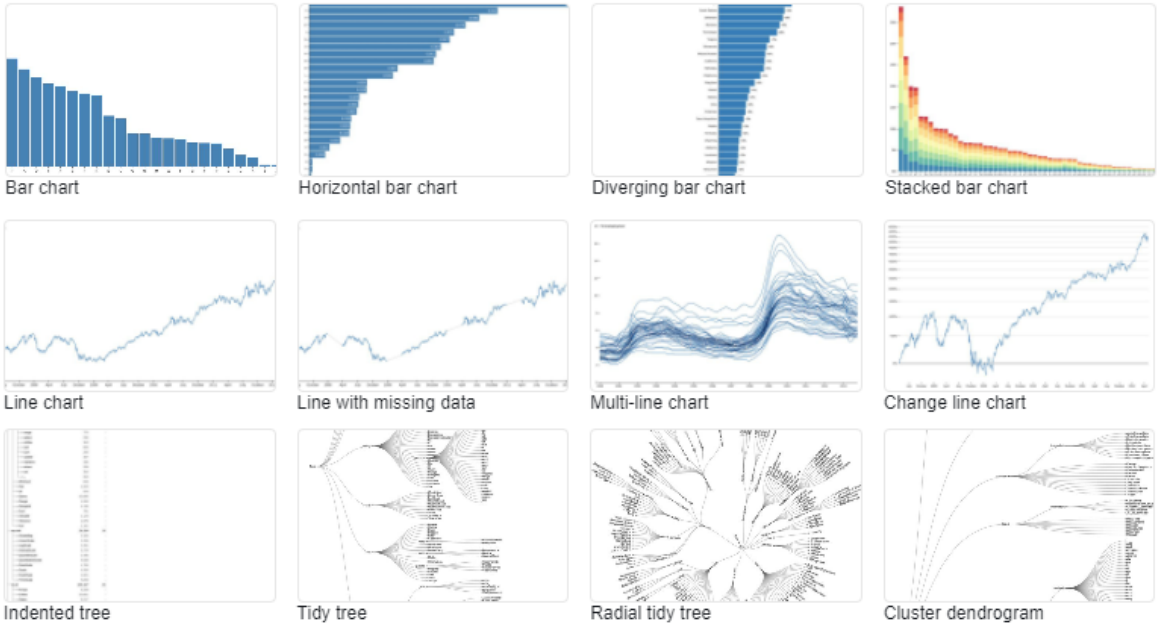
2.6 Val av teknologier

I detta avsnitt berättas valet av grafbibliotek, som framställer graferna samt valet av hanteringssystem för relationsdatabaser för arbetet.

2.6.1 Grafbibliotek

Det finns många olika JavaScript-bibliotek som hjälper till att skapa ett diagram eller en graf. I Neotides kodbibliotek fanns det från tidigare JavaScript-filer som använder sig av D3.js. Därför föll valet naturligt på att använda D3.js eftersom det både fanns exempel från tidigare filer i företaget och att arbetskamraterna hade erfarenhet av D3.js-biblioteket.

D3.js är ett öppet källkodsbibliotek i JavaScript. Namnet D3 står för Data-Driven Documents och det har utvecklats sedan år 2011 av Mike Bostock. D3.js biblioteket handlar om datamanipulering och -visualisering. Kombinerat med HTML, SVG och CSS är D3.js ett kraftfullt hjälpmedel för att kunna skapa till exempel diagram, grafer eller andra sorters visualiseringar av data (se Figur 5). (Teller, 2013) Det som är bra med D3 är att det är snabbt och stöder stora datamängder. Det stöder också dynamiska interaktioner och animeringar. På samma gång möjliggör D3 återanvändning av kod genom mångsidiga samlingar av moduler. (Bostock, 2021)



Figur 5. Exempel på olika sorters D3.js grafer. (Observable, Inc. 2021)

Det finns också andra varianter av grafbibliotek. Någon av dem skulle också kunna användas till graffframställningen, om inte D3.js blivit valt. Några exempel är Highcharts, Chart.js och Charist.js.

2.6.2 Hanteringssystem för relationsdatabaser

Alla databaser som hanteras i projektet är relationsdatabaser. Med relationsdatabas menas, att strukturen på databasen tillåter att identifiera och komma åt data genom relationer med annan data som finns i databasen. Ofta är relationsdatabaser uppdelade i olika tabeller. (Codecademy, 2021)

För att kunna hantera relationsdatabaser finns det olika hanteringssystem att välja mellan. Eftersom Neotides kunder redan använde sig av Microsoft SQL Server så fanns det inga valmöjligheter i arbetet. Alternativa hanteringssystem skulle kunna vara MySQL, PostgreSQL eller Oracle DB. Av dessa är MySQL och PostgreSQL speciellt populära eftersom de är öppen källkod och lättanvända (Codecademy, 2021).

3 Utförande

I kapitlet presenteras planeringen av arbetet samt hur arbetet framskred.

3.1 Planering

Inledande tanken kom från ett möte den 28:e september 2020, varpå idéer för nya mätare till grafiska rapporten diskuterades. Vid mötet diskuterades mätarnas innehåll och hur de skulle visas. Från detta kom själva grundidén till mätarna och hur de möjligtvis skulle kunna se ut. En månad senare hölls ett till möte där diskussion hölls om hur utvecklingen av mätarna kunde bli ett examensarbete. Anteckningarna från detta möte kom också att bli upplägget för målsättningen i arbetet.

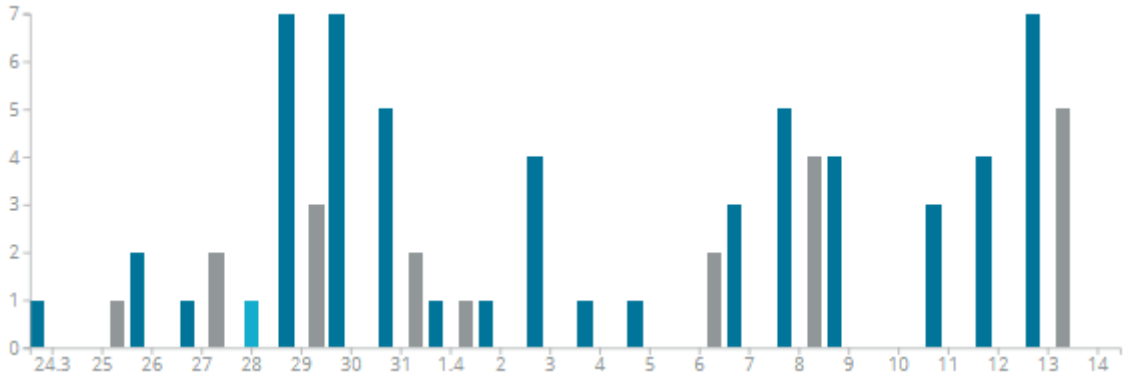
Mätarnas uppgift är främst att förse kunder med relevanta data gällande Covid-19-smittsituationen. På samma gång behövde mätarna vara flexibla eftersom det inte fanns några specifika krav på hur mätarna skulle se ut. Hela utvecklingen är en idé från företagets sida. Detta ger en möjlighet att kunna erbjuda kunderna sätt att presentera olika sorters information om Covid-19, som samlats in i deras sjukvårdsdistrikt.

Koden för mätarna skapas inte helt från grunden, utan kodbibliotek kan användas från tidigare grafer och mätare för att på detta vis kunna framställa de nya mätarna i enlighet med övriga grafers struktur och tema. Vidare existerar det skillnader mellan kunders data och hur denna information benämns eller hänvisas till, vilket ger en svårighet i utvecklingen. Det vill säga, liknande data kan benämnas olika eller ha olika egenskaper och därför måste man ta viss hänsyn till nämnda aspekter vid hämtningen från databaser och vid behandlingen i koden. Koden utvecklas så allmänt som möjligt för att sedan kunna specificeras och parametriseras enligt behov.

3.2 Strukturen på mätarna

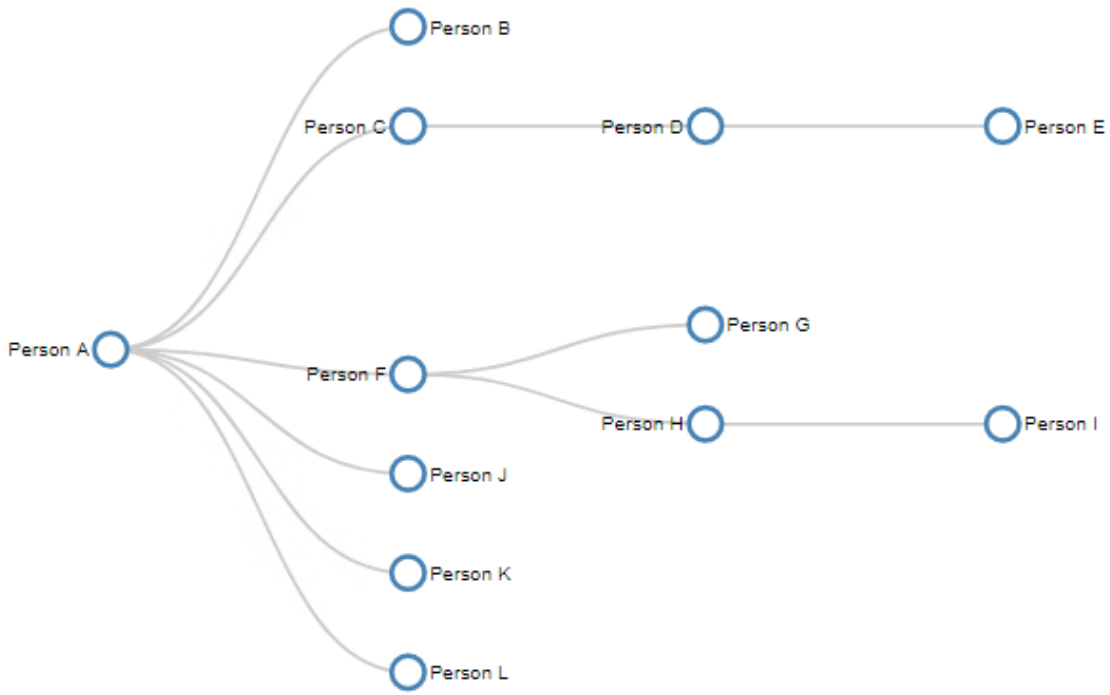
Alla tre mätare behöver ha en enhetlig struktur för att passa in i den grafiska rapporten. Mätarna kan inte ta upp för mycket plats men de skall ändå inte förvrängas eller göras oläsliga. På samma gång har alla mätare unika värden och därför kan själva graftypen variera för varje mätare.

Den första mätaren skall visa nya smittkällor under de senaste 21 dagarna. Intervallet 21 dagar valdes efter diskussioner under de tidigare nämnda mötena. Mätaren skall också visas i tre olika kategorier för nya smittor: smittor i eget distrikt, smittor i annat distrikt och smittor utomlands. Baserat på detta passar ett stapeldiagram bra. På y-axeln sätts antalet smittade och på x-axeln läggs datum. X-axelns datum uppdateras kontinuerligt för att alltid visa de senaste 21 dagarna. Därefter kan de tre olika kategorierna presenteras med tre olikfärgade staplar på varje datum (se Figur 6). Varje kategori kan väljas från deras etiketter, som syns under grafen. På så sätt kan man presentera en, två eller tre kategorier på samma gång. Det finns också en info-knapp i det högra hörnet för att ytterligare förklara för användaren vad som visas i grafen och varifrån värdena kommer.



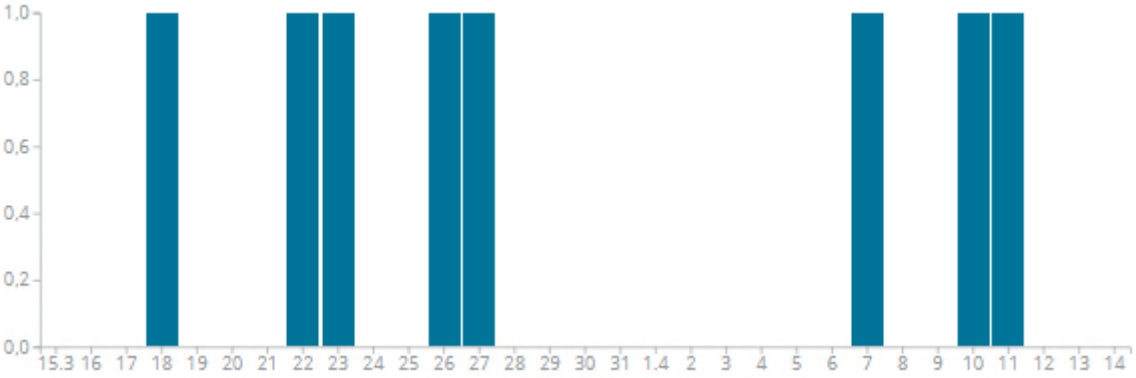
Figur 6. Exempel på ett stapeldiagram med tre kategorier.

Däremot kan inte den andra mätaren skapas med samma struktur som den första. Eftersom den andra mätaren skall visa en listning över smittkedjor så kan ett stapeldiagram inte anses passande. En smittkedja kan vara lång eller kort och olika mängder personer är involverade. Med tanke på detta kan ett träddiagram vara lämpligt (se Figur 7). Träddiagrammet behöver visa smittkällan och hur många personer som är exponerade av denna. Det behövs också ett alternativ att reglera tidsintervallet. Slutligen skall möjligheten finnas att kunna välja en kedja för att lista hela kedjans innehåll.



Figur 7. Exempel på ett träddiagram.

Vidare kan den tredje mätaren liknas med den första. Den tredje mätaren skall visa sekundära smittkällor, alltså personer som har blivit exponerade och under sin karantänstid därefter testat positivt. Ett stapeldiagram kan användas igen eftersom den tredje mätaren presenterar antal smittade under en viss tidsperiod, på samma sätt som den första mätaren (se Figur 8). Datum sätts på x-axeln och antal på y-axeln. Däremot är tidsperioden inte rullande, utan den är fastställd och en valmöjlighet finns i en rullgardinsmeny under grafen.



Figur 8. Exempel på ett stapeldiagram.

3.3 Covid-19-data

All Covid-19-information som används i Exreport kommer först från MMKR-registret. I MMKR fyller kunder i ett särskilt Covid-19-formulär och från formuläret körs data in i en databas. Denna databas är integrerad med SAI-systemet. När Exreport kopplas ihop med databasen är det möjligt att hantera och presentera informationen i rapporter och grafiska rapporter. Databasen är uppdelad i olika tabeller och vidare presenteras de relevanta Covid-19-tabellerna. En tabell, ”Carrierregistry”, om positivt testade personer och två tabeller, ”Exposureregistry” och ”Exposureregistryrow”, om exponerade personer. Alla tabeller kommer användas för att få fram och kunna presentera data på rätt sätt. Tabellerna innehåller känslig information, vilket innebär att de inte får visas i examensarbetet men de bör ändå nämnas eftersom den viktigaste informationen härstammar därifrån. Personnummer för identifiering av en patient, datum för upptäckten av ett fall, vilka personer som har exponerats för viruset av en positivt testad person och om en exponerad person har testat positivt, för att nämna några viktiga.

3.3.1 Positiva

Tabellen för smittade personer, däribland Covid-19-personer, kallas för ”Carrierregistry”. Den innehåller många olika kolumner med data, som kommer att användas när alla mätare skall skapas (se Figur 9). Till exempel finns en datumkolumn om när en person blivit registrerad som positiv, en kolumn som berättar vad personen har registrerat för symptom och mycket annat. Kolumnerna som behövs mest i utvecklingen för mätarna är:

- Datumkolumn för när en person blivit registrerad positivt smittad.
- Kolumnen med personbeteckningar. Användningen är endast för att kunna räkna unika personbeteckningar för att få ett antal, som sedan kan användas i grafen.
- Kolumnen för smitta. Denna kolumn innehåller koder för olika smittor och därför sökes koden för Covid-19-viruset och olika mutationer av det.
- Kolumnen med koden för negativt eller positivt resultat. Visas med en sifferkod.

Notera att kolumnerna inte anges med deras namn eller exakt vad de skall innehålla, eftersom dessa kan variera från server till server. Naturligtvis kan också andra kolumner vara relevanta beroende på behovet av information. Dessa kolumner utgör en grund, varpå det senare kan tilläggas begränsningar för att hantera data som behövs till mätarna.

```
SELECT TOP 1000 [rowid]
      ,[archive_timestamp]
      ,[socsecno]
      ,[microbe]
      ,[patname]
      ,[patsex]
      ,[state]
      ,[origin]
      ,[epidemystrain]
      ,[exposuredistrict]
      ,[foundfacility]
      ,[contcareorg]
      ,[patnotified]
      ,[careunitnotified]
      ,[patdead]
      ,[arrivaldate]
      ,[kunta]
```

Figur 9. Exempel på några kolumner i ”Carrierregistry”.

3.3.2 Exponerade

Data över exponerade personer är samlat i två tabeller. Dessa tabeller kallas för ”Exposureregistry” och ”Exposureregistryrow”. De berättar information om hur många personer som blivit exponerade, vilka personerna är och om de är satta i karantän, när de har blivit exponerade, om de sedan testat positivt i karantän och så vidare.

”Exposureregistry” består av kolumner som berättar om personer som har blivit positiva och på samma gång exponerat andra personer. En jämförelse med personbeteckningen från den positiva tabellen (se kap. 3.3.1) till denna tabell visar att den smittade har exponerat x-antal personer. I tabellen syns också när exponeringen har skett (se Figur 10).

```
SELECT TOP 1000 [rowid]
      ,[exposedby_socsecno]
      ,[exposedby_finddate]
      ,[exposedby_exposure]
      ,[exposedby_microbe]
      ,[positive_socsecno]
      ,[positive_finddate]
      ,[positive_exposure]
      ,[positive_microbe]
      ,[original_rowid]
      ,[deleted]
  FROM [sai].[sai].[exposureregistry]
```

Figur 10. Exempel på kolumner i ”Exposureregistry”.

Vidare kan man i ”Exposureregistryrow” få fram mera exakt information om vem den exponerade personer är, om den blivit kontaktad och försatt i karantän, vilket datum som karantän utfärdats och så vidare (se Figur 11). Till exempel en kolumn som kommer att användas för den andra mätaren kommer från denna tabell. Kolumnen berättar om en person som blivit exponerad och försatt i karantän blivit positiv, d.v.s. testat positivt under sin karantänstid.

```
SELECT TOP 1000 [rowid]
      ,[exposureregistry_rowid]
      ,[timestamp]
      ,[socsecno]
      ,[firstname]
      ,[lastname]
      ,[address]
      ,[telephone]
      ,[caretaker_contact]
      ,[email]
      ,[first_contact_date]
      ,[first_contact_person]
      ,[last_contact_date]
      ,[exposed_date]
      ,[exposed_place]
      ,[sample_taken_date]
      ,[sample_result]
      ,[symptoms]
      ,[symptom_description]
      ,[sick_during_quarantine]
```

Figur 11. Exempel på några kolumner som finns i ”Exposureregistryrow”.

3.4 Mätare 1

Det fanns från tidigare kodbibliotek i användning som skapade grafiska mätare av olika slag. Allt från linjegrafer och stapeldiagram till pajdiagram och trafikljusmätare. Allting studerades och utvärderades för att i någon mån kunna utnyttjas igen. Tanken var att kunna skapa en ny klass för varje mätare som sedan ärver från tidigare skapta klasser och på så sätt kunna spara kodrader och optimera utvecklingen.

Efter att det mesta som relaterade till grafiska mätare hade undersökts konstaterades att den första mätaren, som visar positiva inom vissa områden, behöver en ny tidsaxel. Därmed skapades en ny klass som ärvde av en tidigare mätare, som visade värden med ett rullande 30 dagars intervall. Förändringen var inte stor utan 30 dagar blev i stället 21 dagar, som tidigare nämnts (se Kodexempel 1). Ytterligare behövs ett tillägg för att kunna ha flera kategorier i denna första mätare som nämndes tidigare (se kap. 3.2). För att visa flera staplar

över samma datum behövde mätaren ha ett attribut som säger att staplarna inte får läggas ovanpå varandra. Detta uppnåddes genom att från den ärvda klassen definiera en egenskap för layout som sade att staplarna skall vara i en grupp, i stället för ovanpå varandra.

Kodexempel 1. Exempel på funktion för att få ett 21-dagars intervall.

```
@staticmethod
def _get_start_date():
    timedelta = {}
    date = datetime.datetime.today() - datetime.timedelta(days=21)
    timedelta['days'] = 1
    return date.replace(hour=0, minute=0, second=0, microsecond=0), timedelta
```

Vidare var hämtningen av data, alltså SQL-satserna för varje stapelkategori, den största utmaningen för den första mätaren. Eftersom det i mätaren behöver visas tre olika staplar på samma gång över ett värde på x-axeln så krävdes tre SQL-satser också. Med hjälp av SSMS kunde SQL-satser konstrueras och samtidigt prövas för att se att de gav rätt resultat. På detta sätt kunde nödvändiga begränsningar utarbetas och prövas. Det gemensamma med alla SQL-satser var att allihop gör en SELECT för att visa datum för positivt testad samt hur många unika personbeteckningar, som uppfyller kraven i WHERE-delen. Därtill finns också en kontroll som ser till att det inte förekommer dubbletter i resultatet. Slutligen skulle en gruppering göras över datumkolumnen, för att få en resultatrad med positiva per datum.

Skillnaderna som uppstod i SQL-satserna var kontrolleringen av distrikt. Den första satsen skall ge ett resultat med positiva i det egna distriktet. För att kunna få fram rätt resultat behövs en JOIN med en tabell, ”DistrictKunta”, som visar olika distrikt samt kommuner i dem. Förhållandet mellan tabellerna finns i kolumnerna som innehåller kommunkoder. Därefter kan en begränsning göras på distriktet, som finns definierat i en variabel i koden vid namn ”district_name”. Variabelns värde fås in i SQL-satsen genom att genomföra en strängformatering, alltså att berätta för Python koden var den skall sätta in en variabels värde som en textsträng (se Kodexempel 2). Nästa SQL-sats skall ge alla positiva som inte finns i serverns distrikt, det vill säga alla andra distrikt utom det som definierats i ”district_name”. Den sista satsen skall ge ett resultat som visar positiva utomlands. Kontrolleringen kan inte ske på distriktet, utan i stället finns en kontroll av kommunkoder och där en kod står för utomlands.

Kodexempel 2. SQL-sats för positiva i ett distrikt.

```
SELECT finddate, COUNT(DISTINCT socsecno) FROM carrierregistry
 JOIN districtkunta
  ON districtkunta.kuntanro = carrierregistry.kunta
  AND districtkunta.district = %(district_name)s
 WHERE NOT EXISTS (
   SELECT 1 FROM carrierregistry c2
   WHERE c2.socsecno = carrierregistry.socsecno
   AND c2.exposure = carrierregistry.exposure
   AND c2.finddate > carrierregistry.finddate
   AND c2.state != carrierregistry.state
  )
  AND carrierregistry.finddate >= %(start_date)s
  AND carrierregistry.deleted IS NULL
  AND carrierregistry.exposure = %(exposure)s
  AND carrierregistry.state = '2'
 GROUP BY finddate;
```

Efter en start med enbart SQL-satser kunde det hela förenklas genom optimering av koden. Genom att göra en koppling mellan den vanliga rapporten och den grafiska kunde all information som behövdes för att visa stapeldiagrammet nås. Denna koppling uppnås genom att i den grafiska rapporten definiera den vanliga rapportens resurs. Med resurs menas rapportens URI, som står för Uniform Resource Identifier. I URI:n plockas parametrar som används i kopplingen, till exempel att det är positivt smittade som räknas och att resultatet skall sorteras efter datum (se Figur 12). Med denna förenkling av kod uppnår vi samma resultat som med fullständiga SQL-satser.

```
En rapport URI-link.
http://localhost:12900/mmkr_exposed/render?ordering=cr_finddate&orderdesc=cr_finddate&rows=cr_finddate&_update_values_a

-Får ta bort:
render?ordering=cr_finddate
&orderdesc=cr_finddate
&_update_values_and_form=%3Frows%3Derr_exposed_place%26update_values_and_form%3Drows&
&show_widget_row3=1


-Vad som ska bli kvar och dessa används i data_url:
http://localhost:12900/mmkr_exposed/rows=cr_finddate&reporting=cr_count&counting=count&district=EPSHP

rows=cr_finddate
reporting=cr_count
counting=count
district=EPSHP

'data_url': Gauge.link('mmkr_exposed', columns='year', month='', counting='count', reporting='cr_count',
                       reporting1=None, rows='cr_finddate', district=batch_config.district_name.upper())
```

Figur 12. Exempel på hur en ”data_url” skapas från en rapports URI.

Slutligen görs också en koppling den andra vägen, alltså från den grafiska rapporten till den vanliga (se Kodexempel 3). Genom att välja en stapel i diagrammet så laddas den vanliga rapporten och man kan se hur många som var positivt smittade under valt datum. Vidare kan man också borra sig ner i denna siffra men det är inte relevant för mätarens uppgift. Det hör till den vanliga rapportens funktioner.

Kodexempel 3. Exempel på funktion för en koppling till vanliga rapporten.

```
def get_urls(self, measure):
        today = datetime.datetime.today()
        date, timedelta = self._get_start_date()
        d = []
        while date <= today:
            d.append(self.link('mmkr_exposed', organisation_hierarchy=None, ealatop_hierarchy=None, year=None,
                               month=None, reporting='cr_count', reporting1=None, rows='cr_finddate',
                               district=batch_config.district_name.upper(), cr_finddate_interval_start=date.date(),
                               cr_finddate_interval_end=date.date()))
            date += datetime.timedelta(**timedelta)
        return d
```

3.5 Mätare 2

Den andra mätaren visade sig vara en stor utmaning eftersom det för det första inte fanns kodklasser som kunde ärvas för mätaren. För det andra så behövde all information hämtas och kontrolleras på ett speciellt sätt för att kunna konstruera data med smittkedjor som också kan presenteras i ett träddiagram. Därför kommer mätaren att skapas i två etapper. Först skapas en modul som hämtar och samlar data. Sedan skapas trädstrukturen och hur informationen visas i den.

Till en början behöver vi definiera hur en smittkedja byggs upp. Efter en hel del brainstorming konstaterades följande:

- En kedja kan börja med en positivt testad person, som inte har exponerats av en annan person inom de senaste 14 dagarna från datumet för sitt test. Ifall en exponering skett inom de 14 senaste dagarna, från datumet för testet, tillhör personen redan en kedja och kan inte vara en början.
- Därefter räknas personer som blivit exponerade av den positiva personen.
- Om exponerade personer ger ett positivt resultat, inom 14 dagar, så är de nästa del i trädstrukturen.
- Då granskas ifall den nya positiva i sin tur exponerat personer.
- Slutligen kontrolleras samma saker om och om igen tills en positiv inte exponerat någon eller om de exponerade inte utvecklar ett positivt test.

Dessa punkter fick stå som riktlinjer för hur mätarens information byggs upp. Notera att detta inte är en enda kedja som börjar från patient noll utan en början måste fås någonstans ifrån och den skall vara relevant för användaren. Därmed kan det också finnas flera olika kedjestarter.

Modulen skapas från grunden och den hämtar data som behövs för att senare kunna presentera ett träddiagram. Uppbyggnaden av modulen består av olika funktioner och baserar sig på kraven som ställdes tidigare. Till exempel behövs en funktion för att hämta alla positiva, en funktion för alla som exponerats och en funktion som kontrollerar och sätter ihop smittkedjor.

De positiva och exponerade kolumnerna kan hämtas genom SQL-satser. Från positiva tabellen, ”Carrierregistry”, hämtas kolumnerna för personsignum, datum för när den positiva registrerats, datum när personen tagit sitt Covid-19-test samt två kolumner till som indikerar att personen har smittats av Covid-19. Från exponerade tabellerna, ”Exposureregistry” och ”Exposureregistryrow”, hämtas kolumnerna för personsignum för personen som exponerats, personsignum på den person som har exponerat andra, datumkolumner som indikerar att personen sedan testat positivt samt kolumnerna som indikerar att personen exponerats av Covid-19. Dessa kolumner utgör grunden när smittkedjorna skall kontrolleras och konstrueras.

Vidare bildas kedjorna först med en kontroll för positiva som kan börja en kedja. Som tidigare nämnts är det en positivt testad person som inte blivit exponerad inom de 14 senaste dagarna från datumet för sin provtagning. Personen ska inte heller finnas i tabellerna ”Exposureregistry” eller ”Exposureregistryrow”. Dessa sätts in i en ”dictionary”, som är en mappning mellan en nyckel och dess värde. Nyckeln innehåller de tidigare nämnda positiva kolumnerna och värdena blir två tomma listor som kallas ”exposed” och ”positive” (se Kodexempel 4). I den tomma ”exposed” listan skall de exponerade komma och i ”positive” skall det komma personer som exponerats och därefter blivit positiva.

Kodexempel 4. Exempel på hur början på en smittkedja skapas.

```
def get_tree(self, all_positive, all_exposed):
    result = []
    for positive in all_positive:
        if positive['base_node']:
            result.append({'key': (positive['socsecno'], positive['finddate'], positive['exposure'],
                                   positive['microbe']), 'exposed': [], 'positive': []})
    for case in result:
        self.get_sub_cases(case, all_exposed, all_positive)
    return result
```

Kontrolleringen för att hitta personer som blivit exponerade sker genom en jämförelse av personsignum från ”Carrierregistry” och ”Exposureregistry”. Ifall det positivt testade signumet återfinns i ”Exposureregistry” så finns det åtminstone en exponerad person. Dessa läggs in i listan ”exposed”. Sedan kontrolleras ifall exponerade blivit positiva genom att se ifall det givits ett positivt resultat inom 14 dagar efter att exponeringen skett. Dessa positiva sätts in i listan ”positive”. För att skapa en helhet av allting så körs allt som hämtats igenom och skapas som en hierarkisk struktur (se Kodexempel 5). Denna struktur består av ”name”, som då är ett personsignum och en lista med ”children”. Listan kan fyllas med flera personsignum och deras tillhörande listor om den tidigare personen smittat andra. Slutligen kommer denna hierarkiska struktur skapas som en JSON-sträng för att det var lättast att hantera i skapandet av trädet som följer.

Kodexempel 5. Exempel på hur smittkedjan gås igenom och görs till JSON-sträng.

```
def get_json(self, node):
    result = {
        'name': node['key'][0],
    }
    children = []
    for child_node in node['positive']:
        children.append(self.get_json(child_node))
    result['children'] = children
    return result
```

För att sedan kunna visa själva trädet med smittkedjor i den grafiska rapporten så behövs ett par nya klasser och en JavaScript-fil. Det behövs en grund för själva diagrammet som kommer att visas eftersom det inte finns något tidigare kodbibliotek som ärvs. Grundklassen definierar grafen och hur den skall framställas (se Kodexempel 6). Denna grundklass använder sig av en JavaScript-fil. JavaScript-filen innehåller funktioner för skapandet av

trädet, hur information appliceras på alla noder i trädet samt hur dessa noder beter sig. Sedan skapas en till ny klass i den grafiska rapporten (se Kodexempel 7). Denna klass ärver av grundklassen för att kunna visa ett träd. Klassens data fås fram med hjälp av den tidigare nämnda modulen. Därefter är trädet användbart och man kan granska kedjor genom att välja noder så att de öppnar eller stänger sig beroende på önskad vy.

Kodexempel 6. Grundklassen "CollapsibleTreeGauge" och en del av dess uppbyggnad.

```
class CollapsibleTreeGauge(BaseGauge):
    external_js['javascript/dashboard/collapsibletree.js']

    def graph(self, **kwargs):
        # behandlar och skapar grafen och dess data.

    @template.html
    def render_gauge(self, **kwargs):
        # skapar mätaren med innehållande SVG för träddiagrammet.
```

Kodexempel 7. En del av klassen "Tartuntaketjut" i grafiska rapporten som ärver "CollapsibleTreeGauge".

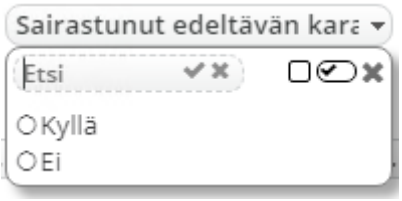
```
class Tartuntaketjut(CollapsibleTreeGauge):
    header = 'Tartuntaketjut'
    name = 'tartuntaketjut'
    measure_data = {
        name: {
            'data_url': True
        }
    }

    # Klassen fortsätter...
```

3.6 Mätare 3

På samma sätt som den första mätaren var det möjligt att ärva från en tidigare klass som ger en stadig grund att sedan bygga vidare på. Klassen som ärvdes var också den samma som för den första mätaren, alltså ett stapeldiagram med rullande 30 dagars intervall. Allt som behövs för denna nya klass är att definiera hur och varifrån informationen kommer. Informationen kommer från tabellen "Exposureregistryrow" där kolumnerna "sick_during_quarantine" samt "exposuredate" är de som är mest relevanta. Med hjälp av

dessa kan man se om en exponerad person blivit positiv under sin karantänstid med hänvisning till dennes exponeringsdatum.

Eftersom det finns endast en stapelkategori så var hanteringen av data inte en stor utmaning. Det fanns också redan en begränsningsmöjlighet i den vanliga rapporten för det resultat som skulle uppnås. Begränsningen skedde i den vanliga rapporten genom 'Ja/Nej' kryssrutor och när begränsningen var aktiv så visades personer som blivit sjuka under sin karantänstid (se Figur 13). Om man på samma gång sökte efter antal exponerade enligt deras exponeringsdatum så fås resultatdata som behövs för mätaren.



Figur 13. Exempel på en "Ja/Nej"-meny i Exreport.

Behovet av att skapa en SQL-sats för mätaren fanns inte i det här fallet då den vanliga rapporten redan kunde ge allt som behövs. På likadant sätt som i den första mätaren görs en koppling mellan den vanliga rapporten och den grafiska. Som tidigare nämnts, plockas parametrarna som behövs till kopplingen från vanliga rapportens URI. Den viktiga parametern som behöver vara aktiv, är den med exponerade som blivit sjuka under sin karantänstid. Avslutningsvis skall också mätaren kopplas andra vägen till den vanliga rapporten på samma sätt som den första mätaren gjordes.

Värt att notera är att servrar kan ha olika kriterier för hur en person anses ha insjuknat under sin karantän. Därför kan olika specificeringar ske i framtiden för denna mätare. Dessutom blev inte valbarheten för tidsaxeln implementerad i denna mätare, trots att det var tanken från början. Som tidigare nämnts, har mätaren ett flytande 30 dagars intervall i stället.

4 Resultat

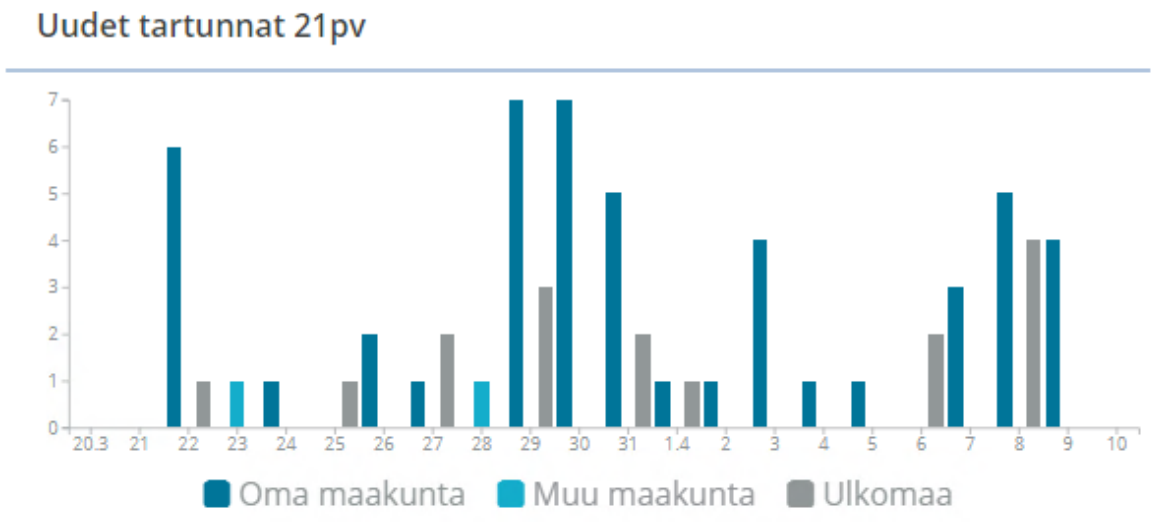
Detta kapitel innehåller resultatet av examensarbetet. Resultatet beskriver och visar slutprodukten som den ser ut på en server, i Exreports grafiska rapport.

För tillfället används inte mätarna på någon produktionsserver men de kan implementeras med en relativt enkel process ifall efterfrågan finns. På samma gång är också mätarna färdiga att visas upp i en demo-miljö och kan därmed säljas som tillägg till kunder.

4.1 Mätare 1

Resultatet för den första mätaren blev ett stapeldiagram med tre olika kategorier. Namnet för själva mätaren blev ”Uudet tartunnat 21pv” för att en användare snabbt skall förstå vad som visas (se Figur 14). Om man översätter namnet så blir det ”Nya smittor 21 dagar”. Samtidigt finns också en infoknapp på mätaren ifall mer information om varifrån data kommer eller om vad som visas på mätaren behövs.

Jämfört med målsättningen för mätaren uteblev beaktningen av kända och okända smittkällor i framställningen. I stället togs bara alla positivt testade personer och därefter gjordes områdesindelningen. I övrigt uppfyller mätaren kraven som ställts.

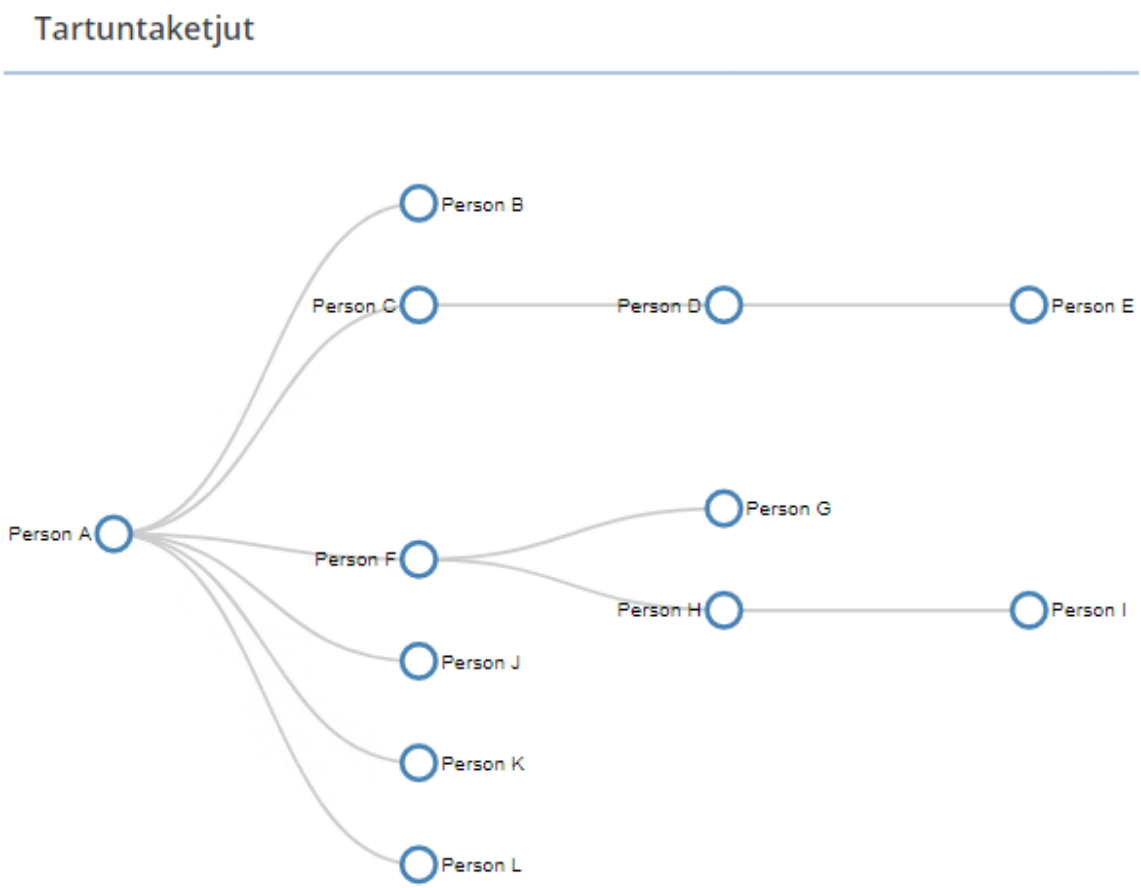


Figur 14. Exempel på mätare 1 – ”Uudet tartunnat”.

4.2 Mätare 2

Andra mätaren var egentligen uppdelad i två delar, en för att hämta informationen och en för att presentera den. Eftersom fokus ligger på det grafiska och hur mätaren visas så kommer inte modulen för datahämtningen (se kap. 3.5) att presenteras. Här presenteras bara den grafiska delen med träddiagrammet, som fick namnet ”Tartuntaketjut”. Namnet ”Tartuntaketjut” syftar på de smittkedjor som visas i diagrammet. Översatt blir namnet ”Smittkedjor”.

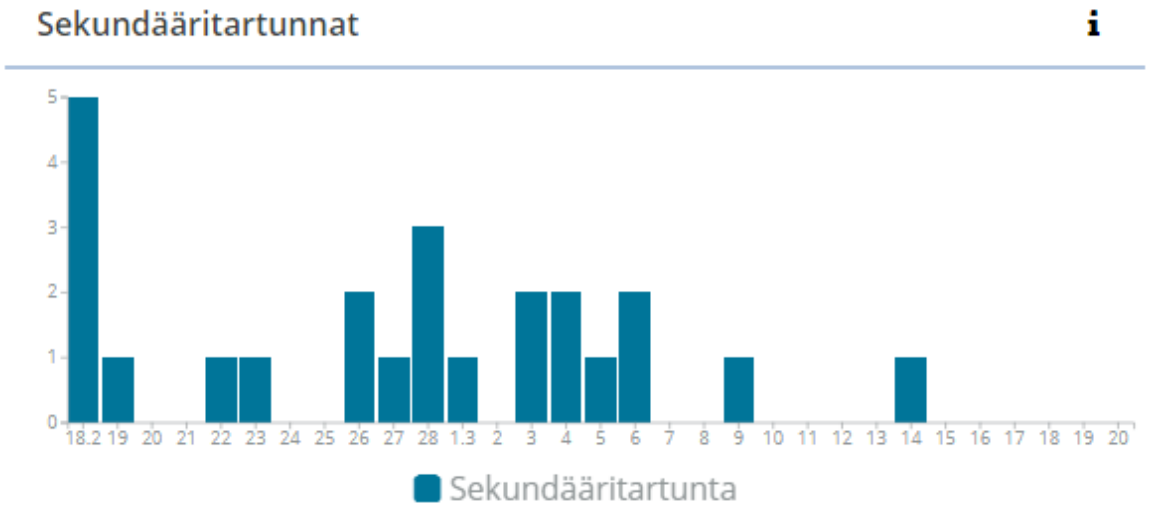
Trädet består av olika noder, där varje nod representerar en positivt smittad person. Därefter går länkar vidare till andra noder som då indikerar att den positivt smittade personen har exponerat x-antal personer. De exponerade kan därefter utveckla ett positivt testresultat och kedjan fortsätter (se Figur 15). På grund av att trädet innehåller personsignum visas inte dessa i figuren. I stället visas en figur med allmänna namn.



Figur 15. Exempel på mätare 2 – “Tartuntaketjut”.

4.3 Mätare 3

Den tredje mätarens kod påminner om den första mätarens ganska mycket och därför blev också resultatet liknande. Ett stapeldiagram med endast en kategori och namnet för mätaren blev ”Sekundääritartunnat” eller ”Sekundärsmittor” på svenska (se Figur 16). Namnet ”Sekundääritartunnat” kommer från idén att en exponerad person som försatts i karantän inte är ett direkt smittfall, eftersom personen ännu inte testat positivt. I stället klassas det som en sekundär smitta eftersom som personen kan utveckla ett positivt resultat i ett senare skede på grund av exponeringen. En primärsmitta kan anses vara en person som testat positivt och sätts i karantän därefter. Det finns också en info-knapp som innehåller förklaring till hur mätarens data är framställda. Skillnaden från målsättningen blev att mätaren visar ett intervall på 30 dagar i stället för att användaren skall kunna välja sitt intervall själv.



Figur 16. Exempel på mätare 3 – “Sekundääritartunnat”.

5 Diskussion

Målsättningen för arbetet var att presentera tre mätare som visar olika sorters information om Covid-19. Enligt min åsikt blev mätarna lyckade och de följer ganska långt kraven som ställdes i målsättningen. Små avvikelser jämfört med målsättningen, men sådant kan man räkna med eftersom ett projekt kan ändras med tiden.

En idé om att utveckla en mätare, som visar massexponeringar och smittor som kommit från dem, fanns också. Utvecklingen av den senareläggs på grund av problemet med att det finns olika data hos olika sjukvårdsdistrikt. Därmed finns det ingen klar linje att gå efter, vilket hindrar utvecklingen av mätaren. Det måste undersökas mera ifall det finns data som kan användas eller om det samlas in nya data som kunde vara till hjälp. Jag kommer att undersöka och utveckla mätaren ifall möjligheten öppnar sig.

Det finns rum för förbättringar i projektet och mätarna. Till exempel kan det komma förbättringsförslag från kundens sida efter att de haft mätarna i användning en tid. Jag tycker också att det finns förbättringar att göra i hur man till exempel hämtar information eller presenterar den. Man kan till exempel snabba upp hämtningen av data. Speciellt i modulen som skapades för träddiagrammet ”Tartuntaketjut” finns det möjligheter att försnabba processen. Det kan också komma förändringar i mätarna ifall data som samlas in ändrar struktur eller dylikt.

Personligen tycker jag att projektet har varit lärorikt. Jag har fått ta del av att vara en del av ett projekt och dess utveckling. På samma gång har jag utvecklat mina kodningsfärdigheter och min datahantering. Genom veckomöten med kollegor från Neotide har jag också fått lära mig att hantera min tid effektivare för att kunna presentera framsteg inför alla möten. På det sättet hålls projektet rullande, vilket jag tycker är en bra sak. Alla dessa saker har utvecklat mig som kodare och som person och jag tror jag kommer ha nytta av dem i arbetslivet framöver.

Avslutningsvis vill jag passa på och tacka mina kollegor på Neotide för all hjälp och stöd jag fått med projektet. Jag vill också tacka min handledare från skolans sida, Kaj Wikman, för hans handledning under arbetets gång.

Källförteckning

Bostock, M. (den 14 April 2021). *D3 Data-Driven Documents*. Hämtat från https://d3js.org/#introduction

Codecademy. (den 24 Februari 2021). *What is a Relational Database Management System?* Hämtat från https://www.codecademy.com/articles/what-is-rdbms-sql

Institutet för hälsa och välfärd. (den 23 Mars 2021). *MRSA*. Hämtat från https://thl.fi/sv/web/infektionssjukdomar-och-vaccinationer/sjukdomar-och-bekampning/sjukdomar-och-sjukdomsalstrare-a-o/mrsa

Microsoft. (den 12 Februari 2021). *Object Explorer*. Hämtat från https://docs.microsoft.com/en-us/sql/ssms/object/object-explorer?view=sql-server-ver15

Microsoft. (den 12 Februari 2021). *What is SQL Server Management Studio (SSMS)?* Hämtat från https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

Neotide Ab. (den 23 Juli 2020). *MMKR Käyttöohje.* Hämtat från [Internt dokument]

Neotide Ab. (u.å.a). *Exreport*. Hämtat från https://neotide.fi/exreport_sv.html den 22 Januari 2021

Neotide Ab. (u.å.b). *Företaget*. Hämtat från https://neotide.fi/foretaget.html den 19 Januari 2021

Neotide Ab. (u.å.c). *Produkter*. Hämtat från https://neotide.fi/produkter.html den 22 Januari 2021

Neotide Ab. (u.å.d). *SAI*. Hämtat från https://neotide.fi/sai_sv.html den 10 Januari 2021

Python Software Foundation. (den 11 Februari 2021). *General Python FAQ*. Hämtat från https://docs.python.org/3/faq/general.html

Social- och Hälsovårdsministeriet. (den 14 April 2021). *Sjukvårdsdistrikt och specialupptagningsområden*. Hämtat från https://stm.fi/sv/sjukvardsdistrikt-och-specialupptagningsomraden

Stephens, R. (2008). *Beginning Database Design Solutions.* Hämtad från: https://ebookcentral-proquest-com.ezproxy.novia.fi/lib/novia-ebooks/detail.action?docID=427853.

Teller, S. (2013). *Data Visualization with d3.js.* Hämtad från: https://ebookcentral-proquest-com.ezproxy.novia.fi/lib/novia-ebooks/detail.action?docID=1389320.