



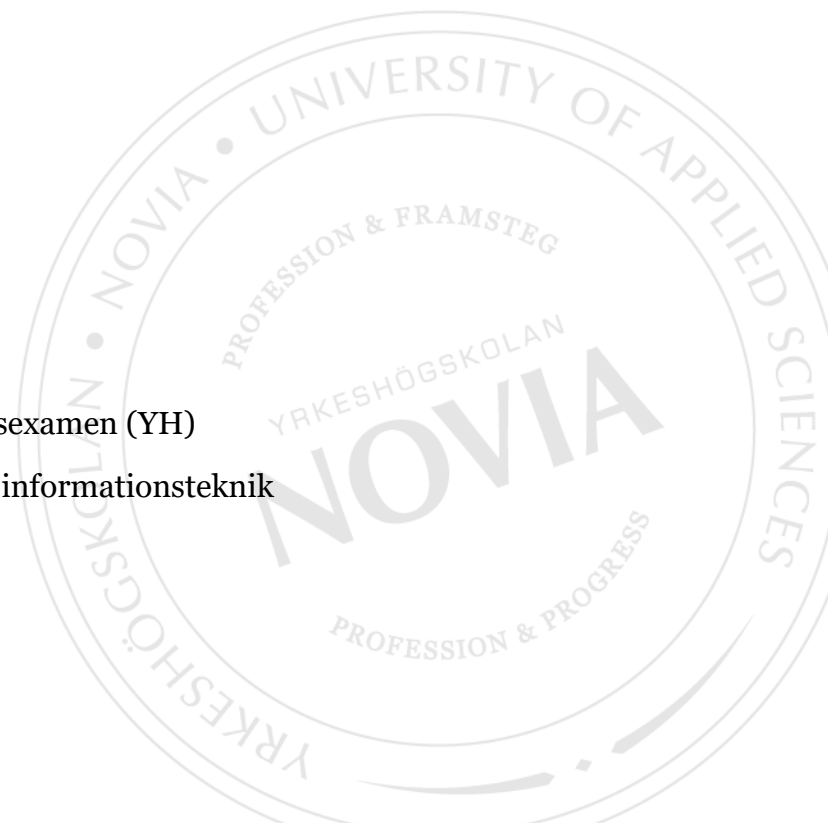
Utveckling av kartverktyg för mättjänst

Simon Liljeström

Examensarbete för ingenjörsexamen (YH)

Utbildningsprogrammet för informationsteknik

Vasa 2015



EXAMENSARBETE

Författare: Simon Liljeström
Utbildningsprogram: Informationsteknik, Vasa
Handledare: Susanne Österholm

Titel: *Utveckling av kartverktyg för mättjänst*

Datum: 01.12.2015

Sidantal: 33

Abstrakt

Detta ingenjörarbete har gjorts på begäran av Vasa Elnät som är ett av Vasa Elektriskas dotterbolag. Uppgiften var att skapa ett kartprogram som visar ny mätardata varje dag för Vasa Elnäts alla kunder. Programmet skulle visa data som kan filtreras för att hitta probleplatser och även ge en överblick över alla kunders mätarinformation och kopplingssituation.

Uppgiften utvecklades med Leaflet som är ett JavaScript-bibliotek med öppen källkod för mobilvänliga interaktiva kartor. Uppgiften gjordes med bland annat PHP, JavaScript, jQuery, AJAX och SQL. Informationen om kund- och mätardata samlades in från företagets olika databassystem till en ny databas.

Resultatet blev ett kraftfullt verktyg som underlättar det dagliga arbetet för Vasa Elnäts anställda.

Språk: svenska

Nyckelord: PHP, JavaScript, JSON, jQuery, AJAX, SQL

BACHELOR'S THESIS

Author: Simon Liljeström
Degree Programme: Information technology, Vasa
Supervisor: Susanne Österholm

Title: *Development of a Map Tool for Measurement Service*

Date: 01.12.2015 Number of pages: 33

Abstract

This Bachelor's thesis was done on behalf of Vasa Elnät, which is one of Vasa Elektriska's subsidiaries. The task was to create a map system that gathers new reading data every day for Vasa Elnät's customers and can be filtered to find problem areas or give a quick overview of all customers' readings and coupling situation.

The job was developed with Leaflet, an open-source JavaScript library for mobile-friendly interactive maps. This was done with PHP, JavaScript, JQuery, AJAX and SQL. The information about customers and readings was gathered from the company's different database systems into a new database.

The result is a powerful tool used by Vasa Elnät today, a tool which makes their daily work easier.

Language: Swedish Key words: PHP, JavaScript, JQuery, AJAX, SQL

OPINNÄYTETYÖ

Tekijä: Simon Liljeström
Koulutusohjelma: Tietotekniikka, Vaasa
Ohjaajat: Susanne Österholm

Nimike: *Mittauspalvelun karttatyökalun kehittäminen*

Päivämäärä: 01.12.2015

Sivumäärä: 33

Tiivistelmä

Tämä insinöörityö on tehty Vaasan Sähköverkon pyynnöstä. Tilaaja on yksi Vaasan Sähkön tytäryhtiöistä. Tehtävänä oli luoda karttaohjelma, joka lataa joka päivä Vaasan Sähköverkon kaikkien asiakkaiden uudet mittaritiedot, joita voidaan käyttää paikantamaan ongelma-alueita tai antamaan yleiskuva asiakkaiden mittaritiedoista ja kytkimien tilasta.

Tehtävän suorittamista varten käytettiin ohjelmaa Leaflet, joka on JavaScriptin avoin lähdekoodikirjasto mobiililaitteille ja sopii interaktiivisiin karttoihin. Tämä tehtiin käyttäen PHP, JavaScript, jQuery, AJAX sekä SQL ohjelmia, joiden avulla asiakkaiden ja mittareiden tiedot kerättiin yhtiön eri tietokantajärjestelmistä uuteen tietokantaan.

Tulos oli voimakas työväline, joka helpottaa Vaasan Sähköverkon jokapäiväistä työtä.

Kieli: ruotsi

Avainsanat: PHP, JavaScript, jQuery, AJAX, SQL

Innehållsförteckning

1	Uppdragsgivare.....	1
1.1	Bakgrund	2
1.2	Uppgift.....	2
2	Teknik.....	4
2.1	HTML.....	4
2.2	CSS.....	4
2.3	PHP.....	6
2.4	AJAX.....	7
2.5	SQL.....	8
2.6	JavaScript.....	8
2.7	jQuery.....	9
2.8	JSON/XML.....	9
3	Utförande	11
3.1	Programutvecklingsmetodik	11
3.2	Iterationer och planering.....	13
3.3	Kartsystem.....	14
3.4	Analys av system	15
3.5	Databasdesign.....	17
3.6	Filter	18
3.6.1	Datumfilter.....	18
3.6.2	Sökfunktion och listfilter.....	19
3.6.3	Mätartypfilter	21
3.7	Skript för dataöverföring	23
3.8	Markörer.....	26
3.9	Tillägg till kartverktyget.....	28
4	Resultat och diskussion	30
5	Källförteckning.....	32

Ordförklaringar

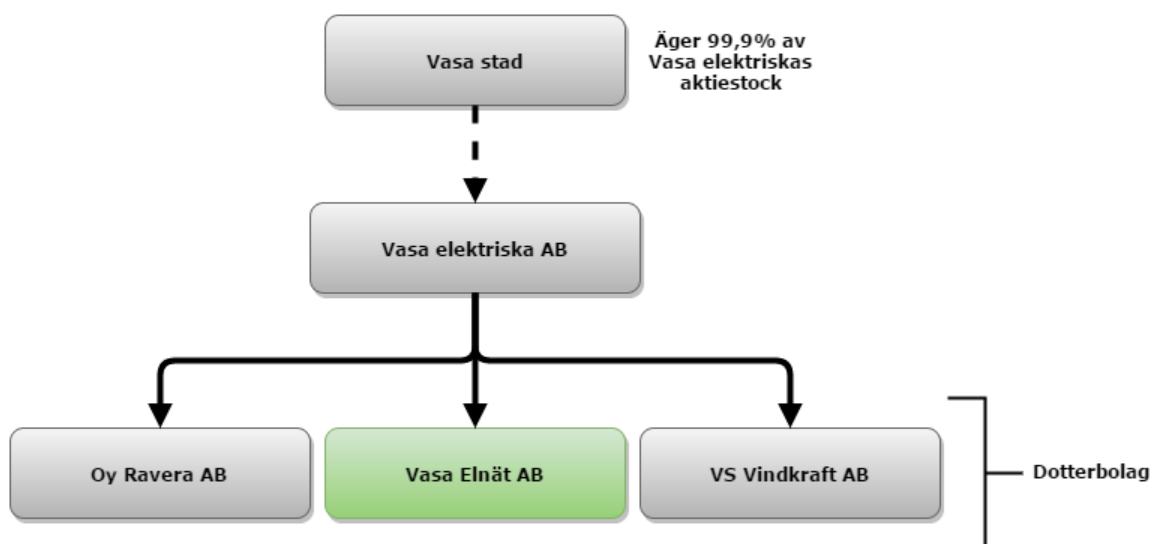
API	Application Programming Interface är en uppsättning av rutiner, protokoll och verktyg som används för att bygga applikationsprogramvara.
PHP	HyperText Preprocessor är ett skriptspråk som körs på server-sidan i webbutveckling och generell programmering.
CSS	Cascading Style Sheets är ett språk som beskriver hur en webbsida presenteras i en webbläsare.
AJAX	Asynchronous Javascript and XML är en samling av tekniker som kan sända och ta emot data från en server asynkront utan att störa presentationen eller funktionen av en webbsida.
XML	Extensible Markup Language är ett sätt att sända data mellan olika informationssystem i ren text.
HTML	Hypertext Markup Language är standarden för att skapa webbsidor.
JSON	JavaScript Object Notation är dataobjekt i textbaserat format som används i utbytet av data.
DOM	Document Object Model är ett språk- och plattformsoberoende gränssnitt som ger möjligheten för programmeringsspråk att läsa och uppdatera formatering och struktur av ett dokument innehåll dynamiskt.
CSV	Comma-separated values sparar tabelldata som ren text där varje rad är en datapost.
SQL	Structured Query Language är standardspråket som används i kommunikation med databaser.
UML	Unified Modeling Language är ett modellspråk i programutveckling som är standarden som används när man visualiserar designen av ett system.
UNICODE	Unicode är en industristandard som tillåter datorer att hantera text skriven i alla världens skriftsystem.
CRONTAB	Crontab är ett systemprogram under Unix som kör tidsbaserade jobb med användarvalda intervaller.
LEAFLET	Leaflet är ett Javascript-bibliotek med öppen källkod för interaktiva kartor.
MIT-licens	MIT-licensen är en programvarulicens som tillåter användare att fritt modifiera och distribuera programvara på nytt.
IIFE	Immediately-invoked function expression är ett JavaScript-designmönster som bland annat skyddar mot att ha för många globala variabler och ger tillgång till metoder utan att privata variabler blir tillgängliga.

1 Uppdragsgivare

Vasa Elnät är ansvarigt för överföringen och distributionen av elektricitet och tillhörande tjänster för invånare i Vasa, Laihela, Vörå, Korsnäs, Korsholm, norra delen av Närpes och Malax. Vasa Elnät är ett av tre dotterbolag som ägs av Vasa Elektriska och har i uppgift att bedriva elnätsaffärsverksamheten. Vasa Elektriska grundades i april 1892 och har på 123 år växt till ett mångsidigt nationellt energibolag med ca 110 000 kunder. Vasa stad äger nu 99,9 procent av Vasa Elektriskas aktiestock. Vasa Elnät hade år 2014 en omsättning på ungefär 29,8 miljoner euro med en arbetskraft på 33 personer och överför 962 GWh till över 68 608 kunder. [1]

De två andra dotterbolagen som Vasa Elektriska äger heter VS Vindkraft och Ravera. VS Vindkraft byggdes år 1991 och är Finlands första vindpark. Parken är stationerad i Korsnäs och har ungefär 1 300 MWh årlig elproduktion. [1]

Ravera grundades 2006 och har för tillfället 60 anställda och är specialiserat på drift, underhåll och byggande av vägbelysningen samt eldistributionsnät. Ravera har sin verksamhet i Kristinestad, Kurikka och Vasa. [2]



Figur 1. Företagshierarki.

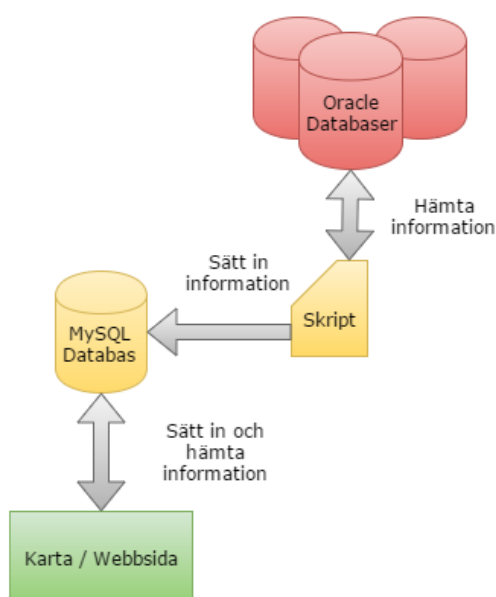
1.1 Bakgrund

Vasa Elnät har många olika applikationer som hanterar och lagrar information om kunder, mätare och förbrukningsplatser. Man önskade sig ett sätt att visualisera all denna information på ett sätt som är enkelt, användbart och ändamålsenligt. Problemet med de nuvarande lösningarna var att man måste gå in i flera olika applikationer och jämföra data för mätare och kunder.

Vasa Elnät kom med idén att visa allting på en interaktiv karta, som skall kunna visa information för personalen som arbetar i driften samt för övrig personal som kan ha användning av informationen. Kartan skulle kunna lokalisera enskilda kunder och mätningsplatser med en sökfunktion och kunna filtrera mellan olika datum. Programmet skulle även kunna visa de senaste 30 dagarnas mätningsvärden för varje kund som bolaget har.

1.2 Uppgift

Uppgiften blev således att skapa ett interaktivt kartprogram. Informationen skulle hämtas från Oracle databaser i de nuvarande systemen och sedan sammanföras till en ny MySQL-databas. Den nya databasen skulle i sin tur användas av kartprogrammet. Struktur och innehåll för den nya databasen skulle diskuteras fram tillsammans med Vasa Elnäts förmän vartefter som arbetet framskred.



Figur 2. Visualisering av uppgiften.

Till uppgiften hörde också att välja programmeringsspråk och en programutvecklingsmetodik. Metodiken behövde vara sådan att det skulle vara möjligt att snabbt lägga till nya idéer genom de olika arbetsfaserna. Programmet behövde även kunna styras med hjälp av en extern konfigurationsfil innehållande enkla parametrar.

2 Teknik

Detta kapitel beskriver de olika teknikerna som användes beträffande programmeringsspråk, databaser och bibliotek med öppen källkod samt motiveringar till varför vissa tekniker valdes framför andra.

2.1 HTML

Utifrån de tekniker som valdes var det kristallklart redan från början av projektet att HTML behövdes, eftersom programmet skulle fungera som en slags hemsida man kan gå in på via företagsnätverket.

Märkspråket HTML eller "HyperText Markup Language" utgör med HTTP och TCP/IP standarden för WWW (World Wide Web). I allmänhet skrivs webbsidor som HTML och med hjälp av HTTP överförs webbsidorna över internet. Med HTML ges möjlighet att bland annat ange dokumentets struktur med rubriker och metainformation. [11]

I HTML-kod kan man infoga information från andra filer som inte är av samma typ, till exempel Java/JavaScript, CSS-filer och vanliga bilder. HTML utvecklas kontinuerligt och HTML5 är den femte och senaste revisionen som blev färdig och publicerades av "World Wide Web Consortium", W3C, den 28 oktober 2014. Den tidigare versionen, HTML4, blev standard år 1997. [11]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample page</title>
  </head>
  <body>
    <h1>Heading</h1>
    <p>This page is just a demo. Also,
    I'm a paragraph.</p>
  </body>
</html>
```

Figur 3. Exempel på HTML kod. [11]

2.2 CSS

CSS eller "Cascading Style Sheets" är en stilmall som fungerar som ett språk, vilket beskriver hur ett dokument presenteras, allt från färg, textstorlek, typsnitt till vilken position som ett element skall visas på i en webbläsare. Namnet CSS kommer från

möjligheten att återanvända ett definierat utseende på flera dokument som alla har olika innehåll. Med hjälp av CSS-filer kan man alltså få ett dokument att ta hänsyn till olika typer av datorer, deras skärmapplösning och installerade typsnitt för att få en mera konsekvent upplevelse för olika användare. [12]

```
.menu-ui2 {
  background:#fff;
  position:absolute;
  top:170px;right:10px;
  z-index:1;
  border-radius:3px;
  width:120px;
  border:1px solid rgba(0,0,0,0.4);
}
.menu-ui2 a {
  font-size:13px;
  color:#404040;
  display:block;
  margin:0;padding:0;
  padding:10px;
  text-decoration:none;
  border-bottom:1px solid rgba(0,0,0,0.25);
  text-align:center;
}
.menu-ui2 a:first-child {
  border-radius:3px 3px 0 0;
}
.menu-ui2 a:last-child {
  border:none;
  border-radius:0 0 3px 3px;
}
.menu-ui2 a:hover {
  background:#f8f8f8;
  color:#404040;
}
.menu-ui2 a.active,
.menu-ui2 a.active:hover {
  background:#3887BE;
  color:#FFF;
}
```

Figur 4. CSS-kod från programmet som beskrev mätarfiltret.

2.3 PHP

Till arbetet kunde egentligen vilket språk som helst användas, men PHP i kombination med SQL användes redan från tidigare. Därför valdes dessa språk även för det nya programmet.

PHP betyder "HyperText Preprocessor" och är ett språk som främst används för att köra dynamiskt innehåll åt webbsidor på serversidan. För detta behövs en webbserver med PHP installerat och en webbläsare som kan läsa PHP-utskriften från servern. Språket kan även användas på andra sätt med t.ex. CLI (Command Line Scripting), vilket gör att man kan köra koden utan en server eller webbläsare; bara en PHP-parser behövs. Detta är användbart om man till exempel kör "cron" på Linux eller "Task scheduler" på Windows. En annan fördel med PHP är att det kan köras på alla de största operativsystemen: Windows, Linux, Mac OS X och många olika Unix-distributioner som Solaris och OpenBSD. [10]

Det som även gjorde PHP till ett utmärkt val för arbetet är att det fungerar på nästan vilket operativsystem eller webbserver som helst utan hinder och att man enkelt kan skapa en koppling till en eller flera olika databasinstanser. [10]

```
<?php
//go up one directory, where database.php is located
$file_path = realpath(dirname(__FILE__) . '/../');
require_once($file_path . '/database.php');
ini_set('memory_limit', '-1');
ini_set('max_execution_time', 0);

$db = new Db();

//connect to dev server
$conn = $db->connect("dev");

$sql = "SELECT columns
        FROM place p
        INNER JOIN database.someTable st on p.columnB=st.columnA
        WHERE criteria
        AND misc";

$db->mysqli_try_catch_query('SET CHARACTER SET utf8');
$result = $db->mysqli_try_catch_query($sql);

while($row = $result->fetch_assoc()){
    $data[] = $row;
}
echo json_encode($data);

unset($conn, $sql, $db, $row, $result, $data);
gc_collect_cycles();
?>
```

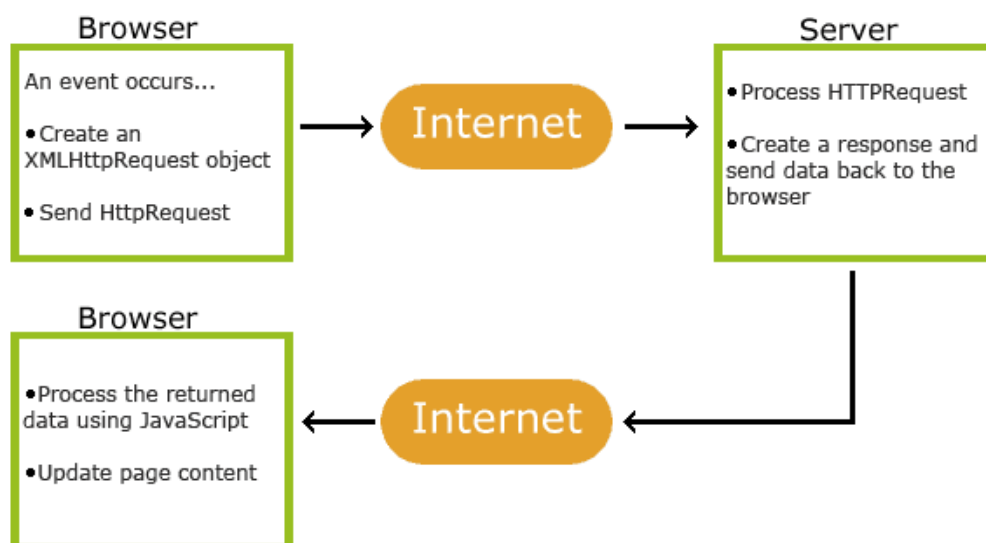
Figur 5. Förvrängd PHP-kod från programmet.

2.4 AJAX

Till uppgiften hörde att en förbrukningsplats skulle kunna visa de senaste 30 dagarnas mätningsuppgifter som skulle hämtas från databasen. Problem uppstår dock när webbsidan laddat klart och man exempelvis vill se de senaste dagarnas mätningsdata. Man ville inte att sidan skulle behöva laddas om, för då försvinner allt pågående också och alla förbrukningsplatser skulle då hämtas på nytt. Tekniken som kan användas för att undvika detta problem kallas AJAX och tillåter att man hämtar data från en webbserver utan att hela sidan måste laddas om, och istället bara de delar man vill.

AJAX är förkortning av "Asynchronous JavaScript and XML" och är en samling av tekniker som används på klientsidan för att skapa asynkrona webbapplikationer. Traditionellt måste en sida laddas om för att uppdatera när man skickar eller tar emot data. Med AJAX kan webbsidan, under tiden annat händer, sända och ta emot information från en server utan att behöva uppdatera sidan. Tack vare detta kan sidan utan avbrott visas åt en användare. Trots sitt namn behöver man varken använda sig av XML eller göra sina förfrågningar asynkront. [13]

AJAX är inte ett nytt programmeringsspråk, utan bara ett nytt sätt att använda de standarder som redan finns. AJAX använder sig av standarderna XMLHttpRequest object, JavaScript/DOM, CSS och XML. XMLHttpRequest-standarden gör ett asynkront utbyte av information med en server, medan själva JavaScript/DOM behövs för att visa och manipulera den hämtade informationen. [13]



Figur 6. Exempel på hur AJAX fungerar. [14]

2.5 SQL

En stor del av projektet utgjordes av att man skulle kunna hämta och manipulera information från databaser. För att kunna kommunicera med en relationsdatabas behöver man använda sig av kommandospråket SQL.

SQL är en förkortning av "Structured Query Language" och är ett kommandospråk som specialiserar sig på att hantera data som är sparade i en databas. Det utvecklades först av Raymond Boyce och Donald Chamberlin i början av 1970-talet och var baserat på Edgar F. Codd's relationsmodell i företaget International Business Machines (IBM) och släpptes år 1979 av Relational Software Inc., som numera är känt som Oracle Corporation. I språket SQL finns sätt att definiera data, modifiera data och kontrollera data för en databas. Kommandon som ofta används är: create, drop, alter, select, insert, update, delete, grant och revoke. Det vanligaste man gör med SQL är förfrågningar mot en databas med hjälp av kommandot "select". [15]

```
SELECT owner,Table_name
FROM all_tables
WHERE Table_name
LIKE '%CUSTOMER%'
ORDER BY owner,Table_name;
```

Figur 7. Exempel på en SQL select-sats.

2.6 JavaScript

JavaScript behövdes bland annat för att skapa en interaktiv karta. Tekniken som användes, och som förklaras i större detalj i kapitlet 3.3, heter "Leaflet" och är ett JavaScript-bibliotek med öppen källkod för interaktiva kartor.

JavaScript är ett prototypbaserat skriptspråk som vid sidan om HTML och CSS är den teknik som används i skapandet av innehåll man hittar på webben. JavaScript skapades ursprungligen år 1995 av Brendan Eich, som arbetade för Netscape Communications Corporation. Största delen av alla sidor på internet använder sig av JavaScript och alla moderna webbläsare har stöd för språket utan nerladdning av insticksmoduler. Trots sitt namn har skriptspråket JavaScript inget att göra med det robusta programmeringsspråket Java. Båda är objektorienterade programmeringsspråk och är härledda från C-språket, men där slutar likheterna. [16]

```
function getAllMarkers(){
    var data = <?php include 'example_get_markers.php';?>;
    //Format data and send marker data to 'values'
    mt_namespace.values = setMarkers(data);
    //Add values to markers
    mt_namespace.markers.addLayers(mt_namespace.values);
    //set searching parameters to new layer
    setSearch(mt_namespace.markers);
    //add markers to map
    mt_namespace.map.addLayer(mt_namespace.markers);
    //returns visible markers
    visible_markers(mt_namespace.values);
}
```

Figur 8. Javascript-kod som hämtar alla markers med hjälp av PHP och IIFE. [24]

2.7 jQuery

I applikationen behövdes funktionen att gömma eller visa element. Exempelvis då man klickar på en knapp skall programmet hämta information från databasen eller filtrera någonting specifikt. Detta görs enkelt med jQuery.

jQuery är ett JavaScript-bibliotek som underlättar skriptningen på klientsidan av HTML. jQuery är gratis att ta i bruk under MIT-licensen och är installerat på största delen av de populäraste webbsidorna idag. Med jQuery kan man enkelt skapa animationer, hantera händelser, underlätta AJAX och mycket mera med en API som är lätt att förstå och använda. [17]

```
$(document).on("click", "#clear", function(){
    $("#styled").val('');
});
```

Figur 9. jQuery-kod som tömmer "styled" elementet då man klickar på "clear-knappen".

2.8 JSON/XML

JSON, eller JavaScript Object Notation, är ett öppet standardformat som använder sig av en struktur som är lätt för människor att läsa och överföra som dataobjekt. Detta används som ett alternativ till XML och är det primära dataformatet som tas i bruk när asynkron kommunikation utförs mellan en webbläsare och en server. Det används exempelvis av AJAX. Fastän JSON från början var härlett från JavaScript-språket är det i ett format som är oberoende av enskilda programmeringsspråk och kan i stort sett

användas av alla moderna programmeringsspråk, inklusive Perl, Python, Java, JavaScript, C-familjen C, C#, C++ och många flera. [18]

Då AJAX användes för att hämta information från databasen returnerades data som JSON-objekt. Detta gjordes eftersom kartsystemet Leaflet bäst kunde hantera objekt av denna typ för att skapa "markers" effektivast. En marker är en punkt på kartan som i detta fall skulle innehålla information om förbrukningsplatsen. Alla Vasa Elnäts kunder skulle alltså visas som enskilda markers. I ett JSON-objekt finns bland annat information om kundens namn, adress, förbrukningsplatsnummer och koordinaterna som styr så att kunden visas på rätt position på kartan.

XML är förkortning av "Extensible Markup Language" och är liksom HTML också ett universellt märkspråk. År 1998 blev XML en W3C-rekommendation (World Wide Web Consortium). I rekommendationen beskrivs strukturen på XML och vad en XML-tolk behöver för att kunna läsa informationen. Syftet med XML är att skapa dokument i ett format som kan både förstås av människor och vara läsbart för maskiner. Målet är alltså att det skall vara enkelt och lättanvänt över hela internet. Det är ett textformat designat för att lagra och överföra data och stöds av Unicode för att representera de olika språk människor använder runtom i världen. [12]

```
{ "employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Anna", "lastName": "Smith" },
  { "firstName": "Peter", "lastName": "Jones" }
]}
```

Figur 10. JSON-exempel på ett "anställda" objekt som har en Array med tre olika anställda. [19]

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Figur 11. Samma exempel på ett objekt med tre anställda i XML. [19]

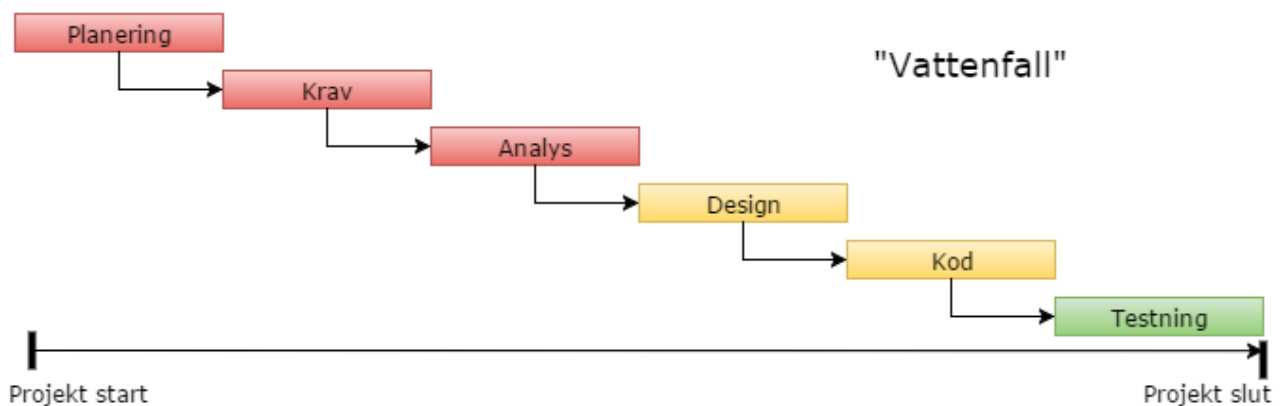
3 Utförande

I detta kapitel beskrivs projektets planeringsfaser, problemlösningar och motiveringar för tekniken som valts under examensarbetets utveckling från start till slutprodukt.

3.1 Programutvecklingsmetodik

Innan man kan börja programmera behövs en plan över hur arbetet skall framskrida. Planen inkluderar hur man delar in arbetet i mindre delar och milstolpar för hur lång tid varje del uppskattas ta. Milstolparna behöver vara kartlagda så tidigt som möjligt i projektet.

En så kallad programutvecklingsmetodik är ett system där man kan se faserna i projektet på ett visst sätt. Dessa system har ofta olika vikt placerad på de olika faserna.



Figur 12. Vattenfallsmodellen. [3]

Den vanligaste modellen kallas för vattenfallsmodellen. Enligt den ska en fas i projektet påbörjas och avslutas innan man kan övergå till nästa fas. I vattenfallsmodellen läggs stor vikt vid planering i början av projektet så att problem kan upptäckas tidigt och hanteras innan de uppstår. Fördelar med denna modell är att den är enkel och förstärker goda vanor, som att definiera innan man designar, designa innan man kodar och skapandet av en robust dokumentation. [4]

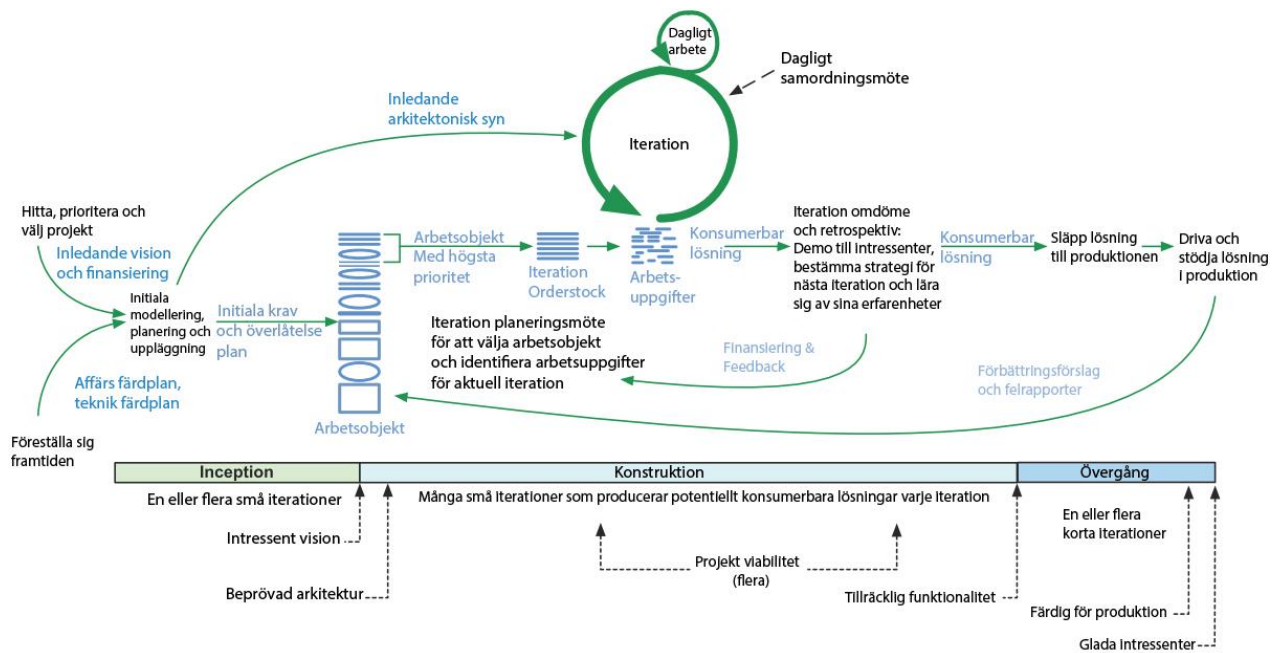
Den här modellen har dock problem med att matcha verkligheten. Även om den är bra för dokumentation är den i mindre projekt inte till fördel när man skapar nya program, eftersom modellen tidigt i projektet kräver en alldeles för hög nivå av precision. Om en fas inte lyckas, eller om man märker att något måste ändras i en utförd fas, blir det alltså väldigt dyrt i tid och pengar att gå tillbaka för att skriva om dokumentationen och starta projektet från början igen. [4]

Valet blev därför att avstå från denna traditionella modell, och istället försöka hitta en lösning där det är lätt och effektivt att uppdatera eller helt ändra delar av programmet. Agile-systemutveckling, som fokuserar mer på uppvisande av resultat än på en alltför ingående byråkratisk dokumentation, uppfyllde dessa krav. [5]

Agile-systemutveckling är de filosofier och principer som formulerades av en grupp programmerare i "The Manifesto for Agile Software Development" år 2001. Grundtanken bakom agile-metoderna är att det skall vara inkrementellt och iterativt, vilket betyder att man följer en plan och utvärderar och förbättrar regelbundet genom ett nära samarbete mellan utvecklare och köpare, ofta med dagliga möten. På detta sätt kan man konstant försäkra sig om att det som utvecklas är det som beställaren vill ha, samtidigt som man visar framsteg och resultat mycket oftare än om man valt en annan programutvecklingsmetodik. [6]

Det finns många olika agile-modeller som sätter olika vikt vid hur de olika iterationerna hanteras. Adaptive software development (ASD), Disciplined agile delivery (DAD), Extreme programming (XP) och Lean software development (LSD) är exempel på sådana systemutvecklingsmetoder som övervägdes tillsammans med uppdragsgivaren att tas i bruk. Slutligen valdes DAD för att den är lättförstådd och lätt skalbar till olika projektstorlekar. [5][7]

DAD fungerar som ett beslutsramverk där man enkelt och planenligt kan ta beslut runt iterationer. DAD bygger på många av de metoder som används av agile-gemenskapen, såsom "Scrum", vilket i korthet betyder att man arbetar som en enhet och har möten med alla i teamet och visar resultat och ändringar dagligen för att nå sitt mål. "Agile modeling" är en metodik för modellering och dokumentation som baserar sig på den bästa praxisen och LSD. Metodiken går ut på att hålla projektet så magert som möjligt, det vill säga att inte koda något i onödan eller ha alltför detaljerad dokumentation, utan leverera så snabbt som möjligt och fatta beslut så sent som möjligt. [7][8]



Figur 13. Disciplined agile delivery (DAD) livscykel. [9]

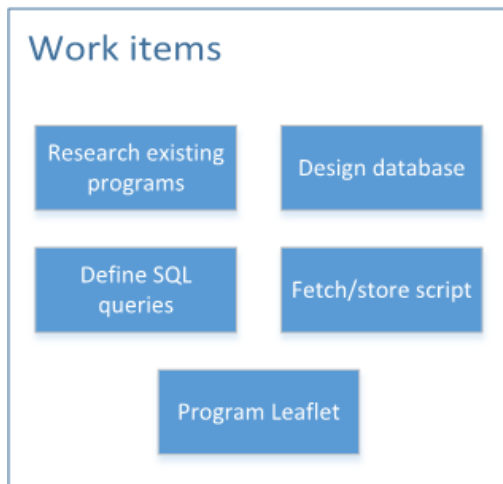
3.2 Iterationer och planering

Den programutvecklingsmetodik som valdes var alltså agile-metoden DAD och den fungerade som en röd tråd man kunde följa genom utvecklingen av programmet för att lätt få en överblick hur långt man kommit i projektet. I figur 14 kan man se hur hela DAD-processen ser ut. Enligt planen börjar man med att försöka kartlägga de olika delarna man kan se i projektet som mindre eller större iterationer. Då man har gjort en plan över hur iterationerna ser ut, kan prioriteringar göras över vad som behöver hända och i vilken ordning.

För att lyckas med planen måste man analysera och tänka efter vad som behövs för att nå slutmålet, fastän uppgiften i sig inte är helt definierad. Från början var det inte allt för svårt att se att man behöver förstå teknikerna som redan är i användning. För att göra det behövdes systemens databaser, varifrån informationen skulle hämtas, undersökas för att sedan utifrån det kunna designa en databas som lagrar informationen på rätt sätt. Dessa två var de första iterationerna som kunde definieras. Eftersom programmet skulle hämta information från databasen gick det även att se att SQL-förfrågningar behövdes som hämtar informationen åt programmet. Utöver detta behövdes även en funktion som automatiskt hämtar färsk information till databasen varje dag.

Det behövdes med andra ord ett skript som automatiskt körs samma tid varje dag med SQL-förfrågningar som både fyller databasen med information samt kan köras för att använda programmet. Det var de två följande iterationerna som definierades.

Att programmera allting från grunden skulle vara väldigt tidskrävande och därför blev en iteration att hitta och ta i bruk något slags öppet bibliotek för interaktiva kartor.



Figur 14. Iterationerna för projektet.

3.3 Kartsystem

En av de viktigaste milstolparna i projektet var att välja ett kartsystem som klarar av att visa informationen och som har de funktioner som kan tänkas behövas i framtiden, samt att man kan skapa egna lösningar ifall det inte skulle finnas ursprungligen i kartans bibliotek. Under forskningsfasen studerades biblioteken Leaflet, Google Maps och OpenLayers. Eftersom uppdragsgivarna ville ha en lösning med öppen källkod kunde Google Maps, även om det är robust, laddar snabbt och används av miljontals webbsidor, uteslutas tidigt i projektet.

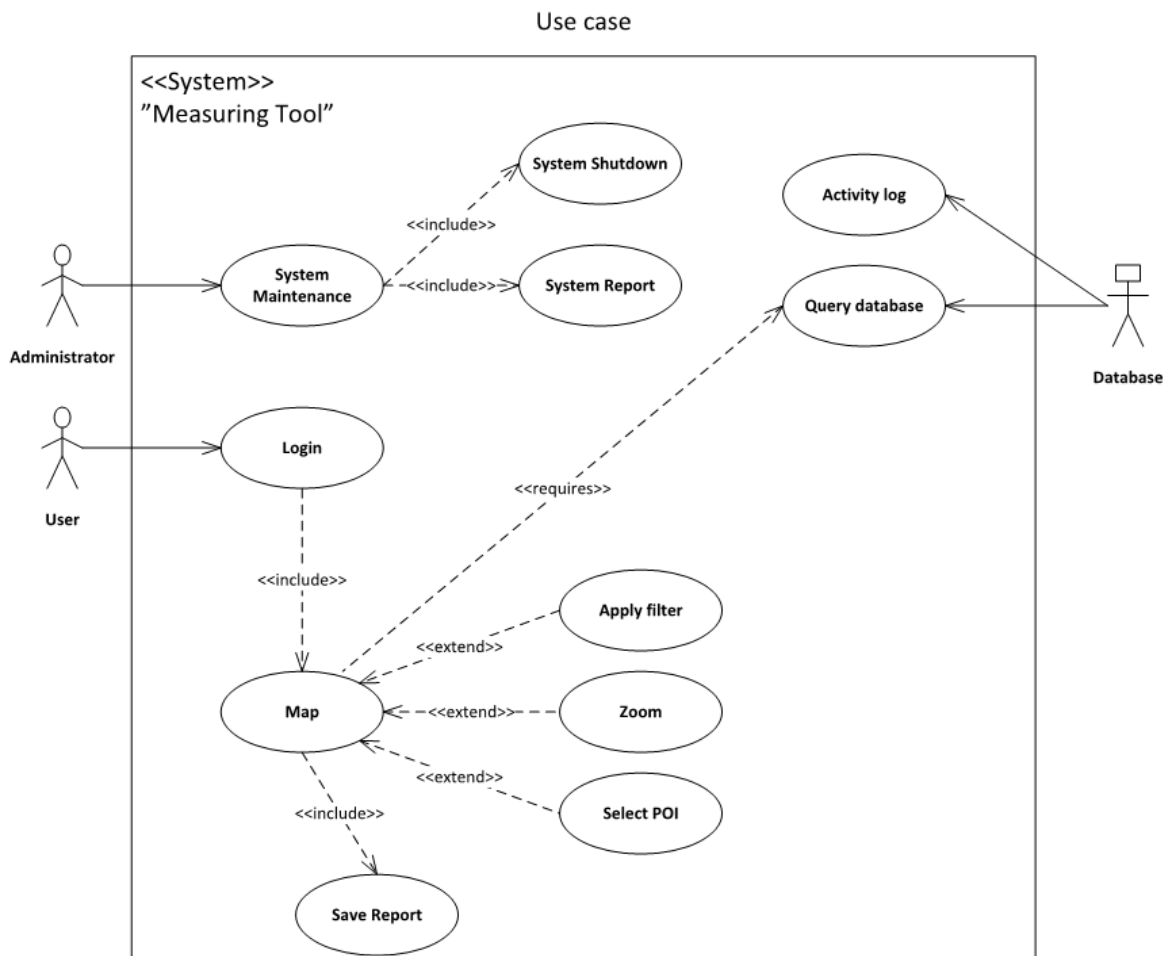
OpenLayers är som Leaflet ett JavaScript-bibliotek med öppen källkod som används för att visa kartdata i webbläsare. Båda liknar varandra, de har båda en stor användargrupp och det finns många exempel på skräddarsydda lösningar som användare har utvecklat och delat med sig på internet. OpenLayers är ett äldre (från 2006) och mera moget bibliotek och kommer därför med mera funktionalitet, men med kostnaden av en större filstorlek. Leaflet har en mycket bra API och har en mycket mindre filstorlek eftersom att det är relativt nytt (från 2011), men inte samma mängd funktioner som OpenLayers. [20][21]

Valet stod mellan ett äldre bibliotek som skulle kunna underlätta med sina många funktioner, möjligheter och flexibilitet eller ett nyare bibliotek som är lätt att använda, men kanske inte hade allting som behövdes. Valet föll på Leaflet.

3.4 Analys av system

Efter att programutvecklingsmetodiken valts och iterationerna etablerats måste de nuvarande systemen som företaget har i användning analyseras så att den databas som skapas för programmet skulle kunna fungera optimalt. Innan en databas för programmet kunde skapas analyserades dessa system med tanke på vilken information de lagrade och vilken information de hade som skulle behövas av kartprogrammet. Dessa system var stora och lagrade miljontals rader och kolumner av data.

De data som uppdragsgivarna önskade skulle visas på kartan måste diskuteras fram med dem på många korta möten under projektets gång. Många kolumner som skulle tas från de nuvarande databaserna och visas på kartan hittades snabbt, men en del av kolumnerna måste beslutas i ett senare skede efter feedback från användarna. Ett "use case" skapades över hur programmet skulle fungera. Detta illustreras i figur 15.



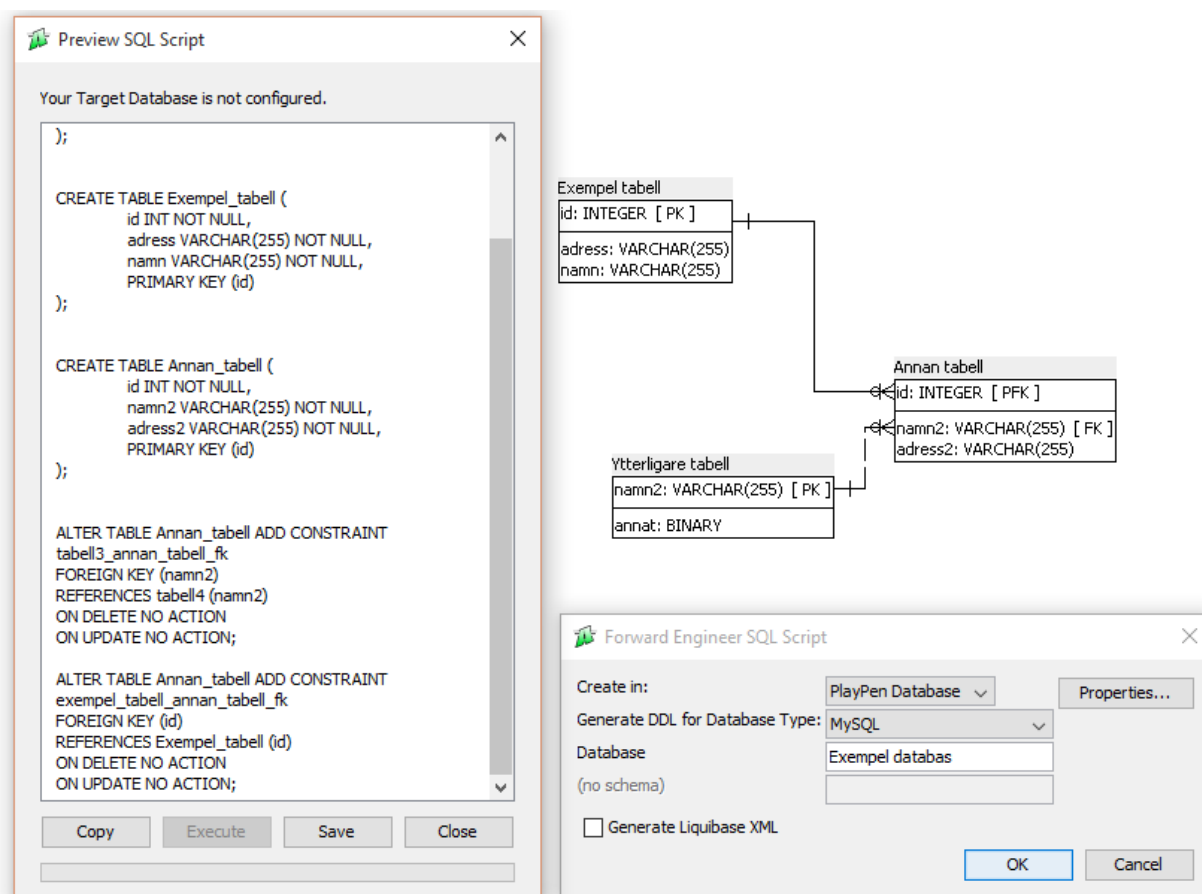
Figur 15. UML use case som överblick av programmet.

I programmet skulle användaren självklart kunna zooma in och ut, vilket man kan förvänta sig av ett kartprogram, men då även välja POIs, eller Points of interest, som är de enskilda förbrukningsplatserna dit alla kunder hör. I "Apply filter" finns en hel del olika funktioner som förklaras mer detaljerat i kapitel 3.6. Från figur 15 kan man också se att det går att spara en rapport från kartan och att kartan är beroende av förfrågningar till databasen. Databasen sparar också information om viktiga händelser i systemet. Utöver den normala användaren finns även en administrator som sköter om systemet. Till administratörens rättigheter hör att stänga systemet och att se på systemrapporter. "Login" nämns i figur 15, men den funktionen slopades senare i projektet, eftersom programmet endast är tillgängligt för användare inom företagets nätverk och skulle inte fylla någon meningsfull funktion.

3.5 Databasdesign

När forskningen och analysen av de system som redan fanns i företaget var avslutad skapades databasen. Det finns många tillvägagångssätt när man skapar en databas. Ett bra utgångsläge är att man skapar en ER-modell och sedan utifrån det skriver en SQL-sats som skapar en databas. Här används dock en applikation med öppen källkod från internet.

Programmet heter SQL Power Architect och har flera olika versioner. Vill man ha alla funktioner är det dyrt. En "Community edition" hade öppen källkod och hade alla funktioner som behövdes för att enkelt rita upp på dator hur databasen skulle fungera. Man kan rita kopplingarna mellan tabeller som sedan med hjälp av programmet skapar en SQL-sats, som inte bara skapar tabellerna utan också skapar kopplingar och begränsningar. I figur 16 kan man se ett exempel på hur man skapar en enkel databas med hjälp av SQL Power Architect och får SQL-kodutskrift. [22]



Figur 16. Exempel på "SQL Power Architect Community Edition". [22]

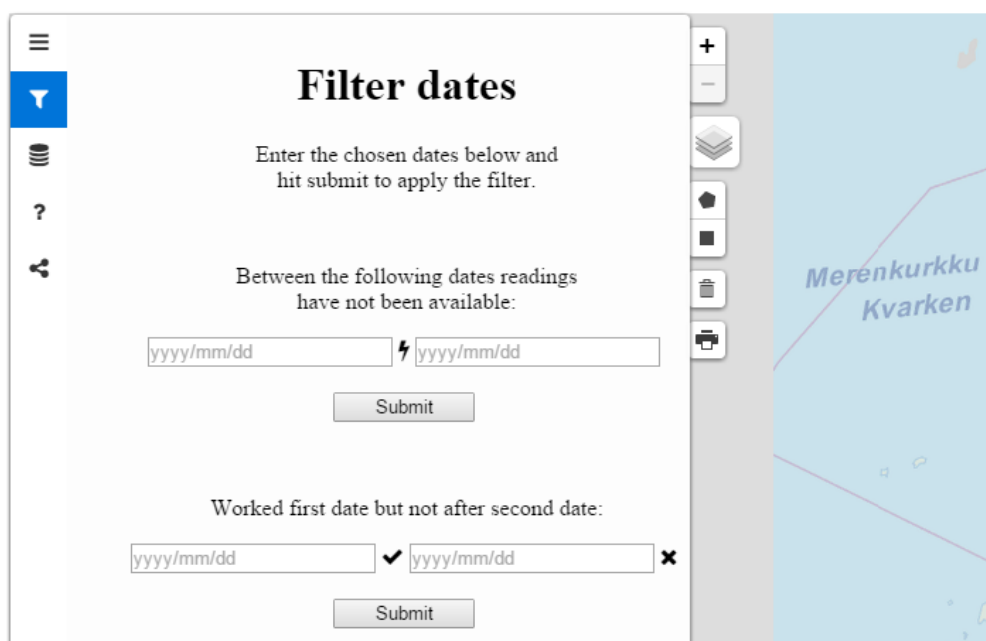
3.6 Filter

I detta kapitel förklaras i detalj hur filtreringen skapades i programmet. Detta var en så pass stor del av projektet att det krävs lite mera förklaring. Hela orsaken till att göra programmet var att man skulle kunna filtrera fram information snabbt. Därför inkluderar denna del många kodexempel från programmet, förklaringar och problemställningar.

3.6.1 Datumfilter

Företaget ville ha olika filter för att kontrollera vilka förbrukningsplatser som skulle visas i olika situationer. Ett verkligt exempel på hur ett sådant scenario skulle kunna se ut är exempelvis situationen efter ett kraftigt åskväder. Då behövs ett sätt att snabbt filtrera enligt de platser som fungerat fram till ett visst datum, med andra ord alldeles före åskvädret, men inte efter åskvädret, så att företaget snabbt kan sända personal till de drabbade områdena för att åtgärda problemen.

Ett annat filter som behövdes var att man skulle kunna visa endast de förbrukningsplatser som inte fungerat mellan två datum. Dessa två olika datumfiltreringar gjordes med en kombination av jQuery och AJAX. För att välja ett datum måste man klicka på en textruta som illustreras i figur 17. När man klickat på rutan dyker det upp en kalender med hjälp av jQuery-funktionen "datepicker" och man kan välja ett datum som i sin tur automatiskt skrivs in i textrutan.



Figur 17. Programmets datumfiltrering.

När man valt två datum och aktiverat filtreringen sparas datumen i variabler och en AJAX-rutin körs. Datumvariablerna skickas iväg till en PHP-fil där datumen sätts in i en SQL-förfrågan. Därifrån returneras informationen som JSON-objekt, som sedan formateras så att data kan visas på kartan för förbrukningsplatser.

```
//när knappen klickas på körs detta
$(document).on("click", "#filterDatesWorkedAfter", function(){
    //Spara datum till variabler
    var dateWorked = $('#dateWorked').val();
    var dateAfter = $('#dateAfter').val();

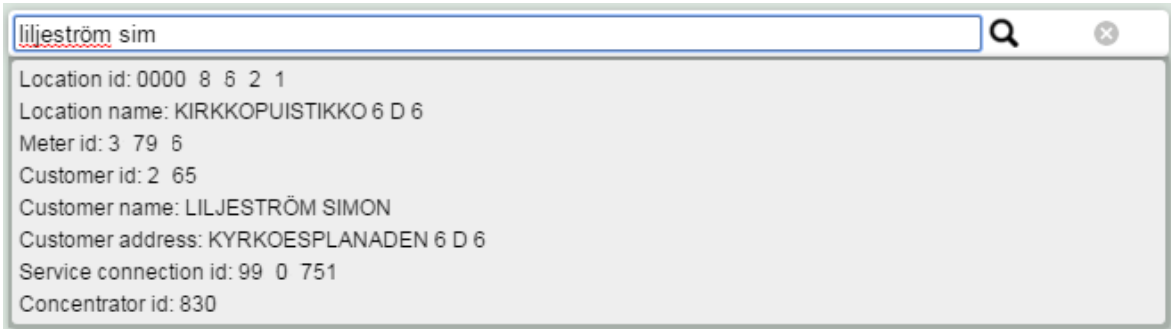
    if(dateWorked && dateAfter){
        $.ajax({
            type: "POST",
            url: 'EXEMPEL.php',
            data: {dateWorked: dateWorked, dateAfter: dateAfter},
            success:function(data){
                //Formatera JSON objekt för kartan
            }
        });
    }
});
```

Figur 18. Förenklad och förvrängd kod för datumfiltrering i programmet.

3.6.2 Sökfunktion och listfilter

Ytterligare behövdes filter för en sökfunktion. Man ville ha en funktion som automatiskt föreslår vartefter man skriver och dessutom fungerar som ett "magiskt" fält där man kan skriva in vad man vet om förbrukningsplatsen, till exempel kundens namn, plats-id och mätar-id. Ett sådant tillägg fanns för Leaflet, skapat av Stefano Cudini. Det gjorde nästan allting som behövdes, men tillägget behövde därtill kalibreras till applikationen och lite kod behövde sättas till. Exempelvis kunde tillägget bara söka efter ett datafält till en början, medan det behövdes möjlighet att flera datafält kontrolleras medan användaren skriver i fältet.

Tillägget ändrades så att man kan söka efter allting enskilt som beskrivs av en förbrukningsplats. Detta illustreras i figur 19. Medan man skriver filtreras saker som är nära sökningen tills endast det man angett visas. Klickar användaren på de rutor som visas eller på retur-tangenten ändrar kartvyn till ett inzoomat läge ovanför den sökta punkten och visar endast den platsen.



Figur 19. Exempel på namnsökning med censurerade fält.

Ett problem som uppstod var att när markers laddas in på kartan och tillägget sätts till kartan, fungerar det endast för det som är inladdat från programmet när man först startat det. Vad detta betyder i praktiken är att när en användare ser ursprungsläget av kartan kan sökningsfunktionen användas, men genast filtren ändras använder tillägget fortfarande ursprungliga data och inte de filtrerade värdena. När användaren då söker på något efter att kartan filtrerats visas gamla data och om man söker på en av dessa platser zoomas kartan in på platser som inte finns.

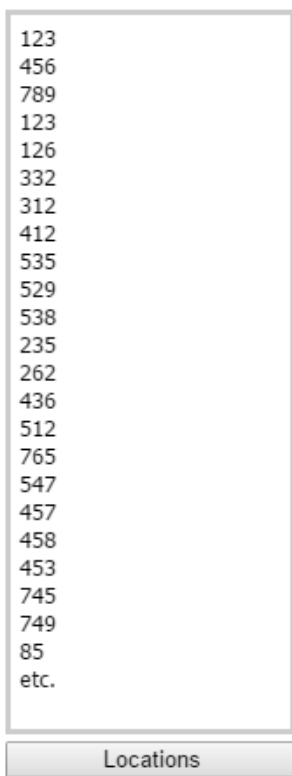
Lösningen på problemet var att för varje filtrering då ett JSON-objekt returneras från en AJAX-rutin kontrollera om söktillägget existerar och sedan ta bort och skapa den på nytt, vilket illustreras i figur 20. Denna metod fungerade som man tänker sig och orsakade inte en långsammare programkörning.

```
function setSearch(layer){
  if (mt_namespace.controlSearch != undefined){
    mt_namespace.controlSearch.removeFrom(mt_namespace.map);
  }

  mt_namespace.controlSearch = new L.Control.Search({
    layer: layer,
    initial: false,
    position: 'topright',
    tooltipLimit: 10,
    autoType: false,
    zoom: 20,
    //markerLocation: true,
    //markerIcon: marker.setIcon(redIcon)
    //animateLocation: true,
    circleLocation: true,
    propertyName: 'search'
  });
  mt_namespace.controlSearch.addTo(mt_namespace.map);
}
//set initial search for markers
setSearch(mt_namespace.markers);
```

Figur 20. Lösningen för sökfunktionen.

Under utvecklingen av programmet tyckte företaget att det vore bra om det även fanns ett sätt att filtrera enligt en färdig lista med förbrukningsplats-id:n. Om en färdig lista kunde klistras in i programmet skulle det underlätta deras jobb mycket. Lösningen som skapades var att i ett enkelt HTML textområde klistra in listans förbruknings-id:n. Dessa aktiverades med hjälp av jQuery och AJAX till en annan fil för att hämta den sökta informationen, formatera den och sedan returnera data som ett JSON-objekt.



Figur 21. Filtrering av inklippt exempellista.

3.6.3 Mätartypfilter

Majoriteten av förbrukningsplatserna har en mätare och varje mätare har en mätartyp. Det är väldigt användbart om man utifrån ett filter snabbt kan växla mellan de mätartyper man vill se på kartan. Att behöva fylla i och komma ihåg vad alla mätare heter är dock inte speciellt effektivt. Därför skulle ett mätartypfilter skapas som är snabbt att använda och filtrerar de synliga värdena på kartan.

Det skulle dessutom vara bra om nya mätartyper automatiskt lades till i programmet, så att användaren alltid har tillgång till de mätartyper som finns i systemen för tillfället. Att skapa en SQL-sats som hämtar unika mätartyper var inte ett problem och gick med en enkel "SELECT DISTINCT"-förfrågan.

```
$sql = "SELECT DISTINCT mätartyper FROM exempel_plats GROUP BY mätartyper";
```

Figur 22. Förvärgt SQL-urklipp från kod.

Efter att en Array har skapats med de olika mätartyperna blir problemet hur man skall skapa elementen på rätt position med rätt funktionalitet.

I koden i figur 23 skapas element dynamiskt på webbsidan med hjälp av funktionen "createMeters" och värdena från "meter_types"-Array. Mätartyperna hämtas vid programstart med hjälp av en extern PHP-fil som gör en SQL-förfrågan efter unika mätartyper. Elementen läggs till i området "menu-ui2" och får formatering och toggle-funktion från en CSS-fil.

```
for(var i = 0; i < mt_namespace.meter_types.length; i++){
    document.getElementById('menu-ui2').appendChild(createMeters(mt_namespace.meter_types[i].meter_type));
}

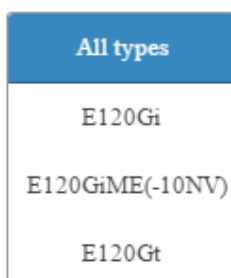
function createMeters(array){
    // Create the list element:
    var list = document.createElement('nav');
    var item = document.createElement('a');
    item.setAttribute('href', '#');
    item.setAttribute('id', array);

    for(var i = 0; i < array.length; i++) {
        // Create the list item:
        item.appendChild(document.createTextNode(array[i]));

        // Add it to the list:
        list.appendChild(item);
    }
    //Return the constructed list:
    return list;
}
```

Figur 23. Skapandet av mätartyp-element till "menu-ui2"-området med funktionen "createMeters".

Största problemet med att skapa dynamiska element som filtrerar data var koden som ser till att den faktiskt filtrerar de markers som visas på kartan utifrån namnet på knappen, och utan att det laddar om aktuella markers. För att kunna förstå hur detta fungerar måste man se på hur CSS-filen hanterar elementen som hör till "menu-ui2".



Figur 24. Exempel på hur en mätartyp-knapp ser ut när den är vald.

När man klickar på en knapp går den från vit bakgrund till blå. Det som händer i bakgrunden när en knapp blir vald är att den får en tagg med namnet "active", och när man tar bort fokus genom att klicka på knappen igen återgår namnet till " ", det vill säga tomt. I lösningen var detta avgörande.

```
//Pushes the name of all meters with the classname "active" to "all_types_array"
$(".menu-ui2 .active").each(function(){
    mt_namespace.all_types_array.push(this.id);
});

for(var i = 0; i < mt_namespace.values.length; i++){
    if(mt_namespace.all_types_array.contains(mt_namespace.values[i]._meter_type) ){
        mt_namespace.new_values.push(mt_namespace.values[i]);
    }
}
```

Figur 25. Kodurklipp från mätartyp-filtreringsfunktion.

Det är mycket annat som händer i funktionen, men i koden ovan kan man se att varje element i "menu-ui2" som har "active" trycker in elementets id i "all_types_array". Id var samma som mätartypnamnet från funktionen "createMeters" och kan därför i fortsatsen under jämföras för att se om de mätartyper som finns i "all_types_array" också finns i de markers som visas på kartan just då. Om de finns, sätts värdena in i en "new_values"-Array, som sedan i koden uppdaterar det användaren ser på kartan. Resultatet är att man kan välja flera mätartyper åt gången, och att kartan genast uppdateras med de markers som visas just då.

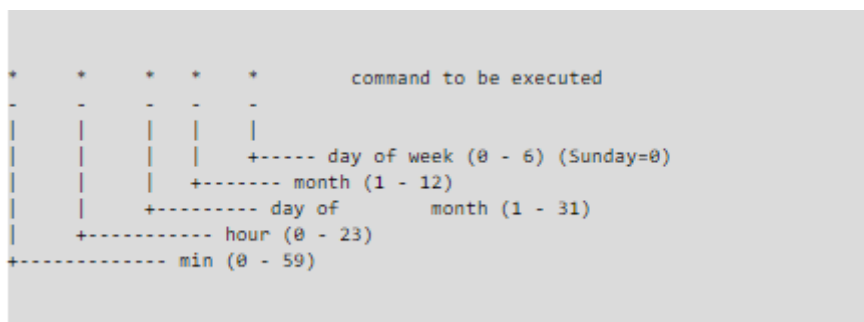
3.7 Skript för dataöverföring

Programmet hämtar sin information från databasen varje gång man startar webbsidan, sätter på ett filter eller interagerar på något sätt med applikationen. Information som kommer från mätare är ny för varje dag och kunddata kan också ändras med ojämna mellanrum. Med andra ord behöver det finnas ny information för varje dag man startar programmet.

Problemets utgångspunkt var att på valfritt sätt hämta data från fyra olika Oracle-databaser och sätta in endast den viktiga information som programmet behöver regelbundet varenda dag. På grund av de tekniker som valts och vad som skulle fungera bäst med dem valdes PHP-programmering. Tanken var att skapa ett PHP-skript som

körs varje dag regelbundet. Databaserna fanns på en Linux-maskin i företaget. Lösningen på problemet var Crontab.

Crontab är en fil som innehåller ett schema av cron-arbeten som körs vid tidpunkter utsatt av ägaren till filen. I exemplet i figur 27 kan man se möjligheterna för en rad i crontab.



Figur 26. Exempel på en rad i crontab, (* betyder att man använder alla värdena för positionen). [23]

Med hjälp av crontab skapades en rutin som körs varje dag tidigt på morgonen innan användarna hinner ha behov av programmet. Det som kvarstod var koden som skulle utföra arbetet.

Skriptet som skapades bestod av flera filer. Det är inte nödvändigt att sprida ut filerna, men det är mycket lättare och tydligare för nästa person som tar över koden, om alla filer har hand om en sak. En databasfil skapades som styrde vart kopplingarna skulle föras samt hade funktioner för olika saker man vill utföra med ett databasobjekt, exempelvis hämta, förbereda och exekvera.

För att logga in i olika system måste man naturligtvis ha den rätta inloggningsinformationen. Ett logiskt ställe att ha det skulle kunna vara just i databasfilen; den sköter ju kopplingarna redan. Men som nämnts tidigare ville man ha en separat fil som styr olika parametrar i programmet samt detaljerna till inloggning. All sådan information sattes i en konfigurationsfil, som sedan hämtas med hjälp av PHP då det behövs. Orsaken till att man ville ha en konfigurationsfil är att det blir mycket lättare för andra användare, speciellt för sådana utan programmeringskunskaper, att snabbt och enkelt ändra hur programmet fungerar eller hur man får åtkomst till det, utan att behöva veta någonting om vad som händer i bakgrunden.

Till varje enskilt system skapades filer som hämtar data med olika SQL-förfrågningar och -funktioner. Skriptet hämtar, förbereder och exekverar data från Oracle databaser

och sparar den i en variabel som sedan gör samma sak. Istället för en hämtning sattes data in i MySQL databasen med hjälp av en for-sats. Orsaken till att det måste förberedas två gånger är att det är två olika databastyper som är med i ekvationen, Oracle och MySQL, som gör att det krävs lite annorlunda kod för hanteringen av de olika typerna. En överblick av databaser och skript finns i figur 2, kapitel 1.3.

En fil som hanterar en viktig del av programmet fungerade lite annorlunda. Filen hade som uppgift att hämta läsningar som mätare gjort de senaste 30 dagarna. När man först ser på problemet verkar det enkelt; pseudo-kod kunde tänkas vara i stil med "SELECT last 30 days FROM place". Problemet är att alla, eller de flesta, enskilda mätare har många olika rader av data som måste hämtas en i gången. Med andra ord kunde det finnas 200 000 rader med data för varje dag från mätarna och det behövs för 30 dagar. Det handlar alltså om miljontals rader där också själva SQL-förfrågan måste vara så pass enkel, men specifik, att det inte tar hela dagen att filtrera och hämta informationen.

Efter att själva SQL-förfrågan var på plats behövdes det också sättas in i databasen. Problemet är att om man sätter in rad för rad, miljoner rader, kommer det att ta väldigt länge. Det tog uppemot tre timmar att göra det på det här sättet. I och för sig kunde skriptet köras tidigt på morgonen, men då blir problemet att vissa mätare kanske inte ännu har skickat sin information och det som visas i programmet blir ofullständigt.

Man vill alltså att skriptet skall köras så sent som möjligt, och då måste det vara snabbt för att inte vara otillgängligt för användarna då programmet behövs. För att få ner tiden det tar att köra måste raderna sättas in som bulk, det vill säga så många rader som möjligt per gång istället för en rad i taget. Även detta tog flera timmar. Lyckligtvis fanns en metod som gör att man kan sätta in data i en databas upp till 20 gånger snabbare än andra metoder. Metoden heter "LOAD DATA INFILE" och sätter data från en fil till en databas snabbare än någon annan metod, med vissa undantag som inte är aktuella i detta projekt.

```
$bulk_query = "LOAD DATA LOW_PRIORITY LOCAL INFILE '$this->file_path./bulk.csv'  
  IGNORE INTO TABLE `EXAMPLE`.`PLACE`  
  FIELDS TERMINATED BY ','  
  OPTIONALLY ENCLOSED BY '\"' |  
  LINES TERMINATED BY '\n' ";
```

Figur 27. Den förvrängda koden som kör in en "bulk.csv"-fil till rätt tabell.

Lösningen var alltså att hämta informationen, sätta den in i en fil och sedan köra in hela filen med "LOAD DATA INFILE" till databasen. Denna metod tog ner tiden från ca 3 timmar till ca 15 minuter, och valdes för användning omedelbart efter att koden testats ett tiotal gånger i olika situationer för att se om den faktiskt fungerade varje körning.

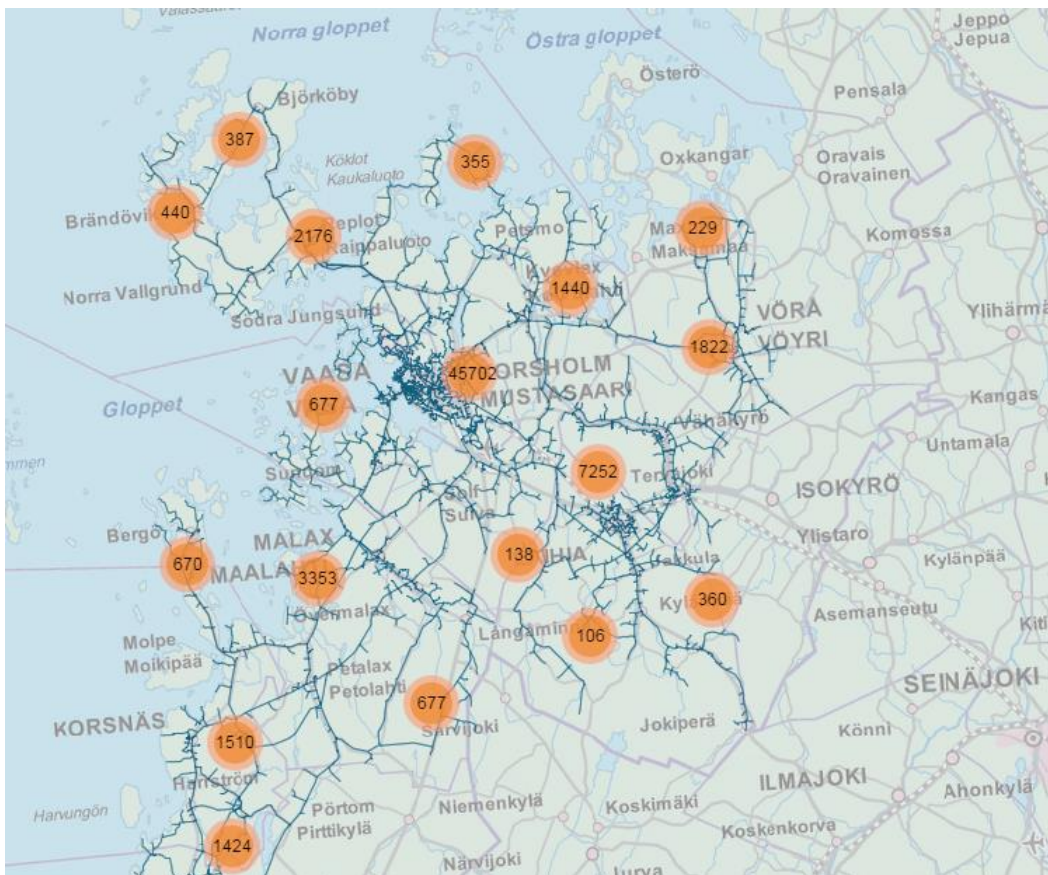
3.8 Markörer

Varje markör eller förbrukningsplats har information om kunden, platsen, mätaren, kopplingsätt och en kopplingskedja, om en sådan finns för platsen, samt eventuella IP-adresser.



Figur 28. Markör-ikon som används i programmet.

Som tidigare nämnts finns relativt många förbrukningsplatser. När de först laddades in på kartan lades det till ca 69 000 punkter, väldigt tätt packade. Utöver det kraschade programmet ibland, eftersom det var för mycket att rita upp på en gång. Lyckligtvis fanns det lösningar på det problemet. Det finns metoder för att slå ihop markere till grupper, så att istället för exempelvis 7000 punkter ovanpå eller nära varandra ser man ett par små grupper med kanske 200-1000 markere i varje. Det som är mycket bra med denna lösning, som är ett Leaflet-tillägg med namnet "Markercluster", är att när man zoomar in eller ut på kartan skingras dessa grupper eller slår sig samman. Vid maximal zoom är det bara de punkter som är exakt ovanpå varandra som förblir grupperade. Klickar man på en sådan grupp ser man alla markörer utplacerade i en spiral och kan klicka på enskilda markörer.



Figur 29. Exempel på Markercluster-tillägget.

När en användare klickar på en markör dyker ett popup-fönster upp med ovannämnd information. Då man väl har upp fönstret kan användaren även klicka på en knapp för att få de senaste 30 dagarnas mätningar för platsen. När man klickar på mätningsknappen körs en AJAX-rutin. Förbrukningsplats-id skickas iväg till en PHP-fil där de används för att hämta de senaste 30 dagarnas mätningar och skickar dem tillbaka som ett JSON-objekt. Utifrån informationen som returneras ritas trafikljus upp i popup-fönstret med datum för att visuellt visa en mätares status.

Anledningen till att mätningarna är styrda med en knapp istället för automatiskt, är att användarna inte behöver se mätningarna varenda gång. Ofta behöver användarna endast se vilken kund som hör till en förbrukningsplats.



Figur 30. Trafikljus-ikon som används i programmet.

Om en mätning finns från upp till tre dagar tillbaks visas grönt ljus. Har ingen mätning kommit från uppemot fem dagar visas gult, och allting över fem dagar visas som rött. Att det är tre dagar för grönt bestämdes bara enligt vad som just då tycktes vara

ändamålsenligt. Eftersom detta antagligen kommer att ändras i framtiden och för att användaren, som inte alls vill se kod, kanske vill ändra för testningssyften, styrs detta också genom en konfigurationsfil.

I figur 31 ser man en "switch case" som utifrån variabeln "day_difference" väljer vilket trafikljus som skall visas för dagen. Koden körs i en for-sats endast en gång för varje markör. PHP-koden som visas tolkas i raden som en siffra som hämtas från konfigurationsfilen.

```
switch(true){
  case(day_difference < <?php echo $config["green"];?>):
    marker._popup.setContent(marker._popup.getContent() + "<p>" + obj[i].consumption_time + "</p><img border='0' src=script/images/green-icon.png>");
    break;

  case(day_difference >= <?php echo $config["yellow_start"];?> && day_difference < <?php echo $config["yellow_end"];?>):
    marker._popup.setContent(marker._popup.getContent() + "<p>" + obj[i].consumption_time + "</p><img border='0' src=script/images/yellow-icon.png>");
    break;

  case(day_difference >= <?php echo $config["red"];?>):
    marker._popup.setContent(marker._popup.getContent() + "<p>" + obj[i].consumption_time + "</p><img border='0' src=script/images/red-icon.png>");
    break;

  default:
    marker._popup.setContent(marker._popup.getContent() + "<br><img border='0' src=script/images/traffic-icon.png>");
}
```

Figur 31. Kodurklipp för trafikljus.

3.9 Tillägg till kartverket

Då projektet började närma sig sitt slut sattes fokus på fler olika funktioner i programmet. Alla av dem var inte nödvändiga för att projektet skulle vara lyckat, men de underlättade och gjorde programmet mer interaktivt. Den viktigaste funktionen som utvecklades var att man skulle kunna markera flera förbrukningsplatser och utifrån det få en lista eller en rapport som snabbt visade och formaterade all väsentlig information inom området.

Det finns många tillägg i Leaflet som gör att en användare kan rita upp ett område. Det som valdes hette "Leaflet draw" och är ett av de populäraste. När man sätter till tillägget i Leaflet får man verktyg för att dra kvadrater, polygoner och linjer. Vad man sen gör med det är upp till programmeraren. Som basfunktion är det endast menat att visualisera viktiga områden på en karta, men man kan även använda informationen som hör till det ritade området för att skapa en rapport. När ett område skapas kan man göra en procedur som kör efter att användaren ritat färdigt.

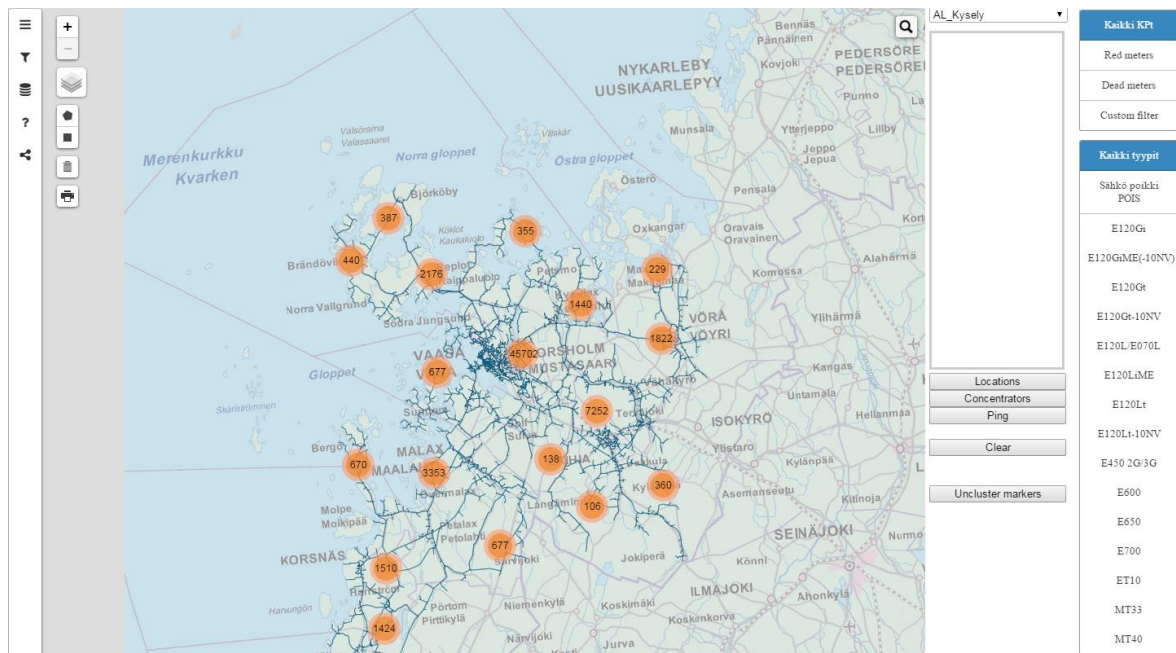
I proceduren jämfördes områdena med koordinaterna för de förbrukningsplatser som fanns på kartan för att se om de var inom området, och om de fanns skickades förbrukningsplats-id:n för att bli formaterade till en rapport sida där användaren kan välja att skriva ut resultatet.

```
for(var i = 0; i < mt_namespace.active_values.length; i++){  
  if (bounds.contains([mt_namespace.active_values[i]._coords])) {  
    inBounds.push(mt_namespace.active_values[i]._consumption_location);  
    $("#styled").get(0).value += mt_namespace.active_values[i]._consumption_location + "\n";  
  }  
}
```

Figur 32. Kod som jämför det ritade området med koordinaterna för aktiva markörer.

4 Resultat och diskussion

Resultatet var ett slutfört projekt och en webbsida som uppfyllde uppdragsgivarnas samtliga krav och mer därtill. Den slutliga koden överlämnades till mina förmän på företaget, som kommer att vidareutveckla programmet i framtiden. Programmet kommer egentligen aldrig att bli helt färdigt. Det kommer antagligen att finnas önskemål eller områden som kan vidareutvecklas.



Figur 33. Startsidan i programmet.

Jag är väldigt medveten om hur ovanligt det är för en person utan lång erfarenhet av programmering att få fria händer att skapa ett program, att få välja vilket språk man programmerar med, vilka metoder man använder och hur man löser problem. Jag är tacksam för att ha fått möjligheten att skapa ett sådant program för ett välkänt företag som Vasa Elnät, i en miljö som var stressfri, trevlig och professionell.

Under projektets gång har jag ifrågasatt mycket, gått tillbaka, redigerat och helt bytt ut delar av uppgiften för att lösa de problem som uppstått. Det fanns alltid möjlighet att begära hjälp av förmännen då jag hade frågor eller behövde få ett annat perspektiv på ett problem. Jag har lärt mig mycket om programmering, webbspråk, hur olika människor tolkar problem och hur saker hänger ihop under den period som jag arbetade i företaget. Speciellt JavaScript hade jag i början inte tänkt använda så mycket som jag slutligen gjorde. Eftersom det är ett språk som används mer och mer på webben är jag glad att jag fick lära mig mera om språket i en icke-konstgjord situation.

Alla krav på projektet som framställdes i början blev slutförda innan tiden var ute. Eftersom programutvecklingsmetodiken var "agile" lades fler funktioner till under programmets utveckling, vilket resulterade i att det slutliga programmet gjorde mera än vad jag hade förväntat mig då projektet startade.

Jag har förut från skolans sida varit med och programmerat lite åt företag och programmerat i grupp, men det här är första gången jag varit anställd av ett företag för att i stort sett själv programmera och utveckla ett projekt. Jag är mycket nöjd med resultatet och tacksam för möjligheten: Det har varit en viktig milstolpe för mig att slutföra detta projekt.

5 Källförteckning

- [1] Vasa elektriska. [Online]
<http://www.vaasansahko.fi/> [hämtat: 7.9.2015].
- [2] Oy RAVERA Ab. [Online]
<http://www.ravera.fi/> [hämtat: 7.9.2015].
- [3] Liemur software development, *Waterfall model*. [Online]
<http://liemur.com/wp-content/uploads/2013/04/Risk Based Software Development image004.jpg>
[hämtat: 13.9.2015].
- [4] Munassar, N., M., A. & Govardhan, A., 2010. *A Comparison between Five Models of Software Engineering*. [Online]
<http://www.ijcsi.org/papers/7-5-94-101.pdf> [hämtat: 13.9.2015].
- [5] Agile Alliance. [Online]
<http://www.agilealliance.org/> [hämtat: 13.9.2015].
- [6] Highsmith, J., 2001. *History: The Agile Manifesto*. Agile Alliance. [Online]
<http://agilemanifesto.org/history.html> [hämtat: 16.9.2015].
- [7] Serena Software, Inc., 2007. *An Introduction to Agile Software Development* [Online]
<http://www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf>
[hämtat: 17.9.2015].
- [8] Ambler, S., W., 2011. Agile Modeling (AM) Home Page. [Online]
<http://www.agilemodeling.com/> [hämtat: 17.9.2015].
- [9] Disciplined Agile Consortium, 2015. *Disciplined Agile 2.0*. [Online]
<http://www.disciplinedagiledelivery.com/> [hämtat: 17.9.2015].
- [10] The PHP group, 2015. *What can PHP do?* [Online]
<https://secure.php.net/manual/en/intro-whatcando.php> [hämtat: 19.9.2015].
- [11] HTML living standard. [Online]
<https://html.spec.whatwg.org/multipage/> [hämtat: 21.9.2015].
- [12] W3C, 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. [Online]
<http://www.w3.org/TR/REC-xml/> [hämtat: 22.9.2015].
- [13] Refsnes data, 2015. *AJAX Introduction*. W3Schools. [Online]
http://www.w3schools.com/ajax/ajax_intro.asp [hämtat: 23.9.2015].

- [14] Refsnes data, 2015. *How AJAX works*. W3Schools. [Online]
<http://www.w3schools.com/ajax/ajax.gif> [hämtat: 23.9.2015].
- [15] Oracle corporation, 2014. *Database SQL Language Reference*. [Online]
http://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm [hämtat: 24.9.2015].
- [16] Koch, P-P., 2006. *ppk on JavaScript*. New Riders.
- [17] The jQuery Foundation, 2015. *What is jQuery?* [Online]
<https://jquery.com/> [hämtat: 26.9.2015].
- [18] JSON. *Introducing JSON*. [Online]
<http://www.json.org/> [hämtat: 27.9.2015].
- [19] Refsnes data, 2015. *JSON Tutorial*. W3Schools. [Online]
<http://www.w3schools.com/json/> [hämtat: 27.9.2015].
- [20] OpenLayers History, 2015. [Online]
<http://trac.osgeo.org/openlayers/wiki/History> [hämtat: 25.10.2015].
- [21] Lovelace, R., 2014. *Testing web map APIs - Google vs OpenLayers vs Leaflet*. [Online]
<http://robinlovelace.net/software/2014/03/05/webmap-test.html> [hämtat: 25.10.2015].
- [22] SQL Power Group Inc., 2015. *SQL Power Architect*. [Online]
http://www.sqlpower.ca/page/architect_download_os [hämtat: 29.10.2015].
- [23] Sharma, H., 2014. *Crontab – Quick Reference*. [Online]
<http://www.adminschoice.com/crontab-quick-reference> [hämtat: 29.10.2015].
- [24] Simpson, K., 2015. *You Don't Know JS: Up & Going*. O'Reilly Media.