CHALMERS

Visuell skadeanalys i webbmiljö

Examensarbete inom Data- och Informationsteknik

CAROLINE KABAT
SIMON PLANHAGE

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Visuell skadeanalys i webbmiljö
CAROLINE KABAT
SIMON PLANHAGE

Examinator: Christer Carlsson

Examensarbete 2016

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik
Göteborg 2016

Abstract

The Swedish public transport company Skånetrafiken uses a system for planning, reporting and follow-up work. This system is developed by JETAS Quality Systems. Part of this system focuses on the reporting of vandalism that has occurred on Skånetrafikens property.
When using a system based on a large amount of data, it can become problematic for the end user to get a proper overview of all the information. This thesis is about developing a web application that visualizes and compiles the reported information for end users at Skånetrafiken. The visualization takes the form of a Single Page Application consisting of a map, graphs and a summary table.
Development occurs mainly in the front-end portion of a web application, where JavaScript, HTML and CSS is used. Part of the development is done in the back-end, using ASP.NET MVC and Microsoft IIS to maintain compatibility with other parts of the existing system.

This thesis report will describe the implementation of the developed web application. The result fulfills the purpose and aim of the project, and will be integrated into the existing system that Skånetrafiken uses. The thesis is written in Swedish.

Sammanfattning

Skånetrafiken använder idag ett system för planering, inrapportering och uppföljning av arbete.
Detta system är utvecklat av JETAS Quality Systems. En del av detta system fokuserar på inrapportering av skadegörelse som inträffat på Skånetrafikens egendom. Vid användning av ett system som baseras på en stor mängd data kan det bli problematiskt för slutanvändaren att få en översiktsbild av all information. Examensarbetet går ut på att utveckla en webbapplikation som visualiserar och sammanställer inrapporterad information för slutanvändare. Visualisering sker i form av en "Single Page Application", bestående av en karta, grafer och en summeringstabell. Utvecklandet av webbapplikationen sker främst i front-end, där JavaScript, HTML och CSS används. En del av utvecklingen sker i back-end, och använder ASP.NET MVC och Microsoft IIS, för att bibehålla kompabilitet med andra delar av det befintliga systemet.

Rapporten beskriver utvecklingen av webbapplikationen. Resultatet uppfyller examensarbetets syfte och mål, och kommer att implementeras i det system som används av Skånetrafiken.

Förord

Denna rapport behandlar ett examensarbete inom institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola. Projektet genomfördes hos JETAS Quality Systems.

Vi vill tacka David Gullmarsvik och Joel Steen Timle på JETAS för deras handledning och stöd under examensarbetet. Vi vill även tacka Joachim von Hacht som varit våran handledare på Chalmers.

Innehållsförteckning

1. Inledning

1.1 Bakgrund

JETAS Quality Systems AB är ett svenskt företag, baserat i Göteborg. Företaget har kunder både inom Sverige och internationellt. De flesta av kunderna finns inom branscherna fastighetsservice och tung industri.

JETAS främsta produkt, JetasMetoden, bygger på att komponenter i ett system har blivit utrustade med avläsningsbara koder. Exempelvis kan ett ventilationsdon (komponent) vara utrustat med en kod, vilken i sin tur är en del av ett större ventilationssystem (system). När ett arbete, t.ex. underhåll eller reparation, utförs på komponenten avläses koden och eventuell nödvändig information matas in i en handdator eller smartphone för att sedan skickas och sparas till databas. Detta leder till ett smidigare sätt att utföra inrapportering, analys och uppföljning av arbete.

Inom kollektivtrafiken läggs mycket stora summor på åtgärder till följd av skadegörelse. En behovsanalys, utvecklad av tvärvetenskapliga designbyrån BOID [1] under Chalmers Industriteknik [2], anser att det går att göra stora besparingar genom att kartlägga och analysera skadegörelsen för att tidigare kunna sätta in preventiva åtgärder. Målsättningen är att tydliggöra vilken typ av skadegörelse som förekommer, var de uppstår geografiskt och hur man på ett miljömässigt optimalt sätt åtgärdar dessa samt hur förändringsåtgärder skall mätas, analyseras och värderas.

Med behovsanalysen som grund håller JETAS på att utveckla en skräddarsydd variant av JetasMetoden, kallad Public Transport, som är beställd av Skånetrafiken.
Skånetrafiken är ett bolag som har hand om kollektivtrafiken i Skåne.

1.2 Syfte

Syftet med arbetet är att göra det möjligt för Skånetrafiken att grafiskt visualisera skadegörelse över tid, på en karta och i grafer, för att få en översiktlig bild av omfattningen.

1.3 Mål

Målet för projektet är att ta fram en webbaserad applikation för visualisering av skadegörelse enligt ovan.

1.4 Avgränsningar

Handledaren på plats kommer att implementera ett API som förser projektet med testdata som kan användas i utvecklingsfasen.

2. Metod

2.1 Metodik

JETAS utvecklingsteam använder en lättrörlig s. k. agil metod. Arbetssättet har sitt ursprung i Scrum-metodiken. Förslaget från JETAS är att projektet också följer samma metod. Det agila arbetet kommer att ske i veckovisa s.k. sprints , med sprintplanering på måndagar och sprint review på fredagar. Planeringen kommer att utgå från en backlog, vilken kommer att uppdateras löpande under arbetet. Processen kommer att visualiseras på en whiteboard och i Google Docs Kalkylark.

3. Teknisk bakgrund

Den här delen behandlar tekniker och principer relevanta för projektet.

3.1 Tekniker

JSON: JavaScript Object Notation
Textbaserat format för datautbyte. Syntaktiskt påminner det mycket om JavaScript-objekt. Formatet är språkoberoende [3].

JSONP (JSON with Padding)
Inbyggda säkerhetsåtgärder i de flesta webbläsare blockerar hämtning av JSON-data från servrar utanför den egna domänen. Användande av JSONP kringgår förbudet mot detta genom att omsluta JSON-datan i en funktion [4].

Scalable Vector Graphics (SVG)
Ett format för vektorbaserad grafik. Grafik hanteras som vektorer istället för punkter, vilket gör att grafiska element kan skalas om utan någon förlust i kvalitet [5].

AJAX: Asynchronous JavaScript and XML
En teknik för att skapa interaktivitet i webbapplikationer utan att sidan behöver laddas om.
Med hjälp av webbläsarens XMLHttpRequest API anropas webbservern, denna returnerar data, ofta i form av JSON. Anropet sker asynkront, vilket innebär att resten av webbapplikationen inte låses upp och behöver vänta på svar innan arbetet kan fortsätta [6].

Databindning
En teknik för att synkronisera ett grafiskt användargränssnitt med den underliggande datamodellen. Tekniken garanterar att den information som visas för användaren i vyn är konsistent med informationen i datamodellen [7] .

MVC: Model-View-Controller
Designmönster för mjukvaruutveckling av applikationer med grafiskt användargränssnitt. Applikationen delas in i tre distinkta komponenter, Model, View och Controller [8].

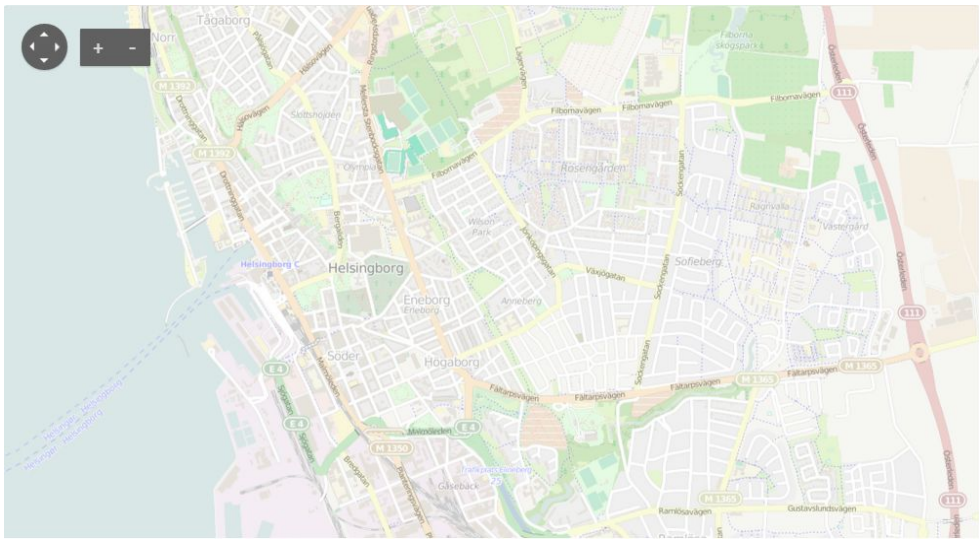
MVVM: Model-View-ViewModel
Designmönster som är utvecklat av Microsoft. Grundkonceptet med MVVM är att använda databindning för att synkronisera vy och modell [9]. MVVM-möstret består av tre komponenter: Model, View och ViewModel. Designmönstret har många likheter med MVC-mönstret, men Controllern har bytts ut mot ViewModel. ViewModel innehåller alltid en representation av vad som för tillfället visas i vyn, och hanterar logik för att behandla data så att informationen kan visas på önskat sätt i vyn. Komponenten tar även emot händelsenotifikationer från vyn.
View innehåller databindningar för att koppla vyn till ViewModel. Dessa databindningar kan även innehålla validering och formatering av data som ska visas eller hämtas av användaren. Då databindningarna måste deklareras i View-komponenten har MVVM-mönstret lite lägre separation

mellan komponenterna än MVC-mönstret. Detta på grund av att det krävs en viss kunskap om strukturen i ViewModel-komponenten vid deklarationen av databindningarna.

3.2 API:er

OpenStreetMap

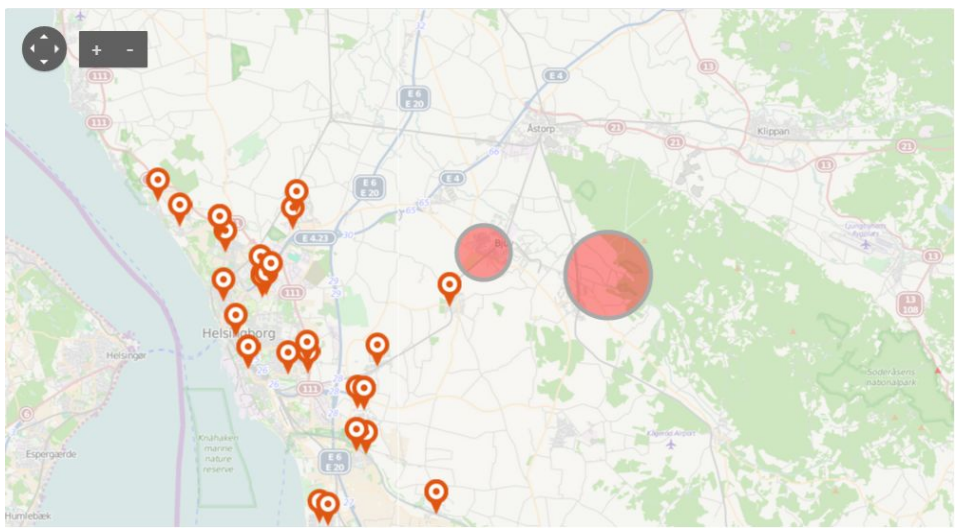
OpenStreetMap är ett öppet projekt för framtagning av kartor (figur 3.1) och annan geografisk information. Projektet bildades med målet att kostnadsfritt distribuera geografisk data, och finns tillgänlig genom ett API [10].



Figur 3.1. Exempel på karta utan ytterligare funktionalitet.

Telerik Kendo UI Map

Telerik Kendo UI Map är ett JavaScript-API som tillhandahåller kartfunktionalitet [11]. API:t bygger på att utvecklaren väljer ett underliggande kart-API, i vårt fall OpenStreetMap, API:t tillför ytterligare funktionalitet för arbete med kartor. Exempel är ritning och hantering av händelser för markörer (figur 3.2) och centrering av kartan.



Figur 3.2. Exempel på karta med markörer och cirklar, med Teleriks API

Google Maps Geocoding API
Google Maps Geocoding API tillhandahåller ett API för geokodningstjänst [12]. Genom ett AJAX-anrop kan ytterligare information om en adress eller koordinatpar hämtas.

Highcharts
Highcharts är ett API för att rita grafer i HTML5/JavaScript med hjälp av SVG-grafik [13].

3.3 Ramverk

Knockout.js
Knockout.js är ett JavaScript-ramverk på klient-sidan som implementerar databindning. Ramverket underlättar att applikationen struktureras enligt designmönstret MVVM [14].

Twitter Bootstrap
Bootstrap är ett CSS/JavaScript-ramverk utvecklat av Twitter, och har inbyggt stöd för responsiva komponenter för webbsidor [15]. Exempel på komponenter är dropdown-menyer, tabeller, knappar, navigationsmenyer och textinmatingsfält. Ett av ramverkets styrkor är att alla komponenter har ett enhetligt grafiskt utseende.

ASP.NET MVC
ASP.NET MVC är ett ramverk på server-sidan för att utveckla webbapplikationer enligt MVC-modellen. Ramverket är utvecklat av Microsoft.

3.4 Bibliotek

jQuery
JavaScript-bibliotek [16] som förenklar manipulation av DOM-trädet, hantering av händelser, AJAX-anrop och animeringar. Koden blir något färre rader och mer lättläst än med ren JavaScript.

Jämförelse (lista 3.1) av AJAX-anrop med jQuery och ren JavaScript [17][18]:

Med jQuery	Ren JavaScript

`$.ajax({`
`  type: 'GET',`
`  url: '/some/url',`
`  success: function(res) {`
`  // Hantera mottagen data`
`  },`
`  error: function() {`
`  // Hantera fel`
`  }`
`});`
	`var req = new XMLHttpRequest();`
	`req.open('GET', '/some/url', true);`
	`req.onload = function() {`
	`  if (req.status >= 200 && req.status < 400) {`
	`  // Mottaget resultat`
	`  var res = req.responseText;`
	`  } else {`
	`  // Mottaget fel`
	`  }`
	`};`
	`req.onerror = function() {`
	`  // Hantera fel`
	`};`
	`request.send();`

Lista 3.1

4. Genomförande

4.1 Kravspecifikation

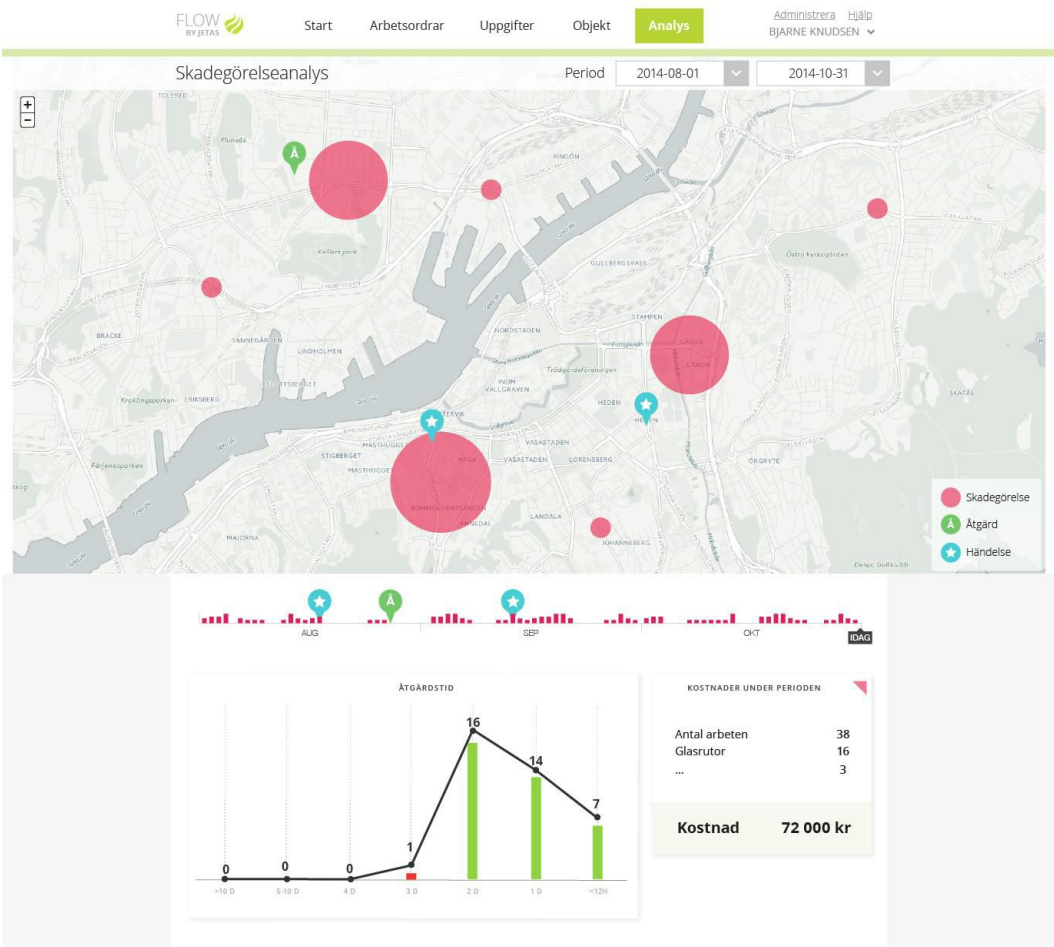
Utveckling
JETAS använder som standard Microsoftplattformen med ASP.NET MVC 5 och språket C#.
Front-end skrivs i JavaScript, HTML och CSS. Valet att använda Knockout.js som ramverk gjordes i samråd med handledare på JETAS. Detta på grund av eftersträvad enhetlighet, då andra befintliga Single-Page-Applikationer i systemet följer MVVM-designmönstret.

Verktyg
Vi har utvecklat i Visual Studio 2013, då utvecklingsmiljön har bra stöd för både C# samt JavaScript. Microsoft har utvecklat både Visual Studio 2013 och IIS, vilket ger strömlinjeformat stöd för uppladdning av applikationen till servern. Git används för versionshantering.

Användargränssnitt och funktionalitet
Som grund för hur JETAS applikation kan se ut finns en samling grafiska designförslag av användargränsnittet, framtagna av BOID. Dr Alexandros Nikitas vid Chalmers Tekniska Högskola har på JETAS beställning gjort en studie för att organisera och identifiera kundens krav för applikationens funktionalitet. Studien omfattar hela JETAS system och är samlad i en rapport.

Vi har valt att ha designförslaget (figur 4.1) från BOID som kravspecifikation för det grafiska användargränssnittet eftersom förslaget redan figurerat i samtal med Skånetrafiken, och responsen från dem har varit positiv.

Kravstudien av Dr Alexandros Nikitas är enligt oss inte tillräckligt specifierad kring vår del av applikationen samt är ej kopplad till det grafiska desingförslaget. Vi har dock valt att ta med studien som diskussionsunderlag för funktionaliteten och användarflödet i applikationen.



Figur 4.1. Designförslag framtaget av BOID

Baserat på designförslaget identifierades fyra delkomponenter i applikationen:

- filtreringsmeny för val av tidsperiod,
- karta med markering av platser,
- graf med statistik,
- panel med detaljerad information.

4.2 Analys

Användningsfall

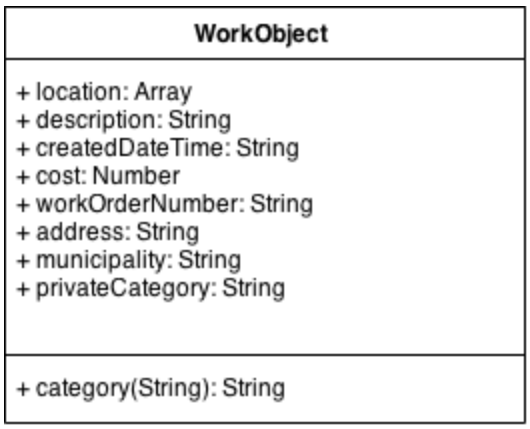
Mål: Användaren vill se jämförelse av skaderapporter över tid, under en viss period i en eller fler kommuner.

Aktivitet: Användaren ställer in tidsperiod och väljer en eller flera kommuner.

Resultat: Användaren ser en karta med markerade områden där skador rapporterats, en graf med detaljer om händelserna under tidsperioden, samt en informationspanel med en tabell som visar jämförelser och total kostnad.

Domänmodell

Från JETAS API hämtas data om skaderapporter som JETAS har i databasen. För vår del av applikationen behövs inte all information om varje skaderapport. Det som är relevant data från varje rapport används för att skapa nya objekt i applikationens modell. Som grund för modellen i applikationen har vi en klass kallad WorkObject (figur 4.2).



Figur 4.2. WorkObject - modell i JavaScript

Beskrivning av attributen

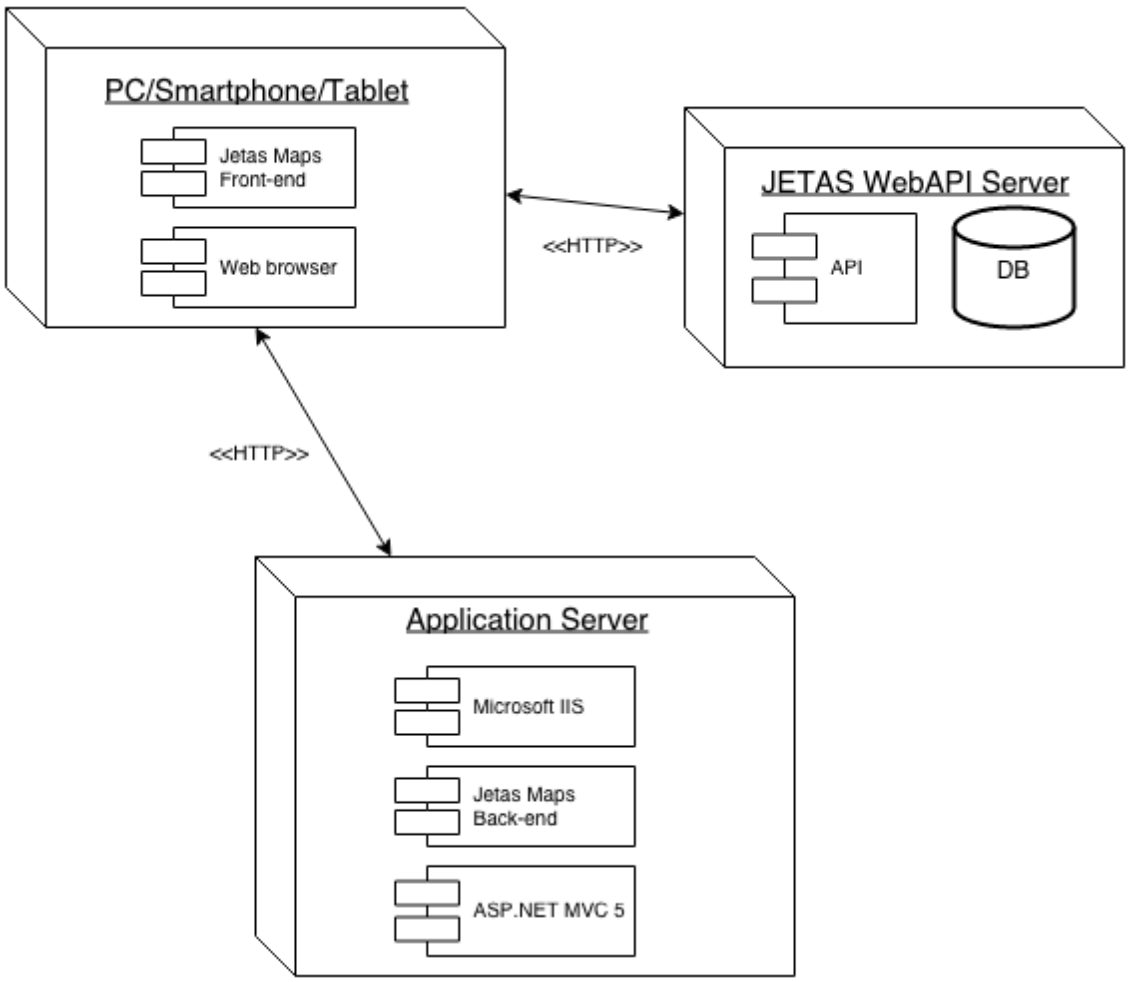
Location:	Innehåller latituden och longituden för GPS-positionen av skadan.
Description:	Beskrivning av skadan.
CreatedDateTime:	Tidpunkt som skadan rapporterades in.
Cost:	Kostnaden i SEK för åtgärdandet av skadan.
WorkOrderNumber:	Skaderapportens interna identifikationsnummer.
Address:	Gatuadressen som skadan skedde på.
Municipality:	Kommunen som skadan skedde i.
PrivateCategory:	Hjälpvariabel för parsning av kategori.
Category:	Metod som parsar kategori, “glasskross” och ”klotter”. Är kategorin tom alternativt inte passar in på glaskross eller klotter så sätts den till “övrigt”. Metoden hanterar felstavningar och varianter av kategorier som syftar till glaskross eller klotter.

4.3 Design

4.3.1 System

Systemet består av tre noder (figur 4.3):

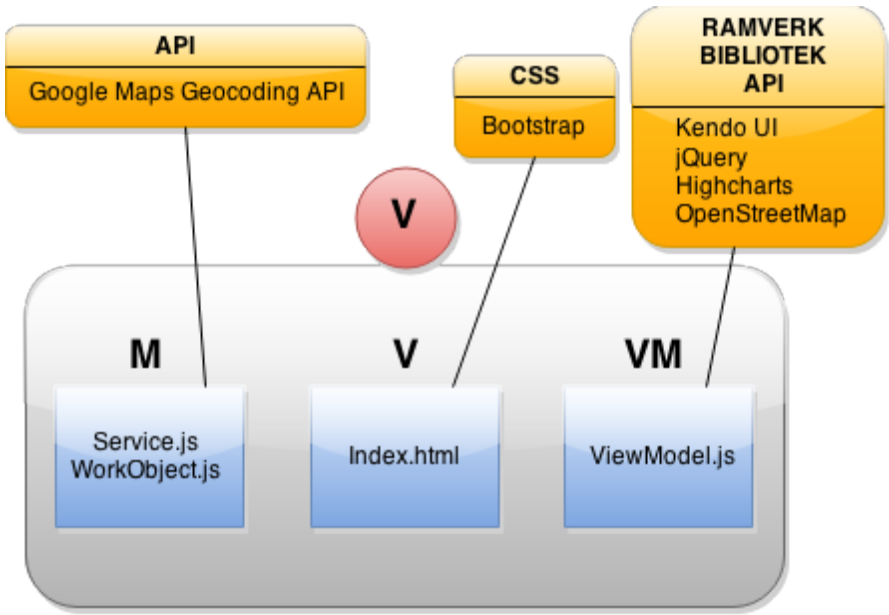
- Klientdel av vår applikation. Körs i webbläsare. Beskrivs vidare i kapitel 4.3.2.
- JETAS WebAPI Server. Fungerar som endpoint där all data relaterad till JETAS tjänster hämtas. Servern använder operativsystemet Windows Server 2008 R2. IIS hanterar mottagning av anropen och all data lagras i Microsoft SQL Server 2014. Data hämtas via AJAX-anrop.
- Serverdel av vår applikation. Körs i IIS, med ramverket ASP.NET MVC 5. Används som grund för att förenkla implementering av applikationen till den större delen av systemet. Är skriven i C#.



Figur 4.3. Deployment diagram

4.3.2 Klientnod

Vår del av applikationen befinner sig främst inom View:n, V:et i MVC. Applikationen använder Knockout.js som ramverk i front-end. Detta medför att strukturen inom vyn (figur 4.4) följer designmönstret MVVM, med komponenterna Model, View och ViewModel.



Figur 4.4. Överblick applikationsdesign MVVM

Model

Modellkomponenten har funktioner som hanterar AJAX-anrop till JETAS API. Data hämtas i form av JSONP och används för att skapa de JavaScript-objekt som används i vår domänmodell. Att lägga funktioner för AJAX-anrop i modellkomponenten följer Knockout.js rekommenderade struktur.

View-Model

Mellan modellen och vyn ligger en komponent, View-Model, för hantering av data och stadie i vyn. I komponenten sköts logiken som behövs för att presentera modellen i vyn. Komponenten är helt skriven i JavaScript.

View

Vyn är skriven i HTML och CSS. Vyn är en så kallad Single Page Application och består av en enda webbsida. Viss logik sker i vyn i form av databindningar. Se mer detaljer om databindningar i kapitel 4.5.2. I HTML-dokumentet för vyn sker även import av script för ramverk och API:er.

4.4 Användarintervju och kravdiskussion

För att få en mer klar bild av hur applikationen ska stämma överens för slutanvändarens syften gjordes det en intervju med representanter för Skånetrafiken. Frågorna i intervjun var främst kopplade till detaljer av hur exempelvis grafer och kartvy skulle kunna se ut och fungera tillsammans, då detta inte var tillräckligt specifierat i kravspecifikationen och designförslaget från BOID.

I och med att det fanns mycket data från JETAS att använda oss av ansågs det intressant att veta mer konkret vad som kunde vara nyttigt att visualisera i graf och informationspanel.
Från mötet tog vi med oss inhämtad kunskap angående:

- önskad detaljnivå vid visualisering,
- flöde vid användning av användargränssnittet.

4.5 Implementation

Realisering av användningsfall

Kommande avsnitt behandlar applikationens flöde vid realisering av användarfallet från kap 4.2.
Viss kod samt representering av data under körning visas.

4.5.1. Välja start- och slutdatum

Användaren väljer start- och slutdatum från respektive datumväljare. Dessa datumväljare från Kendo UI har initierats och konfigurerats i filen viewModel.js (lista 4.1). Vid val av datum körs funktionen change(). I funktionen sätts ett minimum/maximum-värde till respektive datumväljare. Detta så att man inte ska kunna välja ett slutdatum som ligger tidigare än startdatumet, eller ett startdatum som ligger senare än slutdatumet. Variabeln startDateString kommer alltid att innehålla det för nuvarande valda startdatumet. Detsamma gäller stopDateString för slutdatum. Slutligen ser vi om det finns valda kommuner i municipalityList, detta för att inte köra funktionsanrop som uppdaterar kartan om ingen kommun/kommuner har valts ännu.

viewModel.js

```
self.startTimeField = $("#start").kendoDatePicker({

    change: function () {
        var date = self.startTimeField.value().toLocaleDateString("sv");
        self.stopTimeField.min(date);
        self.startDateString = date;

        if (self.municipalityList.length > 0) {
            self.refreshMap();
        }
    }
}).data("kendoDatePicker");
```

Lista 4.1

4.5.2. Välja kommuner

Användaren väljer nu en eller flera kommuner från en dropdown-meny. Dropdown-menyn har vid initiering populerats med kommuner, som hämtats med hjälp av AJAX-anrop från JETAS API. Endast kommuner som har någon typ av skadegörelse visas i dropdown-menyn.
Nedan visas implementation av databindning mellan dropdown-menyn i HTML-dokumentet (lista 4.2) och listan med kommuner i ViewModel.js (lista 4.3).

Index.html

```
...
<ul data-bind="foreach: municipalityList">
    <li>
        <input type="checkbox" data-bind="checked: isSelected">
        <span data-bind="text: name"></span>
    </li>
</ul>
...
```

Lista 4.2

ViewModel.js

```
function ViewModel() {
    var self = this;
    self.municipalityList = ko.observableArray([
        {name: “Malmö”, isSelected: false},
        {name: “Helsingborg”, isSelected: false},
          … representering av data under körning av applikationen

    ]);
    self.selectedMunicipalities = ko.computed(function () {
        // Kollar vilka kommuner som är valda
        ...
        return selected;
    });
   ...
};

var vm = new ViewModel();
ko.applyBindings(vm);
```

Lista 4.3

I attributen “data-bind” deklareras bindningar mellan DOM-trädet och ramverket Knockout.js. I dessa attribut kan även uttryck definieras. Bindningen “foreach: municipalityList” säger till Knockout.js att skapa ett `<li>`-element för varje objekt i listan municipalityList. Uttrycket “checked: isSelected” kopplar checkbox-rutans status till ViewModel. Uttrycket “text: name” säger att det är fältet “name” från objekten i municipalityList som ska skrivas ut.

4.5.3. Hämtning av data

Relevant data hämtas från JETAS API. Detta görs med ett AJAX-anrop som skickas via HTTPS. Filtrering av efterfrågad data sker genom att använda en *query string* vid anropet. En query string är ett tillägg i webbadressen där man kan skicka information till servern. Detta görs i vårat fall för att undvika att begära mer information till klienten än vad som är nödvändigt. I query string skickas även en *token*, som identifierar användaren. Nedan följer ett exempel på en URI med en query string som innehåller efterfrågat start och slut-datum, samt kommun.

```
'https://api.jetas.se/api/workorders?Token=xxx&FromDate=2015-01-01T00:00:00&ToDate=2015-02-01T00:00:00&Municipality=Helsingborg'
```

Webapplikationen får svar i form av ett JSONP-objekt med arbetsordrar. Av varje arbetsorder i JSONP-objektet skapas ett WorkObject-objekt. Dessa objekt utgör modellen för applikationen.

4.5.4. Skapa modell

Objekten läggs i en lista med typen Observable Array. Den här typen av lista är central vid användande av Knockout.js, då det är den som står för databindning mellan ViewModel och View. Listans uppgift är att lagra alla de WorkObject-objekt som för närvarande visas i karta och graf. Detta enligt MVVM-mönstret, som säger att ViewModel ska vara konsistent med View.

service.js

```
   ...
function getWorkObjects(startDate, stopDate, municipalityList) {
    $.ajax({
        url: // url med query string skapas från parametrar,
        type: "GET",
        dataType: 'jsonp',
        cache: false,
        contentType: 'text/javascript',
        success: function (data) {

            $.each(data, function (key, item) {

                var o = new WorkObject(item.Latitude, item.Longitude, ... );
                mapViewModel.workObjectList.push(o);
            })
};
  ...
```

Lista 4.4

När användaren har valt start-, och slutdatum, samt en eller flera kommuner, hämtats data i filen service.js (lista 4.4) om skadegörelse inom valda intervall och kommuner till klienten från JETAS API.

4.5.5. Kartans lager

Ett funktionsanrop kallar på kart- och grafobjekt för att rita ut information i kart-, graf-, och informationspanel.
Applikationens karta består av tre lager. Det understa lagret består av OpenStreetMap. Detta lager ändras inte efter initiering av kartobjektet. Nästa lager består av de markörer som ritas ut i kartan. Det tredje och översta lagret innehåller bubblor. För att placera kartan vid vald kommun hämtas koordinater till kommunens centrum från Google Maps Geocoding API.

Funktionsanrop på kartobjektet talar om att ändringar har skett i ViewModel, och kartan ritar ut markörer på de platser som definieras i varje WorkObjects location-fält.

Sedan kalkyleras data på följande sätt (lista 4.5, lista 4.6):

1. Räknar ut antal dagar mellan start- och slutdatum, skapar en lista med detta antal som längd, och populerar alla fält i listan med initiell kostnad 0.
2. Skapar tre kopior av listan för att dela upp skadestatistik i olika kategorier.

chart.js

```
   ...
function calcGraphData (startDate, stopDate, municipalityName,
relevantWorkObjects) {

(1) var firstDate = new Date(startDate);
    var secondDate = new Date(stopDate);
    var diffDays = Math.ceil(Math.abs((firstDate.getTime() -
secondDate.getTime()) /    millisecondsInADay));

    var yAxisList = [];
    for (var i = 0; i < diffDays; i++) {
        yAxisList[i] = 0;
    }

(2) var klotterList = yAxisList.slice(0);
    var glaskrossList = yAxisList.slice(0);
    var annatList = yAxisList.slice(0);
```

Lista 4.5

3. Differens mellan sökt startdatum och aktuell workObjects datum räknas ut för att populera listor med kostnad på rätt fält.
4. Parsning av skadegörelsekategori för att placera kostnad i rätt lista.

chart.js

```
    for (var i = 0; i < relevantWorkObjects.length; i++) {
(3)     var searchedFirstDateMs = (new Date(startDate)).getTime();
        var currentWorkObjectDateMs = (new
Date(relevantWorkObjects[i].createdDateTime.substring(0, 10))).getTime();
```

```
        var yPos = (currentWorkObjectDateMs - searchedFirstDateMs) / msInADay

(4)     if (relevantWorkObjects[i].category().toLowerCase() === "glaskross") {
            glaskrossList[yPos] += relevantWorkObjects[i].cost;
        }
        else if (relevantWorkObjects[i].category().toLowerCase() === "klotter") {
             klotterList[yPos] += relevantWorkObjects[i].cost;
        } else {
            annatList[yPos] += relevantWorkObjects[i].cost;
        }
    }
```

Lista 4.6

De tre listorna används för att populera graferna med data, där varje position i listan motsvarar ett dygn.

chart.js

```
...
    $('#chartContainer').highcharts({
    ...
        xAxis: {
            type: 'datetime',
        ...
            }
        },
        series: [{
            name: "Glaskross",
            data: yAxisData[0],
            pointStart: dateObj.getTime(),
            pointInterval: 24 * 3600 * 1000
        }, {
            name: "Klotter",
            data: yAxisData[1],
            pointStart: dateObj.getTime(),
            pointInterval: 24 * 3600 * 1000
        }, {
            name: "Övrigt",
            data: yAxisData[2],
            pointStart: dateObj.getTime(),
            pointInterval: 24 * 3600 * 1000
        }]
...
```

Lista 4.7

Det viktigaste att notera i chart.js (lista 4.7) är följande:

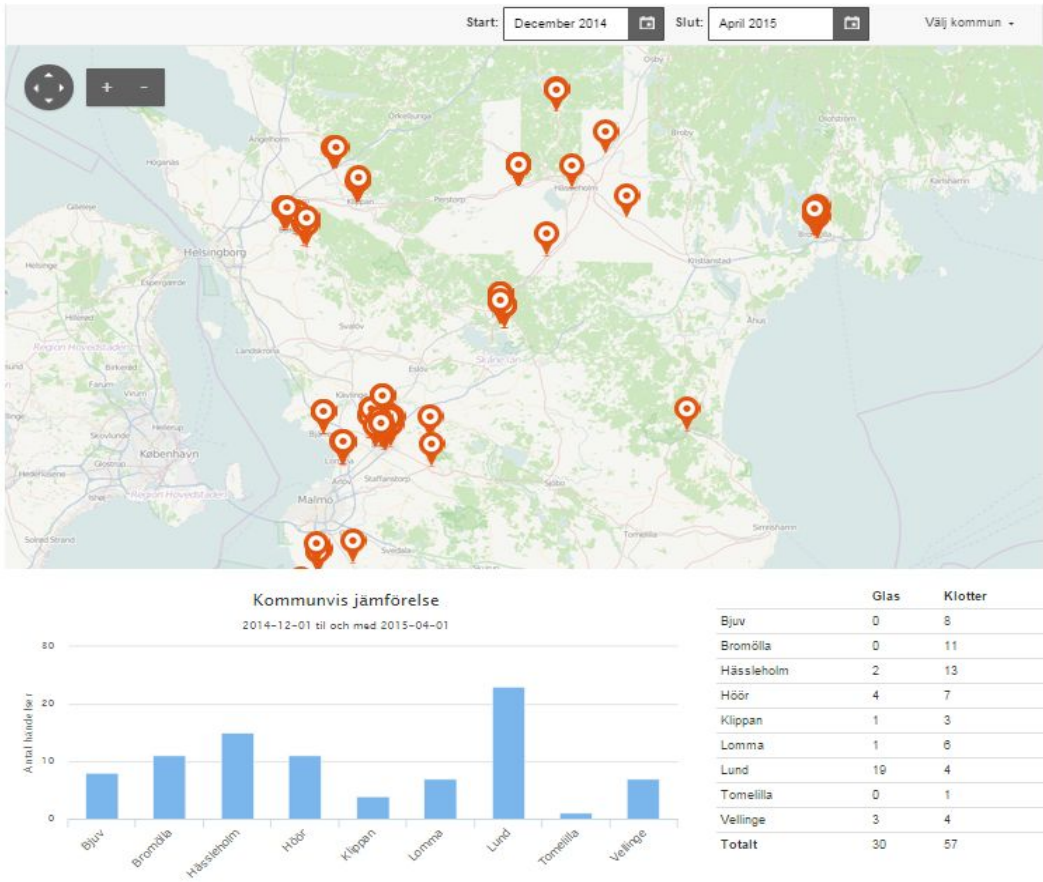
- xAxis har typen “datetime”. Detta gör att vi inte behöver utveckla speciallfall när man vill jämföra över månad- eller årsgränser, då Highcharts API tar hand om dessa fall.
- I series har vi tre objekt, ett för varje kategori av stapel. Detta för att kunna visa kategorier som olika färger i staplarna på y-axeln.

Uppdatering av grafen i DOM-trädet sker via metodanrop på grafelementet, som fås ut med hjälp av jQuery. Ytterligare användarfall realiseras på samma sätt, förutom att kalkyleringen av data till grafen skiljer sig åt beroende på graftyp.

5. Resultat

I figur 5.1 - 5.6 redovisas resultatet av applikationens utseende och funktion.

Överblick utseende (figur 5.1)
Överst: Filtreringsmeny och dropdown-meny
Mitten: Karta
Nederst: Graf och informationspanel

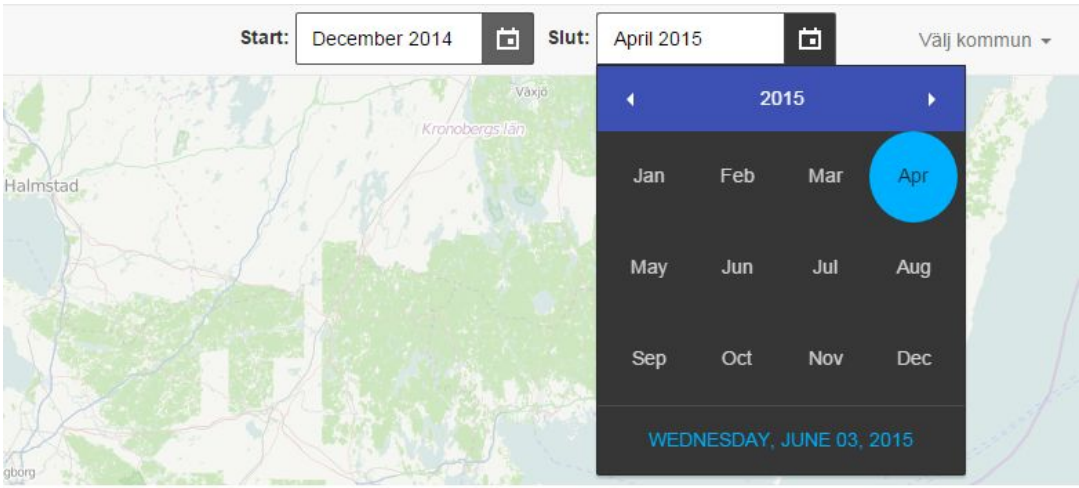


	Glas	Klotter
Bjuv	0	8
Bromölla	0	11
Hässleholm	2	13
Höör	4	7
Klippan	1	3
Lomma	1	6
Lund	19	4
Tomelilla	0	1
Vellinge	3	4
Totalt	30	57

Figur 5.1. Överblick applikationens utseende

Filtreringsmeny

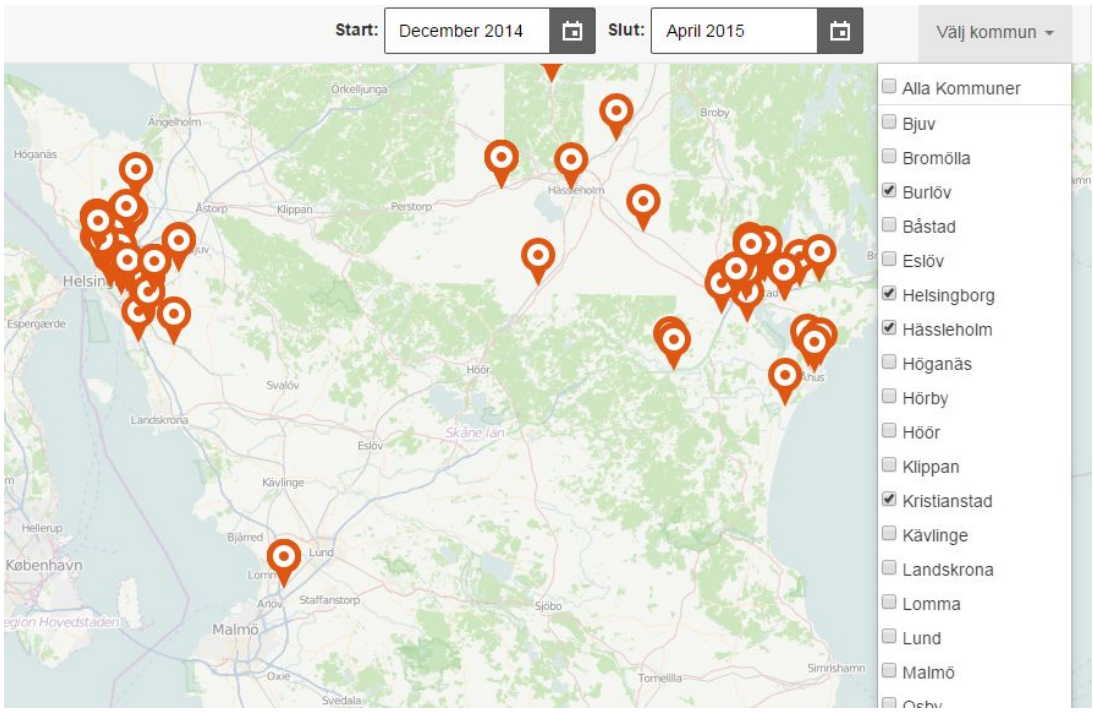
I filtreringsmenyn (figur 5.2) kan användaren välja start- och slutdatum att se skaderapporter från.



Figur 5.2. Val av start- och slutdatum

Karta och dropdown-meny

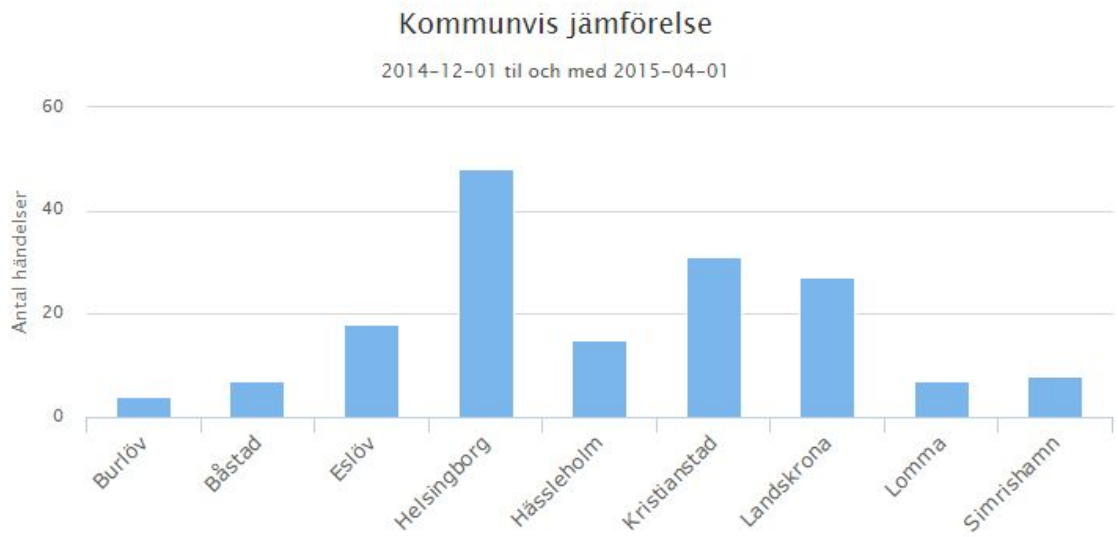
I kartan (figur 5.3) visas markörer för varje skaderapport inom den tidsperiod och i de kommuner som användaren valt. Valet av kommuner sker från en dropdown-meny (figur 5.3). Det går att välja en, flera eller alla kommuner från dropdown-menyn.



Figur 5.3. Karta och dropdown-meny

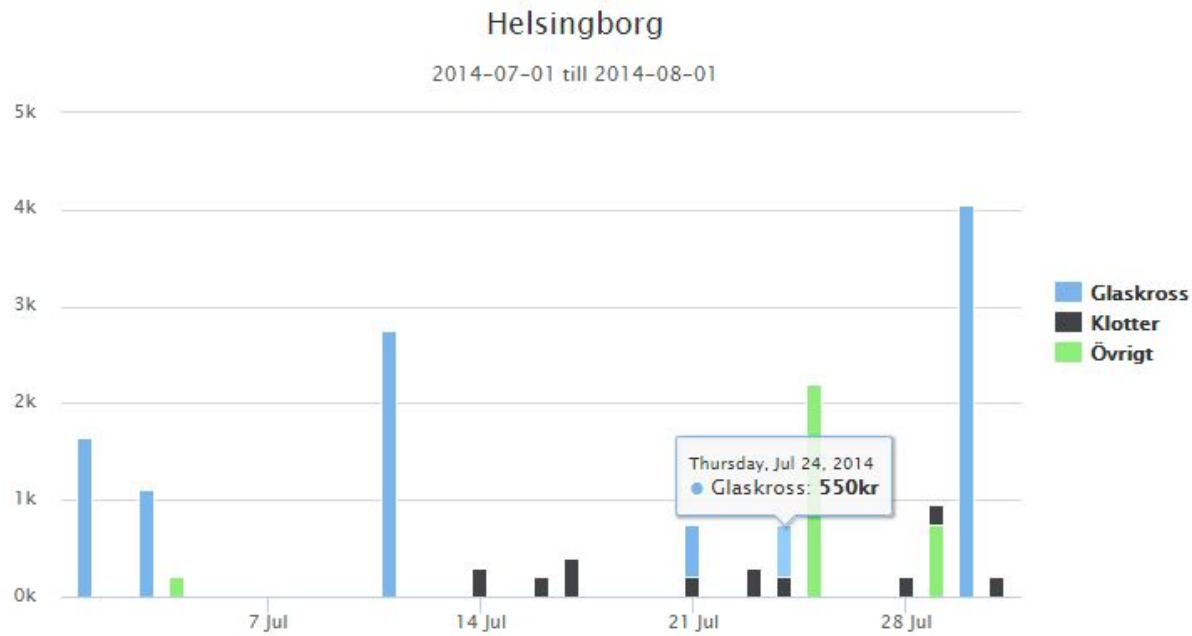
Grafer

Två typer av grafer är implementerade. Den ena grafen (figur 5.4) visar kommunvis antalet skador som rapporterats in mellan de start- och slutdatum och i de kommuner som användaren har ställt in.



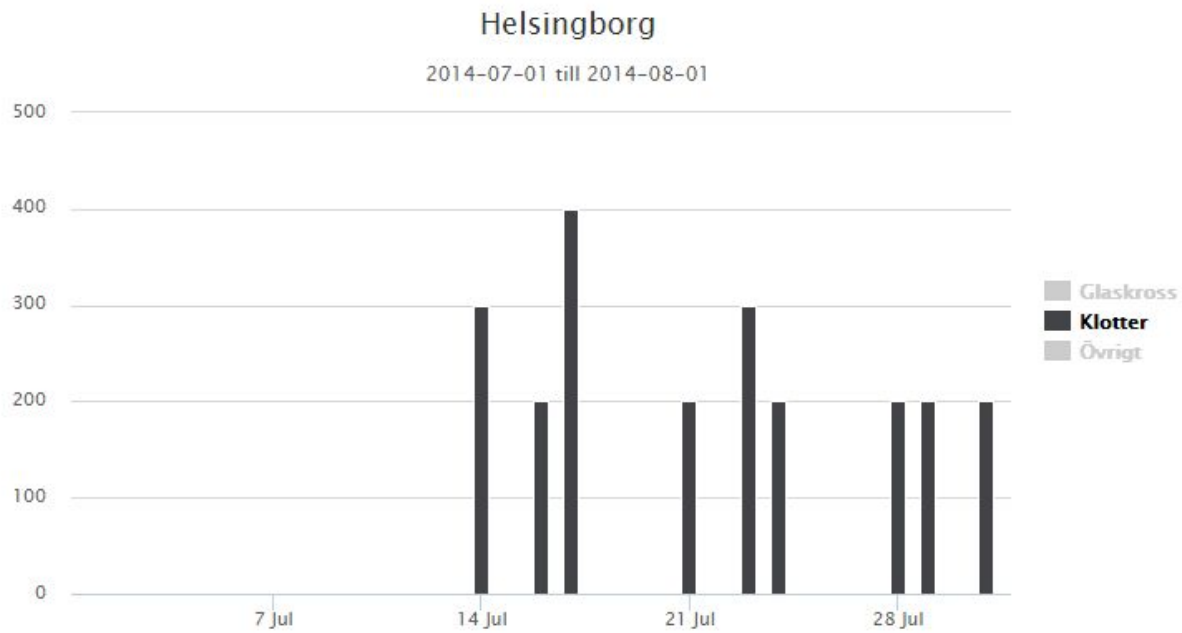
Figur 5.4. Antal skaderapporter i valda kommuner

Den andra grafen (figur 5.5) visar dagsvis summering av kostnaderna för åtgärdandet av skadorna i en kommun. Färgerna på staplarna visar vilken kategori skadorna har. För att se detaljerad information kan användaren hålla muspekaren ovanför en viss stapel. Då visas en inforuta med datum då skadorna rapporterades, kategori, samt kostnaden för åtgärdandet av skadorna.



Figur 5.5. Kostnader för åtgärder av skador, dagsvis i en kommun

För att endast se kostnaderna för exempelvis klotter kan användaren avmarkera de kategorier som användaren inte vill se (figur 5.6). Grafen animeras då om och visar bara staplar med relevanta kategorier.



Figur 5.6. Samma graf som i figur 5.5, filtrerat på skaderapporter om klotter

Informationspanel

Panelen innehåller en tabell (figur 5.7) som visar summering av skaderapporternas kategorier för varje kommun under vald tidsperiod. Uppdatering av tabellen sker automatiskt när användaren väljer till kommuner från dropdown-menyn.

	Glas	Klotter
Burlöv	3	1
Båstad	0	7
Eslöv	3	15
Helsingborg	13	35
Hässleholm	2	13
Kristianstad	5	26
Landskrona	8	19
Lomma	1	6
Simrishamn	0	8
Totalt	35	130

Figur 5.7. Panel med summering skaderapporter av glas/klotter i en tabell

6. Slutsats, diskussion och vidareutveckling

Resultatet stämmer överens med syftet och målet för projektet. Vi har ut utvecklat en webbapplikation ger en visuell överblick över insamlad data om skadegörelse.

I och med att projektet genomfördes under en begränsad tid valde vi att fokusera på, enligt oss, den mest grundläggande funktionaliteten och komponenterna i applikationen. Det grafiska designförlsaget från BOID gav en tydlig överblick av hur applikationen kunde se ut och var enkel att följa och efterlikna.
Mest intressant var undersökandet av hur funktionaliteten i applikationen skulle vara, då den inte var helt specifierad från början. Att arbeta enligt SCRUM visade sig således passa bra för den här sortens projekt. Vi kunde ofta visa upp en fungerande applikation, även om inte alla komponenter alltid var sammankopplade alternativt delvis saknade funktionalitet. Att arbeta iterativt gjorde det möjligt att kunna undersöka olika funktionalitet och att snabbt ändra den efter diskussion med slutkund och med handledare på JETAS.

Det finns en del saker som vi tror går att förbättra samt idéer på mer funktionalitet att implementera. Vi har två olika typer av grafer implementerade, men det saknas möjlighet att växla mellan graferna beroende på vad användaren önskar se. En annan idé är att ha en funktion för att exportera vald filtrering till PDF eller excel, alternativt spara filtreringen och kunna använda den i query-string vid förfrågningar till JETAS API. Detta för att göra det enklare för användaren att dela med sig av kartvy och grafer som diskussionsmaterial vid möten med mera.

Implementation av datavisualiseringen kan på håll leda till en effektivare planering av arbete hos de anställda som ska åtgärda skadegörelsen. Detta speciellt hos arbetsledare och personalansvariga då man har en bättre översiktlig bild att motivera sin resursfördelning med. Fältteknikernas körsträckor kan effektiviseras och på så sätt minska bränsleförbrukning.

Referenser

[1] BOID - Tvärvetenskaplig designbyrå (2015)
URL **http://boid.se**

[2] Chalmers IndustriTeknik (2015)
URL **http://www.cit.chalmers.se**

[3] JSON Format - Hemsida (Maj 2015)
URL **http://json.org**

[4] Specifikation JSONP
URL **http://www.json-p.org**

[5] Specifikation SVG
URL **http://www.w3.org/TR/SVG/**

[6] Specifikation AJAX
URL **http://www.w3.org/TR/XMLHttpRequest/**

[7] Dokumentation Knockoutjs Data-Bindings
URL **http://knockoutjs.com/documentation/binding-syntax.html**

[8] Specifikation MVC
URL **https://msdn.microsoft.com/en-us/library/hh848246.aspx**

[9] Specifikation MVVM
URL **https://msdn.microsoft.com/en-us/library/ff649643.aspx**

[10] Open Street Map (2015)
URL **https://www.openstreetmap.org**

[11] Telerik Kendo UI - Hemsida och API Dokumentation (2015)
URL **http://www.telerik.com/kendo-ui**

[12] Google Maps Geocoding API - API Dokumentation (2015)
URL **https://developers.google.com/maps/documentation/geocoding**

[13] Highcharts - Hemsida och API Dokumentation (2015)
URL **http://www.highcharts.com**

[14] Knockout.js - Hemsida och API Dokumentation (2015)
URL **http://knockoutjs.com**

[15] Twitter Bootstrap - Hemsida och Dokumentation (2015)
URL **http://getbootstrap.com**

[16] jQuery - Hemsida och Dokumentation (2015)
URL **https://www.jquery.com**

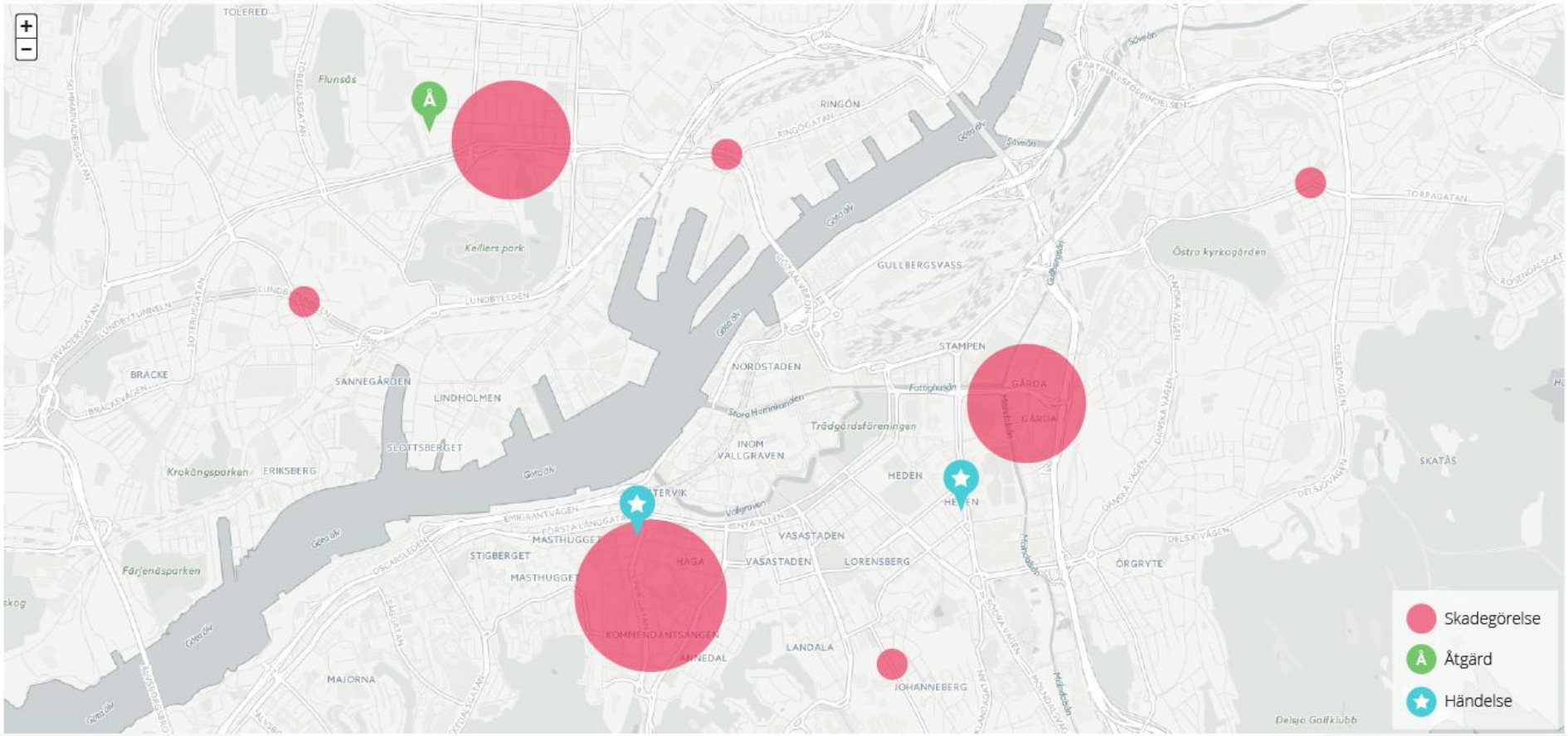
[17] N. C. Zakas. Professional JavaScript for Web Developers. John Wiley & Sons, Inc (2012)

[18] Jämförelse jQuery och JavaScript (Maj 2015)
URL **http://youmightnotneedjquery.com**

Appendix A.

Skadegörelseanalys

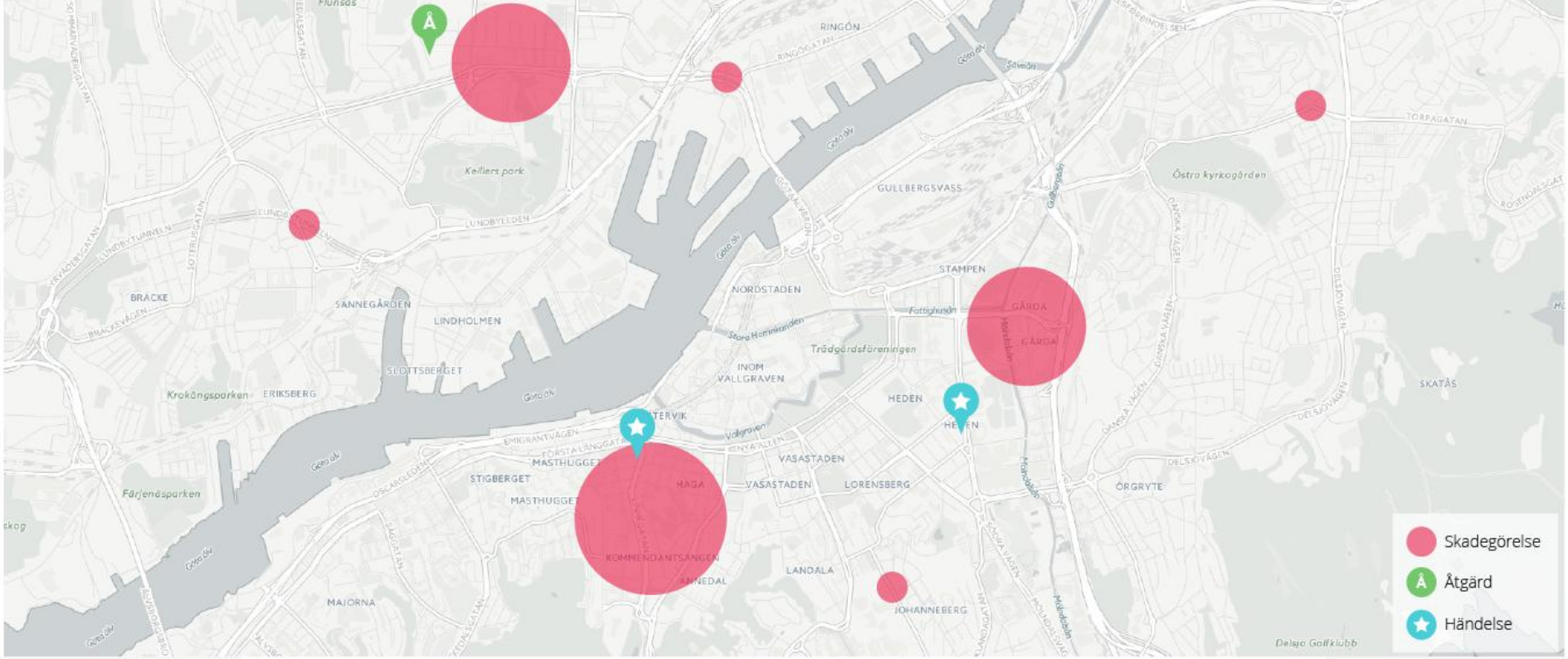
Period 2014-08-01 2014-10-31



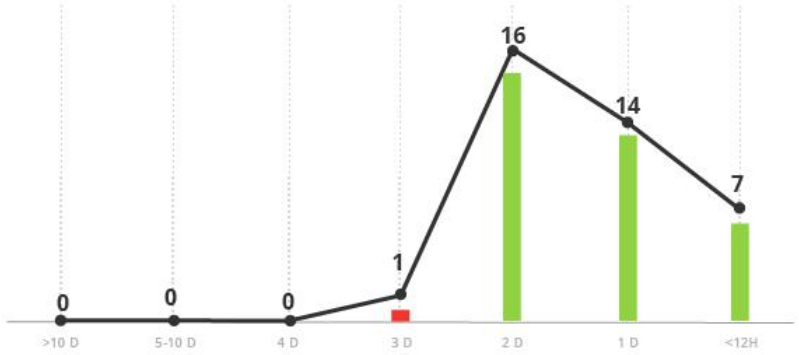
ÅTGÄRDSTID

KOSTNADER UNDER PERIODEN

Antal arbeten	38
Glasrutor	16
...	3



ÅTGÄRDSTID



KOSTNADER UNDER PERIODEN

Antal arbeten	38
Glasrutor	16
...	3

Kostnad **72 000 kr**