Dashboardlösning för EDI-system

Av

Simon Wictorsson

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Sammanfattning

Bland alla de avancerade system (bokningssystem, kundsystem, industrisystem) som utvecklas idag behövs det ett sätt att få en snabb överblick av systemet för att till exempel underlätta underhåll och övervakning. Detta löses ofta genom manuell sammanställning av data från systemet i program som Excel och görs oftast av en person vid olika tillfälle. Nackdelen med denna lösning är den inte är snabb och tar oftast lång tid att sammanställa.

En lösning till detta kommer då i form av en dashboard. Dashboardens uppgift är att sammanställa och visualisera data från systemet i realtid. Ett exempel är instrumentbrädan i en bil som visualiserar viktig data i form av hastighet m.m. till föraren. Dashboardens användare får då en snabb överblick över viktiga parametrar i systemet och blir snabbt informerad om någon parameter inte är som förväntat.

Tyringekonsult AB som ligger i skånska Tyringe behöver en lösning till detta för deras EDI-system. EDI är en metod för överförande av information enligt ett avtalat format. Målet var att ta fram en prototyp till en dashboard och undersöka vilken data från EDI-systemet som är viktigt att visualisera i dashboarden samt hur datan visualiseras på ett tydligt och bra sätt. Undersökningen av viktig data att visualisera gav tre parametrar från EDI-systemet:

- Belastning av CPU
- Användning av disk
- Antalet filer/dokument som systemet bearbetar

Till var och en av dessa tre parametrar togs det fram två till tre prototyper. Dessa prototyper bearbetades och de slutgiltiga prototyperna som implementerades valdes efter respons från de på Tyringekonsult som ska använda dashboarden.

Nyckelord: Dashboard, EDI-system, visualisering, Wicket, webbapplikation

Abstract

Among all the advanced systems (booking, customer, industrial) that are being developed today there is a need for a way to get a fast overview of the system. The solution for this is often through manual compilation of data from the system in programs such as Excel. This is often managed by one person on different occasions. This solution is not fast because the compilation is often time consuming.

One solution for this comes in the form of a dashboard. The dashboards task is to compile and visualize data from the system in real time. An example of this is the dashboard in a car that visualize the cars speed etc. The user of the dashboard then gets a fast overview of important parameters in the system and can then react fast if some parameters are out of the ordinary.

Tyringekonsult AB, who is based in Tyringe, need a dashboard solution for their EDI-system. EDI is a method for data transfer with an agreed format. The goal was to develop a dashboard prototype and research what data from the system is important to visualize and how that data is visualized in a good and clear way. The result was three important parameters from the system:

- CPU load
- Usage of disc
- The amount of processed files/documents in the system

Two to three prototypes were created each for the three parameters. These prototypes were processed and the final prototype implemented was selected after feedback from the ones that are going to use the dashboard at Tyringekonsult.

Keywords: Dashboard, EDI-system, visualization, Wicket, web application

Innehållsförteckning

1. Inledning

Kapitlets inledning ger en beskrivning av examensarbetets helhet genom att introducera bakgrund, syfte, mål, problemformulering och motivering.

1.1. Bakgrund

Examensarbetet utförs i samarbete med Tyringekonsult AB. Tyringekonsult levererar EDI-lösningar till företag inom bygg, logistik, handel, och bilindustri. EDI (Electronic Data Interchange) är en metod för att överföra data elektroniskt enligt ett avtalat format. Då många filer passerar genom Tyringekonsults system dagligen är det viktigt att snabbt kunna få överblick på status (överbelastat, hög CPU- & hårddiskanvändning) för deras EDI-system samt hårdvaran (servern) som det körs på. Examensarbetet har i uppgift att utveckla en dashboardsprototyp för övervakning/support av företagets system och hårdvara samt att ta fram hur detta ska presenteras i dashboarden. Den automatiska presentation av data som en dashboard tillhandahåller underlättar det manuella arbete som tidigare gjorts för att sammanställa data. Prototypens målgrupp är i första hand de på Tyringekonsult som jobbar som utvecklare och. Tyringekonsult har i nuläget ingen lösning för att få denna snabba överblick och man får därför lägga tid på att få fram datan manuellt.

1.2. Syfte

Syftet med examensarbetet att förenkla övervakning/support av Tyringekonsults EDI-system och den hårdvara som systemet körs på samt hur den data som används i övervakningen/supporten kan presenteras. Denna information kan sedan också appliceras på andra företag som arbetar med EDI-lösningar.

1.3. Målformulering

Målet med examensarbetet är en prototyp till en webbapplikation i form av en dashboard. I dashboarden kommer olika typer av data från Tyringekonsults system att visualiseras. Vilken data från

systemet är då viktig att visualisera och i vilken form är den data lämplig att visualiseras med.

1.4. Problemformulering

Genom målformuleringen kan då tre kategorier av frågor tas fram som presenteras nedan.

1.4.1. Data som ska visualiseras i dashboarden.

1.4.1.1 Vilken data är viktig att visualisera i dashboarden?

1.4.2. Visualisering av data.

1.4.2.1 Vilken form är mest lämplig för att visualisera datan (stapeldiagram, linjediagram, tårtdiagram m.m.)?

1.4.2.2 Hur visualiserar man realtidsdata tillsammans med historikdata?

1.4.2.3 Vilka/vilket ramverk/bibliotek lämpar sig bäst för visualiseringen?

1.4.3. Uppdatering av data.

1.4.3.1 Hur löser man att data uppdaterar sig i realtid istället för vid specifika tidpunkter eller intervall?

1.5. Motivering av examensarbete

Sökandet av examensarbete började i min hemtrakt runt Tyringe. Jag lyckades komma i kontakt med Tyringekonsult AB. Företaget i fråga har under en längre tid tänkt att ta fram en dashboard för att underlätta övervakning/support av sitt system. Eftersom företaget haft mycket annat att göra har dashboarden fått lägre prioritet. De valde då att erbjuda utvecklingen av dashboarden som ett examensarbete. Detta lät intressant och tillsammans med företaget bestämdes det att gå vidare med examensarbetet.

Tyringekonsult AB är inte det enda företaget som sysslar med EDI-lösningar. Hur olika företags EDI-system är uppbyggda kan

skilja sig men i grund och botten fungerar de ändå på ett liknande sätt när det kommer till at ta emot filer/dokument & översätta till avtalat format. Vilken data som är viktig och hur denna data ska visualiseras kan vara av allmänintresse eftersom det alltså skulle kunna appliceras på andra företags EDI-system.

1.6. Avgränsningar

Prototypen till dashboarden kommer att utvecklas för integration med Tyringekonsults nuvarande system och hårdvara.

2. Teknisk bakgrund

2.1. EDI

EDI (Electronic Data Interchange) är en metod för att överföra data elektroniskt enligt ett avtalat format. Det finns ett antal olika format för detta och de som Tyringekonsult stödjer är till exempel; XML, EDIFACT, VDA, Excel-filer, record-taggade filer och databasfiler. EDI kan användas internt inom ett företag eller levereras som en tjänst av ett annat företag. Filer och dokument som vanligen skickas mellan företag över en EDI-tjänst är till exempel, lagersaldo, ordrar, fakturor och fraktsedlar. [11]

Tyringekonsult har i huvudsak tre olika processer för deras EDI-syetem:

Den första processen kallas för routing av filer. Vid denna process tar Tyringekonsult emot en fil från en part över i stort sett vilken kommunikationsmetod som helst, till exempel FTP eller epost. När filen tagits emot uttolkas det till vem eller vilka filen ska vidarebefordras till. Filnamn, filens innehåll eller metadata till filen (del av kommunikationsprotokollet som använts) bestämmer detta.

Den andra processen är mer komplex och startar också med att en fil tas emot från en part av Tyringekonsult. Systemet tolkar filen och bestämmer hur den ska processas då olika typer av filer hanteras olika. För att filen ska kunna processas strukturerat ”packas den upp”. När filens mottagare och processmetod är identifierad delas den upp i dokument. Denna del av processen kallas för översättning. De skapade dokumenten läses av och de olika applikationer som Tyringekonsult har, uppdateras med den avlästa datan.

Den tredje processen liknar process två fast omvänt då filer istället ska skickas vidare till en part. Det som triggar denna process är antingen en utskrift (användare som ”beställer” sändning) eller en process från systemet själv. Dokument med den data som ska skickas skapas. När dokumentet skapats skall det paketeras själv eller med andra dokument till en fil. I vissa fall krypteras och signeras denna fil för att styrka innehåll och avsändare. När filen är skapad sänds den

till en eller flera mottagare på den specifika partens bestämda kommunikationsform.

2.2. Dashboard

En dashboards uppgift är att ge en snabb överblick över specifika parametrar i ett system. Ett tydligt exempel på en dashboard är instrumentpanelen i en bil. Här kan föraren snabbt få en överblick ocj information om fordonets hastighet, varvtal och bensinnivå m.m. Då detta ska kunna göras med en snabb överblick är det viktigt att inte visualisera datan alldeles för komplext.

S. Few skriver [12, s.13-14] att *”All That Glitters Is Not Gold”*. Han menar att dashboards är ett bra medel för att visualisera data men att det samtidigt kan vara svårt att göra detta då övervägande del av dashboards fallerar att visualisera data effektivt. Few skriver att en effektiv dashboard inte består av ”söta” [12, s.26] komponenter utan *”/.../ more science than art, more simplicity than dazzle. It is, above all else, about communication.”* [12, s.26] För att undvika detta beskriver Few också ett antal olika principer om dashboards.

- *”Dashboards display the information needed to achieve specific objectives”* [12, s.26]

Det är viktigt att förstå att data som visualiseras i en dashboard kan komma från många olika datakällor då det oftast behövs en samling information som tolkas tillsammans för att uppnå ett mål.

- *“Dashboards are used to monitor information at a glance”* [12, s.27]

Det är viktigt att dashboarden snabbt kan förmedla datan som visualiseras. Dashboardens funktion är att upplysa användaren om att det är något som behöver ens uppmärksamhet eller åtgärd och då inte hur detta ska åtgärdas.

- *“A dashboard fits on a single computer screen”* [12, s.27]

Data som visualiseras i en dashboard måste få plats på en enda sida. Behöver användaren aktivt bläddra i fönstret för att se all data har man inte längre en snabb överblick.

- *"Dashboards are customized"* [12, s.27]

Informationen i en dashboard måste skräddarsys efter de specifika användarna eller funktionen av dashboarden för att den ska tjäna sitt syfte.

S. Few [12, kapitel. 3] går visare in på vanliga misstag som görs i samband av dashboarddesign. Detta illustreras genom exempel på riktiga dashboards som inte fungerat tillfredsställande. Största delen av dessa misstag behandlar hur datan visualiseras på fel sätt. Några av dessa fel är; För många detaljer, dålig arrangering av datan, onödig fluff, och felanvändning eller för mycket användning av färg.

2.3. Visualisering

Visualisering syftar i detta fall på informationsgrafik och presentationen av data. Data kan presenteras på åtskilliga sätt men vid visualisering i en dashboard är det viktigt att datan presenteras tydligt och inte för komplext. Det är lätt att vara blind som designer och inte själv se att visualiseringen kan vara för komplex och otydlig. T. Munzer [24, kapitel 1] skriver om varför validering av själva visualiseringen är viktig men samtidigt svår att göra. Det som gör validering svår är alla frågor som uppkommer när visualiseringen är klar. Ett fåtal av dessa frågor som listas är: *"How do you know if it works? How do you argue that one design is better or worse than another for the intended users? For one thing, what does better mean"* [24, s. 14]. Sammanhanget som datan visualiseras i är mycket viktig. Vad är det för data som ska visualiseras och vad ska de avsedda användarna få ut av visualiseringen. En bra praxis är att visa upp design för de som ska använda det och se om de förstår vad som visualiseras utan att gå in på detaljer.

T. Munzer [24, kapitel 4] skriver också om fyra olika nivåer för validering av visualisering. Den översta nivån av dessa fyra är ”Domain situation”. Denna nivå beskriver själva sammanhanget som

datan ska visualiseras i. I detta examenarbetes fall är sammanhanget en dashboard för ett EDI-system. Viktiga saker att ta reda på är vad användarna vill ha ut av visualiseringen.

Den andra nivån är ”Data/task abstraction”. Denna nivå fungerar som en uppdelning av den information som samlats in i föregående nivå. Det är i denna nivå som vilken data som ska visualiseras bestäms och valideras.

Den tredje nivån är ”Visual encoding/interaction idiom”. Denna nivå går djupare in på den information som tagits fram i föregående nivå. Detta är då på vilket sätt datan ska visualiseras?

Den innersta och sista nivån är ”Algoritm”. Denna nivå behandlar val av implementation för visualiseringen.

Andra sätt att visualisera data som inte har med en dashboard att göra är att bara visualisera en typ av data för sig på en sida. En dashboard ger en snabb överblick men den ger inte en inblick över alla specifika detaljer. Hade varje del av dashboarden haft hur mycket detaljer som helst hade det varit svårt att få just den snabba överblick som man är ute efter.

S. Few [12, kapitel 4] lyfter fram information om visuell uppfattning, hur det undermedvetna behandlar vad ögonen ser innan det uppfattas av en själv. Betydelsen av att visualisera de viktigaste delarna av datan att det står ut. Detta görs till stor del av annat val på färg, position, form eller rörelse på den viktigaste datan.

Beroende på vad man vill få ut av visualiseringen av data kan samma mängd data visualiseras på olika sätt. S. Few [12, kapitel. 6] går igenom olika sätt att visualisera. Först ska det bestämmas om datan ska visualiseras med hjälp av ren text eller grafiskt. Efter detta är bestämt är det lättare att gå in på exakt hur det ska visualiseras. Few går igenom de vanligaste sätten att visualisera data och visar hur samma mängd data visualiseras olika. Vanliga grafiska sätt att visualisera är histogram, linjediagram, cirkeldiagram, punktdiagram m.m. Alla dessa vanliga typer är bra då de är väl etablerade i den mening att många har sett dessa grafer innan och vet hur de ska tydas.

A. Kirk behandlar [23] en mängd vanliga grafer och i vilka situationer dessa grafer passar i beroende på datan som ska visualiseras. Han delar upp graferna i fem olika kategorier:

Jämförande, hierarkiska, förändringar över tid, relationer och geografiska.

Stapeldiagram

Stapeldiagram används för att visualisera jämförelse mellan olika kategorier av data. Jämförelsen av kategorierna görs med höjden eller längden på staplarna i diagrammet beroende på om de sträcker sig vertikalt eller horisontellt. Ett stapeldiagram kan också fungera likt ett linjediagram där x-axeln med staplarna är tidsintervall och y-axeln mängden. Figur 1 visar exempel på ett stapeldiagram.

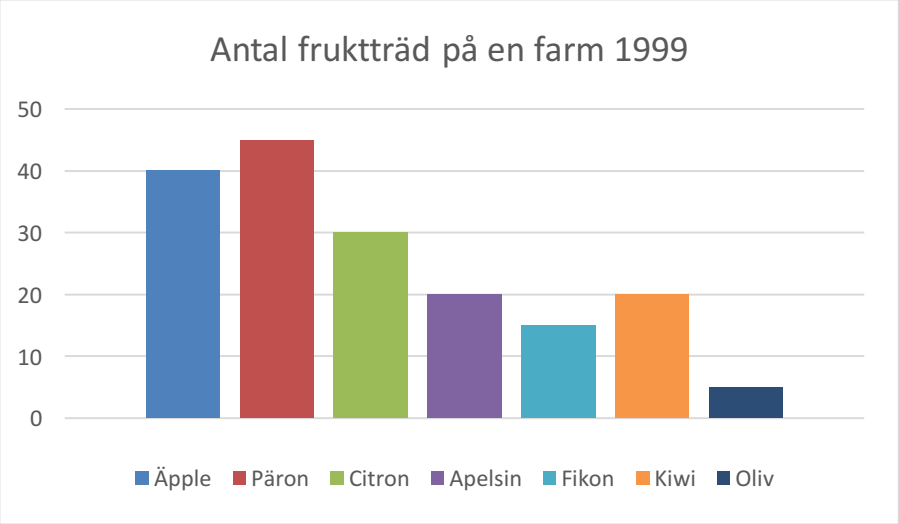

Fig. 1. Stapeldiagram (exempel)

Tårtdiagram

Tårtdiagram används ungefär på samma sätt som ett stapeldiagram. Diagrammets uppgift är att visualisera jämförelse mellan olika kategorier men att istället förmedla detta genom kategoriernas procentuella andel av totalen. Figur 2 visar exempel på ett tårtdiagram med samma data som figur 1.

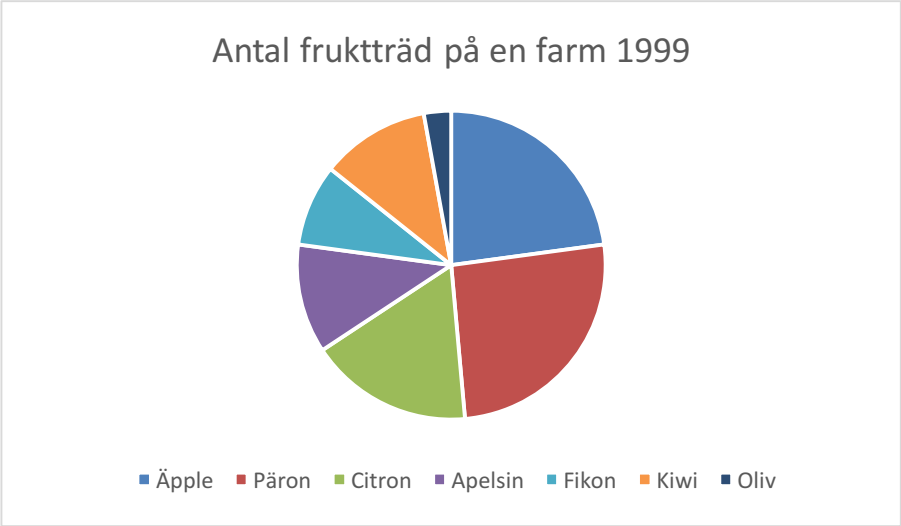

Fig. 2. Tårtdiagram (exempel)

Linjediagram

Linjediagram används för att visualisera kontinuerlig data. X-axeln används för de kontinuerliga variablerna (t.ex. timmar, dagar, månader, år) och y-axeln för storleken (t.ex. antal eller procent). Mellan datapunkterna i diagrammet dras sedan linjer för att visa hur datan ändras över tiden. Jämfört med stapeldiagram behöver ett linjediagram inte visa hela omfånget av värdena (starta från värdet noll) på y-axeln. Flera kontinuerliga datamängder går också att visualiseras tillsammans i diagrammet. Figur 3 visar exempel på ett linjediagram.

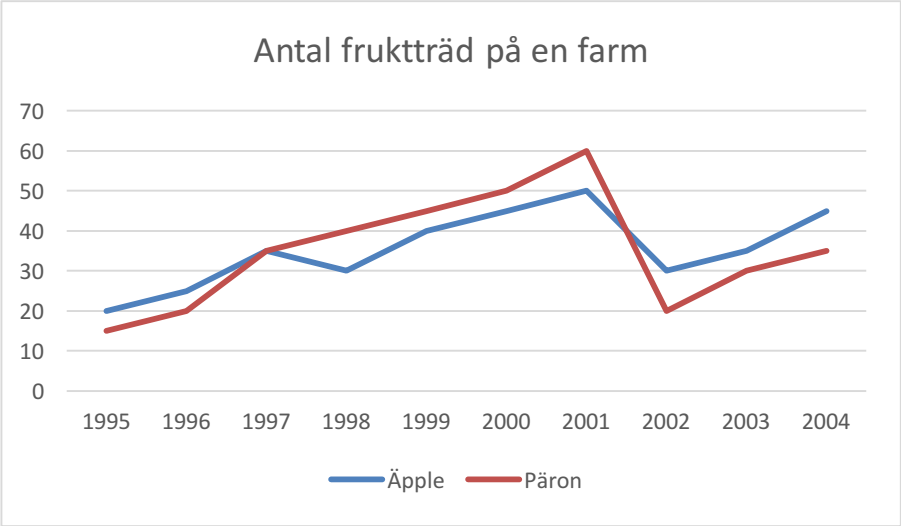

Fig. 3. Linjediagram (exempel)

2.4. Apache Wicket

Som grund till dashboarden ligger Apache Wicket. Wicket är ett ramverk för webutveckling i Java som bygger på Java Servlet API. Till skillnad från andra Java Servlet API ramverk (Strut, Spring MVC) som bygger på implementationen av controllers som hanterar requests och responses bygger Wicket på implementationen av komponenter. Varje komponent kan sedan användas flera gånger på samma sida. Ett exempel på en komponent skulle kunna vara en knapp med en specifik funktion. Varje gång man behöver denna knapp på en sida kan man bara lägga till den komponenten där det behövs. Inom Wicket finns det stöd för Ajax som gör att man specifikt kan uppdatera en komponent med ny data utan att behöva ladda om hela sidan eller andra komponenter. [1][2]

Andra kända väl använda ramverk som liknar Wicket är till exempel Vue.js & Angular. Den största skillnaden mellan dessa är att Vue.js och Angular bygger på JavaScript.

Tyringekonsult valde Wicket då de redan använder det för deras andra webbapplikationer.

2.5. Hibernate ORM

Som grund till interaktionen med databasen ligger Hibernate ORM. ORM (Object/Relational Mapping) bygger på att sammanföra en relationsdatabas till en objektorienterad modell. En tabell i en databas kan då kopplas till klass och då sedan ett objekt för lättare hantering och manipulering av data. Hibernate distribueras under LGPL (Lesser General Public License). [3]

Fördelar med att använda Hibernate istället för till exempel vanlig JDBC (Java Database Connectivity) är att Hibernate inte är beroende av typen av databas. Detta leder till att man som utvecklare inte behöver bry sig om skiljande syntax mellan databastyper. Hibernate ger också bättre prestanda än ren JDBC. Det finns också möjlighet att spara data i en cache för att underlätta hämtandet av data. [13]

Nackdel med Hibernate är att det inte går att använda med databastabeller som inte har en primärnyckel. Det är bra praxis att alltid ha en primärnyckel i tabellen men inte ett måste. [14]

I figur 4 är ett kodexempel på hur data hämtas från en databas med hjälp av Hibernate istället för vanliga queries i SQL. I detta fall skapas ett Criteria objekt som tar den klass som representerar den tabell i databasen man vill hämta data från. På detta Criteria objekt kan sedan en Restriction läggas till. I figur 4 fungerar denna som ”where” i SQL genom att bara hämta den data där fältet chgd är lika med ett specifikt datum.

```
/**
 * Get the amount of documents processed today
 *
 * @return the processed amount
 */
public Integer getDocsToday() {
    Criteria crit = session.createCriteria(Gdocf.class);
    crit.add(Restrictions.eq("chgd", new BigDecimal(getDate(0))));

    return (Integer) crit.setProjection(Projections.rowCount()).uniqueResult();
}
```

Fig. 4. Interaktion mellan hibernate och databas.

2.6. JavaScript

JavaScript är ett högnivåspråk som är interpreterande (koden kompileras ej i förhand av en kompilator). Språket presenterades 1995 som en lösning för att lägga till funktionalitet till webbsidor i webbläsaren Netscape Navigator. Sedan dess har alla större webbläsare infört JavaScript som en standard. Det är JavaScript som ligger till grunden för många av de webbsidor som används idag. I introduktionen av [16] skriver M. Haverbeke att JavaScript inte ska förknippas med programspråket Java då de två språken nästan inte har något med varandra att. När JavaScript presenterades marknadsfördes Java stort. Haverbeke skriver då att någon tyckte att det skulle vara en bra idé att använda sig av Javas framgång och använde då ordet Java i namnet till JavaScript.

När JavaScript började användas utanför Netscape Navigator såg Ecma International till att införa en standard på JavaScript som 1997 fick namnet ECMAScript. [16]

JavaScript utgör en av tre kärnfunktioner till en webbsida, de andra två är HTML och CSS. Med JavaScript blir webbsidor dynamiska och interaktiva. Grafer, interaktiva kartor, animation, och uträkningar är några få exempel av funktioner som JavaScript kan berika en webbsida med.

En stor fördel med JavaScript är att det är väl använt. Detta gör att språket i sig är väl dokumenterat i form av språkspecifikationer, böcker, artiklar samt diskussioner på sidor som Stack Overflow. Antalet bibliotek till JavaScript är enormt vilket gör att det redan finns lösningar för alla möjliga problem. [20]

En nackdel med användningen av JavaScript är säkerheten. När användaren av en klient (webbläsare) besöker en webbsida exekveras den webbsidans JavaScript direkt hos klienten. Detta leder till att illvilliga webbsidor kan utnyttja systemet som klienten körs på. Detta kan såklart undgås av användaren genom att vara vaksam på vilka sidor som besöks. Vanlig funktionalitet i webbläsaren är också funktionen att stänga av JavaScript så att det inte exekveras.

2.7. HTML5 Canvas

Canvas (<canvas>) är ett element som är en del av HTML5. Detta element används för att rita olika sorters grafik (geometriska figurer) på en webbsida. Istället för att lägga in färdiga bilder av figurer på en webbsida kan canvas användas för att rita upp dessa från grunden samt manipulera dem efteråt. Själva ritandet görs med hjälp av JavaScript. En stor användning av canvas är till exempel spel i webbläsaren. [21]

Elementet canvas startade som en uppfinning av Apple som behövde grafik till sina dashboard widgets. Elementet blev snabbt populärt hos andra tillverkare av webbläsare och har då lett till att elementet nu blivet en del av HTML5. [21]

Figur 5 visar ett kodexempel på ritandet av en cirkel med en triangel i sig och figur 6 visar resultatet på webbsidan. [15]

```
<!DOCTYPE html>
<html>
<body>

<canvas id="canvas" width="200" height="100"></canvas>

<script>
var c = document.getElementById("canvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.moveTo(95,10);
ctx.lineTo(135,50);
ctx.lineTo(55,50);
ctx.lineTo(95,10);
ctx.stroke();
</script>

</body>
</html>
```

Fig. 5. HTML5 Canvas, kodexempel

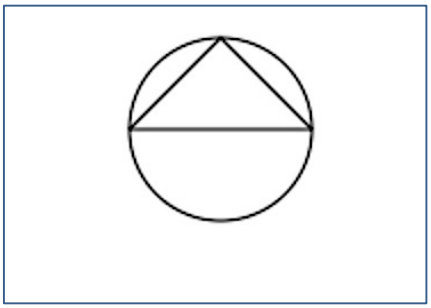

Fig. 6. HTML5 Canvas, resultat från figur 5

Fördelar med canvas är att det funkar väldigt väl med grafikintensiva spel. Canvas elementet som syns på webbsidan går också att spara direkt som en bild. Används canvas för visualisering går det direkt att spara visualiseringen som en bild och sedan använda den i till exempel en powerpoint. [15]

Nackdelar med canvas är att grafiken inte är vektorgrafik, grafiken beror då på upplösningen.

2.8. AJAX

AJAX (Asynchronous JavaScript And XML) är ett verktyg som används på klientsidan av en webbapplikation för att asynkront kunna skicka och hämta data från en server utan att interferera med webbsidans beteende. Ytterligare funktion som AJAX tillhandahåller är funktionalitet för att uppdatera och ändra innehåll på en webbsida utan att hela sidan behöver laddas om. [17]

Ordet AJAX kom först från Jesse James Garret i hans artikel *”Ajax: A New Approach to Web Applications,”* som publicerades för Adaptive Path. I början var folk skeptiska till artikeln men tillslut blev det en samstämmighet att AJAX var användbart. I [22] skriver C. Ullman och L. Dykes att den bästa webbsidan för att visa styrkan med AJAX är Googles Gmail. Användare kan editera mail, bläddra bland mail och inkorgar utan att sidan i sig behöver ladda om.

Denna funktionalitet funkar mycket bra för en dashboard med komponenter. AJAX kan till exempel ställas in för att utföra en uppgift med ett specifikt tidsspann. Genom att låta detta tidsspann vara kort upplevs det som att uppgiften sköts i realtid. Denna uppgift

kan då vara att skicka en query till en databas. När databasen svarar med data behöver komponenten som visualiserar datan uppdateras. Istället för att ladda om hela sidan och alla dess komponenter laddas bara den berörda komponenten om med hjälp av AJAX.

Fördelar med AJAX är att det är väl använt och det finns mycket dokumentation. Funktion för att automatiskt uppdatera komponenter på en webbsida är också en stor fördel.

Nackdel med AJAX är beroendet av att JavaScript är aktiverat i webbläsaren. Om datan i databasen inte uppdateras ofta och man väljer att skicka queries ofta blir det samma data som skicka om och om igen innan ny data hämtas.

2.9. WebSocket

WebSocket är ett kommunikationsprotokoll för tvåvägskommunikation över en socket med TCP. J. Karlström [26] skriver om historiken bakom protokollet. 2010 lämnades ett tidigt förslag på protokollet in av I. Hickson (författare och underhållare av HTML5 specifikationen). Efter ett antal utkast av protokollets specifikation blev WebSockets en del av HTML5

Denna teknik kan då användas mellan en databas och en webbapplikation för att sända data i realtid. Anslutningen över socket är öppen hela tiden vilket gör att data från databasen kan skickas till webbapplikationen när som helst. Om webbapplikationen är en dashboard som ska visa data i realtid kan data från databasen direkt skickas till webbapplikationen när ny data finns tillgänglig. [18]

2.10. Google Chart

Google chart är ett gratis bibliotek för att visualisera data. Biblioteket har funktionalitet för en mängd olika grafer.

Figur 7 visar ett kodexempel på ett tårtdiagram och figur 8 visar hur detta tårtdiagram ser ut i webbläsaren.

```html
<html>
  <head>
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">

      google.charts.load('current', {'packages':['corechart']});

      google.charts.setOnLoadCallback(drawChart);

      function drawChart() {

        var data = new google.visualization.DataTable();
        data.addColumn('string', 'Pokemon');
        data.addColumn('number', 'Antal');
        data.addRows([
          ['Pidgey', 20],
          ['Ratata', 8],
          ['Caterpie', 12],
          ['Weedle', 5],
          ['Pikachu', 2]
        ]);

        var options = {'title':'Antal fångade Pokémon under en dag'};

        var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
        chart.draw(data, options);
      }
    </script>
  </head>
  <body>
    <div id="chart_div"></div>
  </body>
</html>
```

Fig. 7. Kodexempel av tårtdiagram i Google chart

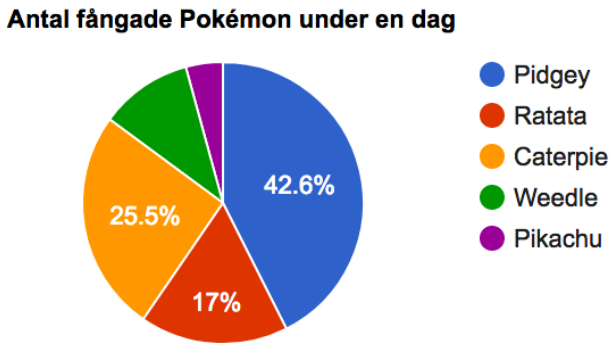

Fig. 8. Tårtdiagram, resultat från figur 7

2.11. Draw.io

Draw.io är ett digitalt verktyg för att rita UML, flödesschema samt vanliga figurer. Det som ritas i verktyget kan sparas för att senare ändras eller direkt exporteras till olika bildformat. Dessa kan då sparas direkt på datorn eller till Google drive. I figur 9 syns ett exempel på hur draw.io kan användas för UML.

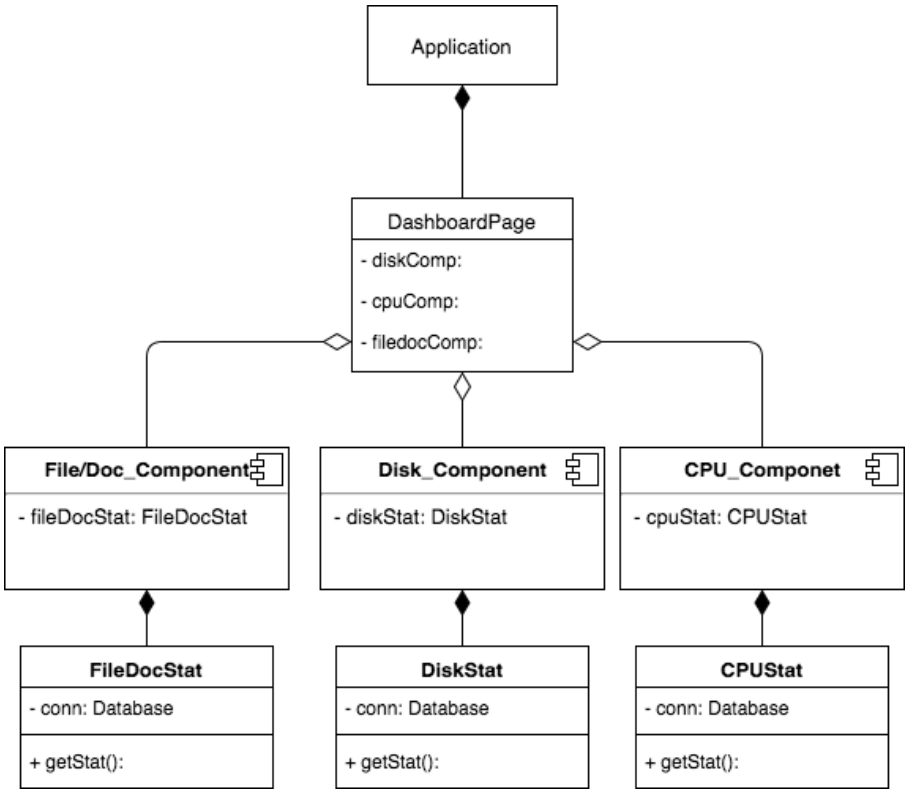

Fig. 9. Exempel på UML ritat i draw.io

2.12. Apache Commons Math

Apache Commons Math är ett bibliotek vars funktion är att lägga till matematisk och statistisk funktionalitet och ta itu med de mest vanliga problem som inte redan finns tillgängliga i Java. [6]

3. Metod

I följande kapitel beskrivs upplägget på examensarbetet. De olika faserna som arbetet befunnit sig i och hur arbetet skiljt sig mellan de olika faserna. Arbetet har inte följt en specifik utvecklingsprocess. Arbetet mellan de olika faserna har gått sekventiellt och är i följande ordning enligt figur 10; Formulering av examensarbetet, inlärning, och utveckling. Fasen utveckling är sedan indelad i tre delkapitel som i sin tur gått iterativt; Undersökning, implementering, och validering. Varje iteration har använts för att undersöka, implementera och validera en specifik komponent till dashboarden.

Fig. 10. Överblick på faserna

3.1. Formulering av examensarbetet

Första fasen inleddes med själva formuleringen av examensarbetet.

Tyringekonsult AB efterfrågade från början en färdig dashboard med för att projektet skulle kunna fungera som examensarbete krävdes det att anpassningar gjordes. Det resulterade i att i stället för att skapa en färdig dashboard skulle en prototyp av den skapas.

Tillsammans med en av de anställda på Tyringekonsult (som tidigare läst till högskoleingenjör i datateknik på LTH) gjordes problemformuleringarna. Utifrån detta skrevs examensarbetets initialbeskrivning.

Efter att initialbeskrivningen lämnats in till handledare (Christin Lindholm) och examinator (Christian Nyberg) och godkänts startade examensarbetet.

3.2. Inlärning

Andra fasen bestod av inlärning av de olika verktygen Wicket och Hibernate som skulle användas inom examensarbetet. Denna fas startade nästan samtidigt med den första fasen. För inlärning av ramverket Wicket lästes boken *Wicket in Action* [1]. I boken är där ett exempel på en webbapplikation som byggs ut för varje ny sak som introduceras i de olika kapitlen. Genom att följa detta exempel och implementera det själv var det till stor hjälp för att börja använda Wicket. Tillsammans med boken ställdes också frågor till utvecklare på Tyringekonsult som agerade handledare för utveckling.

Tyringekonsult använder sig också av ramverket Hibernate. Inlärningen av Hibernate gjordes med hjälp av en introduktion från handledare på Tyringekonsult.

Fasen har också använts för studier om visualisering av data och speciellt visualisering i dashboards.

3.3. Utveckling

Inom den tredje fasen har arbetet gått iterativt mellan tre mindre faser. Dessa tre mindre faser är först fasen undersökning, sedan fasen implementation, och till sist fasen validering enligt figur 11.

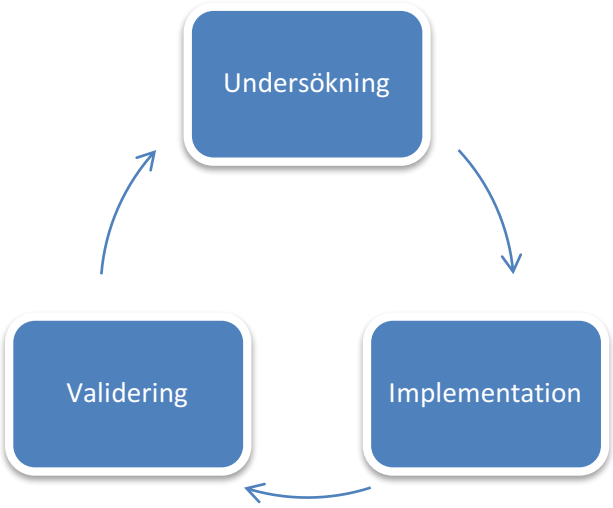

Fig. 11. Överblick på utvecklingsprocessen

Utvecklingen av dashboarden har utförts på plats bland utvecklarna på Tyringekonsult. Detta har underlättat kommunikationen då det snabbt gick att bestämma eller spontant hålla ett möte eller ta en diskussion. Detta gjorde också att det snabbt gick att validera.

3.3.1. Undersökning

Fasen undersökning har använts för att kartlägga hur arbetet ska gå tillväga för att besvara de olika problemformuleringar som ställs inom examensarbetet. Överblick av undersökningen enligt tabell 1.

TABLE I. ÖVERBLICK AV PROTOTYPERNA

Data	**Antal prototyper**	**Typ av prototyper**	**Slutgiltig prototyp**
CPU	3	#1: ”Hastighetsmätare” som visualisera både CPU och Disk i samma. #2: ”Hastighetsmätare” som bara visualisera CPU. Yttre mätpunkter för två äldre värde som referens. #3: ”Hastighetsmätare”. Samma som #2 fast med inre mätpunkter.	#3
Disk	3	#1: Hastighetsmätare som visualisera både CPU och Disk i samma. #2: Linjediagram med värde för disk med 30min mellanrum. #3: Linjediagram med medelvärde för disk under 1	#3

		dag. Värdepunketer under en period av 30 dagar.	
Filer & Dokument	2	#1: Stapeldiagram med antal processade filer/dokument för den innevarande dagen, under de senaste 7 dagarna och under de senaste 30 dagarna. #2: Tabell med samma värden som #1.	#1

Vilken data som skulle kunna vara viktig att visualisera i dashboarden undersöktes först. Detta gjordes genom diskussion/möte tillsammans med de på Tyringekonsult som kommer att ha användning för dashboarden. Mötet började med en introduktion på hur Tyringekonsults system fungerar. Efter detta redovisade de hur de skött den nuvarande övervakningen/supporten av systemet. Detta hade då skötts av en anställd som manuellt hämtat data från databasen och fört in det i ett Excel-ark. Den data som hämtats manuellt (användning av CPU, användning av hårddisk) kunde då fastställas som viktigt data att visualisera i dashboarden och har sedan legat till grund för de andra frågorna i problemformuleringen som behandlar visualisering av data. Vid mötet uppkom också extra tillägg till dashboarden. För att visualisera dessa extra tillägg skulle dock extra hårdvara behövas för att föra in datan i databasen. Detta valdes att läggas som framtida utvecklingsmöjligheter.

Här följer ett exempel som till stora delar beskriver hur arbetet inom fasen gått till. Efter identifiering av vilken data som var viktigt började framtagningen av hur en specifik data skulle visualiseras. Detta gjordes först med olika prototyper som först ritades som en mindre skiss på papper för hand och sedan i ett digitalt ritverktyg (draw.io [5]) enligt figur 13. Prototyper i form av tabeller och grafer ritades upp med hjälp av Excell enligt figur 12.

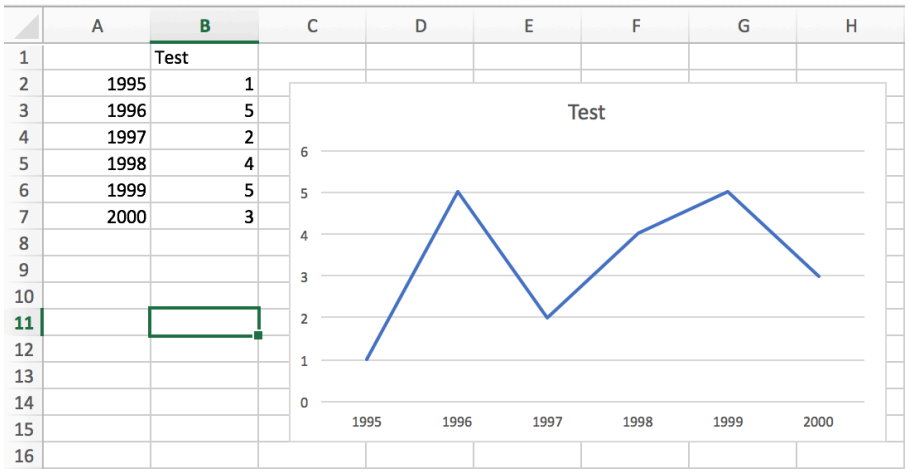

Fig. 12. Exempel på graf i Excell

Till varje typ av data som skulle visualiseras gjordes två till tre prototyper. Text i prototyperna är på engelska enligt begäran från Tyringekonsult.

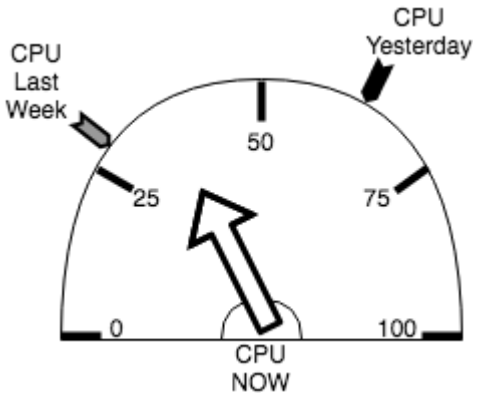

Fig. 13. Prototyp till mätare för CPU med yttre äldre mätpunkter

I samband med framtagandet av prototyperna undersöktes också vilka bibliotek/ramverk som kunde användas för de olika prototyperna. Detta gjordes genom att söka information om grafiska bibliotek/ramverk vi Google. Specifikationer för dessa

bibliotek/ramverk har sedan lästs igenom för att få en överblick om deras funktionalitet.

Efter framtagning av två till tre prototyper hölls möte för att gå igenom prototyperna och bestämma vilken som skulle passa bäst eller om det skulle behövas göra några mer ändringar till prototyperna. Mötet genomfördes med samma personer som var med på mötet om vilken data som skulle visualiseras. Prototyperna visades upp och tankegången bakom dem förklarades. Efter detta gavs feedback på de olika prototyperna. Denna feedback kom från anställda på Tyringekonsult som dashboarden är riktad till (utvecklare, support). Efter diskussion valdes gemensamt en av prototyperna att gå vidare till nästa fas inom utveckling som är själva implementeringen av prototypen.

3.3.2. Implementering

Under fasen implementering har de tre prototyperna som tagits fram i fasen undersökning implementerats. Då dashboarden består av olika komponenter är det lätt att ta fram en prototyp åt gången och sedan implementera den komponenten. Av de två till tre prototyper som tagits fram till varje komponent implementerades bara en. Prototyperna har också varit olika från komponent till komponent. De olika komponenterna är följande; Statistik för CPU, statistisk för användning av hårddisk (kommer förkortas som disk framöver), och statistik för antal filer och dokument som processats.

Implementeringen av en komponent till dashboarden startade med klassen som ska hämta den specifika datan till just den komponenten från databasen. Genom tillgång till databasen kunde klassen valideras att den hämtar korrekt data. När klassen validerats började själva implementationen av komponenten. Varje komponent består minst av en klass för komponentens funktionalitet samt en html-fil. Vid behov av ytterligare funktionalitet för till exempel användning av html canvas skapas en fil med tillhörande java-script. Komponenterna hänger inte ihop och kan därför implementeras en och en oberoende av varandra och när en komponent är klar börjar en ny iteration med undersökning och då med prototypen till nästa komponent. Hur komponenterna hänger ihop med dashboarden och de klasser som hämtar data från databasen finns i figur 14 i form av UML.

Application är webbapplikationen och DashboardPage är dashboarden. Dashboarden har sedan 3 komponenter, File/Doc_Component, Disk_Component, CPU_Component. Dessa komponenter får sedan data från Tyringekonsults befintliga databas via respektive klasser: FileDocStat, DiskStat, CPUStat.

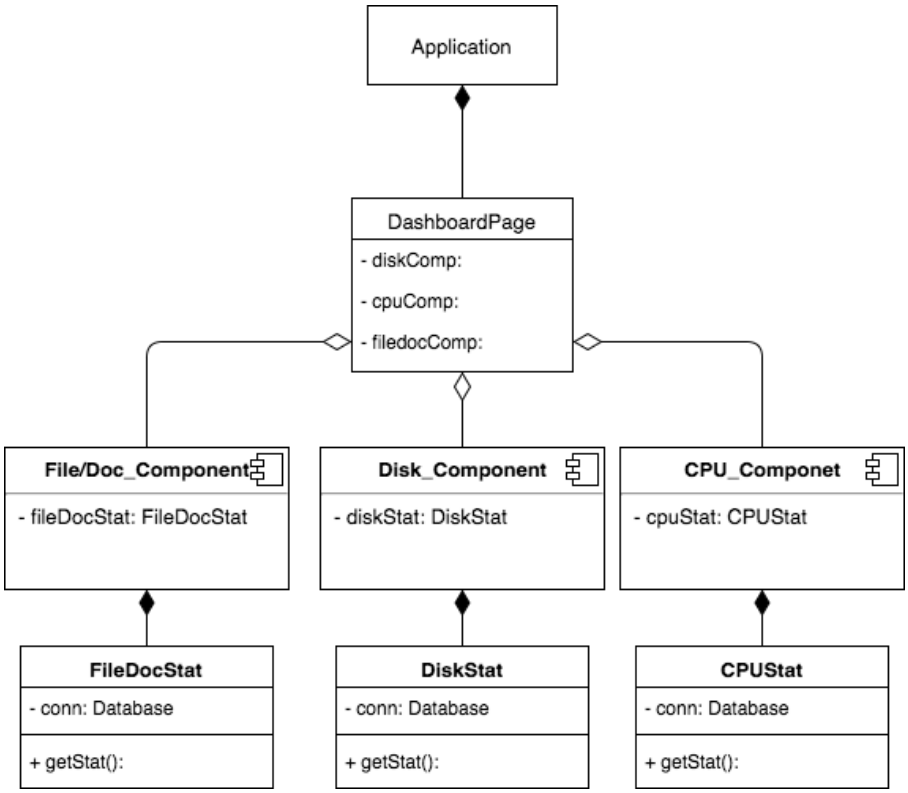

Fig. 14. UML över hur komponenterna hänger ihop med dashboarden

3.3.3. Validering

I slutet av varje fas validerades prototypen. Ändamålet med valideringen av prototyperna var att se till att den data som visualiserades var korrekt samt att de som skulle använda dashboarden tyckte prototypens visualisering var begriplig.

Valideringen av datan som visualiserades i prototypen gjordes genom att jämföra den med den obearbetade datan i databasen.

Valideringen av prototypen i sig gjordes genom att prototypen visades upp på möte för de på Tyringekonsult som har användning för dashboarden. Största delen av denna validering hade redan gjorts vid framtagandet av prototypen innan den implementerats. Detta tillfälle validerade då att implementationen följde prototypen i sig.

3.4. Källkritik

Ett stort antal av källorna som listas är källor från de företag som levererar de olika ramverk/bibliotek som använts inom examensarbetet. Källorna anses vara trovärdiga då dessa är väl använda. Hade felaktig information listats på sidorna borde användare snabbt upptäcka detta och felanmäla. Skulle informationen inte rättas kommer företaget i fråga definitivt tappa sina användares förtroende. Dessa är Apaches wiki om Wicket [2][8][9], Apaches specifikation av Apache Commons Math [6], Hibernates documentation av Hibernate ORM [3][13][14], Googles specifikation av Google Chart [7].

Boken *”Wicket in Action”* [1] är skriven av M. Dashorst och E. Hillenius vilka båda varit aktivt involverade i Wicket över åren. Detta ger boken hög trovärdighet som källa.

Källan för implementationen av ”hastighetsmätaren” med HTML5 canvas [5] anses trovärdig då koden som angetts på sidan använts utan problem. Ytterligare källor som behandlar canvas är w3schools specifikation [15] och boken *”Foundation HTML5 Canvas”* [21]. W3schools specifikation har använts och fungerat vilket ger källan stor trovärdighet, sidan har också sedan innan använts inom ett antal kurser. Boken [21] har hittats via LUBSearch samt att den refererar till ett antal källor vilket gör den till en trovärdig källa. Ytterligare info från W3Schools är källa [17] om AJAX. W3Schools är en väl använd sida som gör att källan är trovärdig. Källan har också exempel som direkt går att testa.

Källa [19] är en muntlig källa om hur Tyringekonsults EDI-system fungerar. Denna person har arbetar på Tyringekonsult vilket gör att källan är trovärdig. Ytterligare info om EDI har hämtats från [11]. Hemsidan anger inte några källor men företaget bakom sidan levererar EDI-lösningar vilket gör att källan anses vara trovärdig. Informationen från [11] styrks också av den muntliga källan [19]

Ytterligare källor som hämtats från LUBSearch eller Google Scholar är [12], [16], [20], [22], [23], [24] och [26]. Alla dessa källor anger källor för information från publicerade vetenskapliga arbeten. Detta gör källorna trovärdiga.

Källa [10] är en artikel från internet. Dessa sorters artiklar brukar inte specificera källor vilket gör de mindre trovärdiga. Denna artikel använder sig dock av källor som kommer direkt från W3C specifikationen av WebSockets vilket då gör [10] till en trovärdig källa.

4. Analys

I kapitlet analys analyseras de olika val, problem och lösningar som uppkommit under examenarbetets gång.

4.1. Prototyper

Under detta kapitel beskrivs valet av prototyper och varför den slutgiltiga prototypen blev som den blev.

4.1.1. CPU

Prototypen för att visualisera CPU började som en hastighetsmätare som visar det nuvarande belastningen i procent men också en mindre visare för belastning på CPU vid ett tidigare tillfälle som referens. För att spara plats var också det procentuella värdet för disk inkorporerat i hastighetsmätaren enligt figur 15. Den yttre ringens fyllnad visade användning av disk. Pilarna visade två olika mätvärden på användning av CPU, den längre visa det nuvarande värdet medans den kortare visade ett äldre värde som referens.

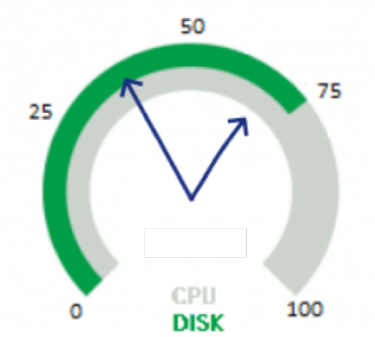

Fig. 15. Tidig prototyp för visualisering av CPU och disk

Denna prototyp uppfattades vara svår att förstå av en utvecklare på Tyringekonsult, detta på grund av att mer än en sort av data visualiserades samtidigt. Vad pilarna exakt pekade på var också förvirrande. I detta skede valdes det då att separera på visualiseringen av CPU och disk. Från denna separation framkom två nya prototyper, figur 16 och figur 17.

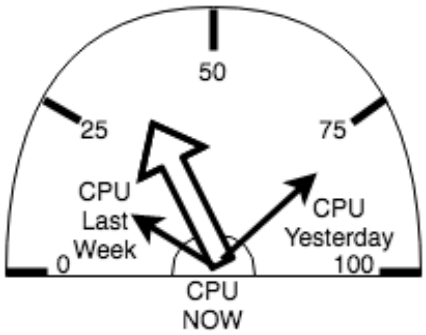

Fig. 16. Prototyp till mätare för CPU med inre äldre mätpunkter

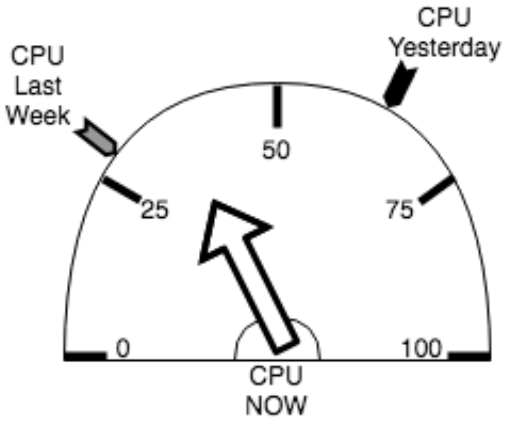

Fig. 17. Prototyp till mätare för CPU med yttre äldre mätpunkter

”Hastighetsmätare” fungerar bra för att visualisera ett momentanvärde som bara kan fluktuera mellan 2 specifika värden

Skillnaden i dessa två prototyper är hur de två äldre mätvärdena ska visualiseras, antigen som markörer på utsidan av mätaren eller som mindre pilar inne i mätaren. Detta ansågs vara lika tydligt och den avgörande faktorn blev komplexiteten att implementera den ena eller den andra. Komplexiteten berodde på om det fanns bibliotek för grafer av denna typ eller om grafen skulle behövas implementeras från grunden.

Sökandet efter ett bibliotek/ramverk med funktionalitet för denna typ av mätare startade. Sökandet var inte bara låst till Java utan JavaScript var också ett alternativ. Antalet bibliotek/ramverk för visualisering av data i olika grafer är enormt men när det kommer till

just mätare som hastighetsmätare blir resultaten avsevärt mindre, i alla fall när det kom till funktionalitet på mätaren. Resultatet av sökandet gav att det inte fanns något gratis bibliotek/ramverk för den funktionalitet som efterfrågades i prototypen. I detta läge var lösningen att antingen gå tillbaka till ritbordet för prototypen eller hitta en annan lösning.

Lösningen som hittades var att implementera mätaren direkt med HTML5 Canvas och JavaScript. Med hjälp av ett exempel som var fritt fram att använda och ändra på [4] gick prototyp figur 17 att implementera utan problem.

4.1.2. Disk

Som nämnt tidigare startade prototypen för visualisering av disk tillsammans med visualiseringen av CPU i figur 15. Disk togs bort från denna prototyp och resulterade i en prototyp för sig.

Denna prototyp visar alltid det nuvarande värdet för hur många procent av disken som används. Utöver den prototypen gjordes också en prototyp i form av ett linjediagram enligt figur 18. Istället för att bara visa det nuvarande värdet kan ett linjediagram visa en större bild i användandet av disk. Linjediagram är också användbara för att visualisera kontinuerlig data. Med en större bild går det att se om användandet av disk har ökat mer än vanligt vid ett tillfälle, vilket då kan signalera att ett problem uppstått vid det tillfället.

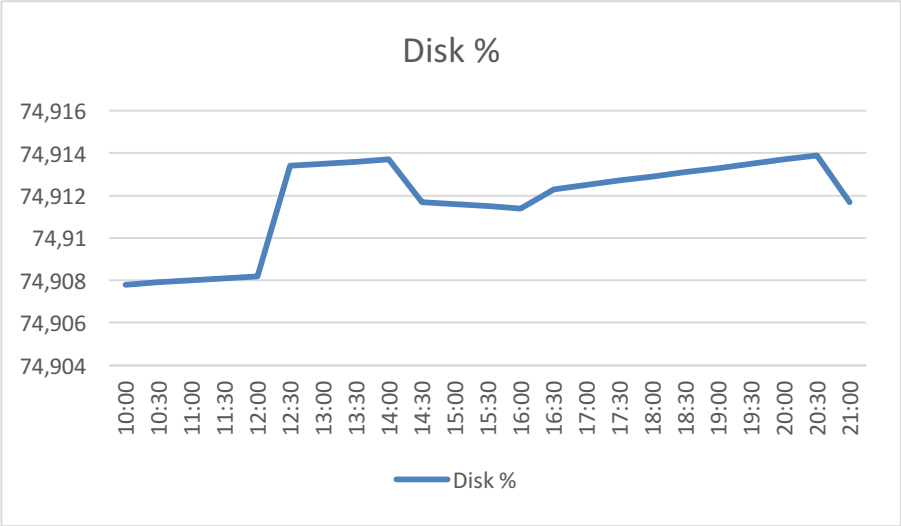

Fig. 18. Diskprototyp, linjediagram med data från databas, x-axel: tidpunkt, y-axel: användning av disk i procent

Linjediagramsprototypen var först planerad att visa datapunkter i spannet om en dag med 30 minuter mellan var datapunkt. Efter genomgång av den data från databasen som sen innan satts in i ett linjediagram i ett Excel-ark framgick det att värdet på disk inte ändrar sig mycket under en dag enligt figur 18. Datan skulle då behöva visualiseras på ett annat intervall för att det skulle ge en bättre överblick. Prototypen ändrades då till istället att ta medelvärdet för disk på en dag och sedan visualisera detta över ett större antal dagar (runt 30 dagar eller mer). Ytterligare sak som framgick var att delar av lagringen på disk ibland rensades för att fria upp minne. Detta gjorde att grafen till disk fluktuerade en del över dagarna som gjorde att det blev svårt att få en överblick av hur användningen av disk ökade under perioden. För att tydligare göra detta lades en trendkurva in i prototypen. I detta skede var Tyringekonsult nöjda med hur linjediagrammet visualiserade disk och denna prototyp (figur 19) valdes att gå vidare till implementation.

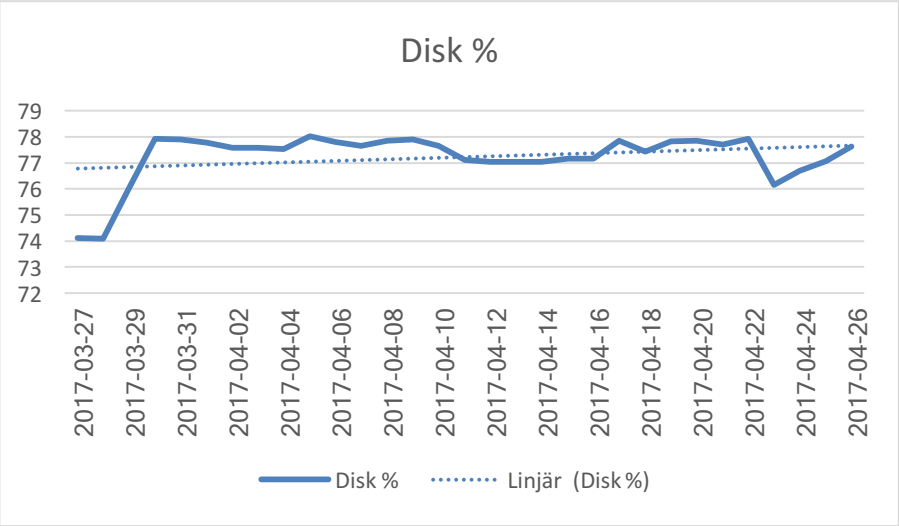

Fig. 19. Diskprototyp, linjediagram och trend med data från databas, x-axel: datum, y-axel: användning av disk i procent

Mängden bibliotek/ramverk för visualisering i form av grafer är stort och funktionerna är många. Valet av bibliotek/ramverk gjordes efter vilket som redan hade inbyggd funktion för att lägga till en trend över den data som matades in i grafen. Skulle denna funktion inte finnas hade den funktionalitet kunnat implementerats med hjälp av Apache Commons Math [6]. I slutändan valdes det att använda Googles Chart API [7] då det hade den efterfrågade funktionaliteten.

4.1.3. Filer & Dokument

För visualiseringen av de filer och dokument som bearbetas av EDI-systemet gjordes två olika prototyper. Komponenten ska visualisera antalet processade filer och dokument under den innevarande dagen samt det totala antalet filer och dokument processade de senaste sju och 30 dagarna.

Den första prototypen börja som ett stapeldiagram med en stapel för varje mätvärde enligt figur 20. Stapeldiagrammets y-axel visar antalet. Stapeldiagram är bra för att visualisera data som hänger ihop och ej är kontinuerlig.

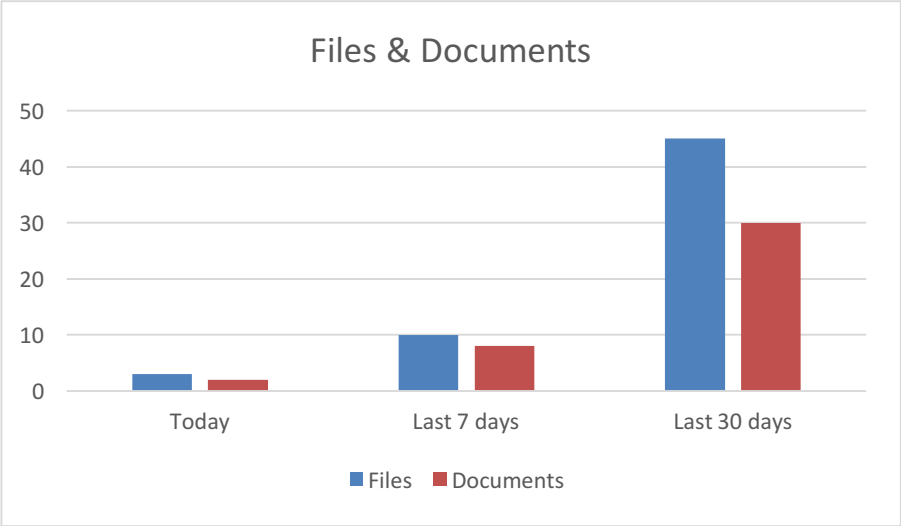

Fig. 20. Prototyp för filer & dokument, stapeldiagram med fiktiv data

Denna prototyp ger en bra överblick av mängden filer och dokument som processats under tidsperioderna. Ett problem som uppstod med denna prototyp var att det snabbt blev svårt att urskilja exakt antal som processats. Figur 21 visar prototypen där data från databasen satts in för att ge en bättre bild hur det faktiskt kan se ut. När antalet över 30 dagar går upp mot 1000 blir det svårt att se antalet som processats under dagen men även de senaste sju dagarna.

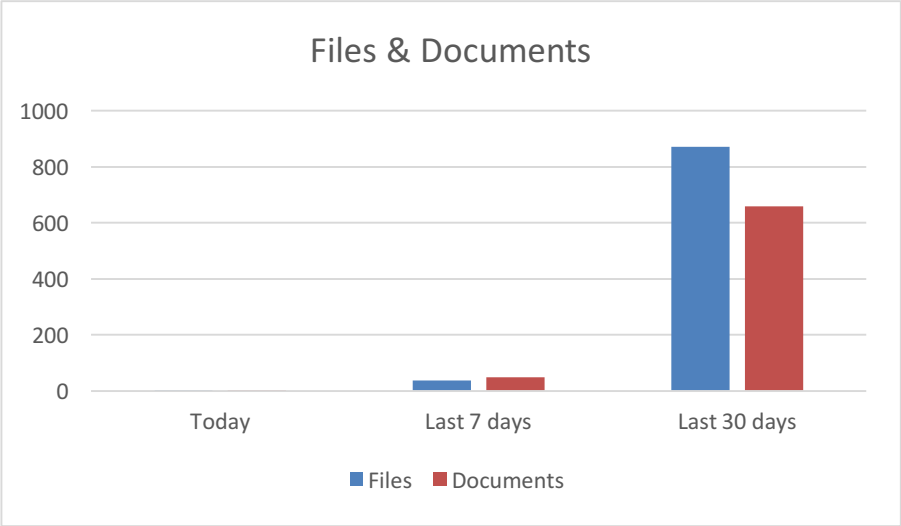

Fig. 21. Prototyp för filer & dokument, stapeldiagram med data från databas

För att motverka detta problem gjordes en prototyp i form av en tabell enligt figur 22. Tabellen ger som stapeldiagrammet också en snabb men mer exakt överblick då man direkt kan avläsa värdena. Även om värdena för 30 dagar stiger mot 1000 går det fortfarande att snabbt avläsa dagens statistisk. En lösning skulle vara att visa värdet under varje stapel men samtidigt kan man då lika väl ha en vanlig tabell om varje värde ändå måste skrivas ut.

Type	*Today*	*Last 7 days*	*Last 30 days*
Files	2	38	871
Documents	1	48	659

Fig. 22. Prototyp för filer & dokument, tabell med data från databas

Prototypen enligt figur 22 valdes då att implementeras efter gemensam diskussion med Tyringekonsult.

Html har bra stöd för tabeller och inget extra bibliotek/ramverk valdes för att implementera tabellen.

4.2. Data i realtid

En av frågorna från problemformuleringen var hur data skulle kunna hämtas i realtid istället för vid specifika tidsintervall. Lösning på hur detta skulle kunna göras har undersökts. Undersökningens resultat har inte implementerats i dashboarden utan lösningarna presenteras bara i kapitlet resultat.

5. Resultat

Detta kapitel går igenom de olika slutgiltiga resultaten för examensarbetet. Resultatet presenteras med bilder på de färdiga komponenterna samt kodexempel för komponenter och klasser som interagerar med databasen. Resultaten har tagits fram genom utveckling av prototyper som sedan visats för de anställda på (utvecklare, support) Tyringekonsult som ska använda dashboarden. De slutgiltiga resultaten speglar vad de anställda ansåg vara tydligast och bäst för att visualisera den specifika datan.

5.1. Data att visualisera

Resultatet med vilken data som ska visualiseras bygger på diskussion med Tyringekonsult om hur deras EDI-system fungerar samt vilken data som sedan innan lagts in manuellt i Excell-ark. Detta resultat var att visualisera belastning för CPU och användning av disk på de servrar som EDI-systemet körs på. Hur många filer och dokument som EDI-systemet bearbetar ska också visualiseras. Varje resultat listas med bild på de slutgiltiga prototyperna.

5.1.1. CPU

Den färdiga implementationen av prototypen för visualisering av CPU blev enligt figur 23.

Fig. 23. Färdig implementation för visualisering av CPU

I figur 23 kan värdet gå mellan 0 till 200%. Detta då denna server kan använda mer processorkraft från en annan CPU som gör att värdet för användning kan gå upp mot 200%.

Protoypen ger en bra överblick över värdet på CPU samt om det nuvarande värdet är rimligt. Att lägga till ytterligare visare i prototypen är lätt. Prototypens nackdelar är implementationen i sig. Den är lätt att använda när den väl är implementerad men om ”hastighetsmätarens” storlek, max värde, värdepunkter, längd på visare ska ändras blir det en del att ändra i koden så att proportionerna stämmer.

De olika värden som ”hastighetsmätaren” visualiserar är enligt tabell 2.

TABLE II. PEKARE I VISUALISERING FÖR CPU

Pekare	**Värde**
Längst	Det nuvarande värdet, inkommer nytt värde var 5:e minut till databasen
Mellan	Medelvärde för gårdagen inom ett intervall av -30 till +30 minuter från det nuvarande värdets tid

Kortast	Medelvärde för samma veckodag förra veckan inom ett intervall av -30 till +30 minuter från det nuvarande värdets tid

Belastning av CPU ser ungefär lika ut varje dag. Därför valdes det att visualisera två äldre värden så att en jämförelse kan göras med det nuvarande värdet. Det nuvarande värdet är också till för att få en snabb överblick om belastningen är hög vid ett tillfälle där systemet upplevs långsamt.

5.1.2. Disk

Figur 24 visar den färdiga implementationen av prototypen för visualisering av disk. X-axeln representerar datum i en period av 30 dagar och y-axeln visar procentuell användning av disk. Den helt linjära kurvan är grafens trend.

Fig. 24. Färdig implementation för visualisering av disk

Värde för procentuell användning av disk läggs in i databasen tillsammans med CPU var 5:e minut. Skillnaden i procent per 5:e minut för disk skiljer sig lite (ändringar på hundradelars procent) att det inte blev någon bra överblick vid användandet av de värdena. För bättre överblick valdes medelvärdet för diskanvändningen under dagen och sedan visualisera detta i ett intervall om 30 dagar. För att

också ge en bättre överblick på hur användandet av disk ändrar sig under denna period lades en trendkurva till.

5.1.3. Filer & Dokument

Hur den färdiga implementationen av prototypen för visualisering av filer och dokument blev enligt figur 25.

Type	Today	-7 days	-30 days
Files	24	93	626
Documents	24	103	414

Fig. 25. Färdig implementation för visualisering av filer & dokument

På grund av den stora skillnaden av antalet bearbetade filer för innevarande dag och för 30 dagar sedan blev det tydligast att visualisera i en tabell. Detta jämfört med tidigare prototyp i form av stapeldiagram där det stora antalet över 30 dagar gjorde att den innevarande dagens värde var svår att utläsa.

5.2. Realtidsdata

Undersökningen av hur data skulle kunna uppdateras i dashboarden i realtid ledde till två olika lösningar, Ajax eller WebSockets. Tillgängligheten för båda lösningarna finns där då det redan finns funktionalitet för båda inom Wicket [8][9].

5.2.1. Ajax

Ajax används redan i dasboarden för att uppdatera data i komponenterna utan att hela sidan i sig behöver laddas om.

Med hjälp av Ajax är det möjligt att inom tidsintervall skicka queries till databasen och sedan uppdatera komponenterna med ny data. Genom att använda sig av en kort tid mellan anrop kan denna lösning upplevas som att datan hämtas i realtid från databasen[1]. Denna lösning kräver inga ytterligare tillägg till servern som databasen ligger på eller databasen i sig.

I Tyringekonsults fall inkommer bara data till databasen var 5:e minut vilket gör att om Ajax skulle vara inställt på att hämta data var 5:e sekund hade samma data hämtats många gånger om. Hur krävande detta skulle vara för nätverkstrafiken mellan dashboard och databas skulle då behöva undersökas vidare.

5.2.2. WebSockets

Istället för att skicka data med specifikt intervall från databasen med Ajax finns lösningen WebSockets.

Med WebSockets hålls anslutningen mellan server och dashboard uppe konstant. När det sker en uppdatering av data i databasen kan serven skicka datan direkt till dasboarden. Detta leder då till att datan kommer till dashboarden i realtid oavsett hur lång tid det är mellan uppdatering av data i databasen.

För att lösa detta skulle dock ytterligare funktionalitet behöva implementeras på den server där databasen ligger i de fall där servern inte redan stödjer WebSockets [10].

6. Slutsatser

Under kapitlet slutsatser sammanfattas det viktigaste resultatet från examensarbetet genom att besvara problemformuleringen och beskriva hur resultatet kommer att användas.

6.1. Problemformulering

Vilken data är viktig att visualisera i dashboarden?

Resultat efter möte och diskussion med Tyringekonsult har gjort att den viktigaste datan att visualisera för ett EDI-system har tagits fram; Belastning av CPU, användning av disk samt hur många filer och dokument som systemet behandlar. Dessa datamängder är de mest grundläggande parametrarna i systemet och beskriver ganska bra belastning m.m. på EDI-systemet.

Denna data hjälper att få den snabba överblick av systemet som Tyringekonsult var ute efter. Hög användning av CPU kan tyda på att något i systemet inte fungerar som det ska. Detta kan sedan jämföras med hur många filer och dokument som systemet behandlat under dagen. Har det inte kommit in många filer och dokument under dagen och CPU är hög så kan det då antyda på att något är fel och behöver vidare undersökning. Att användning av disk ökat mycket kan också antyda på att något i EDI-systemet behöver vidare undersökning.

Även om andra företags EDI-system skiljer sig från Tyringekonsult är ändå den underliggande strukturen med belastning på CPU, användande av disk och att systemet behandlar en mängd data det samma.

Vilken form är mest lämplig för att visualisera datan?

Hur de olika typerna av data har valts att visualiseras har gjorts efter framtagandet av olika prototyper. Dessa prototyper har sedan gåtts igenom tillsammans med planerade användare av dashboarden. De färdiga prototyperna speglar vad som ansåg var tydligast från dessa personer. Då Tyringekonsult länge arbetat med EDI-system anses deras respons på tydlig visualisering av data något som också

skulle kunna anses vara mest tydligt av andra företag som jobbar med EDI-system.

Hur visualiserar man realtidsdata tillsammans med historikdata?

Varje prototyp visualiserar både realtidsdata och historikdata. Datan hanteras inte olika beroende på om det är data i realtid eller äldre historikdata. Varje gång det kommer in ny data till databasen blir den data som räknades som realtid innan nu en del av historikdatan

Vilka/vilket ramverk/bibliotek lämpar sig bäst för visualiseringen?

De bibliotek och ramverk som valts för visualisering har valts efter att de tillhandahöll den funktionalitet som önskades av prototyperna. Bara bibliotek/ramverk som varit under gratis licens för användning kommersiellt har undersökt (till exempel MIT-licens). Bibliotek/ramverk för visualisering med ytterligare funktionalitet skulle därför kunna hittas där en licens måste köpas.

Hur löser man att data uppdaterar sig i realtid istället för vid specifika tidpunkter eller intervall?

Undersökningen för hur data skulle visualiseras i dashboarden i realtid har resulterat i att antingen använda Ajax eller WebSockets. Dessa två lösningar finns redan inbyggd i Wicket som då gör dessa två lösningar till bra kandidater. Dessa två lösningar är också brett använda och tillägget av den ena eller den andra borde inte vara några problem när det kommer till dashboard för andra EDI-system. Reflektion över etiska aspekter

6.2. Reflektion över etiska aspekter

De etiska aspekterna runt datasamling och datalagring är en mycket aktuell diskussion. Ännu mer aktuell är den för tillfället i USA där senaten har röstat bort en lag gällande datainsamling och datalagring. Den lag som upphävdes gjorde att en ISP inte kunde sälja vidare deras abonnenters internethistorik utan ett godkännande från abonnenten i fråga. Data som vilka sidor en specifik abonnent besöker och när dessa sidor besöks kan nu säljas vidare till

reklambyråer. Detta leder också till att vem som helst med rätt mängd pengar kan få ut historik på specifika personer.

Många anser att lagring om vilka sidor man som abonnent besöker är kränkande och gör intrång på ens privatliv. Detta är helt förståeligt, speciellt när situationen har blivit som den är i USA. Problemet är samtidigt att det inte hade varit hållbart för samhället att inte lagra data om ens abonnenter. Datainsamlingen och datalagringen är i grunden där för att motverka och förebygga olaglig aktivitet via internet.

6.3. Framtida utvecklingsmöjligheter

All data som visualiserats inom examensarbetet är data som redan funnits tillgänglig i Tyringekonsults databas. En ytterligare funktion att visualisera i dashboarden var att visualisera om servern som systemet körs på får ström från det vanliga elnätverket eller om strömmen kommer från en generator då det är strömavbrott. Generatorn startar automatiskt vid strömavbrott. Denna data om ström behöver dock extra funktionalitet för att loggas i databasen. Då generatorn behöver bränsle för att fungera hade funktionalitet för push-meddelande om att generatorn startat varit bra.

Implementationen av prototyp för visualisering av CPU skulle behöva förbättring i form av ett bibliotek/ramverk med funktion för ”hastighetsmätare” med mer än en pekare. Implementationen med hjälp av HTML canvas gör det den ska men det blir snabbt krångligt om man skulle behöva ändra på storlek, antalet värdemarkeringar runt mätaren m.m. Undersökningen gav inget gratis bibliotek/ramverk med denna funktionalitet.

Dashboardens funktionalitet skulle också kunna utvidgas till att visa data som inte just berör EDI-systemet i form av att visa temperatur/luftfuktighet på företagets lokaler. Värdet för temperatur kan till exempel loggas med SNMP (Simple Network Management Protocol) som då gör att data inkommer i realtid. Vid detta fall skulle det vara användbart att använda AJAX eller WebSockets för att uppdatera datan i dashbaorden.

Referenser

[1] M. Dashorst, E. Hillenius, *"Wicket in Action,"* Greenwich, Manning Publications CO., 1-932394-98-2, 2009

[2] What is Wicket , hämtad 16 maj 2017, https://cwiki.apache.org/confluence/display/WICKET/Index - Index-WhatisWicket?

[3] What is an ORM, hämtad 16 May 2017, http://hibernate.org/orm/what-is-an-orm/

[4] HTML5 Canvas Speedometer, hämtad 16 maj 2017, http://www.knowstack.com/html5-canvas-speedometer/

[5] Draw.io, drawing tool, använd 16 May 2017 https://www.draw.io/

[6] Apache Commons Math, hämtad 16 maj 2017, http://commons.apache.org/proper/commons-math/

[7] Google Chart, hämtad 16 maj 2017, https://developers.google.com/chart/

[8] Wicket Ajax, hämtad 16 maj 2017, https://cwiki.apache.org/confluence/display/WICKET/Wicket+Ajax

[9] Wicket Native WebSockets, hämtad 16 maj 2017, https://cwiki.apache.org/confluence/display/WICKET/Wicket+Native+WebSockets

[10] Shwetank Dixit, An introduction to WebSockets, hämtad 16 maj 2017, http://www.developerfusion.com/article/143158/an-introduction-to-websockets/

[11] EDI basics, hämtad 16 maj 2017, http://www.edibasics.com/

[12] S. Few, *"Information Dashboard Design : The Effective Communication of Data,"* Sebastopol CA, O'Reilly & Associates, 9780596100162, 2006

[13] Hibernate ORM, hämtad 18 maj 2017, http://hibernate.org/orm/

[14] Hibernate documentation, hämtad 18 maj 2017, http://hibernate.org/orm/documentation/5.2/

[15] HTML5 Canvas, hämtad 18 maj 2017, https://www.w3schools.com/html/html5_canvas.asp

[16] M. Haverbeke, *"Eloquent JavaScript: A Modern Introduction to Programming,"* San Francisco CA, No Starch Press Inc., 978-1-59327-282-1

[17] AJAX intro, hämtad 19 maj 2017, https://www.w3schools.com/xml/ajax_intro.asp

[18] About WebSocket, hämtad 19 maj 2017, https://www.websocket.org/aboutwebsocket.html

[19] H. Alsén, Tyringekonsult AB, beskrivning av Tyringekonsults system, email 12 maj 2017

[20] A. Rauschmayer, *"Speaking JavaScript: An In-Depth Guide for Programmers,"* Sebastopol CA, O'Reilly Media Inc., 978-1-4493-6498-4

[21] R. Hawkes, *"Foundation HTML5 Canvas,"* Ipswich MA, Apress, 978-1-4302-3292-6

[22] C. Ullman, L. Dykes, *"Beginning AJAX,"* Indianapolis IN, Wiley Publishing Inc., 978-0-470-10675-4

[23] A. Kirk, *"Data Visualization: a successful design process,"* Birmingham UK, Packt Publishing Ltd., 978-1-84969-346-2

[24] T. Munzer, *"Visualization Analysis & Design,"* Boca Raton FL, CRC Press, 978-1-4665-0893-4

[25] K. Alexanderson, *"Källkritik på Internet,"* Ödeshög, Danagårds LiTHO, 978-91-979411-3-6

[26] J. Karlström, *"The WebSocket protocol and security: best practices and worst weaknesses,"* Networked Digital Library of Thesis & Dissertations, EBSCOhost, hämtad 27 maj 2017

Figurtabell

Tabellista

Appendix A: Prototyper

CPU #1

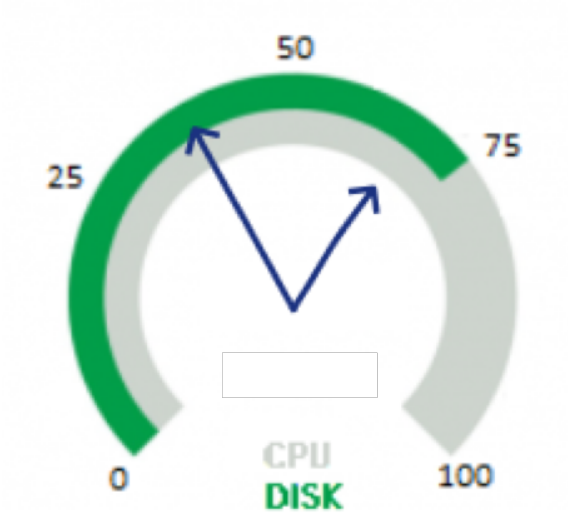

Fig. 26. Prototyp #1 CPU

CPU #2

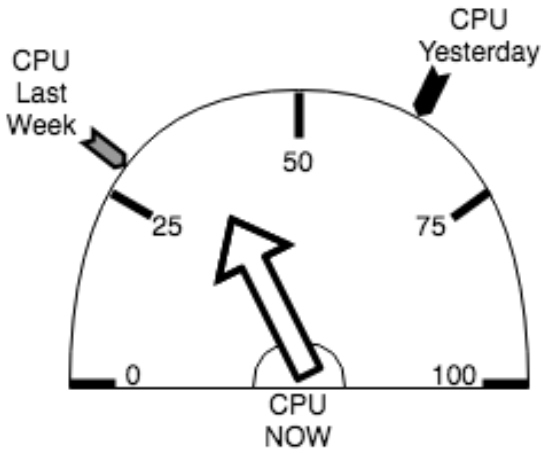

Fig. 27. Prototyp #2 CPU

CPU #3

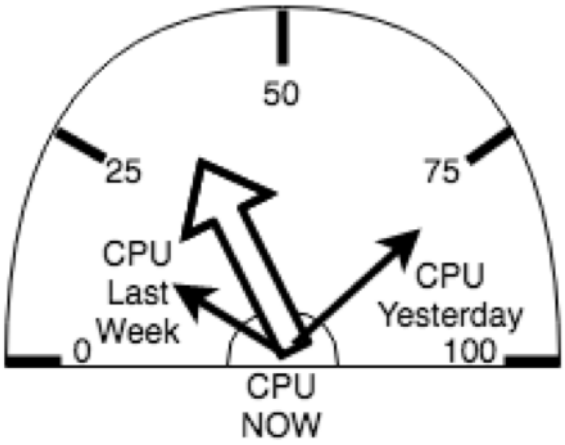

Fig. 28. Prototyp #3 CPU

Disk #1

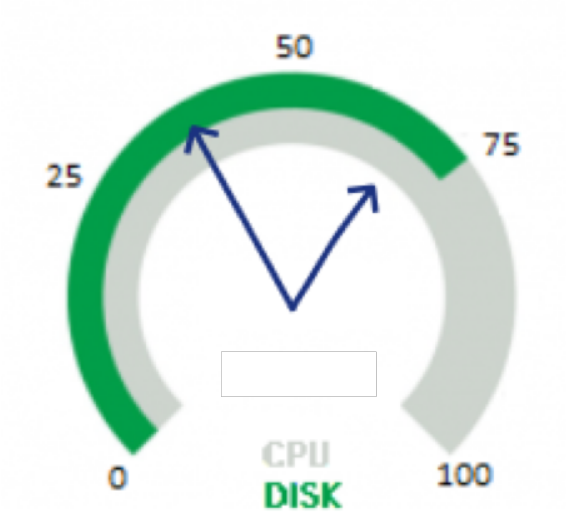

Fig. 29. Prototyp #1 disk

Disk #2

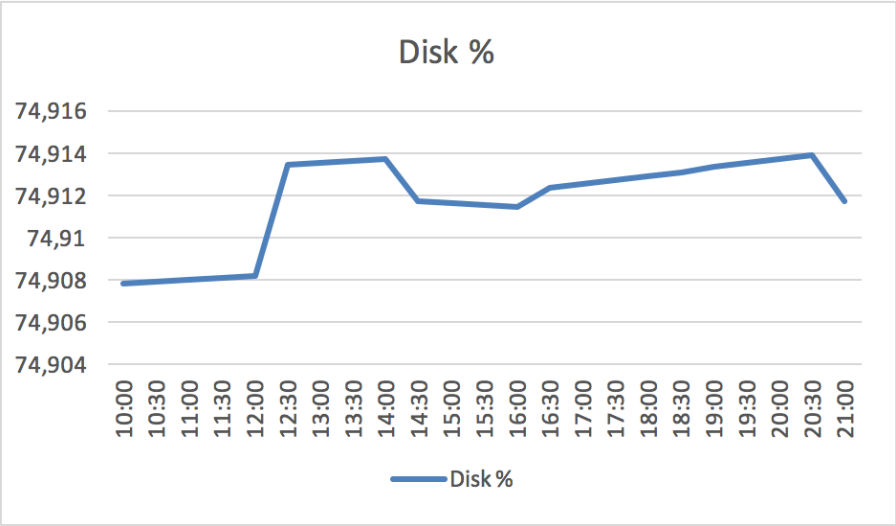

Fig. 30. Prototyp #2 disk

Disk #3

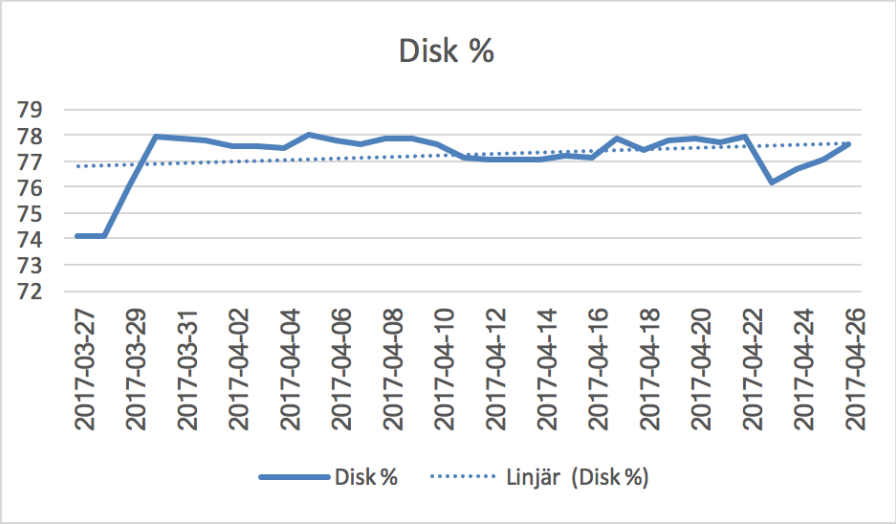

Fig. 31. Prototyp #3 disk

Filer & Dokument #1

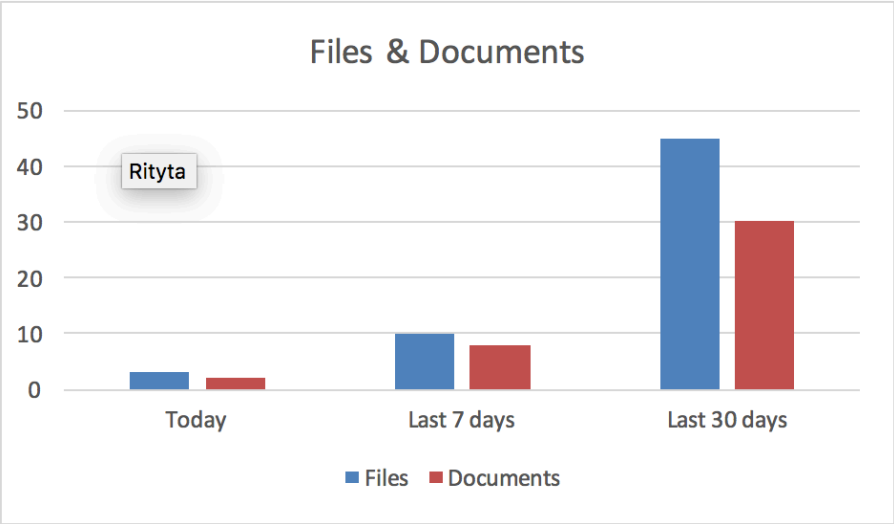

Fig. 32. Prototyp #1 filer & dokument

Filer & Dokument #2

Type	*Today*	*Last 7 days*	*Last 30 days*
Files	3	38	871
Documents	2	48	659

Fig. 33. Prototyp #2 filer & dokument