CHALMERS


# Sensor i konferensrum

## Kontorsklimat med Smart Citizen Kit

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

ALEXANDER EKSTRAND

VIDAR ÅSBERG

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

**EXAMENSARBETE HÖGSKOLEINGENJÖR 2015**

# Sensor i konferensrum

Kontorsklimat med Smart Citizen Kit

**ALEXANDER EKSTRAND**
**VIDAR ÅSBERG**

**Institutionen för data- och informationsteknik.**
**Chalmers tekniska högskola**
**Göteborg, Sverige 2015**

**Sensor i konferensrum**
Kontorsklimat med Smart Citizen Kit

Alexander Ekstrand, Vidar Åsberg


Examinator: Lars Svensson

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

Omslag:
Smart Citizen Kit
fotograf: Alexander Ekstrand

Institutionen för data- och informationsteknik
Göteborg, 2015

# Abstract

The aim of this project was to monitor air quality and the indoor climate in Sigma IT & Managements meeting rooms. The sensor used is developed by the Kickstarter project Smart Citizen. It continuously reads data such as temperature, light, noise, and carbon monoxide. The project was conducted with the working method Scrum. The project resulted in an entire system containing a sensor, database, website and web server with support for the installation of more sensors. The system stores sensor data collected from the rooms in the database. This information is then visualized on a website for employees at Sigma to consult before they book a meeting room. This provides the employees with the ability to easily decide which room has the best conditions for a meeting. The employees at Sigma also answered a survey, it served as a basis for what information about a room is interesting. The survey also answers if and how the data can be effectively displayed.

**Keywords**: Sigma, Sensor, Database, Webserver, Smart Citizen, Website, Conference Room

# Sammandrag

Detta projekt syftade till att övervaka inomhusklimatet och luftkvalitén i Sigma IT & Managements konferensrum. Sensorn som används är ett resultat av Kickstarter projektet Smart Citizen. Den läser kontinuerligt av data såsom temperatur, ljus, ljud, samt kolmonoxid. Genomförandet av projektet utfördes med arbetsmetoden Scrum. Projektet resulterade i ett helt system innehållande en sensor, databas, webbplats samt webbserver med stöd för installation av flera sensorer. Systemet sparar sensorernas insamlade data från rummen i databasen. Denna information visualiseras sedan på en webbplats för de anställa på Sigma att iaktta innan de bokar ett konferensrum. Detta för att de anställda ska ha möjlighet att enkelt avgöra vilket rum som har de bästa förutsättningarna för en konferens. Även en enkätundersökning genomfördes för de anställda på Sigma för att få svar på vilken typ av information om ett rum de ansåg som intressant. Enkäten svarade också på om och hur datan kan visas på ett effektivt sätt.

**Nyckelord**: Sigma, Sensor, Databas, Webbserver, Smart Citizen, Webbplats, Konferensrum

# Förord

Examensarbetet omfattar 15 högskolepoäng vilket motsvarar 10 veckor och är en del i att ta examen vid Chalmers Tekniska Högskola. Arbetet har utförts på Sigma IT&Management på Lindholmspiren 9.

Vi tackar våra handledare Rashwan Lazkani och Joakim Jonsson på Sigma samt Håkan Burden från Chalmers. Extra tack går till vännerna Oskar Selberg, Marcus Thorström, Anton Ekberg och Sebastian Ekstrand för goda råd.

# Definitioner

| | |
|---|---|
| **API** | Application Programming Interface |
| **Arduino** | Open-source electronics platform |
| **Balsamiq** | Verktyg för att göra mockups |
| **CSS** | Cascading Style Sheets, Stilmall för HTML och XML dokument |
| **Databas** | Organiserad data |
| **ER-diagram** | Visuell överblick på hur en databas är uppbyggd |
| **FTP** | File Transfer Protocol |
| **FTPS** | File Transfer Protocol Secure |
| **HTML** | Hyper Text Markup Language |
| **HTML5** | Senaste standarden för HTML |
| **HTTP** | HyperText Transfer Protocol, Kommunikationsprotokoll för att skicka och ta emot data på webbplatser |
| **HTTP Header** | Del av ett paket som skickas via HTTP, som beskriver innehållet |
| **HTTPS** | HyperText Transfer Protocol Secure |
| **Klient** | Beställare eller mottagare av en tjänst |
| **Kodskellett** | Grundfundament till program som logik läggs ovanpå |
| **Mikrokontroller** | Liten dator som innehåller processor, arbetsminne och programminne |
| **Mockup** | Enkel prototyp av en produkt som ska skapas |
| **Plugin** | Insticksmodul till program med utökad funktionalitet |
| **PPM** | Parts per million |
| **Ramverk** | En färdig lösning för ett problem som kan återanvändas |
| **REST** | Representational State Transfer. En server tillhandahåller information via HTTP-anrop |
| **SCK** | Smart Citizen Kit |
| **Server** | Dator i ett klient-server förhållande som delar data |
| **SMHI** | Sveriges meteorologiska och hydrologiska institut |
| **SMTP** | Simple Mail Transfer Protocol |
| **Sprint** | En av flera cykler i projekt där man använder scrum |
| **SQL** | Structured Query Language |
| **URL** | Uniform Resource Locator, en webbadress |
| **Webbapplikation** | Programvara som går att använda i en webbläsare |
| **Webbläsare** | Program som hämtar filer skrivna i HTMP eller XHTML från webbservrar, för att sedan tolka och presentera dessa |
| **Webbplats** | Synonym till hemsida |
| **Webbserver** | Dator specialanpassad till att ta emot och skicka data |
| **XML** | Extensible Markup Language |

# Innehållsförteckning

# 1 Introduktion

Det ligger numera i tiden att överallt samla och bearbeta data i så stor utsträckning som möjligt. Insamlingen kan genomföras med många olika metoder så som enkäter, sensorer, observationer, etc. Några allmänna exempel på detta är SMHIs sensorer och satelliter vars uppgift är att ackumulera data som sedan används för att förutspå framtida temperaturer och väderförhållanden. Med tiden har datainsamling blivit automatiserad och sker digitalt, jämfört med förr då det ofta var människor på plats som hade ansvaret för avläsning av mätinstrument. Datorer, sensorer och kameror är outtröttliga och kan mäta och skicka data dygnet runt. Dessa maskiner har blivit effektivare, mer tillförlitliga och mindre i fysisk storlek för varje år. Runtom i världen sätts hela nätverk av sensorer upp i kontor, hem och industrier.

Miljöfrågan är ett av många viktiga frågor människan idag försöker lösa. Tessa Daniel, Elena Gaura och James Brusey från University of Trento har undersökt detta i en akademisk uppsats där sensorer är uppsatta i passivhus för att lätt kunna se skillnader i olika sorters passivhus [1]. Detta för att kunna se vilka som är effektivaste och mest miljövänliga. De skriver att sensorer är bra för att samla data. Men används de för rätt ändamål så förvandlas sensorn från en datasamlare till en sensor som genererar relevant information som sedan går att använda. De föreslår att sensornätverk placeras i framtida husbyggen för att kunna jämföra vilka metoder som medför de mest miljövänliga husen.

Davide Brunelli, Ivan Minakov, Roberto Passerone, Maurizio Rossi från Coventry University har sett att trådlösa sensorer är både billigare och mer mångsidig än traditionellt trådade lösningar [2]. I deras projekt sattes ett nätverk av 19 sensorer upp som kontinuerligt mäter vibrationer, temperatur, luftfuktighet, ljus och koldioxidnivåer på arbetsplatsen. Projektets syfte var att hitta en balans mellan ett trivsamt inomhusklimat och energikonsumtionen. Månader av tester visade att systemet var både effektivt och applicerbart. Vidareutvecklingen som pågick efter rapporten var att sätta upp fler sensorer och därmed täcka mer av kontoret. Även algoritmer för automatisk analysering av insamlad data och rörelsedetektion höll på att utvecklas.

Detta projekt kommer kretsa kring en sensor vid namn Smart Citizen Kit[3]. Denna sensor kan mäta bland annat temperatur och luftfuktighet. Målet med projektarbetet är att med en webbplats visualisera datan den samlar in på ett informativt sätt.

## 1.1 Syfte

Sigma är ett företag som har som mål att vara i framkant både när det kommer till teknologi och effektivitet. Om sensorer placerades i alla rum skulle det finnas information i realtid om vilka rum som har mest behaglig temperatur, luftfuktighet etc. Examensarbetets syfte är att skapa en tjänst som gör det enkelt för de anställda att få en översikt om var rummet med önskvärda förhållanden finns.

## 1.2 Mål

En sensor ska samla och skicka data kontinuerligt till en webbserver som bearbetar datan för att sedan spara den i en databas. Webbservern ska även ha en webbsida som visualiserar datan i en graf, där det kan väljas hur långt bak i tiden samt vilken typ av data som ska presenteras. Ett mål är att ta reda på vilken typ av data som är intressant och hur den bäst presenteras.

## 1.3 Frågeställningar

- Vilka faktorer är intressanta vid bokning av konferensrum?
- Hur kan Smart Citizen Kit integreras i ett system som övervakar inomhusklimat?
- Hur lämplig är Smart Citizen Kit för datainsamling i en kontorsmiljö?

## 1.4 Avgränsningar

I examensarbetet används endast en sensor i ett rum och datan kommer visas upp på en webbplats. Systemet är endast ämnad för de anställda på Sigma. Sensorn som kommer användas är Smart Citizen Kit version 1.1.

# 2 Teknisk Bakgrund

Detta kapitel redogör för utvecklingsmiljö, programmeringsspråk och andra verktyg som användes under utvecklingen av projektet.

## 2.1 Visual Studio 2013

Visual Studio 2013 är en av de senaste versionerna av en lång rad släpp sedan år 1997 då det första Visual Studio lanserades av Microsoft. Det är en utvecklingsmiljö för operativsystemet Microsoft Windows med avancerade tillämpningar, så som att utveckla webbapplikationer och PC-baserade applikationer. Som utvecklare finns möjligheten att skriva och kompilera kod i språken Visual Basic, Visual C++, Visual C#, Visual J# och Visual F# [4].

## 2.2 C#

C# (C-sharp) är ett objektorienterat programspråk liknande Java. Språket är utvecklat av Microsoft och baserat på C++. C# är plattformsoberoende, vilket betyder att program som skrivs med detta verktyg ska kunna fungera på oberoende av plattform. C# är likt Java, vilket underlättar för personer med Java-erfarenhet att gå över från Java till C# och vice versa. Utvecklare är inte i behov av licenser för utvecklingsverktyget [5].

## 2.3 ASP.NET

ASP.NET är ett ramverk designat för webbutveckling, där ASP står för Active Server Pages. Det används främst för att skapa dynamiska webbplatser, vilket till exempel innebär att en webbsida interagerar med användaren och presenterar efterfrågad data. ASP.NET är utvecklat av Microsoft och skrivs i språk som är kompatibla med .NET-ramverket, till exempel C#. Ramverket möjliggör även kommunikation med en Microsoft SQL Server. Första gången en ASP.NET-webbsida besöks av en användare kompileras koden och sparas i serverns cacheminne. Detta innebär snabbare laddning nästa gång den besöks [6], [7].

## 2.4 Microsoft SQL Server

Microsoft SQL Server är ett databashanteringssystem utvecklat av Microsoft. Detta är en mjukvara med den primära uppgiften att kunna lagra och dela med sig av data som är efterfrågad av applikationer. För att komma åt data på en Microsoft SQL Server används Transact-SQL [8].

## 2.5 MVC

MVC står för Model-View-Controller och är ett designmönster som används i systemutvecklingen av större och mer komplicerade mobil- och dator-applikationer för att göra dem modulära och lättare att förstå. I mönstret separeras data (Model) och presentation (View) då det inte riskerar att datahanteringen får konsekvenser när något ändras i presentationen. Mellan presentationen och datan finns logik som kontrollerar utbytet (Controller) [9].

## 2.6 Code first entity framework

Code first entity framework kallas den metod som utvecklare använder för att skapa en helt ny databas eller skapa nya tabeller i en existerande databas. Detta genom att skriva modellklasser, som det sedan genereras en databas utav. Vilket medför många fördelar, då utvecklare kan fokusera på kodningen och inte på relationerna i databasen [10].

## 2.7 Git

Git är ett system för versionshantering. Det innebär det att efter varje ny tillagd funktion kan dokument, webbsidor, program eller källkod sparas i en egen version. Detta leder i sin tur till att om en bugg skapas som inte kan lösas i projektgruppen så finns det möjlighet att återskapa en tidigare version av mjukvaran. Andra fördelar med versionshantering är att det finns möjlighet till att spåra var och när olika ändringar gjordes i projektet. Det finns även möjlighet till parallell utveckling, där till exempel två personer kan arbeta på samma projekt men i olika filer. Detta medför stora fördelar för större projekt med många personer [11].

## 2.8 JSON

JSON står för JavaScript Object Notation och är ett plattformsoberoende sätt att paketera dataobjekt för överföring. Det är lätt att läsa både för en människa och för en dator, varför det har blivit populärt och det finns väldokumenterade standardlösningar för användning av det [12].

## 2.9 JSON.NET

JSON.NET är ett populärt tredjepartsramverk för Visual Studio för hantering av JSON. Ramverket är utvecklat av Newtonsoft [13].

## 2.10 Arduino

Arduino är en hel plattform av mikrokontrollerkort där både hård- och mjukvara är öppen. På korten kan ytterligare kort läggas för att ge dem utökad funktionalitet. Detta gör Arduino användbar för många olika lösningar. Hårdvaran programmeras med språket wire som påminner starkt om C och C++ [14].

## 2.11 Wiring

Wiring är ett ramverk för C och C++ med öppen källkod för programmering av mikrokontrollerkort. Det finns en gemenskap på internet där både nybörjare och experter delar med sig av sina erfarenheter och problem. Wiring förenklar utveckling av mjukvara till kontrollenheter som är anslutna till kretskort [15].

## 2.12 Smart Citizen

Smart Citizen är ett projekt [16] från ett spanskt företag, Fab Lab Barcelona, som har startat ett Kickstarter projekt [17] där en sensor är en del av projektet. Tanken är att de som köper en sensor kopplar upp den mot Smart Citizens server. Därefter skickar sensorn kontinuerligt sina avläsningar. Datan syns sedan på deras webbplats “smartcitizen.me”. Själva sensorpaketet heter Smart Citizen Kit och förkortas SCK [3]. Det består av en Arduino-enhet (avsnitt 2.11) som med hjälp av sensorer möjliggör avläsning av data. Paketet är nu inne på andra generationen (1.1) med nya delar och optimeringar mot första generationen (1.0). Datan som det senaste kortet kan mäta är temperatur, luftfuktighet, ljud, ljus, kolmonoxid, kvävedioxid och antal tillgängliga WiFi-nätverk.

## 2.13 IIS7

IIS står för Internet Information Services och är utvecklat av Microsoft. IIS är en serverprogramvara för tjänster som är internetbaserade. IIS är kompatibel med nyare versioner av Windows Server, med stöd för HTTPS, HTTP, FTP, FTPS och SMTP. Den har varit en del av Windows NT familjen sedan Windows NT 4.0 [18].

## 2.14 JavaScript

JavaScript är ett objektorienterat programmeringsspråk, mest känt som ett skriptspråk på webbplatser men det är inte begränsat till det utan kan användas i alla möjliga applikationer. Ofta körs JavaScript-kod hos klienten som besöker sidan och inte på servern, vilket bland annat gör att konstant uppkoppling inte krävs från webbplatsen för fullständig funktionalitet [19],[20].

## 2.15 Ajax

Ajax står för Asynchronous JavaScript and XML och är ett samlingsbegrepp för JavaScript som asynkront hämtar data till klienten med HTTP anrop. Det är ett skript som körs på klientsidan, som kommunicerar med en server/databas utan behovet att ladda om webbplatsen [21].

## 2.16 jQuery

Ett av flertalet JavaScript bibliotek är jQuery. Detta är ett funktionsrikt bibliotek som sköter händelsehantering, animering och Ajax-anrop mycket enklare med ett API som fungerar med ett stort antal webbläsare [22].

## 2.17 ChartJS

ChartJS är ett gratis JavaScript bibliotek för att skapa grafer och har sin betoning i att vara enkelt och att graferna ritas på en HTML5 canvas [23].

## 2.18 Postman - REST Client

Postman är ett Google Chrome plugin för att skicka olika HTTP anrop, som till exempel POST, PUT, GET och DELETE. Postman är ett användbart verktyg för utvecklare som behöver felsöka sina applikationer [24].

## 2.19 Balsamiq

Balsamiq är ett verktyg som används på datorer. Verktyget används för att skapa en konceptdesign för användargränssnittet på ett system, applikation eller hemsida. När en konceptdesign är klar kallar man denna för en mockup.

## 2.19 Migrations

Med hjälp av en funktion i Visual Studio är det möjligt att generera en lokal databas utifrån en modellklass. Där klassen har olika variabler som har metoderna get och set för att kunna läsa och ändra variablernas innehåll. När databasen genereras blir klasserna tabeller, innehållande attribut som är klassens variabler. Ändras sedan modellerna i koden så måste även databasen matcha dessa. För att uppnå detta används migrations. Det betyder i praktiken att databasen uppdateras med de nya modellerna; detta kan innebära att hela tabeller eller attribut antingen läggs till eller tas bort. Den tidigare databasen skrivs alltså över med en ny. Det finns även möjlighet att generera en tidigare version av databasen vilket gör att det blir en typ av versionshantering [25].

## 2.20 Razor

Vid utveckling av en webbplats finns det ofta önskemål att använda kod som är skriven i ett annat språk än HTML eller CSS. Razor syntax är ett sätt att lösa detta med. Om programmeraren inleder ett kodstycke med ett “@”-tecken läser webbplatsen detta som C# kod istället för HTML-kod. Detta är användbart om funktioner på serversidan ska användas. Ett exempel är @DataTime.Now som ger en sträng med den nuvarande servertiden som kan visas på HTML-sidan [26].

# 3 Metod

Detta kapitel redogör för hur projektet är tänkt att genomföras, med vilken arbetsmetod som användes och hur lösningsförslag skapades utifrån kravspecifikationen från Sigma.

## 3.1 Kravspecifikationen

Sigma efterfrågar ett system som samlar och sparar data från deras kontor, som även ska visualiseras på en webbsida. Detta för att anställda på Sigma enkelt ska kunna se information om klimatet i de olika konferensrummen.

Här följer de ursprungliga kraven på systemet i punktform:

Sensor

- Sensorn ska kunna skicka sin information till Smart Citizens system

Databas

- Ska kunna lagra information  av olika sensorer
- Ska kunna lagra poster för olika sensorer

Webbserver

- Ska kunna hämta sensordata med hjälp av Smart Citizens REST-API
- Ska kunna spara datan i en databas

Webbplats

- Ska ha stöd för administratörer
- En administratör ska kunna skapa nya administratörer
- Ska kunna visa information som sensorer har samlat

## 3.2 Arbetsmetod

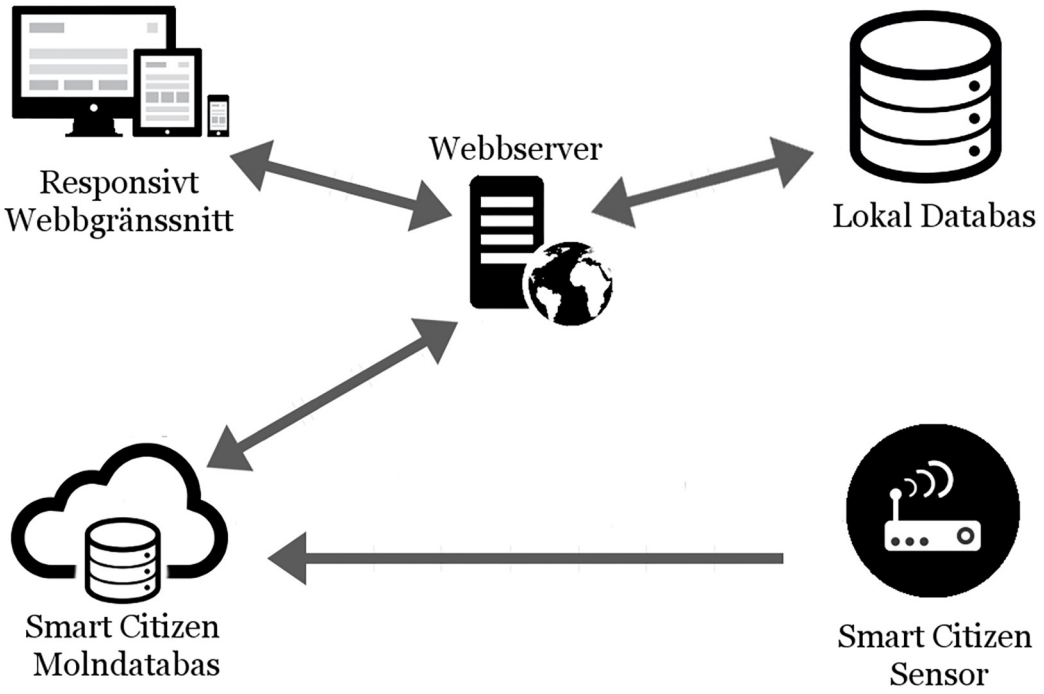
Scrum [27],[28] kommer vara arbetsmetoden för arbetet. ScruM är en agil arbetsmetod som ofta används för systemutveckling. Det innebär att gruppen har en “backlog” att arbeta efter. “Backlogen” består av alla olika “features” som ska finnas i produkten när den är färdig. Arbetet delas upp i “sprints”, som är korta arbetsperioder, ofta två veckor, där det på förhand är bestämt vad som ska göras. Sprints används för att enkelt kunna implementera ny funktionalitet eller förändra inriktning under projektets gång. Sigma som är produktägare använder själva metoden, men även projektgruppen har erfarenhet av det från tidigare projekt. Scrum kommer underlättas av hjälpmedlet Trello [29] som är en virtuell anslagstavla där arbetet delas upp i virtuella lappar. En ny idé ger en ny lapp. Varje sprint börjar med en sprint-planering där gruppen tillsammans med representanter från Sigma beslutar om vad som ska göras under de två nästkommande veckorna. Mot slutet av varje sprint ska en sprint-granskning hållas, även den med Sigma, där sprinten utvärderas, och till sist en sprint “retrospective” där utvecklarna diskuterar vad som kan förbättras i arbetssättet till nästa vecka.

Bitbucket [30] ska användas för versionshantering och delning av programkod. Likt det välkända Github är det en git-klient, men med möjligheten att skapa ett obegränsat antal privata projekt och dessutom utan kostnad. Efter varje ny fungerade funktion läggs koden i molnet på bitbucket. Ifall det i efterhand upptäckts fel i koden finns därmed möjlighet att gå tillbaka till en fungerande version. I första sprinten ska utvecklingsmiljö och git sättas upp och gruppen tittar närmre på C#, .NET och MVC genom en kurs på Microsoft Virtual Academy [31] så att gruppen redan till nästkommande sprint är redo att börja med projektet på allvar.

En undersökning i form av en enkät ska genomföras för att fråga de anställda på Sigma vilka faktorer de anser vara intressanta vid bokning av konferensrum. Enkäten kommer skrivas i Google Forms [32] och deltagarna kommer få en länk till den via Epost.

## 3.3 Lösningsförslag

Utifrån kravspecifikationen har ett lösningsförslag gjorts.




*Figur 3.1. Lösningsförslag*

Den data som ska användas produceras av Smart-Citizen-sensorn. Denna data lagras i molnet, där hämtning sker genom ett get-anrop innehållande en nyckel från Smart Citizen och id-numret för sensorn. Svaret ges i form av en JSON-array, från vilken det är möjligt att ta ut den data som önskas för just den sensorn. Den lokala databasen kommer att innehålla samma data som den molnbaserade databasen som Smart Citizen äger. Detta genom att undersöka kontinuerligt om någon ny data finns, och isåfall hämta denna och spara i den lokala databasen. Det responsiva webbgränssnittet kommer att presentera datan från sensorerna både i realtid och i grafer som kan visa historik för ett givet tidsintervall.

Webbgränssnittet kommer att utvecklas med utgångspunkt att det ska vara en mobilanpassad webbplats. Fokus ligger alltså primärt på mobilgränssnitt och därefter utveckla för större enheter så som plattor, laptops och stationära datorer.

### 3.3.1 Databas

Databasen är en SQL-databas med två tabeller, Sensors och Posts. Sensors har all information om en specifik sensor och har id som nyckel. Posts har timestamp som primär nyckel och sensorID som främmande nyckel (en. foreign key). I tabellen sparas all data sensorn samlar.

### 3.3.2 Webbserver

Webbservern kommer att vara skriven i ASP.NET med ett MVC-mönster, med kopplingar till både webbgränssnitt, lokala databasen och SmartCitizens molndatabas. Den kommer kontinuerligt hämta data från molndatabasen och placera denna data in i vår lokala databas för att det ska vara enklare för webbgränssnittet att komma åt informationen.

### 3.3.3 Webbplats

På webbplatsen ska informationen presenteras så att Sigmas medarbetare med hjälp av datan enklare ska kunna få en bild av hur lämpligt det är att till exempel hålla en konferens i ett visst rum. All data ska visualiseras med hjälp av grafer på webbplatsen, så att det ska vara möjligt att se historik på hur det har varit förr jämfört med nutid.

# 4 Val av verktyg

Detta kapitel redogör för motiveringar bakom vilka verktyg som valdes för projektet.

## 4.1 PHP eller ASP.NET

PHP och ASP.NET är båda populära och vanliga verktyg vid webbutveckling. Båda är verktyg för kommunikationen mellan databas och en tjänst t.ex hemsida. De har även kompatibilitet med den typ av databas som vi använde oss av under detta projekt. Efter undersökning av de båda systemen fick vi fram att PHP har en minimal prestandafördel vid hämtning av data ifrån databasen, men det finns ingen märkbar skillnad. Med tanke på att examensarbetet utfördes på Sigma, där det finns ett antal utvecklare med erfarenhet inom ASP.NET, så fanns det mer assistans att få i ASP.NET än om vi skulle utvecklat i PHP. En skillnad mellan de två systemen var att PHP inte hade trådning vilket medför att webbsidor inte kan laddas dynamiskt medan i ASP.NET var det inbyggt.

## 4.2 Webbhotell, Azure eller lokal server

Backenden i ett system, alltså webbserver och databas, behöver vara aktiv hela tiden. Därför läggs de antingen på webbhotell eller en lokal server. Mindre företag eller privatpersoner använder ofta webbhotell då priset är anpassat efter hur mycket datatrafik som går åt. På samma sätt är det om molnplattformen Microsoft Azure användes. Fördelen med webbhotell och Azure är skalbarhet medan en uppenbar nackdel är att det är en extra kostnad för systemet. Hade projektet varit utan stöd från Sigma hade någon form av webbhotell använts då ingen av utvecklarna har tillgång till någon lokal server; men eftersom Sigma hade en lokal server att tillgå kunde denna användas. Servern var värd för ett antal webb-projekt redan vilket innebar att ingen ökad kostnad skulle komma av att den användes även för detta projekt.

## 4.3 Valda verktyg

Microsofts Visual Studio valdes för att systemet enkelt skulle passa in i Sigmas infrastruktur då majoriteten av Sigmas konsultuppdrag har använt utvecklingsmiljön. Det som utvecklas i projektet kan ligga till grund för oförutsedd användning och vidareutveckling. För att underlätta detta ska projektet vara modulärt. Där igenom blir det lättare att både förstå metoder och att implementera nya funktioner i systemet. MVC-mönstret är lämpligt för detta. Det fanns dessutom en mall för MVC-mönstret i Visual Studio. ASP.NET och Microsoft SQL är integrerade direkt i miljön. Projektet kommer använda Sigmas befintliga server. Det är en IIS7-server som även den är utvecklad av Microsoft. Eftersom det är samma företag bakom många av de valda verktygen borde det innebära ökad kompatibilitet mellan dem.

När Sigma inte kunde bistå med ett verktyg för versionshantering och eftersom projektet skulle vara privat så valdes Bitbucket. Anledningen var att det mer populära Github kräver att det betalas för en licens för att kunna skapa privata projekt.

Trello har använts med framgång på Sigma under tidigare projekt; därför rekommenderades det att det användes även under detta projekt. Med Trello behövs inga fysiska “backlogs”. Dessutom finns en lättöverskådlig historik som visar vem som gjort vad.

Från början var det meningen att projektet skulle använda sensorer som Sigma i Malmö utvecklat. När projektarbetet startades fanns tyvärr inte någon färdig sensor att använda. Handledarna beställde tidigt en sensor från Smart Citizen för att ersätta Sigmas sensorer.

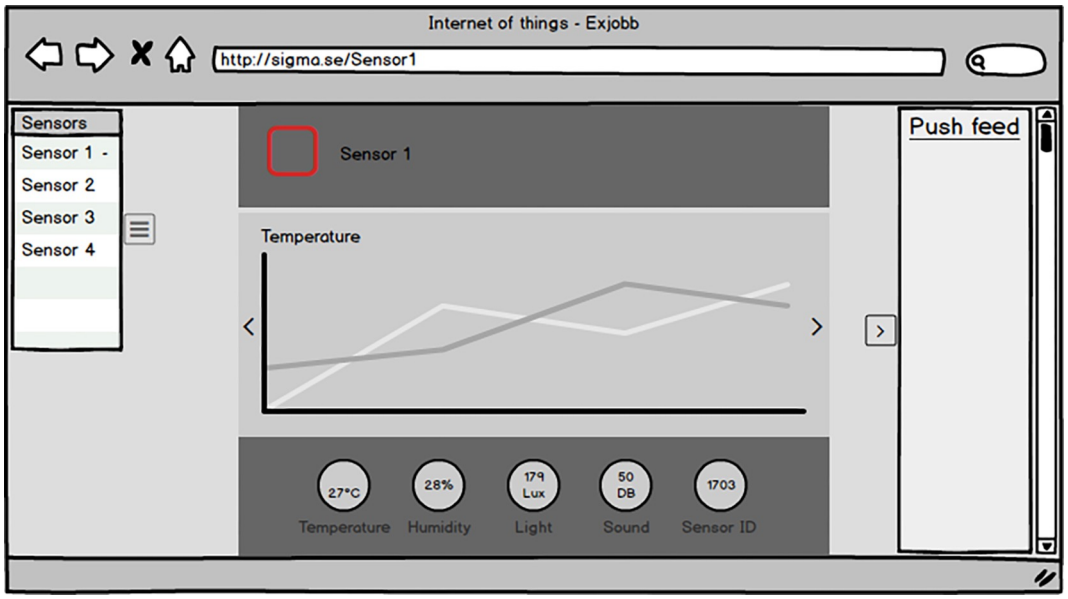
En graf är ett naturligt sätt att visualisera data. För att spara tid kan en färdig lösning för att visa grafer användas på webbsidan. Grafen behövde också kunna manipuleras för att visa olika typer av data. Det fanns många javascriptbibliotek som uppfyllde våra önskemål. Tyvärr krävde merparten av dem en licensavgift för att använda dem. På Sigma hade Chart.JS använts i andra projekt och krävde ingen licens. Det var ett lättanvänt javascriptbibliotek för grafer och verkade ha all funktionalitet som krävdes.

# 5 Genomförande

Detta kapitel redogör för implementationen av systemet.

## 5.1 Implementation

En utvecklingsmiljö sattas upp, men det behövdes en viss inlärningsperiod innan det var aktuellt att skriva kod. En mockup av webbplatsens användargränssnitt för både mobiler och datorer skapades i Balsamiq. Med en godkänd mockup av kunden, lades fokus på utformandet av databas och backend.



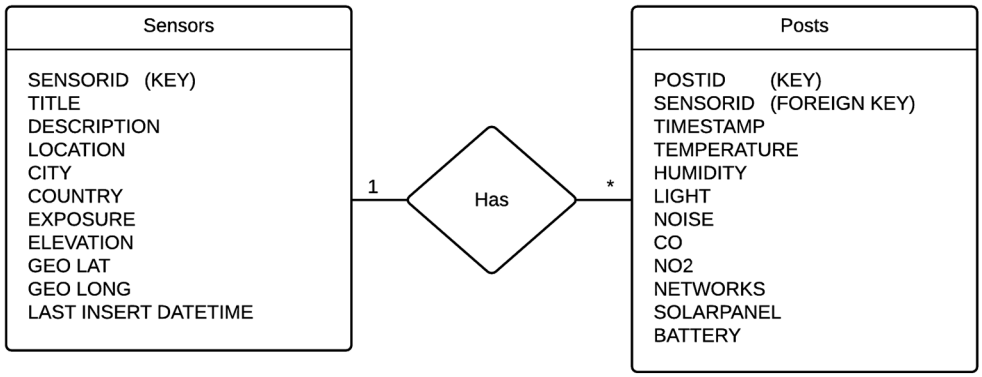

*Figur 5.1. Mockup i Balsamiq*

För att inte missa några essentiella detaljer i backend så granskades Smart Citizen Kits dokumentation - inte bara för att förstå utan främst för att hitta inspiration och idéer. Under de två första veckorna fanns inte fysisk tillgång till sensorn. Här utnyttjades istället Smart Citizens REST-API för att hämta data från vilken sensor som helst som är uppkopplad till SmartCitizens webbserver via ett HTTP GET-anrop. Id-numret för sensorn och mellan vilka tider data önskas skickas med i anropet. Svaret på anropet ser ut som i figur 5.2. Uppbyggnaden där låg till grund för utformandet av tabellerna i databasen.

```
{
"device":{
      "id":"1167",
      "title":"NavyWeather",
      "description":"Temporarily indoors",
      "location":"T\u00e4by,Sverige",
      "city":"T\u00e4by",
      "country":"Sverige",
      "exposure":"indoor",
      "elevation":"12.0",
      "geo_lat":"59.449164100000000",
      "geo_long":"18.120941399999992",
      "created":"2014-05-01 20:49:31 UTC",
      "last_insert_datetime":"2015-03-24 08:39:21 UTC",
      "posts":[
            {
            "timestamp":"2015-03-24 08:39:05UTC",
            "temp":17.6,     //Temperatur i Celsius
            "hum":38.4,      //Luftfuktighet
            "co":344.91,     //Kolmonoxid kOhm
            "no2":171.28,    //Kvävedioxid kOhm
            "light":741.6,  //Lux
            "noise":59.1,    //Decibel
            "bat":100,       //Batteri i procent
            "panel":0,       //Solpanel
            "nets":11,       //Antal synliga wifi-nätverk
            "insert_datetime":"2015-03-24 08:39:21 UTC"
             }
       ]
    }
}
```

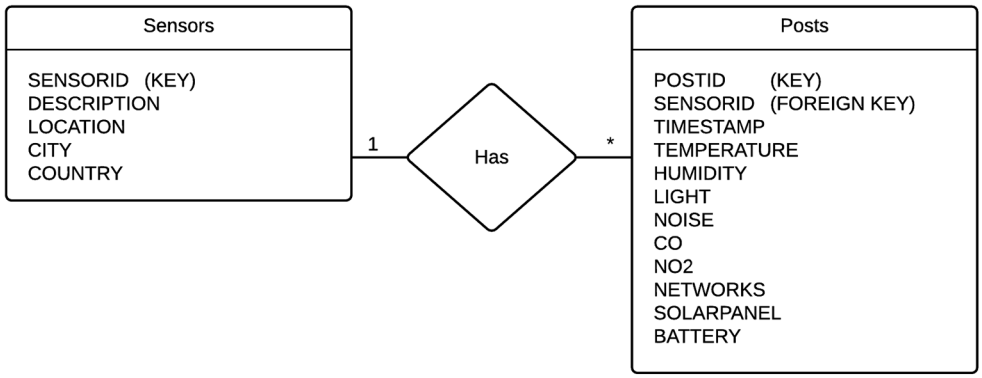

*Figur 5.2. JSON svar från servern*

Efter diskussion om tillvägagångssätt ritades ett ER-diagram för databasen upp som hade stora likheter med Smart Citizens ER-diagram. Databasen innehåller två tabeller, "Sensors" och "Posts". Sensors fick följande attribut: sensorID, title, description, location, city, country, exposure, elevation, geo lat, geo long och last insert datetime, med sensorID som primär nyckel. Posts hade även sensorID som främmande nyckel för att kunna se vilken sensor Posts tillhörde. Posts innehåller sin primärnyckeln postID och attributen timestamp, temperature, humidity, light, noise, co, no2, networks, solarpanel och battery.




*Figur 5.3. ER-diagram för första databasen*

I Visual Studio 2013 skapades ett nytt MVC-projekt med standardinställningarna för .NET framework 4.5. Då genererades ett kodskelett för en webbserver samt logik för inloggning och URL-länkningen med rätt controller och rätt metod. För att sedan skapa databasen användes Code First Entity Framework. Koden för modeller skrevs först, från modellerna genererade Visual Studio en databas. Den genererade databasen skapades först lokalt, där den lokala webbservern kopplades till databasen genom en textsträng i filen web.config, som förklarar för webbservern var den lokala databasen är skapad. Med databasen skapad upptäcktes att den innehöll onödiga attribut i sensors; dessa togs bort och databasen behövde genereras igen. Återgenereringen sköttes enkelt med migrations. Den nygenererade databasen innehöll en uppdatering, där attributen title, location, exposure, elevation, geo lat, geo long och last insert datetime från togs bort från tabellen sensors. Resultatet syns i figur 5.4.




*Figur 5.4. ER-diagram för den slutgiltiga databasen*

Efter testning av lokala databasen togs hjälp av utvecklare på Sigma för att sätta upp databasen på deras nätverk, eftersom arbetsgruppen inte hade någon erfarenhet av denna procedur.

Nästa steg var att skriva metoder för hämtning av data från Smart Citizens webbplats med ett HTTP GET anrop. Anropen gjordes när en användare tryckte på en knapp på webbplatsen. Då hämtades den senaste posten med data från Smart Citizens webbplats, som sedan visades upp på webbplatsen genom två olika flikar. Den ena hette sensorer, där all information om sensorerna gestaltades, den andra namngavs posts, där all information om varje post visualiserades. Problemet var att denna lösning inte var automatisk. Om ingen tryckte på knappen så sparades heller ingen data i databasen. Detta problem skulle kunna lösas med ett skript som konstant hämtade den senaste datan och jämförde tidstämpeln med den senast inlagda. Var tidstämplarna samma sparades den inte men om det var en ny lades den till som en ny post.

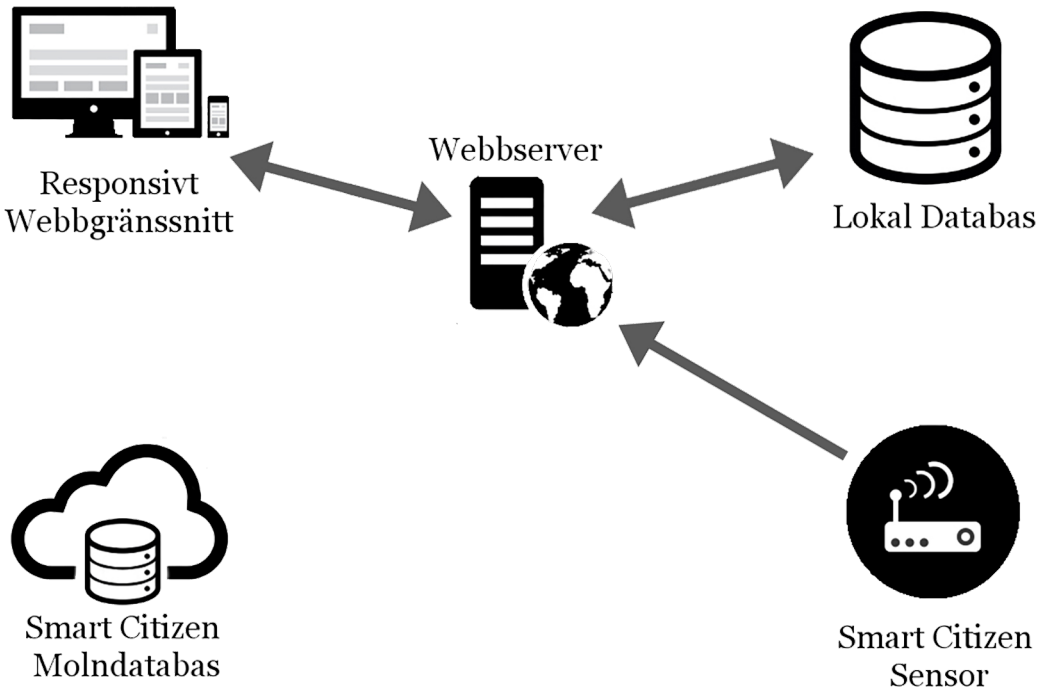
När sensorn levererades hade sensorns uppkopplande mot Smart Citizen hög prioritet. Instruktionerna på Smart Citizens webbplats följdes för att sätta upp sensorn [34]. Koden inspekterades och det upptäcktes att det var möjligt att få sensorn att skicka direkt till Sigmas server, istället för att skickas genom Smart Citizens system. Det var till synes bara att ändra URL-adressen för dit sensorn skickar sin data [35]. Till stor del var detta korrekt men det räckte inte för att servern skulle ta emot datan. För att utröna varför debuggades sensorn där det indikerades att datan skickades iväg. Men i serverloggen på Sigmas server, en Microsoft Server IIS7, fanns det ingen indikation på att något skickades in. Google Chrome-pluginet Postman användes för att testa att skicka data till servern. Postman är ett verktyg för att skicka HTTP-anrop. Detta verktyg användes för att skicka exakt samma data till servern som sensorn och då mottogs datan som förväntat. Detta fick oss att utesluta att det var fel på servern. Därmed konstaterades att det var sensorn som det var något fel på. Eftersom Sigmas serverlogg inte visade hur sensorns anrop såg ut behövdes en server utan dessa restriktioner. En lokal Python-server [36] sattes upp utan spärrar vilket gjorde att all trafik var synlig. Sensorn sattes upp i samma nätverk som servern för att skicka data till den.

När sensorn sedan skickade sin data till den lokala servern registrerades anropet och dess data på servern. Därefter användes Postman för att skicka in exakt samma data för att jämföra de två anropen. Det upptäcktes att Postman hade ett attribut i sin header som sensorn inte skickade, nämligen content-length. Attributet anger exakt hur stort paketet som levereras till servern är. Det

var därför Sigmas server inte tog emot paketen. IIS7 ger felmeddelande 413 när ett HTTP POST-anrop görs utan “Content-Length”. För att lösa detta skrevs en metod på sensorn som räknade storleken på paketet och lade till det i sin header. Metoden behövde vara snål i minnesanvändning då de redan existerande drivrutinerna fyllde i princip allt tillgängligt minnesutrymme.

Som standard var sensorn inställd på att mäta data och skicka den varje minut. Detta ändrades till var femte minut då det ansågs vara onödigt med ett tätare intervall. Trender hade större betydelse än varje enskild avläsning, dessutom krävs inte lika mycket lagringsutrymme.

Med den nya koden installerad på sensorn ser systemet ut som i figur 5.5 där beroendet av Smart Citizens system inte existerar mer, i och med att sensorn skickar sin data direkt till systemets egna server.




*Figur 5.5. Slutgiltig lösning*

Med sensorn integrerad i systemet påbörjades arbetet med webbplatsen och visualiseringen av datan. Från början var allting på webbplatsen helt öppet för alla som besökte sidan då projektet var i utvecklingsfasen. Senare implementerades så att användaren var tvungen att vara inloggad som administratör på webbplatsen för att se allting på webbplatsen. Då var det möjligt att se och redigera de olika sensorerna och posterna. Utan att vara inloggad var det bara möjligt att besöka startsidan och sidan med grafen. Grafen byggdes med Chart.JS och låg i en HTML5-canvas [37]. Denna fylldes sedan med data från webbservern med hjälp av Ajax-anrop. Om grafen skulle visa över hundra punkter blev det en märkbar laddningstid. För en timme tillbaka var detta inget problem då det alltid uppgår till maximalt tolv punkter. Ska ett större tidspann visas behövdes det en lösning. Att hoppa över poster som skickas från serversidan var en effektiv lösning. Det innebar att antalet poster som grafen behövde bearbeta reglerades så att det alltid är maximalt fyrtio punkter.

## 5.2 Enkät för anställda

Personalansvarig på Sigma skickade ett Epost-meddelande till alla 339 anställda i Göteborg. I meddelandet fanns en länk till enkäten och de ombads svara i mån av tid. Större delen av frågorna var flervalsfrågor och ja och nej frågor. En av frågorna ombad de svarande att rangordna vilken information de ansåg som mest relevant. Till slut gavs möjlighet att ge en kommentar eller ställa frågor. I enkäten var alla frågor förutom den sista obligatoriska. Enkäten hittas i appendix A.

## 5.3 Slutprodukt

Produkten som utvecklades resulterade i ett helt system, innehållande en sensor, en databas, en webbserver och en webbplats. I detta avsnitt kommer det redogöras för hur frontend och backend fungerar och ser ut.

### 5.3.1 Frontend

Startsidan för webbplatsen syns i figur 5.6. Överst en horisontell meny med alternativen hem, grafer och logga in. Under menyn hittas information om examensarbetet och kontaktuppgifter till utvecklarna.

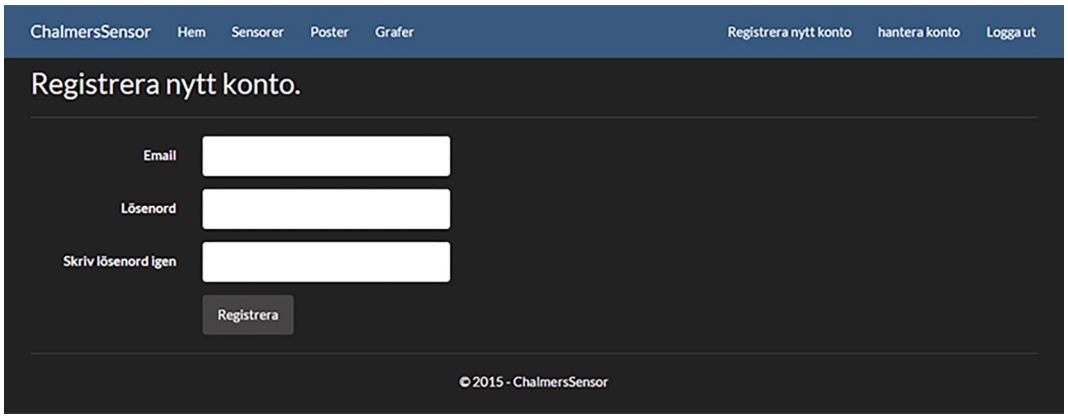

*Figur 5.6. Startsidan*

Ska användare logga in på webbplatsen görs detta under knappen logga in. Vid tryck på denna får användaren ange ett användarnamn och lösenord för att logga in. Användaren har möjligheten att låta webbläsaren minnas uppgifterna och automatiskt logga in nästa gång webbplatsen besöks. När användaren är inloggad har hen möjligheten att även besöka sidorna sensorer, posts, registrera nytt konto och hantera sitt konto, vilket syns i menyfältet i figur 5.7. “Hantera konto” tar användaren till en sida där det är möjligt att byta lösenord på sitt konto.


*Figur 5.7. Startsidan efter inloggning*

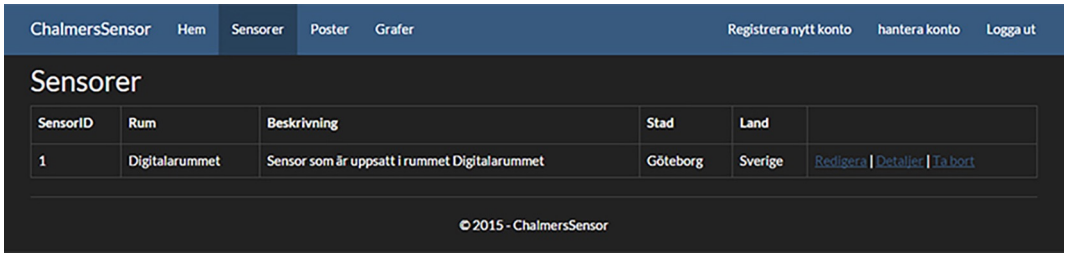
Det är inte möjligt att registrera ett nytt konto utan att vara inloggad. Detta innebär att konton registeras av en administratör som är inloggad. Administratören klickar på “registrera nytt konto” knappen i menyfältet, och ser då sidan i figur 5.8. Användarnamnet för det nya kontot fylls i som

Email. För att kontot ska kunna skapas måste även ett lösenord ges i de två rutorna under Email-adressen. För att fullborda kontoskapandet krävs det även att det trycks på knappen “Registrera”.



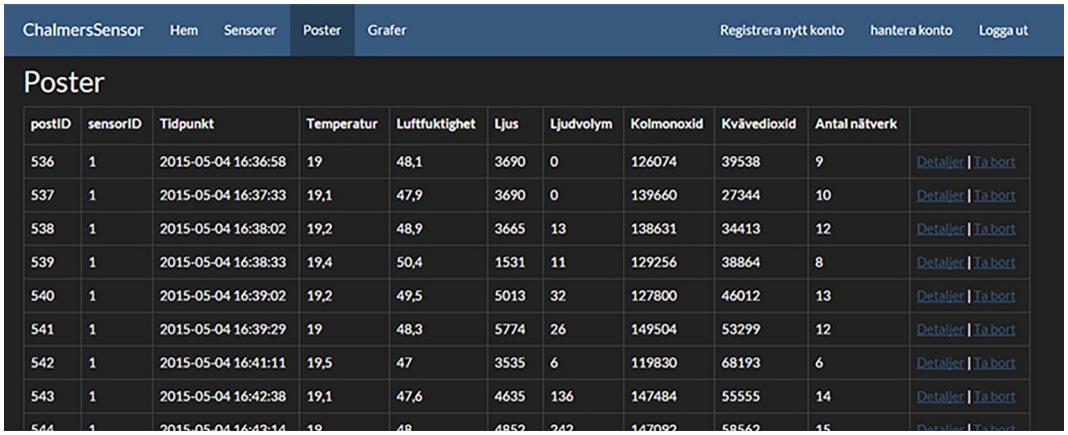

*Figur 5.8. Registrera nytt konto*

Som inloggad administratör på webbplatsen är det möjligt att se information om sensorerna och alla poster som sensorerna skickar. En administratör har möjlighet att redigera informationen för sensorerna så att till exempel placering stämmer överens med verkligheten. Sensorernas information visas under knappen “Sensorer” i menyfältet, se figur 5.9. Där finns det även alternativ för att se detaljer, redigera eller ta bort en sensor.



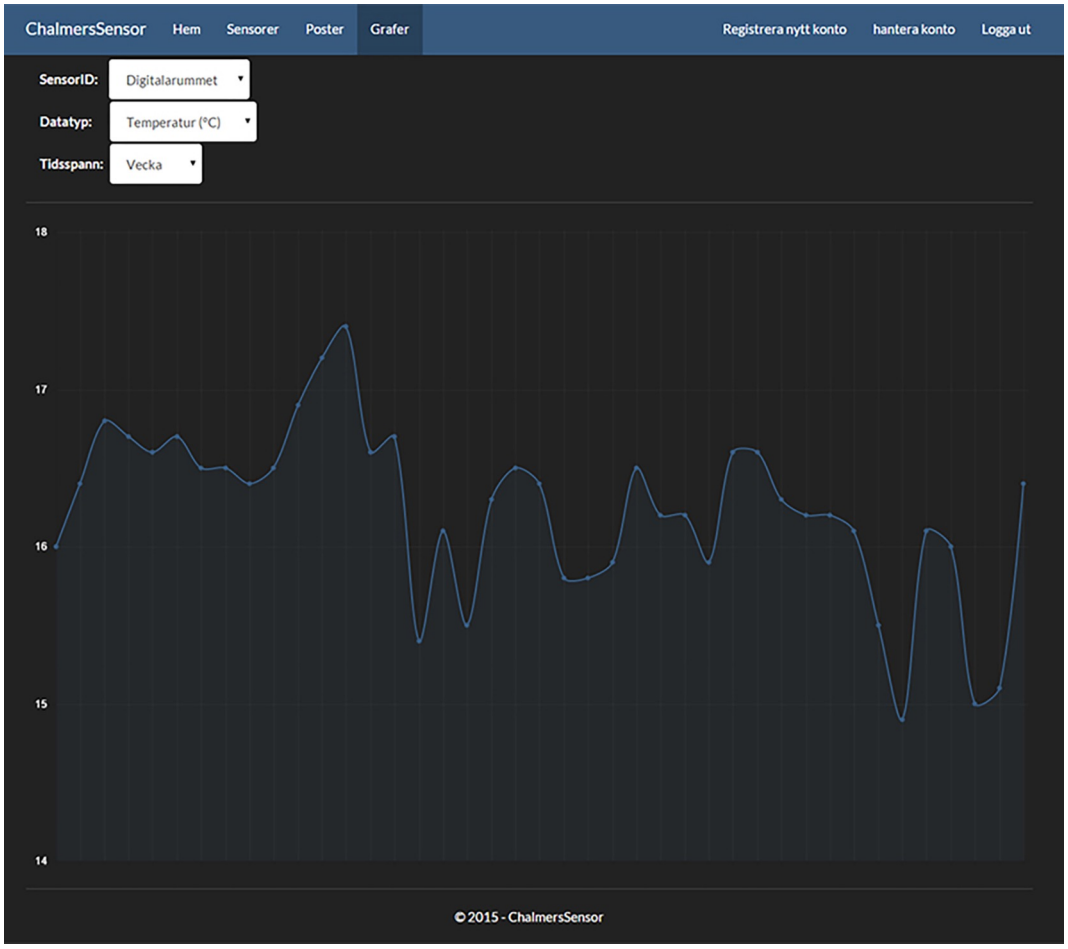

*Figur 5.9. Fliken sensorer*

Alla poster som sensorerna har skickat visas under knappen “Poster” i menyfältet, se figur 5.10 Här kan användaren även se detaljer om posten och ta bort posten.




*Figur 5.10. Fliken poster*

Målet med projektet var att visualisera datan som sensorerna skickar. Detta visas under sidan grafer som användaren kommer till via knappen “Grafer” på menyfältet. Vid tryck på denna knapp presenteras användaren med en webbsida med utseende som figur 5.11.




*Figur 5.11. Sidan med grafen*

### 5.3.2 Backend

Backenden, systemets skelett, är uppbyggt av en webbserver och en databas. Webbserven har ett par metoder som används av webbsidan och sensorn för bland annat att hämta tid och data, men även för att lagra datan som skickas av sensorn i databasen. Metoden som används för att få vad för datum och tid det är just nu heter time. Dit kan ett HTTP GET-anrop skickas och resultatet av det syns i figur 5.12. När denna metod anropas returneras ett svar som ser ut : UTC: 2015,5,25,10,48,43#, där 2015 är nuvarande året, 5 är månad, 25 är datum, 10 är timme, 48 är minut och 43 är sekund.

```
[HttpGet]
public string time()
        {
         //UTC:2015,4,27,11,14,38#
         //UTC:åååå,m,dd,hh,mm,ss#
           DateTime now = DateTime.Now;

            string t = ("UTC:"
                + now.Year + ","
                + now.Month + ","
                + now.Day + ","
                + now.Hour + ","
                + now.Minute + ","
                + now.Second + "#");

            return t;
        }
```

*Figur 5.12. Time-metoden i post-controllern*

Time-metoden anropas av sensorn när den håller på att mäta data för att veta vilken tidpunkt rummet avlästes. Denna tidpunkt följer sedan med tillsammans med datan när den skickas till servern. När sensorn skickar gör den ett ett HTTP POST-anrop där datan finns med i form av ett JSON-objekt. Från objektet hämtar servern sedan ut all data för att använda i de formler som omvandlar den råa datan till de verkliga värdena och sparar denna i databasen. Metoden som tar ut datan och gör om siffrorna till korrekt data heter parsePost och syns i figur 5.13. Här är det endast temperatur och luftfuktighet som omvandlas rätt. Smart Citizen skriver på sin dokumentation att mail ska skickas till dem för att få tillgång till alla omvandlingar av datan som de gör i sitt system. Mail skickades, men dessa blev inte besvarade.

```
public Post parsePost(string jString){
    JObject o = JObject.Parse(jString);
    Post post = new Post{
          sensorID = (int)o["sensorid"],
          temperature = Math.Round(-53
                + (175.72 / 65536) * ((double)o["temp"]),1),
          humidity = Math.Round(7
                + (125.0 / 65536) * ((double)o["hum"]),1),
          co = (double)o["co"],
          no2 = (double)o["no2"],
          light = (double)o["light"],
          noise = (double)o["noise"],
          battery = ((double)o["bat"])/10,
          solarpanel = (double)o["panel"],
          networks = (int)o["nets"],
          timestamp=makeDateTime((string)o["timestamp"]),
    };
    return post;
}
```

*Figur 5.13. parsePost metoden i klassen DatabaseUpdater*

För att populera grafen med data används en metod, getGraphData, som har tre inparametrar, där den första är för vilken typ av data som ska hämtas, till exempel temperatur, luftfuktighet eller ljud. Den andra parametern är hur långt bakåt i tiden användaren vill se data från; antingen en timme, en dag, en månad eller ett år bakåt från idag. Sista parametern är för vilken sensor som data ska visas ifrån.

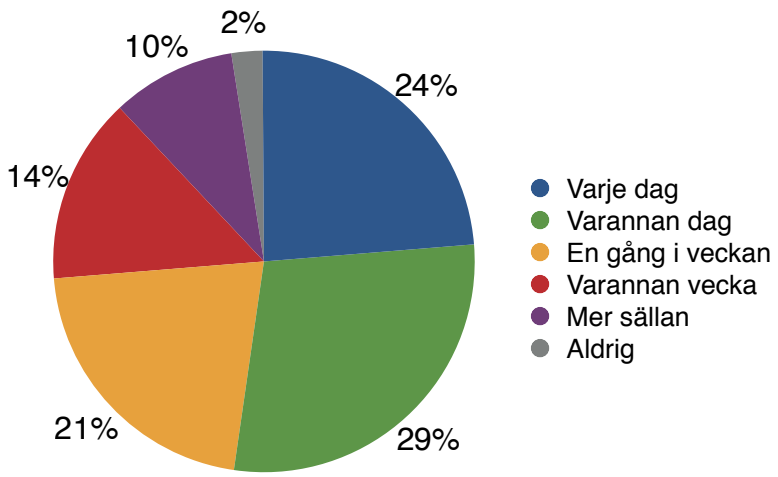
# 6 Resultat

Detta kapitel redogör för de svar som projektet har kunnat ge på de frågor som ställts.

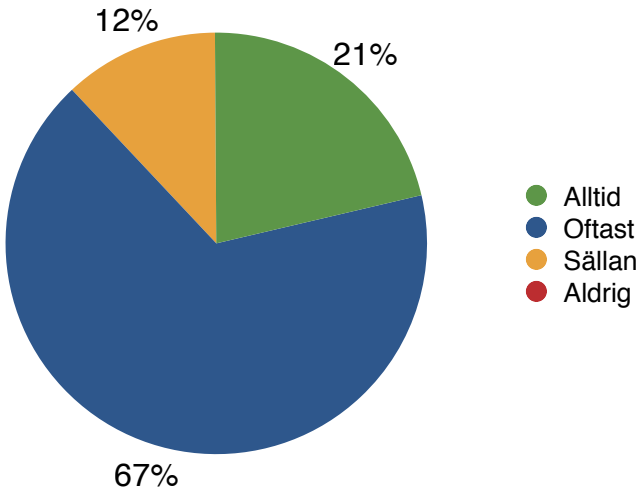
- Vilka faktorer är intressanta vid bokning av konferensrum?
- Hur kan Smart Citizen Kit integreras i ett system som övervakar inomhusklimat?
- Hur lämplig är Smart Citizen Kit för datasamling i en kontorsmiljöer?

## 6.1 Vilka faktorer är intressanta

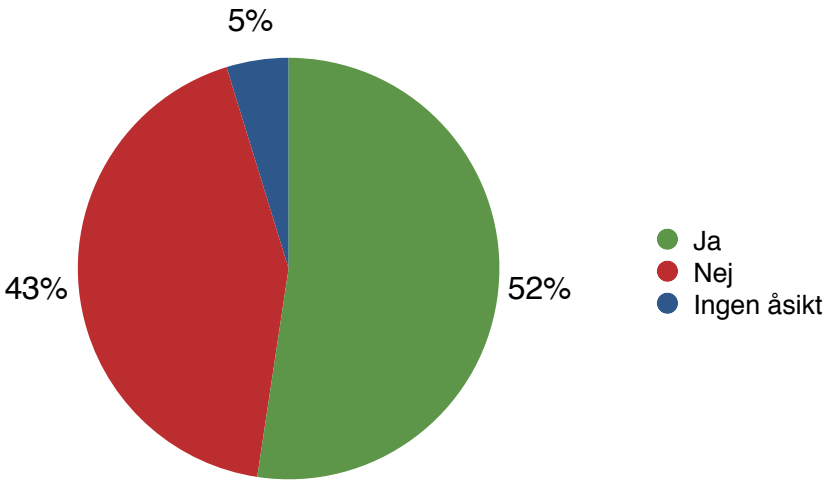
Enkäten besvarades av 42 personer av 339 tillfrågade vilket ger en svarsfrekvens på 12.4%. Nedan är resultatet av de svar som gavs.




*Figur 6.1. Användningsgraden av rum*




*Figur 6.2. Är rummen behagliga i dagsläget*

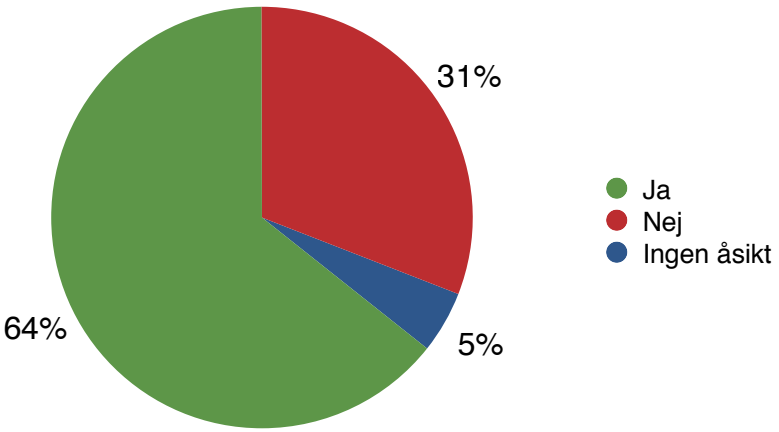



*Figur 6.3. Kommer en hemsida att användas vid bokning av konferensrum för att se sensordata*

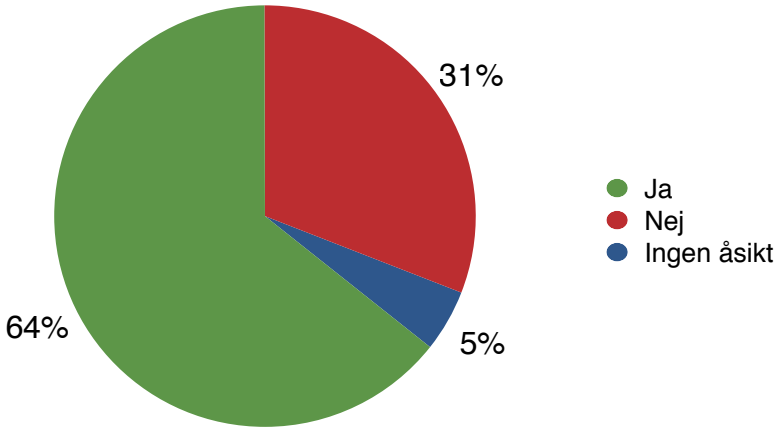
I fråga 4, figur 6.4, blev de svarande ombedda att rangordna de olika typerna av data efter vilken de upplevde som mest relevant för ett bra mötesrum. Viktigast får nummer 1, näst viktigast nummer 2 osv ner till nummer 9. Nedan har ett genomsnitt tagits fram för varje typ och ordnats efter lägst resultat.

| Datatyp | Ranking |
|---|---|
| Temperatur | 3.33 |
| Wifi-styrka | 4.19 |
| Koldioxid | 4.38 |
| Ljus | 4.45 |
| Ljud | 4.83 |
| Antal Wifi-nätverk | 4.93 |
| Luftfuktighet | 5.35 |
| Kolmonoxid | 5.81 |
| Kvävedioxid | 6.45 |

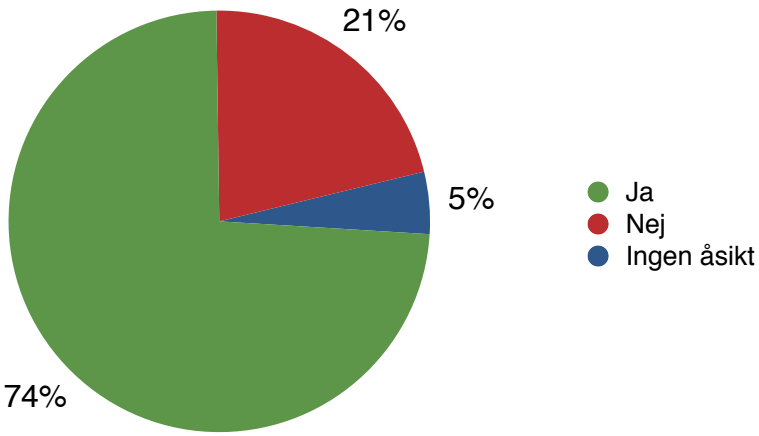
*Figur 6.4. Datatyp t.v och genomsnittlig rankning t.h*




*Figur 6.5. Är det en bra idé att visa temperatur i samband med bokning av rum i applikationen*




*Figur 6.6. Låter det som en bra idé att visa luftkvalité i applikationen där bokning av rum sker*




*Figur 6.7. Finns det intresse för att kunna se data på surfplattorna utanför de olika konferensrummen*

## 6.2 Hur Smart Citizen Kit kan integreras

Under projektet gång har examensarbetarna funnit två olika tillvägagångssätt till att implementera Smart Citizen Kit i sitt egna system. Det första är att använda Smart Citizens egna REST-API och skriva ett skript på servern som hämtar data kontinuerligt som sedan lagras i databasen.Det andra är att följa lösningen som gjordes under detta projekt; att konfigurera koden i själva sensorn så att sensorn skickar direkt till en server av eget val. Där behövs då en metod för mottagande av data.

## 6.3 Utvärdering av Smart Citizen Kit

Smart Citizen anger i sin marknadsföring att en person eller ett företag som köper en sensor får tillgång till deras plattform. Det finns bra instruktioner för att sätta upp sensorn så att den skickar till deras system och köparen blir då en nod i detta system. Sensorn kan mäta temperatur, luftfuktighet, kolmonoxid, kvävedioxid, ljus, ljud och antal nätverk. Huruvida alla dessa datatyper är av intresse i ett konferensrum kan diskuteras.

Ska koden på sensorn redigeras måste minnesåtgång finnas i åtanke då det är begränsat med minne på enheten. Tar en variabel upp för mycket minne kompilerar inte koden. Detta försämrar möjligheten att modifiera och lägga till kod på sensorn.

Formler för omvandlingen av den data sensorn skickar till de riktiga värdena finns ej i Smart Citizens dokumentation. Detta gör att utan formlerna skickar sensorn stora värden och det som kan avläsas är trender och inte faktiska värden. Kolmonoxid och kvävedioxid visas dessutom på Smart Citizens webbplats i kiloohm, alltså en resistans istället för det vedertagna ppm.

# 7 Diskussion och slutsats

Detta kapitel redogör för tolkningar och reflektioner av examensarbetarna.

## 7.1 Resumé

När vi påbörjade projektet fanns frågetecken kring utvecklingsmiljön och nya programmeringsspråk. Arbetet har prövat den teori som vi har lärt oss under vår utbildning genom att kräva ett praktiskt applicerande av kunskaperna. Vi lyckades med målen som sattes upp, övervann svårigheter på vägen och har nu ett fungerade system. Projektet har gett oss viktiga erfarenheter av hur det är att arbeta som mjukvaruutvecklare.

## 7.2 Kritisk diskussion

Detta avsnitt behandlar positiva såväl negativa lärdomar från projektet.

### 7.2.1 Tidsplan och verklighet

Då vi inte hade någon tidigare erfarenhet av utveckling i wire-kod eller .NET var det endast möjligt att grovt estimera tidsåtgång för förstudier och utveckling. Det var även svårt att förutse i vilken ordning saker skulle implementeras. Därför var det svårt att göra ett Gantt-Schema som stämmer med hur det blev när projektet genomfördes (se Appendix B). Detta medförde att tidsplaneringen blev kortare än vad det slutligen tog att genomföra projektet.

Den stora avvikelsen från tidsplaneringen var konfiguration av sensorns kod, där problem uppstod. Detta medförde att vi hamnade en vecka efter i planeringen vilket aldrig vi kunde komma ikapp under projektets gång.

### 7.2.2 Metod och verklighet

Projektets arbete följde metoden enligt planen och det har i huvudsak fungerat bra. I våra sprint-planeringar så tenderade det att bli diskussioner kring hur problem ska lösas och inte fokus på själva planerandet. Trello användes i mindre utsträckning än vad som var tänkt från början. Vi tror det beror på att gruppen bara bestod av två personer.  Det behövdes inget hjälpmedel för att få överblick då det räckte med att fråga varandra. Trello var användbart vid sprint-planeringar och granskningar men även ibland som påminnelse.

Lösningen skiljer sig något mot det lösningsförlag som fanns i början; detta är naturligt då all kunskap inte nödvändigtvis finns till en början. Med Scrum blir inte det något problem utan det snarare välkomnas. På så vis har Scrum passat projektet.

### 7.2.3 Smart Citizen Kit

Detta system anges vara ett open source projekt som ska vara öppet för vem som helst. Detta är bara delvis korrekt, då mycket av koden är kommenterad på spanska, vilket medför svårigheter för personer som inte har kunskaper i spanska. Mail-supporten som finns svarar i genomsnitt en av tre gånger, vilket ger ett oseriöst intryck. Smart-Citizen-sensorn mäter även data som är ointressant i konferensrum, då kolmonoxid och kvävedioxid inte är något som anses relevant i ett kontorslandskap. Enligt figur 6.4 var temperatur, WiFi-styrka och koldioxid de som var de mest intressanta vid val av konferensrum på ett kontor. Alltså är kolmonoxid, kvävedioxid och antal nätverk inte intressant information i konferensrum. Men det är möjligt att dessa värden kan vara relevanta i länder med återkommande smog och dålig luft.

## 7.3 Enkätsvar

Av de 339 personer som fick möjlighet att svara på enkäten så var det 42 som svarade, vilket ger en svarsfrekvens på 12,4%. Eftersom Sigma är ett konsultföretag är många av deras anställda ute och arbetar hos kunder, så är det cirka 90 personer som befinner sig på kontoret minst två gånger under en vanlig arbetsvecka. Detta gör att de 42 personerna som svarade på enkäten inte kan anses vara ett vetenskapligt underlag, men det är fortfarande möjligt att urskilja trender. Värt att notera är att vi inte vet om de 42 personerna som svarade arbetar inne på kontoret eller ute hos kund. Det vi vet är att 75% av de som svarade använder konferensrummen minst en gång i veckan. Det får anses som att de som svarade använder rummen i relativt hög utsträckning. En förändring eller

förbättring baserad på enkäten torde således ske på svar från dem som besvarat enkäten. I dagsläget upplever de flesta att rummen är behagliga. Klimatövervakning kommer kunna omvandla det till siffror.

Den genomsnittliga rangordningen visar vilken information som enkätdeltagarna ansåg var relevant för ett rum när det ska bokas. Temperaturen hade överlägset högst rang. Detta beror troligtvis på att det är en påtaglig faktor som är svårt att bortse ifrån. De flesta har avläst termometrar och sett väderleksrapporter. Detta tror vi gör det enkelt att relatera temperaturen som siffra och därför är denna rankad högst upp av de flesta. Därefter kommer WiFi-styrka. Här är det inte siffran som är viktig som sådan utan med en stark signal minskar risken för att tappa uppkoppling vilket kan vara enerverande. I mitten av rankningen har flera datatyper någorlunda lika genomsnitt, vilket får tolkas som att de varken är viktiga eller oviktiga. Uppfattningen är att de spelar roll men att de inte är nödvändiga. Kolmonoxid och kvävedioxid hamnar längst ner i prioriteringen. Detta är intressant då det är två värden Smart Citizen Kit mäter men efter undersökningen visar det sig att de inte är efterfrågade. Genomgående kommenterades det att halterna är viktiga men att personerna som svarande inte hade någon relation till dem. Människan kan inte uppfatta och särskilja dessa på ett medvetet plan och koppla sin upplevelse av luften till gashalter utan det är känslan av frisk syresatt luft som är betydelsefull. Det är möjligt att fastighetsskötare kan ha intresse av att se dessa värden då till exempel höga halter av kolmonoxid är giftigt.

Många av frågorna handlar om var sensorernas information visas bäst. En hemsida kräver att användarna är tvungna att aktivt besöka den för att få tillgång till datan. Därför trodde endast hälften av det svarande att de skulle ha besökt hemsidan innan de väljer rum. Om informationen istället visades där rummen faktiskt bokas hade det påverkat mer och informationen hade setts av fler. Det var 64% som trodde att informationen i bokningsappen kunde påverka deras val av rum. På så vis kan rummet med bäst luft bokas först och det i sin tur skulle kunna leda till att bokningarna av rummen blir mer jämt fördelade. Givetvis är informationen bara intressant om det finns flera rum tillgängliga och bokningarna inte görs flera dagar i förväg så att värdena blir inaktuella. En majoritet på 73% tyckte att informationen skulle vara intressant att se på surfplattorna utanför rummen. Det fastställer att det finns efterfrågan på datan och är potentiellt något som kan förverkligas i framtiden.

I svarsenkäten kom även pragmatiska förslag om att visa information om rummens utrustning. Information om rummen borde även visa om rummet har TV eller projektor och vilken typ anslutning de använder. Antal stolar och om ethernet-kabel, ljudanläggning eller whiteboardtavla fanns var också intressant.

Vi själva tror att sensorn inte är hela lösningen för att få fram all information om konferensrummen. Det finns nog behov av en inventariedatabas för att få fram vad för möbler, kopplingar och andra tillbehör som finns att tillgå i de olika rummen. Detta tillsammans med sensorer som läser av relevant data, skulle ge personalen ännu bättre förutsättningar för att boka de rummet som är de mest optimala för dem just då.

## 7.4 Vidareutveckling

I dagsläget finns det endast en sensor i ett rum, vilket i praktiken innebär att sensorn är en dyr termometer som mäter fler värden än bara temperatur och som just nu bara visar sin data på en webbplats i form av en graf. Men grunden är lagd för att visa datan på fler platser och även för att installera fler sensorer. Sigma har utvecklat en app för att boka grupprum, i den skulle det vara enkelt att lägga till den senaste avläsningen som sensorn gjort. Letar en anställd på kontoret efter till exempel ett svalt rum är det tydligt vilket som ska väljas. I framtiden när det finns flera sensorer behövs funktionalitet för att sortera på de olika värdena.

Vi tror det är möjligt att integrera Sigmas egna sensorer i systemet. Det skulle då vara möjligt att välja att mäta all relevant data för ett kontorslandskap till exempel som koldioxid och antal personer i ett rum istället för kolmonoxid och kvävedioxid.

# Referenser


1. Daniel, T., Gaura, E., & Brusey, J. (2009). Wireless sensor networks to enable the passive house-deployment experiences. In *Smart Sensing and Context* (pp. 177-192). [Användes 25 maj]

2. Brunelli, D., Minakov, I., Passerone, R., & Rossi, M. (2014, September). POVOMON: An Ad-hoc Wireless Sensor Network for indoor environmental monitoring. In *Environmental Energy and Structural Monitoring Systems (EESMS)*, 2014 IEEE Workshop (pp. 1-6). [Användes 12 maj 2015]

3. SCK, [Online]. Tillgänglig: http://docs.smartcitizen.me/#/ [Användes 25 maj 2015]

4. Visual Studio, [Online]. Tillgänglig: https://www.visualstudio.com/ [Användes 12 maj 2015]

5. C#, [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/vstudio/hh341490.aspx [Användes 12 maj 2015]

6. What is ASP.NET, [Online]. Tillgänglig: http://cplus.about.com/od/introductiontoprogramming/a/asp.htm [Användes 12 maj 2015]

7. ASP.NET Overview, [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/vstudio/4w3ex9c2(v=vs.100).aspx [Användes 12 maj 2015]

8. Microsoft SQL Server, [Online]. Tillgänglig: https://www.microsoft.com/sv-se/server-cloud/products/sql-server/ [Användes 12 maj 2015]

9. MVC, [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/ff649643.aspx [Användes 12 maj 2015]

10. Code first entity framework, [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/data/jj193542.aspx [Användes 12 maj 2015]

11. Git, [Online]. Tillgänglig: http://git-scm.com/book/en/v2/Getting-Started-Git-Basics [Användes 12 maj 2015]

12. JSON, IETF, [Online]. Tillgänglig: https://tools.ietf.org/html/rfc7159 [Användes 1 juni 2015]

13. JSON.NET, [Online]. Tillgänglig: http://www.newtonsoft.com/json [Användes 18 maj 2015]

14. Arduino, [Online]. Tillgänglig: http://www.arduino.cc/ [Användes 12 maj 2015]

15. Wiring, [Online]. Tillgänglig: http://wiring.org.co/ [Användes 12 maj 2015]

16. Smart Citizen, [Online]. Tillgänglig: https://smartcitizen.me/ [Användes 12 maj 2015]

17. Kickstarter, [Online]. Tillgänglig: https://www.kickstarter.com [Användes 18 maj 2015]

18. IIS7, [Online]. Tillgänglig: https://www.iis.net/learn/get-started/introduction-to-iis/introduction-to-iis-architecture [Användes 1 juni 2015]

19. JavaScript, [Online]. Tillgänglig: http://javascript.about.com/od/reference/p/javascript.htm [Användes 13 maj 2015]

20. JavaScript, [Online]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [Användes 13 maj 2015]

21. Ajax, [Online]. Tillgänglig: http://www.seguetech.com/blog/2013/03/12/what-is-ajax-and-where-is-it-used-in-technology [Användes 13 maj 2015]

22. jQuery, [Online]. Tillgänglig: https://jquery.com/ [Användes 13 maj 2015]

23. ChartJS, [Online]. Tillgänglig: http://www.chartjs.org/ [Användes 18 maj 2015]

24. Postman REST Client, [Online]. Tillgänglig: https://www.getpostman.com/features
[Användes 18 maj 2015]

25. Migrations, [Online]. Tillgänglig: http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application
[Användes 1 juni 2015]

26. Razor, [Online]. Tillgänglig: http://weblogs.asp.net/scottgu/introducing-razor
[Användes 22 maj 2015]

27. Scrum, [Online]. Tillgänglig: https://www.scrum.org/ [Användes 11 maj 2015]

28. Schwaber, Ken and Beedle, Mike, 'Agile Software Development with Scrum', Scrum, 2001

29. Trello, [Online]. Tillgänglig: https://trello.com/ [Användes 11 maj 2015]

30. Bitbucket, [Online]. Tillgänglig: https://bitbucket.org/ [Användes 12 maj 2015]

31. Microsoft Virtual Academy ,[Online]. Tillgänglig: http://www.microsoftvirtualacademy.com/
[Användes 1 juni 2015]

32. Google Forms, [Online]. Tillgänglig: https://www.google.com/forms/about/
[Användes 20 maj 2015]

33. Balsamiq [Online]. Tillgänglig: https://balsamiq.com/
[Användes 1 juni 2015]

34. SCK instruktioner, Tomas Diez, 24 maj 2013 "Adding a Smart Citizen Kit, A quick guide to getting started with your SCK", [Online]. Tillgänglig: https://smartcitizen.me/posts/view/6
[Användes 18 maj 2015]

35. Constants.h,Github, [Online]. Tillgänglig: https://github.com/fablabbcn/Smart-Citizen-Kit/blob/master/sck_beta_v0_9/Constants.h
[Användes 15 maj 2015]

36. Python, [Online]. Tillgänglig: https://www.python.org/about/
[Användes 1 juni 2015]

37. HTML5 canvas, http://www.w3.org/TR/html5/scripting-1.html#the-canvas-element
[Användes 18 maj 2015]

# Appendix A: Enkät

Enkäten ställde följande frågor:

- Hur ofta använder du mötesrummen?
  - Varje dag
  - Varannan dag
  - En gång i veckan
  - Varannan vecka
  - Mer sällan
  - Aldrig
- Upplever du generellt sett mötesrummen på kontoret som behagliga?
  - Alltid
  - Oftast
  - Sällan
  - Aldrig
- Rangordna efter vad som är viktigast för dig för ett behagligt rum.
  - Temperatur
  - Koldioxid
  - Kolmonoxid
  - Luftfuktighet
  - Kvävedioxid
  - Ljus
  - Ljud
  - Antal Wifi-nätverk
  - Wifi-styrka
- Om det fanns en hemsida som visade allting från föregående fråga för alla rum på kontoret, tror du att du hade besökt hemsidan innan du väljer rum?
  - Ja
  - Nej
  - Ingen åsikt
- Om temperaturen visades bredvid rummen i GetMeARoom-appen, tror du att det hade påverkat ditt val?
  - Ja
  - Nej
  - Ingen åsikt
- Om den relativa luftkvalitén visades bredvid rummen i GetMeARoom-appen, tror du att det hade påverkat ditt val?
  - Ja
  - Nej
  - Ingen åsikt
- Hade det varit intressant att se temperatur, luftkvalité osv på paddorna utanför rummen?
  - Ja
  - Nej
  - Ingen åsikt
- Finns det något du vill tillägga, kommentar eller fråga, vänligen skriv nedan

# Appendix B: Gantt-Schema

| **Tasks** **Vecka** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Uppsättning av utvecklingsmiljö | | | | | | | | | | |
| Skapa ett lösningsförslag | | | | | | | | | | |
| Skapa databasschema | | | | | | | | | | |
| Skapa usecases | | | | | | | | | | |
| Skapa mockups | | | | | | | | | | |
| Skriva databasen | | | | | | | | | | |
| Utveckling | | | | | | | | | | |
| Rapportskrivning | | | | | | | | | | |
| Redovisningsförberedelse | | | | | | | | | | |