CHALMERS

MoInbaserad webbapplikation för IoT i uppkopplade byggnader

Examensarbete för högskoleingenjörer i datateknik

Valentin Lindblad

Fredrik Mårlind

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2017

Examensarbete

Molnbaserad webbapplikation för IoT i uppkopplade byggnader

Möjligheter och de val som har gjorts vid skapandet av applikationen

Valentin Lindblad
Fredrik Mårlind

CHALMERS

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2017

Molnbaserad webbapplikation för IoT i uppkopplade byggnader
Möjligheter och de val som har gjorts vid skapandet av applikationen
Valentin Lindblad & Fredrik Mårlind

Examinator: Peter Lundin, Data- och informationsteknik.
Handledare: Magnus Almgren, Data- och informationsteknik.

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Göteborg, Sverige 14 juni 2017

Molnbaserad webbapplikation för IoT i uppkopplade byggnader
Möjligheter och de val som har gjorts vid skapandet av applikationen
Valentin Lindblad & Fredrik Mårlind
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet

Sammanfattning

Sensorer och styrenheter förväntas i dagsläget att alltid kunna nås. Denna efterfrågan har medfört att allt fler enheter kopplas upp mot internet. Dessa enheter behöver ofta vara batteridrivna och trådlösa vilket kräver en trådlös nätverksinfrastruktur som är extra energisnål. Efterfrågan leder till utveckling av infrastrukturen som i sin tur leder till att det blir mer kostnadseffektivt att skapa nya tjänster som mäter, samlar in och lagrar information.

Ett område som börjar bli aktuellt är att skapa nya tjänster för bostäder och byggnader. Detta projekt har därav utvecklat en webbapplikation för att redovisa mätvärden ifrån uppkopplade byggnader.

På senare år har utvecklingen tenderat att röra sig mot serverlös arkitektur för att inte behöva hantera underhållning och administration av hårdvara. Därav har systemet utvecklats med denna arkitektur.

Projektets syfte är att dra lärdom och utvärdera arbetsprocessen vid utvecklandet av en molnbaserad webbapplikation. Webbapplikationen utvärderades baserat på dess prestanda och kostnad.

Slutsatsen som drogs var att det i dagsläget inte självklart att välja en molnbaserad lösning, utan det beror på kravspecifikationerna och om systemet kräver några speciallösningar.

Nyckelord: IoT, Sakernas Internet, LPWAN, Molnbaserad, Serverlös arkitektur, Webbapplikation, Microsoft Azure, Uppkopplade byggnader

Cloud based web application for IoT in connected buildings
The possibilities and the choices made during the creation of the application
Valentin Lindblad & Fredrik Mårlind
Department of Computer Science and Engineering
Chalmers University of Technology / University of Gothenburg

Abstract

Control units and sensors connectivity are evolving and is expected to always be available. This demand have brought more devices to get connected to the Internet. These devices often need to be energy efficient and wireless which means that an infrastructure for wireless transmissions that is focused on energy efficiency needs to be developed. The demand leads to further development of the network infrastructure which leads to it being more worthwhile to create new services that measures, collects and stores information.

An area that has been starting to emerge is services for apartments and buildings. This project will therefore develop a web application that presents measurements from connected buildings.

In recent times the development have been trending towards serverless architecture which means that serving and administering of hardware is not necessary for the customer anymore. This project has been developed using this architecture.

The purpose of the project is to gain knowledge and review the process of developing a cloud based web application. The web application was evaluated based on its performance and cost.

The conclusion that was drawn was that currently it is not obvious to choose a cloud based solution, although it depends on the required specifications and if the system requires a special solution.

Keyword: IoT, Internet of Things LPWAN, Cloudbased, Serverless architectures, Web application, Microsoft Azure, Connected buildings

Förord

Den här rapporten gjordes som ett examensarbete på femton högskolepoäng som utfördes de sista tio veckorna under vårterminen 2017.

Vi vill tacka Gabriel Ibanez från Cybercom och Stefan Lindgren från Talkpool för att de introducerade oss till projektet och stöttat oss igenom vårt examensarbete.

Vi vill även tacka Andreas Lindmark och Daniel Öhlund för deras hjälp med organiseringen och arbetsmetodiken som användes igenom detta projekt.

Vi ger Taner Yavuz ett stort tack som en senior mjukvaruutvecklare som hjälpt och varit bollplank åt oss dessa veckor.

Ett stort tack till alla exjobbare och resterande personal som hjälpt oss från Cybercom och Talkpool.

Sakib Sistek tackar vi ödmjukast för hans goda personlighet och omhändertagande. Tack för att du gav oss möjligheten att jobba med Cybercom.

Slutligen vill vi ge ett stort tack till Magnus Almgren för hans hjälp med utformadet av den här rapporten.

Valentin Lindblad & Fredrik Mårlind, Göteborg, juni 2017

Innehåll

Terminologi

AWS	Amazon Web Services vilket är samlingsord för företaget Amazons olika webbtjänster.
Azure	En samling molntjänster utvecklade och tillhandahållna av Microsoft.
Backend	Bred benämning av den bakomliggande mjukvaran för en applikation.
CSS	Cascading Style Sheet, används inom webbutveckling för att ge ett utseende på HTML.
Frontend	Benämning för mjukvara som ansvarar för det som en användare ser så kallat grafiskt gränssnitt.
Git	En versionhanteringslösning för att hantera källkod.
HTML	Hyper Text Markup Language, är ett taggspråk som används inom webbutveckling.
IDE	En sammansatt utvecklingsmiljö för underlätta vid programmerande.
IoT	Internet of Things eller sakernas internet på svenska, principen där internetanslutning ges till mikrodatorer och andra enheter.
JIRA	Ett digitalt verktyg för att hantera och strukturera arbetsuppgifter i ett projekt.
JS	JavaScript, ger funktionalitet till en hemsida.
JSON	JavaScript Object Notation, är ett kompakt sätt att strukturera textbaserad data.
LPWAN	Low-Power Wide-Area-Network, trådlöst nätverk med fokus på lång räckvidd och att vara energisnålt.
LoRa	Long Range, ett LPWAN med fokus på extra långt avstånd på överföringarna och på att dra lite energi.
MVC	Model-View-Controller är ett arkitekturmönster.
REST	Representational state transfer, en variant av en webbtjänst för att ge interoperabilitet mellan datorsystem över internet.
Sprint	Är en tidsperiod som man utför specifik utveckling för att leverera en produkt för granskning.
VM	En Virtuell Maskin är när man skapar en virtuell dator vars resurser är allokerade från fysisk dator.
VPS	Virtuell Privat Server är en virtuell dator som man hyr från företag efter specifikationer.

1

Inledning

Trots att internet fortfarande är relativt ungt i jämnföresle med många andra teknologier utvecklas det allt mer. Vad som bara för några år sedan låg på företags egna servercenter och krävde flertal IT-specialister har de senaste åren börjat flyttas allt mer mot molnbaserade och serverlösa lösningar.

Med allt fler enheter uppkopplade mot internet utvecklas allt fler tjänster som utnyttjar internetuppkopplingen. Denna trend syns tydligt med allt fler mätenheter och sensorer som laddar upp mätdata.

Det här projektet skapar en molnbaserad webbapplikation som kan visa sensorinformation i uppkopplade byggnader. Projektet kommer att utvärderas genom dess prestanda och utefter vilken total kostnad det är att driva applikationen. Detta kommer framför allt utvärderas relativt andra webbapplikationer och molnbaserade lösningar.

Projektet kommer att utföras för Cybercom som är ett IT konsultbolag och TalkPool som är Internet of Things (IoT) nätverksspecialister. Cybercom är de som leder projektet framåt och TalkPool är den part som levererar den hårdvara som behövs. TalkPool är även de som vill använda den levererade applikationen.

1.1 Mjukvara

Inom dagens samhälle blir webbapplikationer vanligare och mer eftertraktade på grund av att de är plattformsoberoende och ej behöver installeras på datorn. Nackdelen är att detta medför behov av servrar som kan tillhandahålla applikationen och informationen som hör till den. För att inte behöva köpa in och hantera servrar och allt som det innebär kommer detta projekt använda sig av molntjänster för att skapa, distribuera och hantera program med globala nätverk av datacenter. En annan fördel för detta är lättare expansion både när kraftfullare hårdvara behövs eller om tjänsten senare ska erbjudas i en annan region.

1.2 Hårdvara

IoT eller sakernas internet är en princip där man ger allt fler enheter internetanslutning. Detta förenklar insamling av sensorinformation och kontroll av olika system. I detta projekt kommer ett nätverk av temperatursensorer och luftfuktighetssensorer installerade i varje rum i ett lägenhetshus användas. För att kunna samla in all sensorinformation på ett smidigt sätt utan fasta anslutningar så kommer ett energisnålt långavstånds trådlöst nätverk även känt som Low-Power Wide-Area-Network (LPWAN) att användas. Genom att skapa ett mer specialiserat LPWAN relativt dagens mobilnätverk kan batteridrivna enheter lättare skapas och implementeras i IoT-nätverk.

1.3 Syfte

Projektets syfte är att dra lärdom och utvärdera arbetsprocessen vid utvecklandet av en molnbaserad webbapplikation.

Med dagens utveckling börjar allt fler enheter anslutas till internet. Detta medför teknologiförändringar vilket leder till nya produkter och tjänster. Allt detta har lett till att IoT har blivit aktuellt de senaste åren. Detta projekt ämnar att utveckla en IoT-nätverksapplikation som kan användas i så kallade smarta byggnader.

Applikationens syfte är att ge användarna en bättre insikt över deras energiförbrukning och kostnader. Detta för att lättare kunna följa nya EU-direktiv kring energisparande. Användare kan exempelvis vara hyresgäster, bostadsägare och statliga myndigheter.

1.4 Avgränsningar

I detta projekt kommer det inte att ske någon utveckling utav de uppkopplade sensorer som ska användas inom projektet.

Projektet kommer inte resultera i en färdig produkt utan den utvecklas för att bevisa konceptet.

Applikationen kommer inte att ha någon form utav inloggningssystem som särskiljer olika användare.

Inga tolkningar av den insamlade informationen kommer göras av datorprogrammet.

2

Teknisk bakgrund

Första delen av detta kapitel kommer att ge en inblick i de kommunikationsteknologier som använts. Andra delen beskriver de tekniker och teknologier som använts på serverdelen av projektet. Den tredje delen tar upp de utvecklingsverktyg som har varit ett bra stöd under utvecklingen. Fjärde delen tar upp de programmeringsspråk och ramverk som använts under projektets utveckling.

2.1 Kommunikation

Denna sektion täcker de tekniker som används vid kommunikationen mellan sensorerna och molnet. LoRa är det nätverk som används för att få sensorernas data till molnet med MQTT som protokoll.

2.1.1 Protokollet MQTT

Message Queue Telemetry Transport (MQTT) är en ISO standard för ett enkelt lättvikts-protokoll för att skicka medelanden mellan internetuppkopplade maskiner. Protokollet följer Publish / Subscribe design mönstret, vilket innebär att den som publicerar informationen inte känner till de som lyssnar utan gör utskick till alla som lyssnar på ett visst ämne. Prenumeranten vet inte vilka enheter den skall lyssna på utan lyssnar istället på ett ämne och tar emot alla paket som publiceras med detta ämne. Relativt andra liknande protokoll har MQTT fokus på att minimera mängden nätverksdata/bandbredd som används, minimera den prestanda som krävs, ha hög tillförlitlighet och säkerställa att informationen kommer fram [1].

2.1.2 Trådlösa kommunikationssättet LoRa

Long Range (LoRa) är ett Low Power Wide Area Network (LPWAN) som är gjort för att användas i system som är strömförsöjda med batteri. I ett LoRa nätverk är alla slutenheter uppkopplade mot en LoRa nätverksnod också kallad gateway. Via

nätverksnoden skickas informationen vidare till en server över ett vanligt nätverksmedium som 3G eller över en fast nätverksuppkoppling. Fokus ligger på uppladdning av data då nätverksnoden lyssnar på alla slutenheter men slutenheterna lyssnar normalt sätt inte på andra nätverksnoder för att spara energi [3].

2.2 Serverdel

Denna sektion kommer att behandla de tekniker och teknologier som har använts på serversidan av projektet. Serverlös arkitektur är en teknik som har använts för webbapplikationsservern och för alla databaser i projektet. MS SQL är den databas som har använts i projektet för att lagra datan från sensorerna.

2.2.1 Serverlös artitektur

Serverlös artitektur bygger på principen att man bygger gemensamma datacenter och serverlokaler för flera företag och deras produkter. Genom att utveckla system som tillåter datakraft och bandbredd att tilldelas dit det behövs när det behövs kan man låta flertalet tjänster köra på samma maskiner. Detta innebär att man inte har statiskt allokerade resurser utan att man dynamisk tilldelar dem där de behövs, när de behövs. Detta medför besparingar och effektiviseringar av resurser, mindre energianvändningar och mindre kostnader. Däremot krävs uppbyggnad av datacenter och serverlokaler, exempel är Amazons AWS (Amazon Web Services), Microsofts Azure och Google Cloud. Ytterligare fördelar är att man lättare kan skala upp och ner ens lösning, lättare hantera säkerhetkopior och slipper hantera servrarna [13].

2.2.2 MS SQL

Microsoft Structured Query Language (MS SQL) är en relationsdatabas som använder sig av språket SQL för att hantera data. MS SQL valdes som databassystem för detta projekt på grund av dess lätta integration i ASP.NET miljöer och för att det fanns färdigutvecklat som en molntjänst på Microsoft Azure [12].

2.3 Utvecklingsverktyg

Denna sektion täcker de verktyg som har använts under utvecklandet av applikationen.

2.3.1 Visual Studio

Visual Studio är ett utvecklingsverktyg som underhålls och utvecklas av Microsoft. Verktyget kan används huvudsakligen för att utveckla mjukvara i C, C++ och C# [14].

2.3.2 Atom

Atom är en textredigerare som är lätt att modifiera och lägga till ytterligare funktionalitet till. Atom har öppen källkod och kan användas på de flesta operativsystem [9].

2.3.3 Git

Git är ett välanvänt system för att hantera olika versioner av filer i olika typer av mjukvaruprojekt. Med hjälp av Git kan man se vem det var som ändrade vad i vilka filer. Man kan också bättre se en historik över hur saker har förändrats och det blir då enklare att hitta ett fel när det blir problem i projektet. Detta projekt kommer att använda sig utav Git (för mer se sektion 4.3.1).

2.4 Webbutveckling

De språk som har använts under webbutvecklingen av hemsidan är HTML, CSS, JavaScript och React. ASP.NET Core och C# används för att koppla frontend till databasen i backend.

Hyper Text Markup Language (HTML) är ett taggspråk som används inom webbutveckling för att skapa struktur och innehåll till olika hemsidor [15]. HTML är väldigt begränsat i hur det kan påverka utseendet på sidan. Därav används Cascading Style Sheets (CSS) för att ge HTML ett annat utseende.

JavaScript (JS) är det som används inom webbutveckling för att ge hemsidorna ytterligare funktionalitet. Detta för att göra hemsidorna mer responsiva och dynamiska för användarna.

React är ett kodbiliotek i JavaScript skapat av Facebook för att bygga användargränsnitt. Ett React gränssnitt består ofta utav ett flertal komponenter. Fördelen med React är att det kan rendera och ladda sidan på komponentbasis. Detta betyder då att om man har ny data som ska visas i en komponent behöver det bara ladda om den specifika komponenten med den nya datan.

ASP.NET Core är ett ramverk med öppen källkod för att bygga olika typer utav Internetapplikationer som kan användas på flera olika operativsystem. Exempel på

det kan vara webbapplikationer, IoT-appar och backends. I detta projektet kommer det att användas för att bygga en backend.

3

Relaterat Arbete

I detta avsnitt kommer det att tas upp andra arbeten som andra har skrivit som kan liknas med denna rapporten.

3.1 Migrating and testing distributed cloud based web applications

Migrating and testing distributed cloud based web applications av Daniel Arenhage och Fabian Lyrfors [4] är ett examensarbete om att göra om en befintlig applikation till en molnbaserad webbapplikation. De undersökte möjligheter och rimligheten att göra om befintliga applikationer till webbapplikationer. Till skillnad ifrån deras arbete ligger fokus i vårt arbete på att utveckla ett nytt molnbaserat system med ett stort antal sensorer som en webbapplikation. En annan möjlig skillnad är att deras arbete är något utdaterat (2012) vilket gör att de molntjänster som finns tillgängliga har utvecklats och förändrats.

3.2 Universell IoT-enhet för dataregistrering och processtyrning

Universell IoT-enhet för dataregistrering och processtyrning av Rickard Edfast och Oskar Lindström [7] är ett kandidatarbete som handlar om att utforska möjligheten att göra en generell enhet för sakernas internet. Ett webbgränssnitt för enkel konfiguration utvecklades för att kunna anpassa dessa enheter till användarens behov. Deras rapport har stort fokus på utvecklandet av enheterna relativt denna rapport som har färdigutvecklade sensorsenheter vars data ska redovisas i en molnbaserad webbapplikation. En annan skillnad är att Edfasts och Lindströms arbete behandlar styrenheter vilket inte görs i den här rapporten.

3.3 Cloud Service Analysis - Choosing between an on-premise resource and a cloud computing service

Cloud Service Analysis - Choosing between an on-premise resource and a cloud computing service av Keith Augustsson och Jonas Fredriksson [6] är ett annat examensarbete som kan vara relevant för läsare av den här rapporten. Augustsson och Fredriksson utforskar valet mellan att använda en molntjänst och lokala maskiner för att utföra beräkningar. Även om deras arbete skiljer sig ifrån det här projektet kan det vara relevant då de går igenom de för- och nackdelar med molnbaserade tjänster som finns relativt lokala alternativ.

4

Metod

Metodkapitlet handlar om det arbetssätt som projektet kommer att använda sig utav. Kapitlet beskriver den förstudie som planeras inför projektstart. Därefter följer en förklaring av det agila arbetssättet som kommer användas. Slutligen handlar det om de hjälpmedel som används för att programmera effektivt.

4.1 Förstudie

En förstudie om projektet ska göras innan någon utveckling börjar. Förstudien ska ta fram tre olika saker där det första är vilka krav det finns på projektet. Det andra är att ta fram en uppsättning av användarberättelser utifrån de krav som togs fram. Det tredje blir att göra en tidsestimering och prioritering av användarberättelserna för att planera in dem i sprintar.

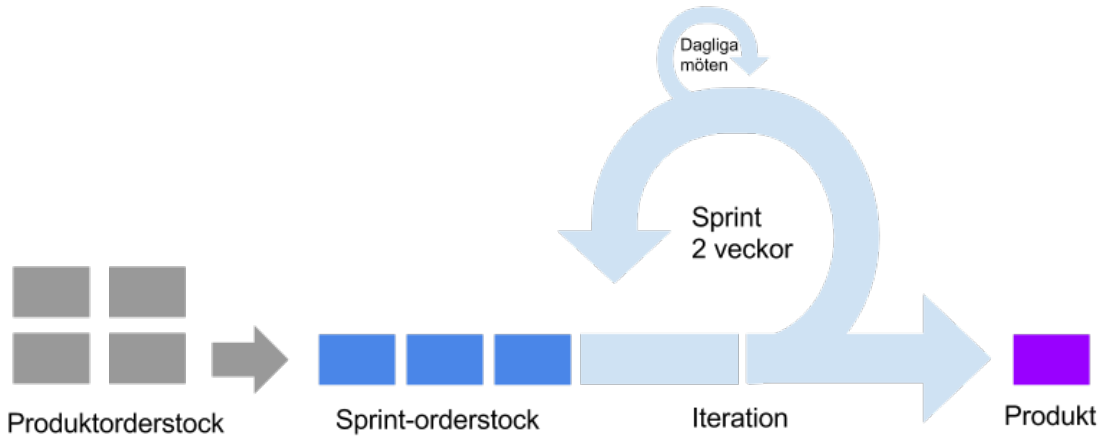
4.2 Agilt arbetssätt vid programmering

Normalt sätt använder agila systemutvecklingsprocesser 12 grundprinciper. Principerna är framtagna från 17 utvecklare som skrev ett manifest om agil utveckling [2]. Nedan är de 12 principerna från manifestet:

- Vår högsta prioritet är att tillfredställa kunden genom tidig och kontinuerlig leverans av värdefull programvara.
- Välkomna förändrade krav, även sent under utvecklingen. Agila metoder utnyttjar förändring till kundens konkurrensfördel.
- Leverera fungerande programvara ofta, med ett par veckors till ett par månaders mellanrum, ju oftare desto bättre.
- Verksamhetskunniga och utvecklare måste arbeta tillsammans dagligen under hela projektet.

- Bygg projekt kring motiverade individer. Ge dem den miljö och det stöd de behöver, och lita på att de får jobbet gjort.
- Kommunikation ansikte mot ansikte är det bästa och effektivaste sättet att förmedla information, både till och inom utvecklingsteamet.
- Fungerande programvara är främsta måttet på framsteg.
- Agila metoder verkar för uthållighet. Sponsorer, utvecklare och användare skall kunna hålla jämn utvecklingstakt under obegränsad tid.
- Kontinuerlig uppmärksamhet på förstklassig teknik och bra design stärker anpassningsförmågan.
- Enkelhet – konsten att maximera mängden arbete som inte görs – är grundläggande.
- Bäst arkitektur, krav och design växer fram med självorganiserande team.
- Med jämna mellanrum reflekterar teamet över hur det kan bli mer effektivt och justerar sitt beteende därefter.

Scrum är en variant av agil utveckling som kommer att användas i detta projekt. Utvecklingen kommer ledas framåt med dagliga möten inom projektgruppen och sprintar på 2 veckor. I slutet av varje sprint ska det ske ett möte med kund för att demonstrera det som skapats under sprinten. Ytterligare möten med kund ska ske löpande under projektets gång. För att enklare bevaka projektet kommer JIRA att användas som Scrum board. En illustrering av hur arbetsflödet kan ske visas i figur 4.1.



Figur 4.1: Illustration av arbetsflödet i Scrum.

Arbetslaget som jobbar med projektet kommer att sitta i samma kontor vilket förenklar vid evaluering av varandras kod och lösningar. Övrig kommunikation inom projektgruppen kommer att ske med hjälp av Slack. En wiki kommer även att användas för att publicera projektartefakter och projektinformation.

4.2.1 Företagets arbetsmetodik

Företaget som webbapplikationen utvecklas hos använder Scrum som en central del i sin arbetsmetodik. Varje dag möts alla olika utvecklingsgrupper och redovisar och diskuterar eventuella framsteg, bekymmer och vad som arbetas på för dagen. Detta hjälper oss att komma framåt i utvecklingen och få hjälp om en arbetsuppgift blockeras av något. Det ger även resterande på företaget en uppfattning om hur alla projekt fortlöper och om någon med specialkompentens kan hjälpa i ett annat projekt.

Företaget erbjuder även utbildningar och informella arbetsseminarium emellanåt. Dessa kommer användas vid relevans för arbetet.

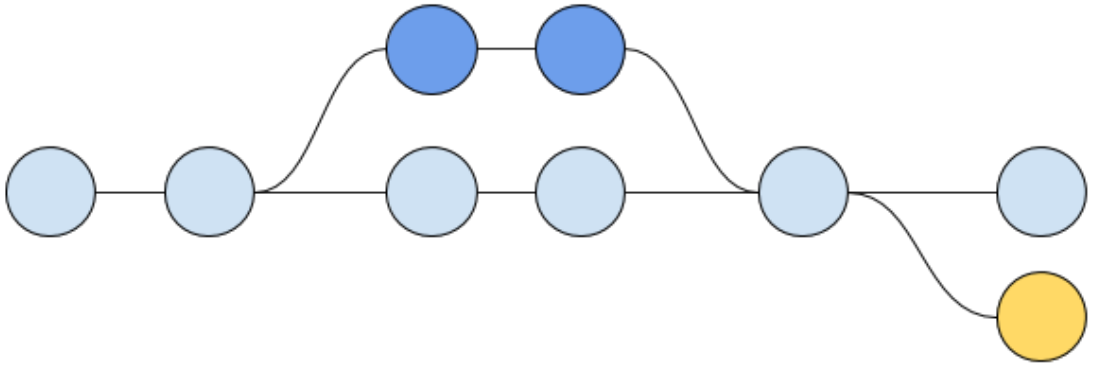
4.3 Programmeringshjälpmedel

Det här avsnittet kommer att gå genom de hjälpmedel som använts under utvecklandet av applikationen. Till exempel Git för att versionshantera koden och kodningskonventioner för att skriva konsekvent kod.

4.3.1 Git

Git kommer att användas för att versionshantera koden. Funktionalitetsgren även känt som Feature Branch arbetssättet kommer att användas för att enklare utveckla ny funktionalitet för baskoden [5].

I figuren 4.2 nedan syns en illustration över hur arbetssättet fungerar. Man har en huvudutvecklingsgren som förgrenas när utveckling av en ny funktionalitet behövs. När den nya funktionaliteten är färdigutvecklad och testad kan förändringarna slås ihop med huvudutvecklingsgrenen.



Figur 4.2: En illustration över hur det kan se ut när man använder funktionsgrenar.

4.3.2 Kodningskonventioner

Kodningskonventioner kommer att följas för ökad läslighet av koden och därmed göra det lättare för andra att fortsätta på arbetet vid vidare utveckling efter projektets färdigställning. De kodningskonvetioner som kommer att följas kommer bero på programspråket som koden skrivs i. Den kodstandard som finns för respektive språk kommer att följas. Genom att följa standarden kan man enkelt dra nytta av extra funktionalitet i de utvecklingsmiljöer som används, ett exempel är automatisk indentering, automatisk formatering och automatisk avslutning på påbörjat funktionsnamn eller variabelnamn.

5

Design

Detta kapitel kommer ta upp den design och de val som finns i skapandet av en molnbaserad IoT applikation. I sektion 5.1 beskrivs de krav som finns på sensorerna. Nästa sektion 5.2 behandlar vilken sorts överföringskanal som behövs för att sensorerna ska få sin data till molnet. Hur den datan sedan ska lagras tas upp i sektion 5.3. Den design och de krav som finns på webbapplikationen finns beskrivna i sektion 5.4.

5.1 Sensorerna

Kraven för sensorerna i detta projekt är framförallt att de är batteridrivna och trådlösa för att tillåta montering utan fysisk el-tillförsel eller nätverkanslutning. För att de ska vara praktiska så kommer även det krävas att batteritiden är i storleksordningen år. Detta så man slipper byta batterier för ofta samt inte får för mycket nertid. Den trådlösa sändaren i sensorerna kommer behöva ha lång räckvidd och bra genomträngning av väggar vid exempelvis montering i källare. Mätning och överföring av informationen kommer inte behöva vara mer frekvent än en mätpunkt varje timme i och med att inomhusklimat inte förändrar sig särskilt snabbt. Detta hjälper med batteritiden då detta innebär att sensorsenheterna endast behöver vara igång en liten stund varje timme då den tar och överför mätdata.

5.2 Överföring

För att överföringen skall fungera med sensorer som är av en passiv intervalsbaserad typ kommer det behövas en annan enhet/nätverksnod som aktivt lyssnar och konstant är igång. Därmed är det mest lämpligt att man har strömförsörning och gör denna nätverksnod stationär. Om man har den stationär kan man även lättare montera större antenner vilket ger längre räckvidd eller att sensorerna inte behöver skicka lika starka signaler. Den större nätverksnoden kan därefter vidarebefordra informationen vart den sen behövs via internet. Detta kan liknas vid mobiltelefoner och telefontorn.

5.3 Databas/Lagring

För att lagra all mätdata så effektivt som möjligt kommer ett strukturerat sätt att lagra all information behövas. Lagrar man mätvärden och informationen om vart de är placerade separerat från varann får man en bra struktur på informationen. Med detta slipper man redundans med andra ord undviker man överflödig data.

5.3.1 Mätvärden

För att kunna förstå mätdatan som skickas så kommer varje mätvärde behöva kopplas med en tidpunkt och en sensorenhet. Varje sensorenhet kommer därav behöva tilldelas ett unikt namn eller ID. För att minimera mängden information vid överföringen så kan man istället för att skicka tidpunkten som mätdatan samlades in, anta att den tid då den mottagande nätverksnoden fick meddelandet kan approximeras till samma tid som när mätvärdet registrerades. Detta är mer resonabelt när mätningarna sker mer infrekvent.

5.3.2 Metadata

För att tolka mätvärderna på ett vettigt sätt är det viktigt med information om var varje sensor är uppsatt och vad det är den mäter. Denna informationen kommer inte att förändras tillräckligt ofta för att behövas skickas med varje paket från sensorn. Därav sparas denna informationen i en annan databastabell för att undvika redundans och spara bandbredd.

5.3.3 Molnbaserad databas

I detta projekt används en serverlös molnbaserad lösning för att hantera och lagra all data. Med en serverlös arkitektur så kan man lätt skala upp eller ner prestandan och lagringsutrymmet efter behov. Samtidigt behöver man ofta inte ha hand om några servrar, installationer och mjukvaror.

5.4 Webbapplikation

Valet att konstruera en webbapplikation baserades på att få en plattformsoberoende, installationslös och lättdistributerad applikation för att låta användare testa applikationen lättare under produktion.

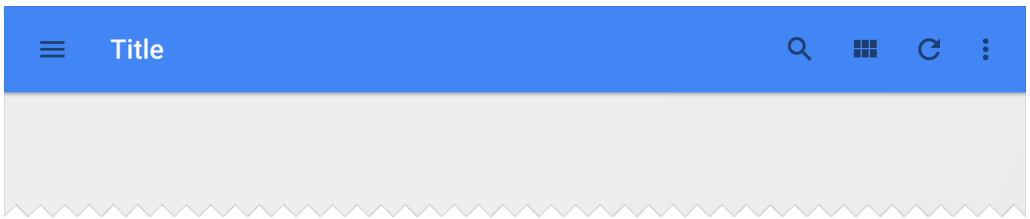
5.4.1 Hårdvara

Projektet kommer att behöva en webbserver för att publicera frontend applikationen på. Det finns flera alternativ för en server men det optimala är att ha en molnbaserad lösning. Detta för att flytta ansvaret att underhålla och hantera servern till en annan part. Andra alternativ som finns är att förskaffa sig en VPS. Man behöver inte underhålla hårdvaran för servern men man behöver underhålla mjukvaran i den. Det tredje alternativet är att publicera applikationen på en egen server men då behöver man tillhandahålla allt själv. Detta inkluderar då hårdvara, elektricitet och server mjukvaran.

5.4.2 Grafisk design

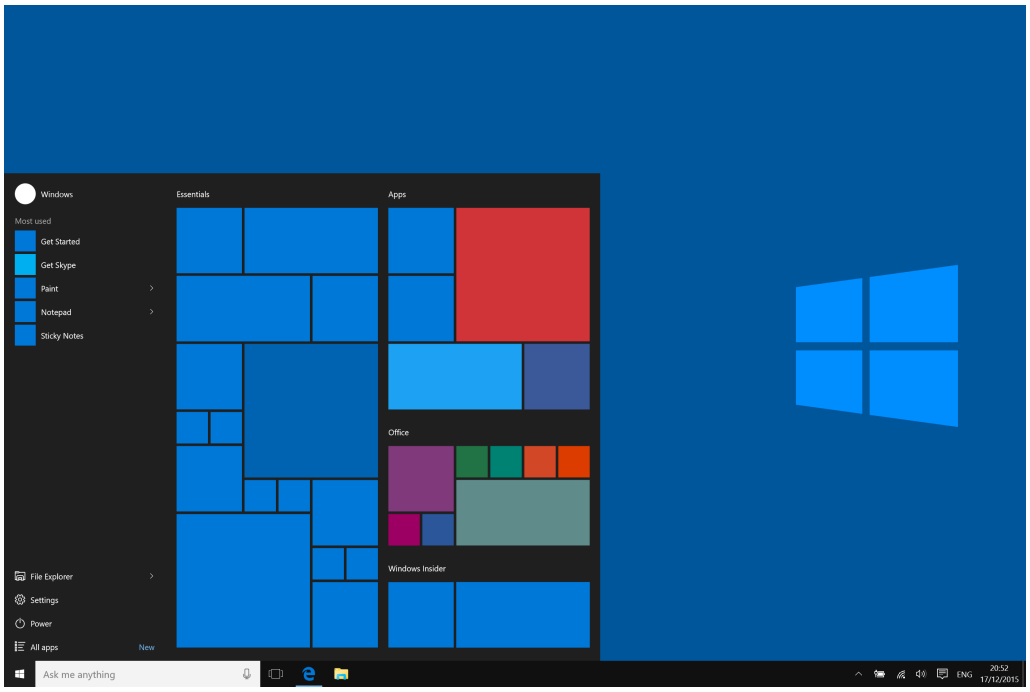
Designen på hemsidan ska vara sådan att användaren lätt kan navigera sig igenom den information som finns tillgänglig. Designen ska till viss del baseras på den standard som finns. Detta för att kunna hålla det konsekvent och att användaren ska kunna känna igen den funktionalitet som kommer med den standarden.

Det finns olika designmallar som man kan använda sig av. En av dessa är Googles Material design, där ett exempel på hur det ser ut ses i figur 5.1. När man ska använda Material design finns det många riktlinjer och dokumentation över hur det ska implementeras. Det gäller till exempel på vad användaren ska få för typ av respons från gränssnittet via en interaktion. Eller vilka avstånd det ska vara mellan elementen i applikationen.



Figur 5.1: Hur ett verktygsfält kan se ut i Material Design, från Google [10].

En annan vanlig design är flat design. Det är en minimalistisk design där det inte finns många strikta regler över hur något ska se ut. Exempel på en design är till exempel startmenyn från Windows 10 som kan ses i figur 5.2.



Figur 5.2: Exempel på en flat design, från Microsoft [11].

En responsiv design för applikationen ska användas för att ge användarna en bra upplevelse. Det kommer också ge möjligheten för användarna att kunna använda applikationen på valfria plattformar utan att det blir en försämrad upplevelse än vad som förväntas.

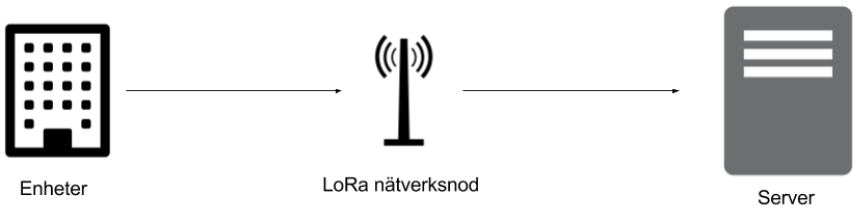
6

Implementation

Det här kapitlet redogör för de val som har gjorts under projektets gång och vid skapandet av webbapplikationen. Först så behandlas sensorsystemet i sektion 6.1. I nästa sektion 6.2, berörs avkodningen av mätdatan och överföringen in till databasen. Sektion 6.3 diskuterar databasen, dess design och implementation. Sist i sektion 6.4 förklaras hela utvecklingsprocessen för webbapplikationen.

6.1 Sensorsystem

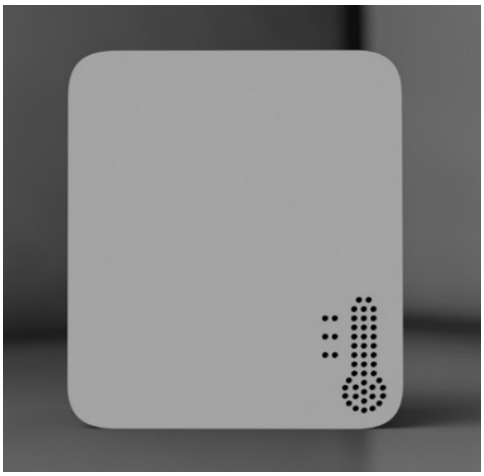
Sensorerna som har använts under denna rapports omfattning av projektet är temperatur- och luftfuktighetssensorer. I början av projektet användes ungefär 200 sensorer vilket har ökat till närmare 700 sensorer vid projektets slut. Den här sektionen behandlar sensorerna och deras överföringssätt. I figur 6.1 nedan syns en överblick av systemet från sensorer till överföring.



Figur 6.1: Överblick av det befintliga sensorsystemet.

6.1.1 Sensorer

Sensorerna som används i projektet är av modell OY1100 av företaget OnYield. OY1100 är en trådlös sensor som mäter temperatur och luftfuktighet och som använder den trådlösa tekniken LoRa. Med hjälp av LoRa har OY1100 över 10 års batteritid, lång räckvidd samtidigt som den har en vikt på 15g. En bild på OY1100 finns nedan i figur 6.2, tagen från OnYields hemsida [17].



Figur 6.2: OnYield OY1100 - Trådlös temperatur- och luftfuktighetssensor, från OnYield [17].

6.1.2 Uppladdning till molnet

Sensorerna överför sina mätvärden till de sammankopplande nätverksnoderna med LoRa. Normalt överför de sin data var sjätte timme och skickar då sin nuvarande mätdata, mätdatan för 2 timmar sen och 4 timmar sen. Nätverksnoderna vidarebefordrar sedan paketet baserat på vilket applikationsidentifikation som sensorn är kopplad med.

6.1.3 Packetbeskrivning

Sensorerna skickar i varje paket sitt unika ID, sekvensnummer och sin data. De sammankopplande nätverksnoderna lägger sedan till extra information vid mottaget paket. I detta paket är tiden det mest relevanta för detta projekt vilket är den tid som används för mätdatan. Fördelarna med att låta en extern enhet hantera detta är att sensorenheterna inte behöver ha klockor som är korrekt tidsinställda och sparar in bandbredd på grund av lägre paketstorlek.

6.2 Hantering av sensordata

Den här sektionen behandlar mottagningen av data från sensorerna, avkodning och insättning i databasen som görs på en VPS. Lösningen som användes för detta projekt var att sätta upp en virtuell maskin på en virtuell privat server som kör en Windows tjänst som agerar MQTT broker och en annan Windows tjänst som sköter avkodningen och insättningen i databasen.

Valet att lägga dessa tjänster på en VPS i molnet istället för att implementera

samma funktionalitet med molntjänster var främst på grund av brist på resurser. I framtiden vore det rimligt att flytta över dessa till en serverlös arkitektur. Dels för att alla andra delar följer det mönstret men även för att helt och hållet slippa serveradministrativa uppgifter och personal.

6.2.1 MQTT Broker

Mottagningen av data sker med hjälp av en så kallad MQTT Broker som är den mottagande och lyssnande delen i ett MQTT system. Till detta projekt valdes en färdigutvecklad MQTT Broker, Mosquitto vilket är en MQTT Broker som även har öppen källkod [8].

6.2.2 Avkodning

Avkodningen av sensorernas mätdata beror på vilken modell av sensor som används. För modell OY1100 som enbart är den modell som använts i detta projekt sker avkodningen på detta sätt:

Exempeldata: 12 1A D7 | 12 1A 46 | E9 19 CE
Motsvarande tid: -4 timmar | -2 timmar | Nuvarande

Här är de första tre hexadecimala paren datan för fyra timmar tidigare än de tre sista hexadecimala paren. De hexadecimala paren avkodas manuellt nedan.

Temperaturdata motsvaras av stora X i detta mönster XX YY XY, stora Y motsvarar informationen om luftfuktighet.

Nedan beskrivs den manuella avkodningen för 12 1A D7 vilket motsvarar datan för fyra timmar innan tidsstämplen.

$$Temperatur = 0x012D * 0.1C^\circ = 30.1C^\circ$$

$$Luftfuktighet = 0x01A7 * 0.1\% = 42.3\%$$

I de följande två formlerna beskrivs den manuella avkodningen för 12 1A 46 vilket motsvarar datan för två timmar innan tidsstämplen.

$$Temperatur = 0x0124 * 0.1C^\circ = 29.2C^\circ$$

$$Luftfuktighet = 0x01A6 * 0.1\% = 42.9\%$$

Nedan redogörs den manuella avkodningen för E9 19 CE vilket motsvarar datan samtidigt som tidsstämplen. I detta fall blir det lite mer komplicerat då temperaturen är negativ.

$$0x0E9C||0xF000 = 0xFE9C = -356 \rightarrow Temperatur = -35.6C^\circ$$

$$Luftfuktighet = 0x019E * 0.1\% = 42.9\%$$

6.2.3 Insättning i databasen

Sista steget för hanteringen av sensordatan är att sätta in de värden som har mottagits av sensorerna redan och lagra dem i en strukturerad databas. Detta sker på två olika sätt, där det första är när sensorn inte är tidigare känd. Då läggs den in i en tabell med sitt enhetsindentifikation som är unikt för databasen, ett ID för vilken typ av sensor det är och sist ett tillfälligt ID för rummet den är placerad i. Däremot om sensorn redan är känd och ligger i tidigare nämnd tabell läggs sensorns data in avkodat i en annan tabell för sensorvärden. Dessa kopplas med en tidsstämpel och sensorns enhets ID för att man sedan ska kunna veta vilken enhet som datan kom ifrån och när. Datan lagras avkodat i JSON med sin sensortyp som nyckelord för att sedan i det grafiska gränssnitt kunna generera grafer för de olika sensortyperna.

6.3 Databas

Följande stycke beskriver design processen och implementationen av databasen som används i detta projekt. Slutligen nämns de fördelar som finns med att använda en serverlös arkitektur vid skapandet av en databas.

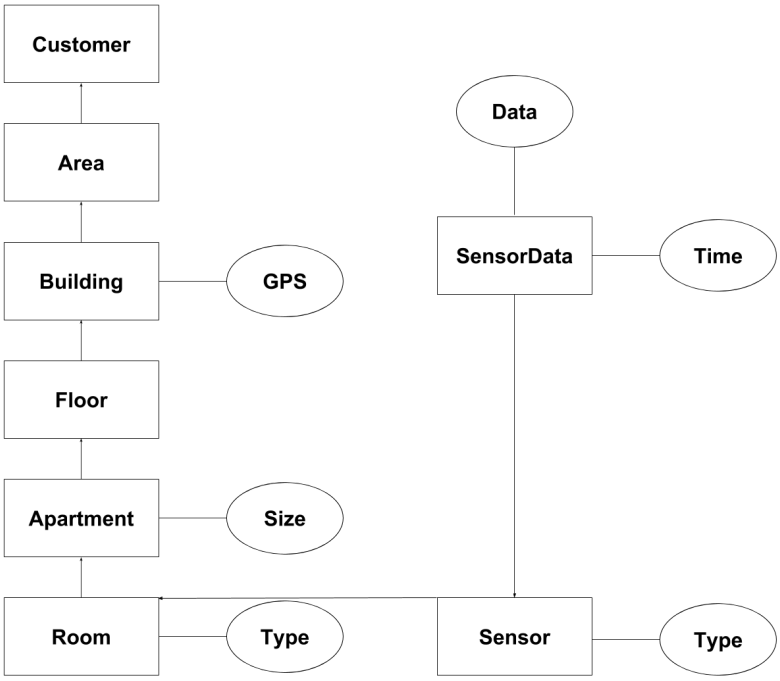
6.3.1 Design

Vid utvecklandet av databasen lades stort fokus på en lätt utvidgningsbar design utan redundans. Enligt specifikationer från kund, skulle sensorerna delas in i ett hierarkiskt träd. Trädet ska utgå ifrån områden där byggnader sedan läggs in. Varje byggnad har ett antal våningar och varje våning har lägenheter. I varje lägenhet finns sedan ett antal rum som innehåller sensorerna. Alla dessa steg: område, byggnad, våning, lägenhet, rum och sensor gjordes till entiteter i en entitet-relationsdatabas med en relation till föregående i listan. Detta visualiseras i figur 6.3 som är det ursprungliga diagrammet som databasen byggdes efter dock med vissa modifikationer.

Varje rum och sensor har sedan en attribut som anger vilken typ av rum eller sensor det är. Varje entitet kan sedan utökas med ytterligare attribut, exempel på detta kan ses i figur 6.3 där GPS koordinater för varje byggnad och storlek för varje lägenhet lades till i tabellen.

Det som är beskrivit hittills är det som anses vara metadata, information som inte kommer att ändras frekvent utan kan till stor del definieras i förväg.

Slutligen så finns sensordata entiteten som är de mätdata som respektive sensor mäter. Datan kopplas med ett enhetsindentifikation för sensorn och en tidsstämpel.



Figur 6.3: Simpel variant på ett entitet-relationsdiagram som användes vid förstudien för projektet.

6.3.2 Implementation

Databasen implementerades i MS SQL som beskrivet i föregående stycke med några modifikationer. Exempelvis gjordes entiteten som representerar kunderna (se figur 6.3) om så att varje kund eller användare får en egen metadata och sensordatatabell. Detta för att enklare kunna kontrollera storleken och kostnaden baserat på kundens användning.

I figur 6.4 kan databasen ses i sin helhet vid slutförandet av projektet. Här ser man även att sensortyp har förändrats ifrån ett attribut i den ursprungliga designen till en egen entitet för att kunna lägga till ytterligare information om en specifik sensortyp.

I figur 6.4 har även en relation lagts till mellan sensordata och sensor med enhetens ID som nyckel.

Figur 6.4: Diagram taget ifrån databasen på MS SQL server.

6.4 Webbapplikation

Denna del kommer att behandla den utvecklingsmiljö som projektet haft i sektion 6.4.1. I sektion 6.4.2 kommer de val som gjorts under utvecklandet av gränssnittet. Hur gränssnittet populeras med data hanteras av sektion 6.4.3. Slutligen behandlar sektion 6.4.4 processen av att publicera webbapplikationen på Azure.

6.4.1 Uppsättning av utvecklingsmiljö

Innan någon utveckling påbörjas behövs en ordentlig utvecklingsmiljö sättas upp för att få en snabb utvecklingsprocess. För detta projekt valdes det att använda sig utav en standard kodmall även kallat boilerplate. Den användes för att snabbt kunna starta en utvecklingsmiljö och tillåta att utvecklandet kunde påbörjas snarast.

Kodmallen skapade en färdig projektstruktur och en grundläggande webbapplikation med inbyggt stöd för React, Asp.net mm. Detta var en bra grundläggande struktur som tillät snabbt och smidigt kodande och implementerande av ny funktionalitet.

En funktionalitet i kodmallen var att en intern webbserver sattes upp som gjorde webbapplikationen tillgänglig för testning på den lokala utvecklingsmaskinen. Utöver detta hade utvecklingsmiljön även koll på om någon av kodfilerna ändrades och då kompilerades och uppdaterades den lokala webbapplikationen automatiskt vilket gjorde att testning och felsökning gick snabbt relativt att behöva omkompilera och starta om webbservern.

Kodmallen hade en färdig filstruktur för både server och klientdelen av webbapplikationen. Det här medförde att själva implementeringen inte behövde göras för projektet, utan endast behövde modifieras för att passa kraven för webbapplikationen.

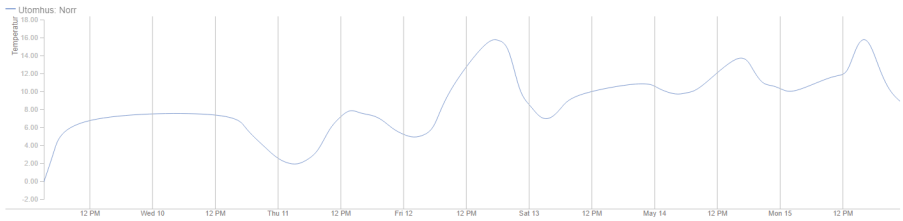
6.4.2 Webbutveckling

Eftersom att webbapplikationen använde sig utav React kunde många färdigutvecklade tredjeparts komponenter användas. Det här tillät enkla byten och testning av olika komponenter för att vad som passar in på detta projektet bäst.

6.4.2.1 Grafer

Under utvecklingsfasen av projektet användes två olika grafbibliotek. Recharts var det första biblioteket som implementerades men under utvecklingen bytes biblioteket ut då det inte var bra på att hantera data med datum. Eftersom syftet med graferna var att visa upp tidsserie diagram behövde valet utav bibliotek utvärderas. React Timeseries Charts blev det grafbibliotek som valdes istället. Det valdes på grund av att bibliotekets syfte var att hantera data med icke kontinuerliga tidsstämplar.

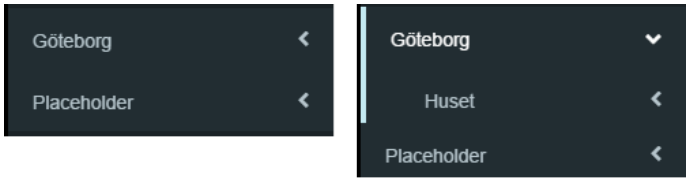
Information som skulle visas i graferna laddas in i komponenten via ett mellanprogram som är skrivet i ASP.net som kan läsas i sektion 6.4.3.



Figur 6.5: En av de grafer som finns i applikationen.

6.4.2.2 Hierarkiträd

För att skapa hierarkiträdet till applikationen behövs en lista med barn där varje nod kan expandera och fällas ihop. React-sidemenu används för att få den typ av funktionalitet som krävs. I figur 6.6 kan man se hur menyn ser ut i applikation i ett ihopfällt läge respektive ett expanderat.



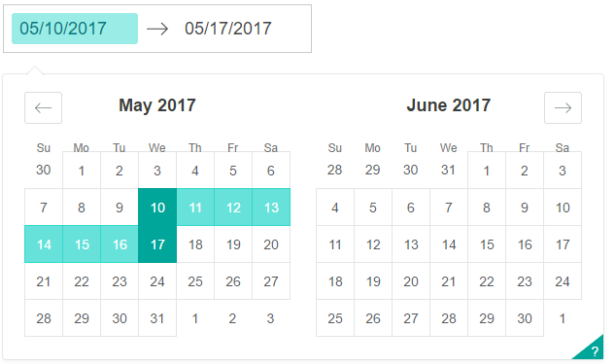
Figur 6.6: En meny i applikationen före och efter expandering.

Menyn används för att välja de sensorer som ska illustreras i graferna. För att välja en sensor behöver man då navigera sig ner i trädet tills att man kommer in i en lägenhet (se figur 6.7) och man kan då välja vilka rum som ska illustreras.

Figur 6.7: En helt expanderad gren i hierarkiträdet.

6.4.2.3 Datumväljare

Biblioteket react-dates användes för att skapa en datumväljare. Vid applikationens start kan användaren se ett intervall som är förvalt. När användaren väljer start- eller slutdatumet som visas kommer en ny ruta att visa sig där användaren kan välja ett annat intervall (se figur 6.8). Det valda tidsintervallet kommer sedan att användas för att välja vilken data som ska hämtas från databasen.



Figur 6.8: Applikationens datumväljare vid interaktion.

6.4.2.4 Verktygsfält

Andra inställningar och val som användaren kommer att kunna göra ska läggas i ett verktygsfält längst upp på sidan. Verktygsfältet består av ett antal knappar som vid knapptryck kommer att visa en gömd meny där fler inställningar och val finns. Menyn är gömd för att inte använda för mycket plats som istället kan användas åt graferna.

6.4.3 Kommunikation med databas

Kommunikationen med databasen i backend sker via det ASP.net mellanprogram som också agerar webbserver åt klienterna. I React är det enkelt att hämta JSON data från ett REST API. Av den anledningen agerar mellanprogrammet som ett API där React kan hämta data ifrån och populera gränssnittet.

Mellanprogrammet använder sig utav entity framework för att hämta data från databasen. Entity framework använder man för att enklare hämta data från en databas. Man behöver inte skriva en query utan det räcker att man använder sin modell som man har definierat. Hur detta görs i C# kan ses i exempel 6.1. Hur man gör med entity framework kan ses i exempel 6.2. En tydlig skillnad är mängden kod som behövs skrivas men datan som hämtas läggs direkt i ett objekt. Det medför att resterande hantering av datan blir enklare eftersom den redan är omvandlad.

Exempel kod 6.1: Hur man hämtar data med C#.

```
String sql = "SELECT * FROM table WHERE id = 10";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["NAME"];
```

Exempel kod 6.2: Hur man hämtar data med entity framework i C#.

```
Item p = repository.GetItem(10);
String name = p.getName();
```

6.4.4 Publicering till webbservern

För att webbapplikationen ska kunna användas på ett smidigt sätt krävs det att den läggs upp på en webbserver. Denna kan användas av de flesta webbläsare genom att ansluta till servern via en URL.

Azure Web Service är den serverlösa webbserver som valdes för detta projekt. Detta för att undvika det extra jobb som kommer med att sätt upp en webbserver. Man fick även möjlighet till ett fleratal olika sätt att publicera applikationen. De sätt som testades i projektet var att publicera via Visual Studio och med hjälp av ett Git repository.

Det sätt som användes i slutet blev ett Git repository eftersom det fanns ett färdigt hjälpmedel för det i den kodmall som använts. När det är dags att publicera kan man då köra ett kommando och efter ett fåtal minuter finns den version man jobbade på i molnet.

Innan man kan göra webbapplikationen tillgänglig för allmänheten behöver projektet byggas och kompileras. Den kod som är skriven i ASP.net kompileras till en dll-fil som webbservern kan köra. När alla huvudfiler finns på webbservern kan applikationen startas. Vid anslutning till klienten skickas en html-fil, en JS-fil och övrig CSS som krävs. All react kod slås ihop till en JS-fil som skickas med html-filen för att klientdelen ska fungera. CSS-filerna används för att formatera och ändra utseendet på webbsidan.

7

Resultat

Projektet resulterade i en fungerande prototyp av webbapplikationen där alla komponenter som krävs för systemet är uppsätta på Microsoft samling av molntjänster kallat Microsoft Azure.

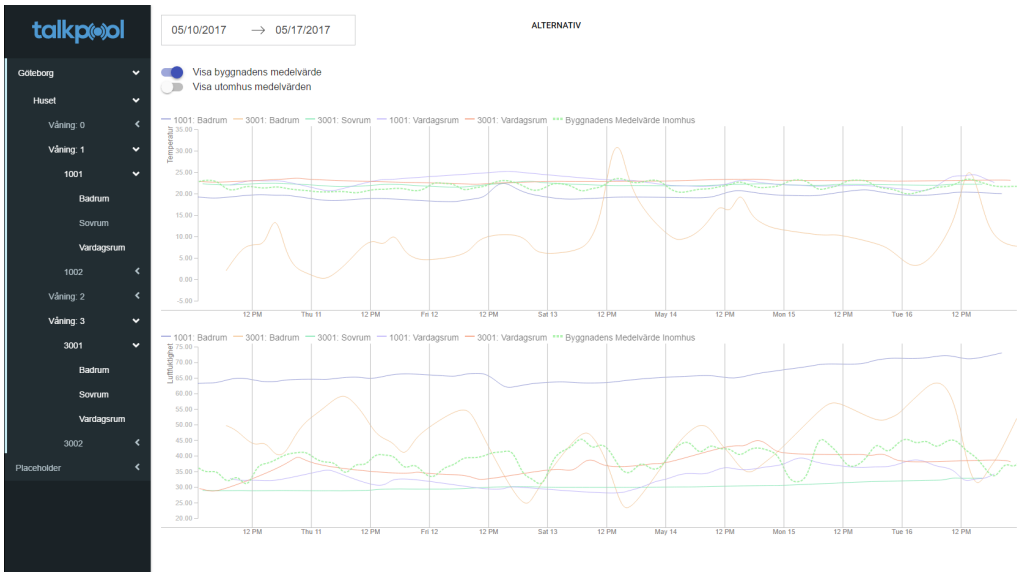
7.1 Produkten av projektet

Det system som har utvecklats innehåller allt som behövs för att få in sensordata från sensorerna via MQTT till att redovisa informationen i en webbapplikation tillgänglig för allmänheten som en webbsida (se figur 7.1).

En MQTT broker har skapats och implementerats på en molnbaserad VPS. Programmet kan ta emot information från de sensorer som är aktiva och sedan skicka denna information vidare till en databas för att lagras.

En databas skapades som en molntjänst. Den är initierad, funktionell och implementerad med skalbarhet i fokus. Den får in mätvärden ifrån sensorer och ger sedan dessa till en webbserver vid begäran.

En webbserver har utvecklats som körs på en molnbaserad tjänst. Servern ansvarar för att hantera klienter som vill få åtkomst till webbsidan. Den agerar även som ett mellanprogram för applikationens kommunikation med databasen.



Figur 7.1: Bild på den färdiga webbapplikationen.

7.2 Webbapplikationens prestanda

Prestandamässigt har flesta operationer i webbapplikationen en svarstid på runt 100 millisekunder (räknat med en internetuppkoppling med en svarstid på 30 millisekunder) vilket är acceptabelt enligt [16]. De tyngsta operationerna såsom sidladdning och de mer avancerade graferna som byggnadens medelvärde tar mellan 800 millisekunder och 1500 millisekunder vilket också anses vara acceptabelt enligt [16].

Vissa operationer upplevs däremot som snabbare, exempelvis sidladdningen på grund av webbapplikationens komponentbaserade uppbyggnad. Så fort en del av webbapplikationen är klar så renderas den vilket gör att en stor del av laddningstiden kan användas av användaren för navigation mm.

7.3 Systemets kostnad

Kostnaden för att implementera hela systemet på Microsoft Azure på samma sätt som under utvecklingen skulle kosta 1937,42 SEK per månad (beräkning baserad på priser från 2017-05-18).

Utav denna summa kommer 590.5 SEK per månad från den SQL server som används. Den virtuella maskin som MQTT brokern ligger på står för 761.3 SEK per månad. Slutligen kostar den webbserver som används 585.62 SEK per månad.

Dessa system är för tillfället överdimensionerade och skulle kunna minskas till en bråkdel av kostnaden om inte många användare och sensorer kommer användas.

8

Diskussion

Detta kapitel kommer först i sektion 8.1 att bestå av en diskussion angående de resultat som observerats under projektets gång. Efter kommer en kort redogörelse över projektets miljöaspekt i sektion 8.2. Sektion 8.3 kommer att behandla vad det finns för möjligheter till vidareutveckling för projektet.

8.1 Utvärdering av resultat

Denna sektion kommer innehålla en diskussion om det resultat som projektet har tagit fram.

8.1.1 Webbapplikationens prestanda

Som beskrivet i resultatskapitlet i sektion 7.2 har webbapplikationens prestanda utvärderats efter Jakob Nielsens arbete angående människans upplevelse av responstid [16]. Under hela projektets gång har webbapplikationens funktionaliteter utvecklats så att de håller dessa ramar med rimlig marginal. Detta innebär att alla nya funktionaliteter ska vara tillräckligt optimerade för att kunna tillföra andra funktionaliteter utan att responstiden ska gå över de tidsgränser angivna i Nielsens arbete.

Denna syn på optimering har använts för att få en vettig balans på hur mycket nya funktionaliteter som implementerats och mängden optimering som gjorts. Ett par undantag har gjorts på grund av tidsbegränsningen i projektet. Dessa är främst optimering vid tidsspann över 3 månader och optimering av hierarkisystemet. Större tidsspann av data har ej testats i stor utsträckning, främst på grund av insamlingen av data endast pågått under ett par månader.

8.1.2 Systemets arkitektur

Systemet har överdimensionerats under utvecklingsprocessen som tidigare nämnt i sektion 7.3. Detta gjordes för att inte lida av prestandabegränsningar under utvecklingsprocessen och för att låta systemet växa obegränsat. Även om omdimensionering

hade varit enkelt med de molnbaserade tjänster som användes var antalet sensorer beroende på annan part. Därav överdimensionerades systemet något för att kunna hantera om någon stor plötslig förändring skulle ske.

8.1.3 Systemets kostnad

Prisjämnförelser gjordes mellan några stora leverantörer av molntjänster. Priserna skiljde sig inte markant utan var ungefär de samma mellan de olika leverantörerna. Största skillnaden ligger främst i de specialtjänster som de erbjuder, dock vid användning av dessa kan man bli beroende av en viss leverantör. En annan skillnad är placering av deras datacenter och serverhallar. Vid utveckling av lokala tjänster kan det vara relevant att välja leverantör baserat på detta för att få bättre prestanda och responstider.

8.1.4 Agil utveckling

Arbetssättet som har använts under projektets gång har fungerat bra. En bra projektledare är väsentligt för att en agil utvecklingsmetod ska fungera på bästa sätt. I detta projekt har vi fått uppleva hur viktigt det är med en bra projektledare. Vi fick en bra struktur på projektet där vi hade planerat in vad som skulle göras och vad som skulle uppnås. Projektledare såg även till att vi följde den planering som gjorts och hjälpte till att leda gruppen i rätt riktning.

8.1.5 Arbetsbörda

Något som inte togs upp i resultat var den mängd arbete som gick åt för att skapa en molnbaserad applikation. Största anledningen till det var att hur långt tid det tog att skapa applikationen kändes irrelevant för denna rapport. Det skulle eventuellt bli relevant om man skulle undersöka om det tar längre eller kortare tid att utveckla en molnbaserad applikation. Men i och med att det är starkt kopplat med de funktionaliteter och vilken typ av webbapplikation som utvecklats blir det irrelevant om inte en väldigt liknande webbapplikation skulle utvecklas.

Personligen tror vi författare av den här rapporten att tidsåtgången för att utveckla en molnbaserad applikation borde vara samma om inte mindre för en som inte är molnbaserad. Detta för att den utveckling som vi gjort för en applikation hade varit likadan oavsett om den ska vara molnbaserad eller inte. Detta eftersom vi fortfarande skapat en webbserver och tillhörande gränssnitt som lika väl kunde ha körts på en lokal webbserver.

Vi tror att det finns möjlighet för att det ska ta mindre tid på grund utav att man slipper göra installationen för de olika servrar som använts. Med serverlösa

molnbaserade tjänster var det bara att specificera vad man ville ha och sedan fanns de resurserna tillgängliga inom kort.

8.2 Projektets miljöaspekt

Syftet med webbapplikationen var att ge bostadsägare en möjlighet att se en översikt över sina bostäder. Denna information kan sedan användas för att hitta problem med luftfuktighetsnivåer och temperaturer vilket kan leda till mindre skador på bostäderna och därmed spara in på natur och kostnader som annars hade krävts vid renovering.

En annan stor punkt är att i viss mån se energikonsumtionen i de temperaturer som olika delar av lägenheten ligger på. Med denna information skulle man kunna införa individuella uppvärmningskostnader, vilket skulle leda till att de enklare kan minska sin energikonsumtion för att inte behöva betala en större elräkning.

8.3 Vidareutveckling

I den här sektionen tas de områden som vore relevanta vid vidare utveckling att bearbeta, göra om eller implementera för att få ett bättre system. Dessa funktionaliteter implementerades inte på grund av brist på tid eller att de först insågs vara ett bättre alternativ för sent i projektet.

8.3.1 Produktorderstock

I projektets förstudie togs det fram en produktorderstock för en komplett applikation som från början var större än den tid som fanns i projektet. Detta för att enklare kunna möjligöra vidareutveckling för projektet.

8.3.2 Fullständig serverlös tillämpning

I och med att lösningen som beskrivs i sektion 6.2 för tillfället använder sig av en VPS istället för en serverlös arkitektur kommer underhåll fortfarande krävas. Därav kan det vara relevant att implementera denna del av systemet på ett annat, serverlöst sätt.

8.3.3 Serverbaserad beräkning och rendering

Vissa beräkningar och rendering som för tillfället görs lokalt på användarens maskin skulle kunna flyttas över till servern som tillhandahåller webbapplikationens resurser. Exempel på arbetsuppgifter kan vara skapandet och renderingen av hierarkiträdet och beräkningen av medelvärde för byggnaden.

Detta skulle ge optimering för många områden för webbapplikationen, mindre data skulle behöva överföras och behandlas på databasen, webbapplikationsservern och på användarens maskin.

Detta skulle även spara energi då många beräkningar endast skulle behöva göras enstaka gånger istället för flertalet gånger på varje användarmaskin.

8.3.4 Stöd för att dynamiskt lägga till nya sensortyper

Ett problem som finns i det system som utvecklades under detta projekt är att lägga till nya sensortyper kräver ytterligare programmering i MQTT broker-delen och webbapplikationen. Detta innebär även att de behöver kompileras om och redistrubuteras till molnet.

Funktionaliteten har delvis implementerats i webbapplikationen där graferna stödjer nya sensortyper i visningen men kräver att en sträng anges för att visa rätt beteckning.

Så istället för att hårdkoda denna information borde den vara med i sensortypstabellen i databasen. Detsamma gäller den avkodningsalgoritm som krävs. Det implementerades inte under detta projekt för att endast sensorstypen temperatur och luftfuktighet och modellen OY1100 skulle användas i detta stadium. Ett annat problem som skulle behöva lösas är att utveckla ett sätt för att få ut sensorns typ från dess unika ID.

8.3.5 Inloggningssystem

För detta projekt skulle det inte läggas någon extra tid på säkerhet i applikationen. Applikationen har därför inget inloggningssytem och all data som samlats in finns tillgängligt för alla att se. Det finns dock inget för tillfället som kan koppla denna datan till en specifik byggnad. Detta eftersom de sensorer som är kopplad till en byggnad blev kopplade utan hänsyn till deras riktiga position.

Ett inloggningsystem behövs för att dels kunna dölja känslig information från allmänheten men även för att olika bostadsägare ska kunna se den information som är relevant för dem.

9

Slutsats

I det här projektet har vi utvecklat en webbapplikation. Fokus låg på att utveckla systemet med serverlös arkitektur. Resultat ifrån systemets kvalitet togs i form av kostnadsanalys och mätning av systemets prestanda. De resultat som erhölls anses vara godtagbara enligt forskning.

Rent teoretiskt sett anser vi att molnbaserade tjänster och serverlös arkitektur är framtiden. Man kan på ett enkelt sätt skala upp eller ner hårdvara efter behov samtidigt som man undviker många andra administrativa arbetsuppgifter. Däremot är dagens implementationer inte de bästa för mer specialiserade användningsområden. Det är ofta som man gör en lösning som använder sig av en virtuell maskin på en virtuell privat server på en bråkdel av tiden än vad det skulle ta att införa samma på ett serverlöst sätt. Slutsatsen som kan dras här är att det i dagsläget inte är ett självklart val, utan det beror på kravspecifikationerna och om systemet kräver några speciallösningar som inte går att implementera lätt med befintliga molntjänster.

Litteratur

[1] ISO/IEC JTC 1. *ISO/IEC 20922:2016*. Juni 2016. URL: https://www.iso.org/standard/69466.html (hämtad 2017-05-02).

[2] Beck et al. *Manifesto for Agile Software Development*. Febr. 2001. URL: http://agilemanifesto.org/ (hämtad 2017-05-09).

[3] LoRa Alliance. *LoRa Technology*. URL: https://www.lora-alliance.org/What-Is-LoRa/Technology (hämtad 2017-04-28).

[4] Daniel Arenhage och Fabian Lyrfors. "Migrating and testing distributed cloud based web applications". Examensarb. Chalmers tekniska högskola, 2012.

[5] Atlassian. *Comparing Workflows*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows (hämtad 2017-03-24).

[6] Keith Augustsson och Jonas Fredriksson. "Cloud Service Analysis - Choosing between an on-premise resource and a cloud computing service". Examensarb. Chalmers tekniska högskola, 2011.

[7] Rickard Edfast och Oskar Lindström. *Universell IoT-enhet För dataregistrering och processtyrning*. 2014.

[8] The Eclipse Foundation. *An Open Source MQTT v3.1 Broker*. URL: https://mosquitto.org/ (hämtad 2017-06-13).

[9] GitHub. *Atom*. URL: https://atom.io/ (hämtad 2017-06-13).

[10] Google. *Structure - Layout - Material design guidelines*. April 2017. URL: https://material.io/guidelines/layout/structure.html#structure-toolbars (hämtad 2017-05-12).

[11] Microsoft. *Windows 10 abstract*. Dec. 2015. URL: https://commons.wikimedia.org/wiki/File:Windows10abstract.png (hämtad 2017-05-12).

[12] Microsoft. *SQL Server dataplatform*. URL: https://www.microsoft.com/sv-se/sql-server/ (hämtad 2017-06-13).

[13] Microsoft. *Vad är Azure – molntjänst från Microsoft | Microsoft Azure*. URL: https://azure.microsoft.com/sv-se/overview/what-is-azure/ (hämtad 2017-06-13).

[14] Microsoft. *Visual Studio IDE, Code Editor, Team Services, & Mobile Center*. URL: https://www.visualstudio.com/ (hämtad 2017-06-13).

[15] Mozilla Developer Network. *HTML*. April 2017. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (hämtad 2017-04-27).

[16] Jakob Nielsen. *Response Times: The 3 Important Limits*. Jan. 1993. URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (hämtad 2017-05-16).

[17] OnYield. *Temperature Sensor LoRa LoRaWAN LPWAN IoT*. URL: http://onyield.com.hk/index.php?route=product/product&product_id=42 (hämtad 2017-05-09).