Norwegian University of Science and Technology

Department of Engineering Cybernetics

Useing Cloud Services for Data Exchange with IoT like devices

By:
Marit Schei Tundal
marittu@stud.ntnu.no

Supervisor: **Geir Mathisen**

Co-supervisor: **Espen Helle**

December, 2017

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Prosjektoppgave

Kandidatens navn:	**Marit Tundal**
Fag:	**TTK4550**
Oppgavens tittel (norsk):	**Bruk av skytjeneste for datautveksling med IoT-lignende enheter**
Oppgavens tittel (engelsk):	**Using Cloud Services for data exchange with IoT like devices**

Oppgavens tekst:

I denne oppgaven ønsker en å se på hvordan små IoT-lignende enheter plassert rundt i omgivelsene kan kommunisere effektivt mot en skyløsning. I skyen skal det foretas noe databehandling og beslutninger/aksjoner skal utføres. Videre skal resultatene sendes til operatør. Operatøren skal også ha muligheter til å kommunisere med, sende meldinger til, enhetene som er plassert i omgvelsene

Oppgaven består av følgende punkter:

1. Gi en oversikt over hvilke tjenester som finnes i forskjellige skyløsninger som tilbys, både løsninger som er beregnet for egen drifting og løsninger hvor tilbyder drifter skyløsningen.

2. De mest kjente tilbyderne av skytjenester.(feks. Amazon, Google, Microsoft) tilbyr forskjellige typer databaser. Sett opp relevante kriterier for vurdering av disse for vårt formål. Vurder styrker og svakheter ved hver av løsningen.
   På samme måte har skyleverandørene forskjellige løsninger for IoT sanntidskommunikasjon mellom enheter. Sett opp relevante kriterier for bruk av disse rammeverkene for fjernstyring.

3. Velg et av konseptene. Implementer en fornuftig strukturert databaseløsning med innsamling av data enhetene. Lag en enkel webløsning som kan presentere dataene.

4. Ta en av løsningene for fjernstyring og implementer slik at en kan se alle sensorene på hver bluetoothnode på web via skylevrandøren i sanntid. På samme måte skal en kunne styre outputs/parametere på hver node.
   Gjerne kombiner de to webdelene for hver sensor.

Oppgaven gitt:	21. august 2017
Besvarelsen leveres:	21. desember 2017

Utført ved Institutt for teknisk kybernetikk

Trondheim, den 21. august 2017

Geir Mathisen

Faglærer

Abstract

Sammendrag

Contents

List of Figures

Chapter 1

Introduction

This paper aims to discuss certain relevant topics related to cloud computing and presents a way of implementing bi-directional communication with IoT-like devices via a web-interface, using cloud services.

1.1 Cloud Solutions

Cloud computing changed the way information technology (IT) services are developed and deployed by providing the opportunity of moving computing processes from local computers to centralized facilities operated by third-party operators. Cloud solution providers are becoming more and more common, there are so many that it might be hard to keep track of the different providers and the services they offer.

1.2 Smart Grids

A Smart Grid is an electric grid that enables digital two-way communication to better utilize the energy infrastructure [1]. The traditional electric grid was built based on the idea that the electricity should be transferred directly from the centralized power plant to households and the communication flowing only one way. Today, with emerging trends toward renewable energy sources, decentralized power production and increased user flexibility, Smart Grids provides a solution to integrate these factors into a new, better and more robust electric grid.

By utilizing cloud services with Smart Grids, one can take advantage of the existing infrastructures that is available in most households today, namely the Internet. The different nodes in the Smart Grid can be organized as IoT nodes which can upload data measurements to the cloud. It is possible to provide a interface for both the electricity company to gather information from costumers, as well as for costumers to keep track of their usage in real-time. This could help distribute electricity consumption to avoid peaks and reduce costs for consumers as they can better monitor their own usage and get an overview of when power is cheaper. Internet-connected nodes also enables two-way communication.

This implementation consists of three peripheral nodes with multiple sensors connected to a central node which is the interface to the cloud.

1.3 Thesis Outline

Chapter 2 presents an overview of cloud computing, as well as some common terms on the subject. Chapter 3 discusses services from different Cloud providers, and assess the services based on certain criteria. Chapters 4 through 6 present a general implementation of the chosen cloud service provider. The result and discussion of the paper will be presented in chapter 7 and 8, respectively. Appendix A gives an overview of the hardware and the specific implementation in this project. Appendix B contains the source code developed in the project.

Chapter 2

Cloud Computing

Cloud computing is a way of sharing computer resources. It provides businesses and other users with the ability to minimize infrastructure costs and maintenance, without reducing performance. Cloud computing provides on-demand computing resources over the Internet, and users can save money with the pay-for-use payment option.

A widely used definition of Cloud Computing was provided by U.S. NIST (National Institute of Standards and Technology)[2],[3]: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.* [4] The definition further states that there are five essential characteristics possessed by Clouds, namely

- On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

- Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data center). Examples of resources include storage, processing, memory, and network bandwidth
- Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

2.1 Service Model

The service model for Cloud Computing is differentiated into three distinct models, namely Software as a Service, Platform as a Service and Infrastructure as a Service.

2.1.1 Software as a Service

Software as a Service (SaaS) is the highest level of abstraction, and provides on-demand access to any application. SaaS provides typically host and manages applications that are directly usable for end-consumers. The user does not have any control or the need to manage the underlying settings or infrastructure. Cloud service providers offer a set of software application, running on platform and infrastructure that the user is unaware of and does not own. The user pays only for what he or she uses and does not need to purchase anything. With

SaaS the users does not need to worry about development or programming of the software, as this is already taken care of by the Cloud provider.

Another definition of SaaS is made by [2], and states that *CLoud consumers release their applications on a hosting environment, which can be accessed through networks from various clients (e.g. web browsers, PDA, etc.) by application users.* This definition implies that the Cloud consumers control the software that they deploy to other users, however, they do not have control over the Cloud infrastructure.

A very commonly used application that is SaaS, are the Google Apps, such as Gmail.

2.1.2 Platform as a Service

By using Platform as a Service (PaaS), the user has more freedom when it comes to developing applications. The user still does not have control over the underlying cloud infrastructure, such as network, servers, operating systems or storage, but can use a development platform to create applications. The Cloud provider support certain programming languages, services, tool and libraries that the user can use to to deploy applications. The benefit of using PaaS is that *It facilitates development and deployment of applications without the cost and complexity of buying and managing the underlying infrastructure, providing all of the facilities required to support the complete life cycle of building and delivering web applications and services entirely available from the Internet* [3].

A well established PaaS is Google AppEngine, where developers write in Python or Java.

2.1.3 Infrastructure as a Service

Infrastructure as a Service (IaaS) gives the user the most freedom when it comes to developing applications. It is a form if hosting, where the IaaS provider takes care of the hardware and administrative services needed to store applications and a platform for running applications [3]. The Cloud provider takes care of managing and controlling the underlying infrastructure, however, the user has control over operating systems, storage, possibly network etc.[4] One of the benefits if IaaS, is dynamic scaling. Costumers only pay for what they use, thus potentially saving a great deal of money by not having to invest in hardware.

IaaS is usually divided into three parts, namely an environment for running virtual machines, storage through data centers, and compute power [3]. The virtual machine is built on top if the two others.

Amazon Web Services Elastic Compute Cloud (EC2) is an example of IaaS. Amazon lets their customers set up and configure virtual servers via a web-based interface.

2.2 Deployment Model

There are a few different deployment models for cloud computing. The three main ones, which will be discussed here, are public clouds, private clouds and hybrid clouds.

2.2.1 Public cloud

Public clouds are owned by organizations selling cloud services to the public. Public clouds are hosted on the Internet, and resources are offered as a service in a pay-per-usage model. Users share the same hardware, storage and network infrastructure.

The main advantages of public clouds are continuous data availability; automatic scalability on demand; limiting hardware and software expenses, as you only pay for the services you use; no maintenance, as this is done by the provider [5]. On the other hand, customers may be unaware of where and how the data is stored, as well as how secure the data is. There is also the issue of privacy, which will be discussed later on.

2.2.2 Private cloud

Private clouds are usually contained within and operated by a single organization. It can be managed by the organization itself, or by a third party. The cloud can only be accessed by the organization, thus providing a more secure infrastructure [5]. A private cloud may be cost saving for the company if it utilizes unused data capacity in the organizations' data center. Private clouds will usually provide the organization with greater control over the resources and infrastructure, as well as providing the organization with the opportunity to customize their layout and infrastructure of the cloud to their needs [6]. Another reason to utilize private clouds is the data transfer costs between local IT infrastructure to a public cloud [2].

The main disadvantages of a private cloud is that when it is compared to public clouds, the costs are higher. The costs of a private cloud are usually composed by the need of purchasing equipment, software, and staffing to maintain the cloud [5]. Another critical disadvantage is the lack of interaction with the "outside-world". This is where the hybrid cloud comes into play.

2.2.3 Hybrid cloud

A hybrid cloud is a combination of a private and public cloud. The users have a private cloud foundation [6] where one would store information sensitive to the organization. However, if one were in need of more storage space or needed to export applications etc. to the public, they would use a public cloud in addition. The private cloud is linked t one or more external cloud services [7], however it is provisioned as a singled unit. There would be no use in having just a private cloud isolated from the rest of the organization's IT resources.

An important factor for choosing hybrid clouds is that it provides the user with more secure control of the data and applications, while allowing third-party users to access information over the Internet [7]. The organization can still keep sensitive assets private while at the same time take advantage of resources provided in the public cloud [8], like accommodating fluctuations in traffic.

Chapter 3

Cloud Service Providers

There are hundreds of different cloud service providers. Following is a overview of the currently six biggest providers [9] [10].

3.1 Criteria for Evaluation

Suitable to hardware used in project Database - NoSQL Information availability Overall Private cloud availability

3.2 Amazon Web Services

Amazon Web Services is a cloud computing provider that offers a simple way to access servers, storage, databases and a broad set of application services over the Internet [11]. In total, Amazon provides over 50 different solutions where the main services include Amazon Elastic Compute Cloud (EC2) for compute, which is virtual servers in the Cloud; S3 for storage, providing scalable storage in the cloud; Aurora which is one of several databases, providing a High Performance Managed Relational Database. Amazon also provides several IoT solutions, for instance AWS IoT Platform, which is *a managed cloud platform that lets* *connected devices easily and securely interact with cloud applications and other devices.*

Amazon Virtual Private Cloud VPC lets you provision a logically isolated section of the Amazon Web Services cloud where you can launch AWS resources in a virtual network that you define. [12]

Amazon Web Services claim to have better cloud security than on-premises infrastructure [11].

3.3 Microsoft Azure

Azure is the only consistent hybrid cloud on the market [8]. Azure is the cloud for building intelligent applications. Data centers in 42 regions. Recognized as the most trusted cloud for U.S. government institutions, taking advantage of Microsoft security, privacy, and transparency. Ability to run any stack, Linux-based or Windows-based. Cloud Virtual Network is a comprehensive set of Google-managed networking capabilities, including granular IO address range selection, routes, firewall, VPN and Cloud Route.

Microsoft virtualization solutions go beyond basic virtualization capabilities, such as consolidating server hardware yo create comprehensive platforms for private and hybrid cloud [12].

3.4 IBM

Focus on enterprise innovation. 50 global data centers. Keep existing solutions on the private cloud. The Bluemix cloud platform is not just about creating new apps or migrating existing ones, on-prem or off-prem implementations, or offering IaaS and PaaS cloud services. It's designed to bring all of these aspects together to help you solve your real, complex business problems in the cloud [6]. Bluemix is a PaaS. IBM offers IaaS, SaaS and PaaS through public, private and hybrid cloud models. SmartCloud Foundation, SmartCloud Services and SmartCloud Solutions. "Cloud data stored at European data centers could still be handed over to American officials, as outlined by US law."

3.5 Google Cloud Platform

Google Cloud Virtual Network, lets you provision your Google Cloud Platform resources, connect them to each other and isolate them from one another in a Virtual Private Cloud (VPC). You can also define fine-grained networking policies with Cloud Platform, on-premise or other public cloud infrastructure [12]. Compute Engine - IaaS providing virtual machines, App Engine - PaaS for application hosting. Bigtable - IaaS massively scalable NoSQL database. BigQuery - SaaS large scale database analutics.

3.6 Salesforce.com

Salesforce is as of August 2017 the biggest (in revenue) cloud service provider on the market. All though most of its revenue comes from customer relationship management (CRM) products.

Brings together all you customer information in a single, integrated platform that enables you to build a customer-centered business from marketing right through sales, customer service and business analysis.

IoT Cloud

Salesforce Platform

3.7 Oracle

Lowest cost and most automated, as well as the industry's broadest and most integrated cloud, with deployment options raging from the public cloud to your own data centers.

3.8 Rackspace

Reviewing needs and helps build a strong business case. Assessing readiness to move to cloud. Design a reliable, scalable, secure and cost-efficient cloud architecture. Migrate data and apps to the right clouds. Cloud always managed by support. Ongoing enhancement and optimization for cloud.

Chapter 4

Implementation of Sensor Network

In order to upload sensor data to the cloud, a Raspberry Pi is used as a interface to the cloud. The Raspberry Pi receives data from the sensor nodes via Bluetooth Low Energy (BLE). Since the sensor nodes used in this implementation are not capable of connecting to the Internet, the Raspberry Pi is used as an interface to the cloud. BLE is used as it enables bi-directional communication. More about this specific implementation can be found in the appendix. This chapter discuses a general way to implement a sensor network with BLE as the communication protocol.

4.1 Bluetooth Low Energy communication

BLE has a range of about 100 meters, and periodically send small data packets. It is typically used for applications like monitoring sensors and remote control, among others [13]. This makes it a very suitable protocol for the application to be implemented in this project.

Following are some of the key characteristics for BLE, as described by Aftab [14].

Attribute Protocol (ATT) determines how data is transferred, by defining how devices are discovered as well as reading and writing attributes. Generic Attribute Profile (GATT) is the layer on top of ATT. It defines the role of the device, whether it is a client or a server. A GATT client, usually the central device, requests ATT data from the GATT server, usually one or more peripherals. The GATT server transfers attributes back to the GATT client. Attributes are defined by a Universally Unique Identifier (UUID), a 128-bit ID. A profile is used to group the attributes together, and consists of services and characteristics. A service consists of a collection of multiple characteristics. In this case, the service is Environment Sensing, and the characteristics are sensor measurements like Temperature and Humidity. The characteristics consists of a value and one or more descriptors. I.E. a temperature characteristic could have 24 as the temperature value and Celsius as a descriptor.

Generic Access Profile (GAP) defines roles as Broadcaster, Observer, Peripheral, and Central used in BLE. A broadcaster sends advertising packets to whoever is listening. Broadcasting packets are generally used to establish a connection between devices. The observers role is to listen to the broadcasting packets. In a master-slave architecture, the central device is the master and the peripheral device is the slave. The central is responsible for making the connections, and can connect to multiple peripherals at once. The peripheral broadcasts advertisement packets for the central to establish connection. The implementation of BLE defines that a device can only have the role as either a peripheral or a central at any given time.

Chapter 5

Setting up Cloud Environment

This chapter describes how the Amazon Web Services (AWS) environment was set up and which resources in AWS were utilized. Python is used through out the implementation to provide consistency. The implementation is based on the AWS Documentation [15].

5.1 DynamoDB

As previously mentioned, DynamoDB is a NoSQL database. In order to access DynamoDB programmatically, an access key must be configured. The key consists of an access key ID and a secret access key. The key is used to sign programmatic requests that are made to AWS. The credentials can be configured by running the command *aws configure* and entering the AWS access key. There are three ways of accessing DynamoDB, either through the console, the command line interface or the API. In the API, application code can be written by using the AWS Software Development Kit (SDK). This project uses the AWS SDK, with Python as the programming language. The AWS SDK for Python is Boto, it allows developers to make use of Amazon services when writing software.

When accessing a Amazon service through the API, the service and the region name must be specified. I.E. *dynamodb = boto3.resource('dynamodb', region_name='eu-central-1')*. Through the *dynamodb* object, items can now be created or queried in the database. In order to distinguish items from each other, each item has one or more key. The primary key is required, and the additional sort key is optional. When only a partition key is used, there can only be one item for each partition key value. If there are multiple items with the same partition key, the sort key can be used to distinguish the items.

In this case, each item consists of the device ID, the time stamp of when the item was created and some sensor data. The device ID is the primary key and the time stamp is the sort key. This makes it easy to query the database for data from a specific device, and these items are sorted by when they were created. An item can be queried by using key condition expression. For instance, all the information from one device can be obtained by evaluating the device ID in a key condition expression.

5.2 Web application with AWS Elastic Beanstalk

AWS Elastic Beanstalk provides the ability to deploy and manages applications without needing to set up the infrastructure which the applications run on. The AWS Elastic Beanstalk service handles things like scaling, capacity provisioning and load balancing automatically. Elastic Beanstalk also requires AWS credentials, like DynamoDB.

When an Elastic Beanstalk application is created, several resources are enabled to create the environment that runs the application. Among the resources is the launch of an Amazon Elastic Compute Cloud (EC2) instance, a virtual machine which runs the web application on the platform of choice. An Amazon Simple Storage Service (S3) bucket is also create to store the source code and other application files. Load balancing increases availability and scalability of the application Elastic Beanstalk also takes care of monitoring the load and automatically scales underlying resources as needed. Boto is also used as the AWS SDK for Python for AWS Elastic Beanstalk applications.

The AWS Elastic Beanstalk is managed and deployed through a virtual environment running on the Raspberry Pi. The virtual environment is also used to create the application environment. In order for Elastic Beanstalk to correctly replicate the environment, all the packages with the correct versions that run inside the virtual environment are stored in a file called *requirements.txt*. The virtual environment help distinguish which package are needed to run the application. The Python application run behind a Apache proxy server with WSGI.

This implementation is created using Django, a high-level Python Web framework which makes it easier to create a web application[16]. The Django application is created inside the virtual environment. In order to deploy the Django application to the Elastic Beanstalk environment, it is only needed to run the command *eb deploy* from within the virtual environment. Django enables an easy framework for displaying data with charts from different chart providers through *Graphos* [17].

Elastic Beanstalk is not actually needed to deploy the application. It is also possible to do it simply with a EC2 instance as a server. However, this requires that everything needed to run the application is set up manually, like server instances and installing required software. As mentioned, Elastic Beanstalk takes care of this and additional things like load balancing and automatically scaling resources. This is arguably not needed for such a small application as the one created here, however it could be useful for possible future development AWS Elastic Beanstalk is a PaaS, where the goal is to separate the application from the platform. AWS EC2 on the other hand is more like a IaaS which enables broader configuration possibilities

5.3 AWS IoT

AWS iot intentionally - timing issues Python - AWSIoTPythonSDK? Mqtt?

Chapter 6

Remote Control of Sensor Network

This chapter explains a general way of implementing remote control from the web application. The specific implementation of the communication between the BLE devices for this project can be found in the appendix.

Remote control of a network will typically be used to set parameters in the sensor network e.g. sampling time of data. Another area of use could be activation of certain features based on data from the network, e.g. turning on a heater based on readings from a temperature sensor.

For this project, lights on the peripheral devices will be turned on or off based on the users input from the web application. A demonstration of this can be seen at [...].

6.1 Web application interface

The web application displays two buttons for each device, one for turning on the lights and one for turning them off, as can be seen in figure 6.1. When a button is pressed, an associated JavaScript function is executed. The JavaScript performs an asynchronous HTTP (Ajax) POST request, using jQuery. The POST requests indicates that data is to be submitted to and processed by a specific source. In this case the request consists of the device ID and whether the request is to turn on or off the lights.

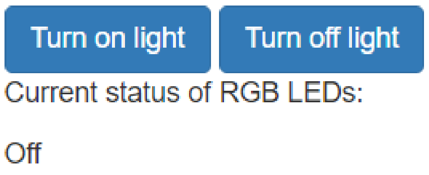

Figure 6.1: Interface for remote control of sensor network.

6.2 DynamoDB

A second DynamoDB table was created to keep track of the status of the RGB LEDs for each of the nodes. This table uses only the device ID as the key. Every time an item is updated, the current value will be overwritten. This means that there will only be one query for each device at any time. This makes for a good way to keep data consistency throughout the system. The value to be posted is given by the POST request from the JavaScript function.

The central device in the sensor network will continuously query the database for the status of each of the sensor nodes and updates them accordingly.

Chapter 7

Results

Why IoT not applicable, due to central device -RPi easier to upload directly to Db.

s Latency

Chapter 8

Discussion

Discuss results in relation to use of smart grids

8.1 Future work

Security

Future work - create users who can only access one device

Scalability?

Latency - i resultatseksjon Maskin-maskin grensesnitt

Sampling frequency for remote control

price?

Chapter 9

Conclusion

Bibliography

[1] *What is the Smart Grid.* URL: https://www.smartgrid.gov/the_smart_grid/smart_grid.html.

[2] T. Dillon; C. Wu; E. Chang. "Cloud Computing: Issues and Challenges". In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications.* 2010, pp. 27–33. DOI: 10.1109/AINA.2010.187.

[3] S. Bhardwaj; L. Jain; S. Jain. "Cloud Computing: A Study of Infrastructure As A Service (IAAS)". In: (2010), pp. 60–63. ISSN: 0976-0253. URL: http://www.academia.edu/1181740/Cloud_computing_A_study_of_infrastructure_as_a_service_IAAS_.

[4] P. Mell; T. Grace. "The NIST Definition of Cloud Computing". In: (2011). URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[5] Goyal; Sumit. "Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review". English. In: *International Journal of Computer Network and Information Security* 6.3 (2014), pp. 20–29. URL: https://search.proquest.com/docview/1624016976?accountid=12870.

[6] *IBM Cloud Computing.* URL: https://www.ibm.com/cloud-computing.

[7] S. Ramgovind; M. M. Eloff; E. Smith. "The management of security in Cloud computing". In: *2010 Information Security for South Africa.* 2010, pp. 1–7. DOI: 10.1109/ISSA.2010.5588290.

[8] *Microsoft Azure.* URL: https://azure.microsoft.com.

[9] *10 Biggest Cloud Computing Companies in the World.* URL: http://www.insidermonkey.com/blog/10-biggest-cloud-computing-companies-in-the-world-518585/?singlepage=1.

[10] *The World's Most-Powerful Cloud-Computing Vendors.* URL: https://www.forbes.com/sites/bobevans1/2017/06/05/meet-the-cloud-wars-top-10-the-worlds-most-powerful-cloud-computing-vendors/#625d34534e1e.

[11] *Amazon Web Services.* URL: https://aws.amazon.com.

[12] *Private Cloud Comparison.* URL: predictiveanalyticstoday.com/top-private-cloud-computing-solutions/.

[13] *Things you should know about Bluetooth Range.* URL: https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range.

[14] Muhammad Aftab. *Building Bluetooth Low Energy Systems.* Packt Publishing, 2017. ISBN: 978-1-78646-183-4.

[15] *AWS Documentation.* URL: http://docs.aws.amazon.com.

[16] *Django makes it easier to build better Web apps more quickly and with less code.* URL: https://www.djangoproject.com/.

[17] *Graphos.* URL: https://github.com/agiliq/django-graphos.

[18] *Using Bluetooth LE with Python.* URL: https://www.elinux.org/RPi_Bluetooth_LE#Using_Bluetooth_LE_with_Python.

[19] *Thunderboard Sense User's Guide.* URL: https://www.silabs.com/documents/public/user-guides/ug250-tb001-user-guide.pdf.

[20] *Thunderboard Sense with Raspberry Pi and Python.* URL: https://www.silabs.com/community/projects.entry.html/2017/03/08/thunderboard_sensew-Scqr.

Appendices

Appendix A

Sensor Network

A Raspberry Pi is used as the interface for all communication with the AWS cloud. It is connected to 3 Thunderboard Sense via BLE. The Raspberry Pi uses Bluepy, a python implementation of BlueZ, which is BLE interface for Linux based devices [18]. Bluepy contains an easy interface for scanning for and connecting to peripheral devices, as well as reading GATT data.

A Thunderboard Sense is a multi-sensor and multi-protocol development board from Silicon Labs, based on the EFR32 Mighty Gecko Wireless System-on-chip [19]. It contains 8 sensors, including temperature, relative humidity, pressure, and UV index. The Thunderboard Sense also consists of 4 RGB LEDs. It is compatible with BLE. Each device has a unique ID, which is utilized in this project.

The implementation of the reading of sensor data is based on this tutorial from Silicon Labs [20]. Each of the Thunderboard Sense have a GATT profile with Environment Sensing as a service. The service includes characteristics for all the sensors implemented on the board. For this project only the temperature, relative humidity, sound, and ambient light sensors are utilized. The Thunderboard Sense is the peripheral device and starts broadcasting its presence. The Raspberry Pi, the central device, initiates a connection when it observes the broadcast message. When the Raspberry Pi scans for services, it discovers the Environment Sense service, and can iterate through it to find the characteristics. The Raspberry Pi requests attribute data for a given characteristic from the Thunderboard Sense, which in turn transmits its GATT profile back to the Raspberry Pi.

The remote control of the sensor network is realized by the central writing characteristics and sending it to the peripheral. As a way to demonstrate functionality, the characteristic that is set is the RGB LEDs. One of two different values is set, on or off, depending on the user input from the web application. The Raspberry Pi reads the status of the RGB LED for each device from the LED Status database and writes the characteristic accordingly. The RGB LED characteristic consists of 4 bytes, where the first byte determines which of the 4 RGB LEDs should be active, and the remaining byte determines the red, green and blue color intensity, respectively.

Appendix B

Code

Following is the source code created in this project.

tbsense.py

```
from bluepy.btle import *
import struct
from time import sleep

class Thunderboard:

   def __init__(self, dev):
      self.dev = dev
      self.char = dict()
      self.name = ''
      self.session = ''

      # Get device name and characteristics
      scanData = dev.getScanData()

      for (adtype, desc, value) in scanData:
         if (desc == 'Complete␣Local␣Name'):
            self.name = value
```

```
        ble_service = Peripheral()
        ble_service.connect(dev.addr, dev.addrType)
        characteristics = ble_service.getCharacteristics()

        for k in characteristics:
            if k.uuid == '2a6e':
                self.char['temperature'] = k

            elif k.uuid == '2a6f':
                self.char['humidity'] = k

            elif k.uuid == 'c8546913-bfd9-45eb-8dde-9f8754f4a32e':
                self.char['ambientLight'] = k

            elif k.uuid == 'c8546913-bf02-45eb-8dde-9f8754f4a32e':
                self.char['sound'] = k
                print(k.read())

            elif k.uuid == 'fcb89c40-c603-59f3-7dc3-5ece444a401b':
                self.char['led'] = k

    #Helper functions to convert GATT characteristic values to
      ↪ numbers

    def readTemperature(self):
        value = self.char['temperature'].read()
        value = struct.unpack('<H', value)
        value = value[0] / 100
        return value

    def readHumidity(self):
        value = self.char['humidity'].read()
        value = struct.unpack('<H', value)
        value = value[0] / 100
        return value

    def readAmbientLight(self):
        value = self.char['ambientLight'].read()
      value = struct.unpack('<L', value)
      value = value[0] / 100
      return value

   def readSound(self):
      value = self.char['sound'].read()
      value = struct.unpack('<h', value)
      value = value[0] / 100
      return value

   #RGB LEDs have 4 bytes to write. First byte determines which
       ↪ LEDs to turn on, next three bytes are colors for red,
       ↪ green and blue, respectively

   def LedOn(self):
      self.char['led'].write('\x0F\x00\x00\x70', withResponse=True
          ↪ )


   def LedOff(self):
      self.char['led'].write('\x00\x00\x00\x00', withResponse=True
          ↪ )
```

tbsense_scan.py

```
from bluepy.btle import *
import struct
from tbsense import Thunderboard
import threading
from post_db import post
from get_leds import read_db
from datetime import datetime
from time import sleep



def getThunderboards():
    scanner = Scanner(0)
    devices = scanner.scan(3)
    tbsense = dict()
    for dev in devices:
        scanData = dev.getScanData()
        for (adtype, desc, value) in scanData:
            if desc == 'Complete␣Local␣Name':
                if 'Thunder␣Sense␣#' in value:
                    deviceId = int(value.split('#')[-1])
                    tbsense[deviceId] = Thunderboard(dev)

    return tbsense

def sensorLoop(tb, devId):

    while True:
        data = dict()

        try:
            for key in tb.char.keys():

                if key == 'temperature':
                        data['temperature'] = tb.readTemperature()

                elif key == 'humidity':
                    data['humidity'] = tb.readHumidity()

                elif key == 'ambientLight':
                    data['ambientLight'] = tb.readAmbientLight()

                elif key == 'sound':
                    data['sound'] = tb.readSound()

        except:
            return

        post(devId, data) #Posts sensor data to database
        read_db(tb, devId) #Read Led data from database
        sleep(1)


def dataLoop( thunderboards):
    threads = []
    for devId, tb in thunderboards.items():
        t = threading.Thread(target=sensorLoop, args=(tb, devId))
        threads.append(t)
        print('Starting␣thread␣{}␣for␣{}'.format(t, devId))
        t.start()


if __name__ == '__main__':

    while True:
        thunderboards = getThunderboards()

        if len(thunderboards) == 0:
            pass
        else:
            dataLoop(thunderboards)
```

post_db.py

```
from __future__ import print_function # Python 2/3 compatibility
import boto3
import json
import decimal
from boto3.dynamodb.conditions import Key, Attr
from datetime import datetime

tstamp = datetime.now()

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)
```

```
def post(devId, data):
    dynamodb = boto3.resource('dynamodb', region_name='eu-central
        ↪ -1')

    table = dynamodb.Table('Sensors')
    timestamp = datetime.now()

    #Posts sensor data to database once every hour
    if ((timestamp - tstamp).total_seconds() > 3600):
        response = table.put_item(
            Item={
                'Device': str(devId),
                'Time': str(timestamp),
                'Sound': data['sound'],
                'Temperature': data['temperature'],
                'Ambient␣Light': data['ambientLight'],
                'Humidity': data['humidity'],
            }
        )
        global tstamp
        tstamp = datetime.now()
```

get_leds.py

```
from __future__ import print_function # Python 2/3 compatibility
import boto3
import json
import decimal
from boto3.dynamodb.conditions import Key, Attr
import tbsense

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)

def read_db(tb, devId):

    dynamodb = boto3.resource('dynamodb', region_name='eu-central
        ↪ -1')

    table = dynamodb.Table('LedStatus')

    response = table.query(
        KeyConditionExpression=Key('Device').eq('%d' % devId)
    )

    for i in response['Items']:
        if( i['Status'] == 'On'):
            tb.LedOn()
        elif(i['Status'] == 'Off'):
            tb.LedOff()
```

views.py

```
from __future__ import print_function # Python 2/3 compatibility
from django.http import HttpResponse
import boto3
import json
import decimal
from datetime import datetime
from rgbled_db import post
from boto3.dynamodb.conditions import Key, Attr
from django.shortcuts import render
from django.template import loader

from graphos.renderers import gchart
from graphos.sources.simple import SimpleDataSource
from django.views.decorators.csrf import csrf_exempt

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)

dynamodb = boto3.resource('dynamodb', region_name='eu-central-1')

table = dynamodb.Table('Sensors')
table2 = dynamodb.Table('LedStatus')

@csrf_exempt
def index(request):

    #Post RGB LED status if button pressed
    if (request.is_ajax()):
       post(request.POST['device'], request.POST['status'])


    name = ['44804', '45452', '45311']

    context = {}

    #Read sensor data from database and create graphic view of
        ↪ data
    for dev in name:
        response = table.query(
            KeyConditionExpression=Key('Device').eq(dev)
        )
        data = [[]for i in range(int(response['Count'])+1)]

        data[0] = ['Time', 'Temperature␣[C]', 'Humidity␣[RH%]', '
            ↪ Ambien␣Light␣[Lux]', 'Sound␣[dB]']
        n = 1

        for i in response['Items']:
            info = [datetime.strptime(i['Time'],'%Y-%m-%d␣%H:%M:%S
                ↪ .%f'), int(i['Temperature']), int(i['Humidity'])
            ↪ , int(i['Ambient␣Light']), int(i['Sound'])]
            data[n] = info
            n += 1

        context[dev] = gchart.LineChart(SimpleDataSource(data=data
            ↪ ), options={'title':'Device␣ID␣'+dev})
        response2 = table2.query(
            KeyConditionExpression=Key('Device').eq(dev)
        )
        for j in response2['Items']:
            context['btn'+dev] = j['Status']


    return render(request, 'index.html', context)
```

rgbled_db.py

```
from __future__ import print_function # Python 2/3 compatibility
import boto3
import json
import decimal

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)

#Post status of RGB LED to database
def post(device, status):

  dynamodb = boto3.resource('dynamodb', region_name='eu-central-1
      ↪ ')

  table = dynamodb.Table('LedStatus')
```

```
response = table.update_item(
   Item={
        'Device': device,
        'Status': status,

    }
)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Prosjektoppgave</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,␣initial-
      ↪ scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
      ↪ bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery
      ↪ /3.2.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js
      ↪ /bootstrap.min.js"></script>
  <script type="text/javascript" src="https://www.google.com/
      ↪ jsapi"></script>
  <script type="text/javascript">
    google.load("visualization", "1", {packages:["corechart"]});
  </script>

  <style>
    .grid-container {
      display: grid;
      grid-template-columns: auto auto;
    }

  </style>
</head>
<body>
```

```
<h1 align="center">Sensor data from each device</h1>

<div class="grid-container">
  <div id="div1">
    {{44804.as_html}}
  </div>
  <div id='div2'>
    <h2> Refresh page after a button is pressed </h2>
    <button id="on1" type="button" class="btn␣btn-primary␣btn-md">
        ↪ Turn on light</button>
    <button id="off1" type="button" class="btn␣btn-primary␣btn-md"
        ↪ >Turn off light</button>

    <p> Current status of RGB LEDs: </p>

    {{btn44804}}

  </div>
  <div id="div3">
    {{45452.as_html}}
  </div>
  <div id='div4'>
    <button id="on2" type="button" class="btn␣btn-primary␣btn-md">
        ↪ Turn on light</button>
    <button id="off2" type="button" class="btn␣btn-primary␣btn-md"
        ↪ >Turn off light</button>

    <p> Current status of RGB LEDs: </p>

    {{btn45452}}

  </div>
  <div id='div5'>
    {{45311.as_html}}
  </div>
  <div id='div6'>
    <button id="on3" type="button" class="btn␣btn-primary␣btn-md">
        ↪ Turn on light</button>
    <button id="off3" type="button" class="btn␣btn-primary␣btn-md"
```

```
        ↪ >Turn off light</button>

    <p> Current status of RGB LEDs: </p>
    {{btn45311}}

 </div>
</div>
</body>
</html>
<script type='text/javascript' src='http://code.jquery.com/jquery
    ↪ -1.8.2.js'></script>
<script type="text/javascript">

$(document).ready(function(){
   $("#on1").click(function() {
       $.ajax({
           type:"POST",
           url:"",
           data: {device: '44804', status: 'On'},
       });
   });

   $("#off1").click(function() {
       $.ajax({
           type:"POST",
           url:"",
           data: {device: '44804', status: 'Off'},
       });
   });

   $("#on2").click(function() {
       $.ajax({
           type:"POST",
           url:"",
           data: {device: '45452', status: 'On'},
       });
   });

   $("#off2").click(function() {
        $.ajax({
            type:"POST",
            url:"",
            data: {device: '45452', status: 'Off'},
        });
    });

    $("#on3").click(function() {
        $.ajax({
            type:"POST",
            url:"",
            data: {device: '45311', status: 'On'},
        });
    });

    $("#off3").click(function() {
        $.ajax({
            type:"POST",
            url:"",
            data: {device: '45311', status: 'Off'},
        });
    });
});
</script>
```

Appendix C

Video demonstration

A demonstration of the remote control of the RGB LEDs can be found on ...