FRAMEWORK FOR INTELLIGENT HOMES

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Prachi Utkhede

FALL
2017

FRAMEWORK FOR INTELLIGENT HOMES

A Project

by

Prachi Utkhede

Approved by:

_____, Committee Chair
Dr. V. Scott Gordon

_____, Second Reader
Dr. William Mitchell

_____
Date

Student: Prachi Utkhede

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

_____, Graduate Coordinator _____
Dr. Jinsong Ouyang Date

Department of Computer Science

Abstract

of

FRAMEWORK FOR INTELLIGENT HOMES

by

Prachi Utkhede

The Internet of Things refers to the connection of everyday objects to the Internet. This allows humans to monitor and interact with these objects from anywhere in the world.

With IoT, we can envision a future where everyday objects such as toasters, coffee makers, vacuum cleaners would be uniquely identifiable and controlled over the internet. With so many wide variety of devices connected to the Internet there is a need for a scalable, fault tolerant, distributed processing framework that can fetch, process, store, display, and analyzes such large amount of data generated by these devices.

For this project, a system was built which can receive data from multiple devices and provide an unified framework to process, store, and display data from these devices in real time. The framework henceforth means collection of environment which we have built. To demonstrate the capabilities of this framework, an Intelligent Home system was simulated which monitors room temperature using a temperature sensor, connected via a Raspberry Pi. The Raspberry Pi acts as a gateway to fetch data from the sensors, which is processed by the framework in real time to show the room temperature trends on the UI. The framework was built using Apache Spark, a fast, distributed, scalable and general engine for large-scale data processing and Apache Cassandra, which is a free and open-

source distributed NoSQL database management system designed to handle large amounts of data. Both Apache Spark and Cassandra are an ideal fit for processing large amounts of data since they provide excellent horizontal scalability, i.e. they can process every increasing amounts of data by adding more hardware capacity without the need to change any framework code.

_____, Committee Chair
Dr. V. Scott Gordon

_____
Date

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Scott Gordon for guiding me throughout this Project. This project would not appear in its present form without his kind assistance and support. I owe a deep debt of gratitude to my advisor for his belief in my abilities, his enthusiasm for this project, and his continuous insistence that I make it better. He provided encouragement, sound advice, good teaching, and lots of good ideas.

I would also like to thank Department of Computer Science at California State University, Sacramento for giving me an opportunity to pursue Masters in Computer Science and guiding me to become a successful student.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figures Page

Chapter 1

INTRODUCTION

1.1 Overview of IOT

The word Internet Of Things known as IOT is coined from two words- “Internet” which is a web of networks and “Things” which are physical objects connected to networks [1]. Internet Of Things(IoT) is a network of interrelated physical objects or devices that transfers data over the wireless network. In theory, interrelated physical objects, also called “Smart” objects, can be virtually anything such as devices like temperature sensor, coffee machine, television, fridge etc., and can be connected to each other with an on/off switch over the Internet. These “Smart” objects can be very helpful in day to-day life; for example, in “Smart Homes” which can help us to reduce electricity consumption and thus can be more cost-efficient. Another application of IOT would be if a person wants to track his routine and receive updates on his mobile phone, he can easily monitor it via getting a text or an email every day according to his needs. There is unlimited scope for IOT with potential for making our life more simpler and comfortable.

1.2 Current Solutions

The current Intelligent Homes market stands at 39.93 Billion in 2016 and is expected to reach USD 79.57 Billion by 2022 [2]. The most popular include Samsung SmartThings, Wink Hub 2, Abode Home Security System. Some of these smart hubs support only a small number of devices made by the manufacturer of the hub, while others offer certification programs for third party device manufacturers to make their devices compatible with their solutions. Even with an ever increasing range of products

to choose from, the market is very fragmented and devices from one manufacturer might not be compatible with the monitoring and control solution of another. This device incompatibility makes it very costly to replace one solution with another.

1.3 Project Statement

In this project, we use Raspberry Pi as an easy to use gateway for integrating various type of devices and build a highly scalable platform, using Apache Spark, that monitors and controls these devices. The gateway and the framework communicate via a message queue which shields these components from failure. Also, the framework is encapsulated from the various types of devices by the gateway by converting all the sensor readings into a common message format before being processed. Common message format is a unified way of representing any reading from any type of device. So the temperature senor or humidity sensor or carbon monoxide sensor or any type of sensor can send the readings in the same format. This makes framework extensible as any new sensor can be connected to the system as long as it sends the reading in common message format.

To demonstrate the capability of this platform we have connected a temperature sensor to our platform using a Raspberry Pi and the readings are displayed on a web interface. A similar approach can be followed to integrate other types of devices to the system. I have used RabbitMQ as a message queue to store temperature data from Raspberry Pi before it is processed by Spark and persisted in Apache Cassandra. I have used Spring Boot to build a RESTful web service API that fetches temperature data for a given time period from Cassandra which is then plotted on the UI using the HighCharts JavaScript library.

Chapter 2

BACKGROUND

The project uses a variety of hardware components such as temperature sensors, Raspberry Pi, breadboards, resistors and software components such as Apache Spark, Apache Cassandra, RabbitMQ, Spring Boot, and HighCharts. The details of each of these components is listed below:

2.1 Raspberry Pi 3 Model B

The Raspberry Pi is a low cost, small, single-board computer developed in the United Kingdom by the Raspberry Pi Foundation [3].

In this project the Pi is used as a bridge to read data from variety of sensors and convert those readings into a common message format to be transmitted to the platform to monitor and control various devices. We are using the Raspberry Pi 3 Model B which is a small yet very powerful device with 1.2GHz 64-bit quad-core ARMv8 CPU, 1GB RAM, 802.11n Wireless LAN, 4 USB ports, 40 GPIO pins and Full HDMI port. The 40 GPIO (General Purpose Input Output) pins makes the Raspberry Pi an ideal choice of an cheap, easy to use, and extensible gateway to add and monitor variety of devices.

2.2 Sensors

Sensors are devices that transmit detected physical data such as temperature, pressure, light and heat from the environment to some other device for further processing. Sensors are an integral part of this Intelligent Homes project since they constantly monitor various parameters of a home such as temperature and moisture, and then transmit any changes to the platform.

In this project, we have integrated a DS18b20 sensor, which is a high-quality temperature sensor with an accuracy ±0.5°C over the range of -10°C to +85°C, to monitor room temperature. The DS18b20 sensor comes with Red, Black, and Yellow wires for power, ground, and data respectively. These wires are connected to the corresponding GPIO pins of the Raspberry Pi to transmit the temperature readings.

2.3 RabbitMQ

RabbitMQ is a message broker or queue manager [4]. In simple words, RabbitMQ is a simple queue in which applications/systems can be connected to both side of queue in order to transfer a message. The sending application is called a producer and the receiving application is called a consumer. Introducing a message queue makes the system adapt to changes since new producers and consumers can transparently join without impacting the existing components.

In this project, use of a message broker enables us to process data from multiple Raspberry Pi's (producer) or in general any device which can send data to the queue over http. At the same time, we can add more hardware capacity to Apache Spark (consumer) to improve the message processing power of the system. Since RabbitMQ stores the message on a persistent storage until they are delivered, in case of either the producer or consumer failure, the messages will be safely buffered in the queue thus avoiding any data loss.

2.4 Apache Spark

Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing [5]. It provides high

level APIs to write applications in Scala, Java, or Python programming languages. **Spark streaming**, is an extension of spark API which provides scalable, fault-tolerant, high–throughput streaming for live data streams.

In this project, the Spark Streaming module continuously monitors the RabbitMQ to process any new temperature information which is then populated in Cassandra database.

2.5 Apache Cassandra

Cassandra is an Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable [6]. It is a distributed NoSQL database which provides scalability across commodity servers. It also provides high availability with no single point failure. It is fault tolerant i.e. even if there is a node failure the failed node is replaced quickly. It is also used for durable applications that cannot afford to lose data. Cassandra is excellent for storing data in sequential form despite of the datatype or size. Cassandra offers excellent read/write latency when the data is stored and retrieved sequentially. In this framework since the streaming data is ordered by time and retrieved in the same fashion, we get excellent read/write latency. A time series data is excellent fit for such scenario. It provides an meaningful way to access time series data to make decisions about future outcome.

In this project, we used Cassandra as a time series database for storing and accessing temperature data through RESTful web service and showing it through Web UI for a specific interval of time.

2.6 Web Services

Web Services are client-server applications that exchange data through collection of open protocol. Software applications communicate over Internet i.e. World Wide Web (WWW) using HyperText Transfer protocol (HTTP). There are two types of Web services :

- SOAP-based: Simple Object Access Protocol uses XML to make requests and get responses for providing messaging services.
- REST: REST is a set of constraints which includes client/server relationship, uniform interface and is stateless. Rest is associated with HTTP and it is used for designing distributed system.

In this project, we have used RESTful web service over SOAP because REST is easier to work, since it supports various data format. Also REST supports browser compatibility, less bandwidth and integrates easily with existing website.

2.6.1 Spring Boot

Spring Framework is an application framework that provides inversion of control and dependency injection for java platform [7]. Spring boot creates standalone, production grade spring based applications which are ready to run. Spring Boot provides an embedded Tomcat server, starter POMs for Maven configuration, and it doesn't require XML configuration.

In this project, we are building a RESTful Web services API using the Spring framework which will fetch the time series room temperature data from Apache Cassandra which will be displayed on the web UI.

2.7 Web UI

JavaScript, abbreviated as JS, is an event-driven , object-oriented interpreted programming language used along with HTML and CSS to make web page dynamic and interactive. HighChart JS is a JavaScript library included to make animated, interactive charts to make website more readable [8]. High Chart provides an easy way to add an interactive charts which takes room temperature data from RESTful web service and show it on the web UI.

In this Project, the temperature readings that are provided through Web Service are parsed and displayed in form of HighChart graphs.

Chapter 3

DESIGN AND IMPLEMENTATION

3.1 System Architecture

The Producer Consumer pattern is an ideal way of separating work that needs to be done from the execution of that work [9]. It contains two major components which are linked by queue. The producer generates the data, puts it into a queue and the consumer consumes the data for further processing.

We have modelled the framework as shown in figure 1 using the Producer Consumer design pattern where multiple sensors, connected via Raspberry Pi, are the message producers and the Spark Streaming application is the consumer with RabbitMQ as a blocking message queue. Further, we created a common message format which is a unified way of representing any reading. The framework can process data from any type of sensor as long as it can send the reading in the common message format to the message queue. Figure 1 represents a high level system architecture, where the Raspberry Pi continuously polls the temperature sensor and transmits this reading to the RabbitMQ. The Spark Streaming application continuously monitors the RabbitMQ and on receiving any new message, it process and stores it in Cassandra. When the user visits the UI, the HighCharts JavaScript framework retrieves the data stored in Cassandra using a RESTful web service API, which we built using Spring framework and displays it.

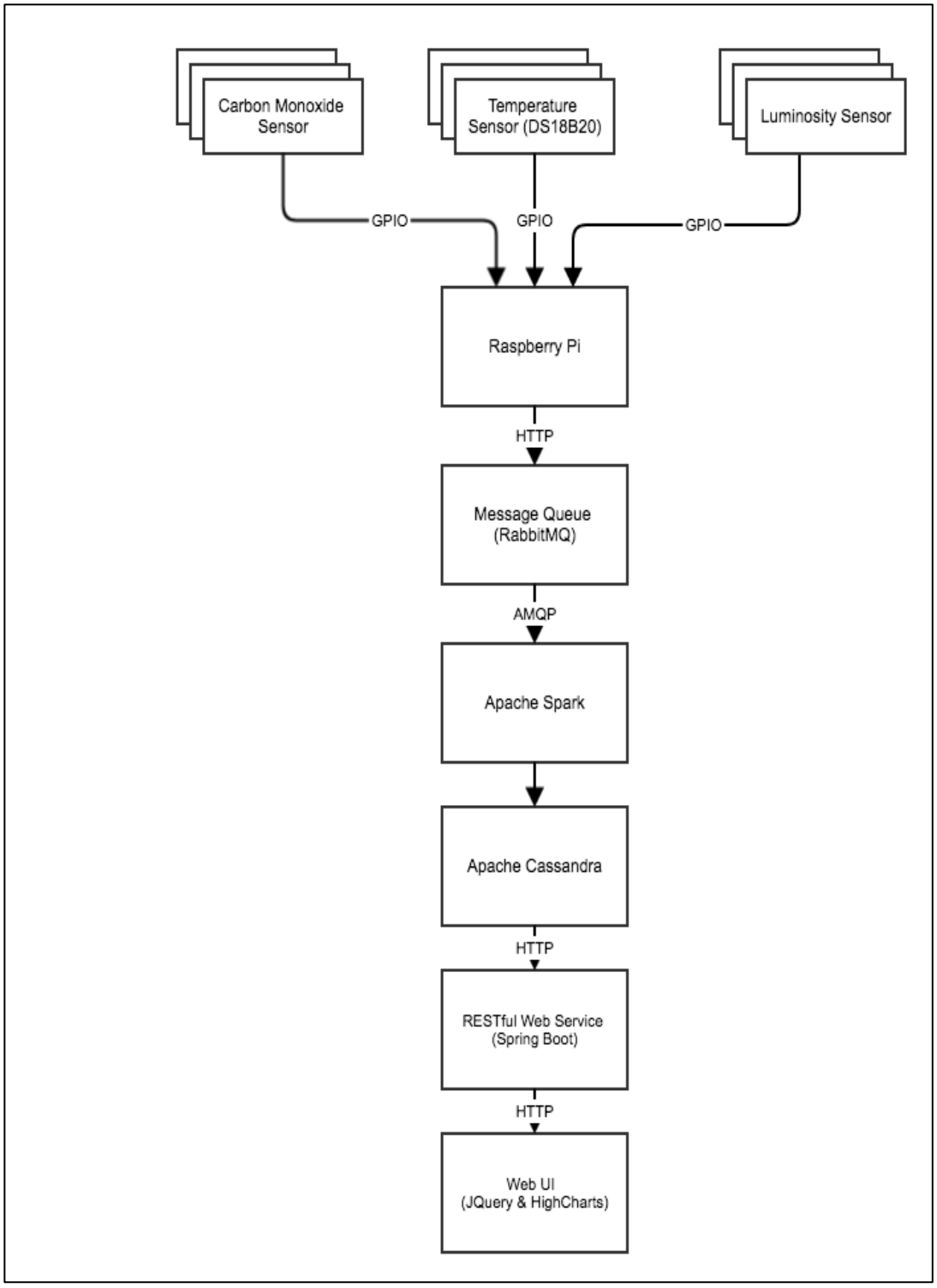

Figure 1 : System Architecture

3.2 Interfaces and Implementation

3.2.1 Temperature Sensor to Raspberry Pi

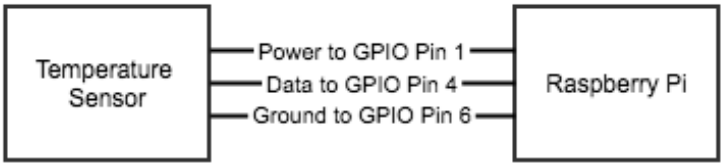

Figure 2 : Sensor to Raspberry Pi

Figure 2 shows temperature sensor connected to Raspberry Pi. Raspberry Pi has a the row of **GPIO** (**General Purpose Input Output**) pins which are used to connect external components to the Pi. This provides Pi with the flexibility to connect and read data from a variety of sensors. In this project we will connect the temperature sensor to the GPIO Pin 4 of the Pi and use a Python script to the read the data and post this data to the RabbitMQ. Following is the code to initialize the temperature sensor and reads the data:

```
# Setup the GPIO Pins
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

devices_dir = '/sys/bus/w1/devices/'
temperature_device_folder = glob.glob(base_dir + '28*')[0]
temperature_device_file = device_folder + '/w1_slave'

fd = open(temperature_device_file, 'r')
temperature = fd.readlines()
```

The framework can scale to any number of devices since we can connect multiple sensors to the Raspberry Pi and any number of Raspberry Pi's can be connected to the system as long as they can send the reading in common message format to the RabbitMQ.

3.2.2 Raspberry Pi to RabbitMQ

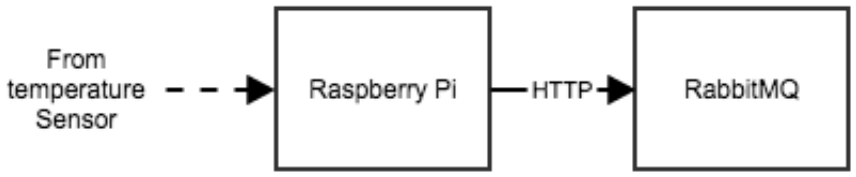

Figure 3 : Raspberry Pi to RabbitMQ

Figure 3 shows the Raspberry Pi reads the data from the temperature sensor and converts it into common message format. Following are the fields in the common message format which abstracts the type of the device and the way the data is fetched from the device:

```
home_id,
zone_id,
measure_name,
reading_timestamp,
measure_numeric_value,
measure_string_value
```

where *home_id*, is the unique id of the home where the device is installed, the *zone_id* specifies the zone in the home, for e.g. living room, game room, basement etc. The *measure_name* specifies the name of reading such as temperature, luminosity, moisture, or carbon monoxide levels. Some of the sensor readings are numeric (such as a temperature) and are stored in *measure_numeric_value* field. Some readings can be string values (such as light ON or OFF status) and are stored in *measure_string_value* field. The time at which the reading is taken is stored in *reading_timestamp* field. Here is a sample temperature reading message:

```
{
    "home_id": "HOME-1",
    "zone_id":"LIVING_ROOM",
    "measure_name":"temperature",
    "measure_numeric_value":75,
    "measure_string_value": "",
    "reading_timestamp": 1507513691894
}
```

This message is sent to the RabbitMQ over HTTP. Here is the code that POSTs the message to the queue:

```
import request

# POST the message to RabbitMQ
response = requests.post(self.publish_url, None,
json.loads(payload), auth=(self.username, self.password))

# Check if the POST was successful
if response.status_code != 200:
    print("Failed to publish message to rabbitmq. HTTP Error
code: " + str(response.status_code))
    print("Error reason: " + str(response.content))
print("Successfully published message to RabbitMQ. Response:
" + str(response.content))
return response
```

By using the Producer Consumer design pattern we have decoupled the Raspberry Pi from the framework, so the framework can process data from any number of Raspberry Pi's as long as they send the reading in common message format to the RabbitMQ.

3.2.3 RabbitMQ to Apache Spark

Figure 4 : RabbitMQ to Apache Spark

Figure 4 shows Spark Streaming module connects to the RabbitMQ and continuously monitor for new messages to be processed. Here is the code that connects and gets the new data from RabbitMQ and parses it:

```
JavaStreamingContext jssc = new
JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(properties.getProperty("spark.stream
ing.batch.interval"))));

JavaDStream<HomeMessage> messages =
rabbitMQConsumer.createStream(jssc, HomeMessage.class,
RabbitMQConfig.getConnectionParams(properties),
messageParser.getMessageParser()).filter(message -> message
!= null);
```

Apache Spark has excellent horizontal scalability, i.e. Spark can process ever increasing number of messages generated by multiple devices by adding more hardware capacity and without the need to make any framework code changes.

3.2.4 Spark to Cassandra

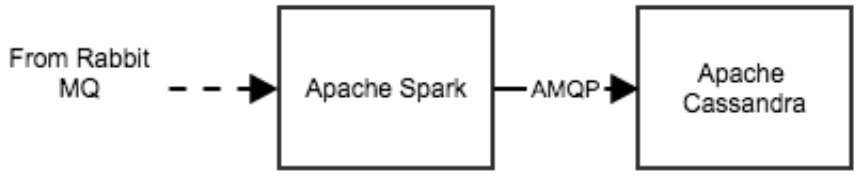

Figure 5 : Apache Spark to Cassandra

Figure 5 shows the sensor readings that are parsed by the framework are stored in Cassandra. Following is the Cassandra keyspace and table schema definition:

```
CREATE KEYSPACE intelligent_homes WITH replication =
{'class':'SimpleStrategy', 'replication_factor' : 1};
```

```
CREATE TABLE intelligent_homes.readings (
    home_id VARCHAR,
    zone_id VARCHAR,
    measure_name VARCHAR,
    reading_timestamp TIMESTAMP,
    measure_numeric_value DOUBLE,
    measure_string_value VARCHAR,
PRIMARY KEY (home_id, zone_id, measure_name,
reading_timestamp));
```

Following is the code snippet that persists the parsed sensor readings data into Cassandra:

```
messages.foreachRDD(rdd -> {
javaFunctions(rdd).writerBuilder(properties.getProperty("cas
sandra.keyspace"),
properties.getProperty("cassandra.table"),
mapToRow(HomeMessage.class, HomeMessage.PAIRS))
.saveToCassandra(); });
```

Apache Cassandra is a distributed NoSQL database which provides scalability across commodity servers. It also provides high availability with no single point failure. By using Cassandra, the framework can store huge amounts of data by adding more hardware capacity.

3.2.5 Cassandra to Restful Web Service

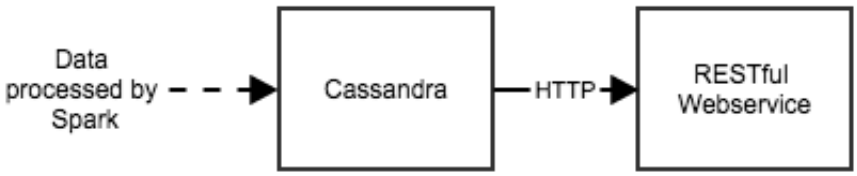

Figure 6 : Cassandra to Web Service

Figure 6 shows the RESTful API, built using Spring Boot, fetches the data from Cassandra using the Spring Cassandra Template. We construct the query that fetches the temperature reading and encode the results in a MeasuresReadingDTO class which is serialized to JSON before being sent to the UI.

```
public class CassandraDataFetcher {

   @Autowired
   private CassandraOperations cassandraOperations;

   @CrossOrigin
   @RequestMapping(method=GET, value =
"/readings/{homeId}/{zoneId}/{measureName}")
   public List<MeasureReadingsDTO> getTimeSeriesData(
                                    @PathVariable String
homeId,
                                    @PathVariable String
zoneId,
                                    @PathVariable String
measureName) {
       List<MeasureReadingsDTO> readings = new ArrayList<>();
```

```
       String query = String.format(READINGS_QUERY, homeId,
zoneId, measureName);

       cassandraOperations.select(query, Readings.class)
                          .stream()
                          .forEach(r -> readings.add(new
MeasureReadingsDTO(r.getPk().getReadingTimestamp(),
r.getMeasureNumericValue())));

       return readings;
   }
}
```

Spring Boot enabled us to build highly scalable web services which are horizontally scalable. These web services can scale to hundreds of request per seconds by adding more hardware.

3.2.6 Web Service to Web UI

Readings from the device using the Web Service is displayed on UI using the HighCharts Javascript Library.

```
<script type="text/javascript">
   function temperatureData (label,number) {
   $('#chartContainer').highcharts({
   chart: {
   type: 'line'
   },
   title: {
   text: 'Room Temperature'
   },
   xAxis: {
   categories : label,
   title: {
       text : 'Time'
   }
   },
   yAxis: {
   title: {
   text: 'Temperature'
   }
   },
   series: [{
   name: 'temperature',
   data: number
   }]
   });
   }

   $(document).ready(function() {
   $.ajax({url: 'http://localhost:8080/readings/HOME-
1/LIVING_ROOM/temperature',
   type: 'GET',
   async: true,
   dataType: "json",
   success: function (data) {
   var labels = [], number=[]
   for(var i = 0; i < data.length; i++) {
```

```
    labels.push( data[i]['readingTimestamp']);
    number.push(data[i].measureNumericValue);
    }
    temperatureData(labels,number);
    }
    });
    });

</script>
```

Figure 7, Figure 8 and Figure 9 are the charts representing the changes in room temperature (in degree Celsius) that were captured by the framework:

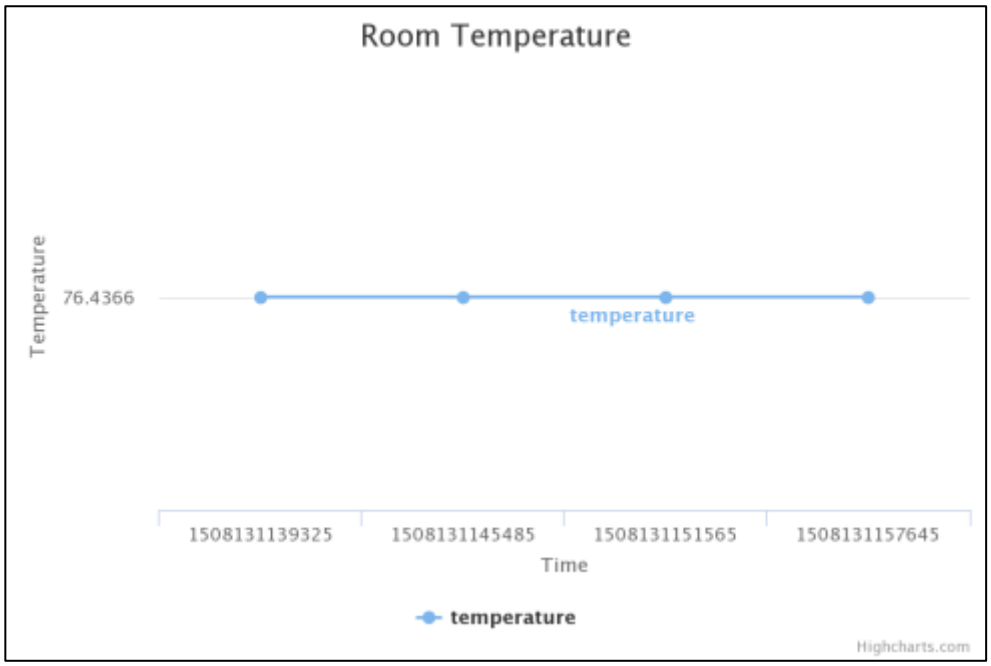

Figure 7: Constant Room Temperature

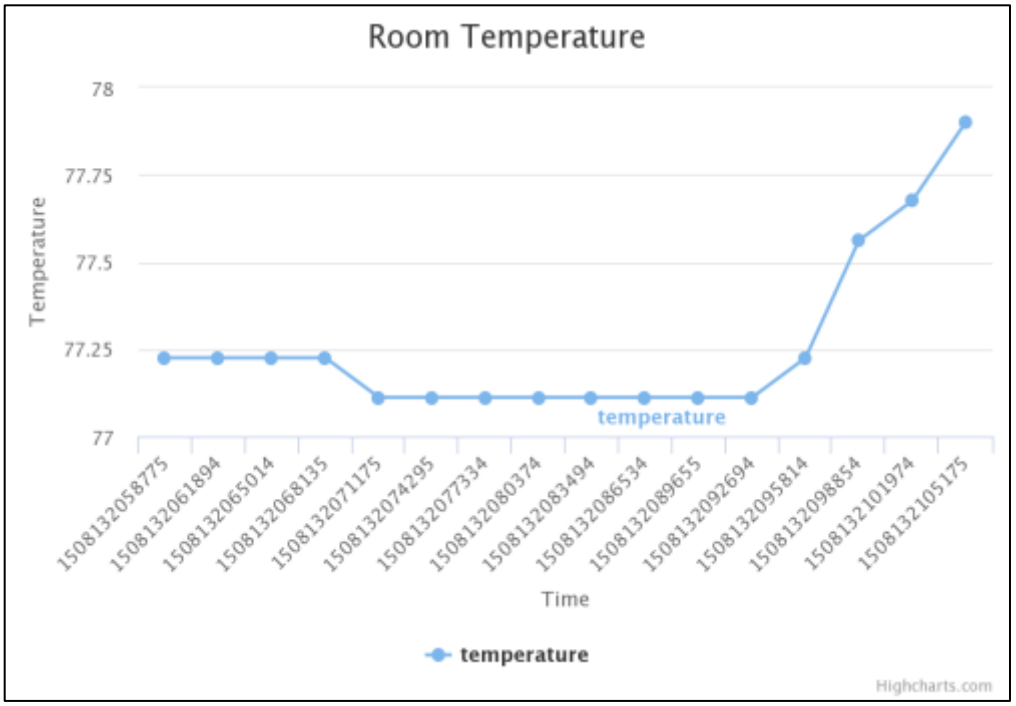

Figure 8: Increasing Room Temperature

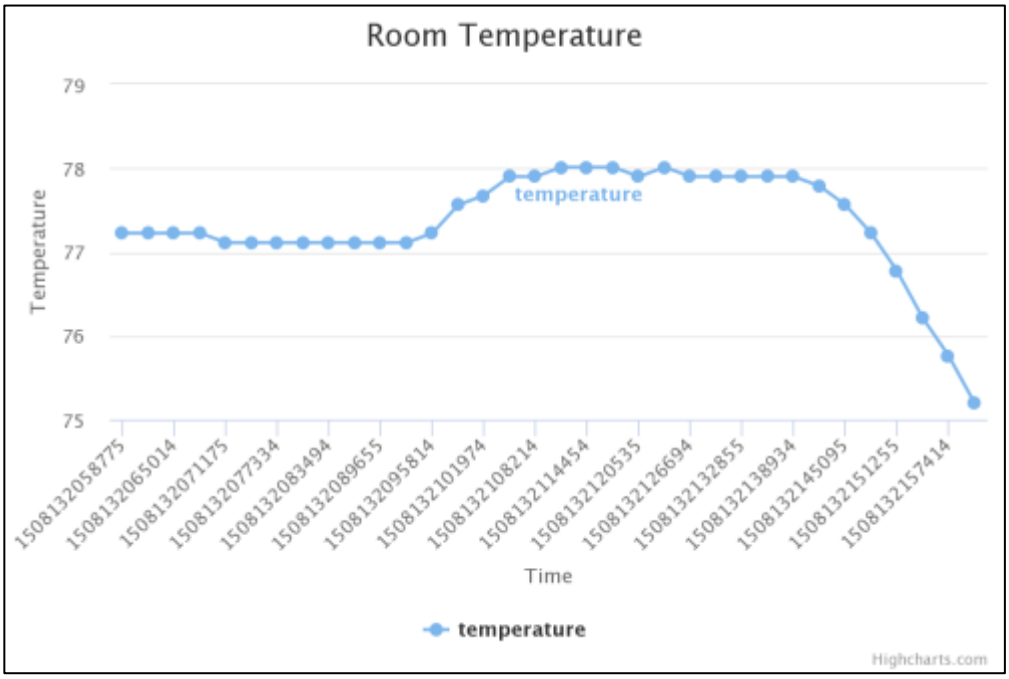

Figure 9: Decreasing Room Temperature

Chapter 4

CONCLUSION

In this project, we build a fast, scalable framework which helps us to monitor our environment in real time and view that information in web UI. Use of technologies such as Apache Spark and Cassandra makes it easy to scale the framework to monitor thousands of devices simultaneously. Also, by modelling the framework on Producer Consumer design pattern we have made it highly fault tolerant as any device failure doesn't impact the framework and in case of framework failure the messages are safely stored in the queue thus avoiding any data loss.

Further, this project helped me to learn about different technologies such as Apache Spark, Apache Cassandra, Spring Boot and HighCharts and how they can be used to build a real time applications.

Chapter 5

FUTURE WORK

The future work will include extending this framework for other Intelligent Environments such as transportation, energy, farming etc. For example, in the Transportation sector, a vehicle can be fitted with variety of sensors which can generate vehicle performance data. By analyzing data from these sensors, we can recognize any upcoming problems and perform preventive maintenance.

We can also enhance the framework to use Rules Engine such as Drools so that user can take actions based on sensor data. For e.g., we can create an IPhone / Android application where users can control the room temperature using their mobile devices.

The framework can be used to integrate different machine learning algorithms so the devices can be controlled to create a fully automated home which can adjust according to user's preferences.

APPENDIX

Source Code

Raspberry Pi Code:

```
# Import Libraries
import os
import glob
import time
import base64
import requests
import json

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

sys_devices_base_dir = '/sys/bus/w1/devices/'
temperature_sensor_base_dir = glob.glob(sys_devices_base_dir + '28*')[0]
temperature_readings_file = temperature_sensor_base_dir + '/w1_slave'


def read_room_temperature():
    f = open(temperature_readings_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def wait_for_device_to_be_ready(lines):
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.5)
        lines = read_room_temperature()
    return lines

def read_temp():
    lines = read_room_temperature()
    lines = wait_for_device_to_be_ready(lines)
```

```
    equals_pos = lines[1].find('t=')

    if equals_pos != -1:
        temp_string = lines[1][equals_pos + 2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f


class RabbitMQPublisher(object):
    def __init__(self, publish_url, username, password):
        self.publish_url = publish_url
        self.username = username
        self.password = password

    def publish(self, topic, base64_encode_msg):
            payload = '{"properties":{"delivery_mode":2}, "routing_key":"' + topic +
            '","payload":"' + base64_encode_msg.decode(
            'ascii') + '","payload_encoding":"string"}'
        print("Payload: " + payload)
             response = requests.post(self.publish_url, None, json.loads(payload),
            auth=(self.username, self.password))
        if response.status_code != 200:
             print("Failed to publish message to rabbitmq. HTTP Error code: " +
            str(response.status_code))
            print("Error reason: " + str(response.content))

        print("Successfully published message to RabbitMQ. Response: " +
        str(response.content))
        return response


class MessageGenerator(object):
    @staticmethod
            def generate_numeric_value_message(measure_name, measure_numeric_value,
                reading_timestamp):
        temperature_message = '{"home_id": "HOME-1", ' \
```

```
                '"zone_id": "LIVING_ROOM", ' \
                '"measure_name": "' + measure_name + '", ' \
                                    '"measure_numeric_value": ' + str(
    measure_numeric_value) + ', ' \
                    '"measure_string_value": "", ' \
                    '"reading_timestamp": ' + str(reading_timestamp) + '}'

    return temperature_message


rabbitMQPublisher =
RabbitMQPublisher('http://10.0.0.245:15672/api/exchanges/%2f/amq.topic/publish',
'test', 'test')

# Print out the temperature until the program is stopped.
while True:
   temperature_celcius, temperature_farhenite = read_temp()
   temperature_msg_json =
MessageGenerator.generate_numeric_value_message("temperature",
temperature_farhenite,
                                                int(round(time.time() * 1000)))
   rabbitMQPublisher.publish('intelligent-homes-topic',
                        base64.b64encode(temperature_msg_json.encode()))
   time.sleep(2)
```

Cassandra Code:

```
CREATE KEYSPACE intelligent_homes WITH replication = {'class':'SimpleStrategy',
'replication_factor' : 1};

CREATE TABLE intelligent_homes.readings (
   home_id VARCHAR,
   zone_id VARCHAR,
   measure_name VARCHAR,
   reading_timestamp TIMESTAMP,
   measure_numeric_value DOUBLE,
   measure_string_value VARCHAR,
PRIMARY KEY ((home_id, zone_id), measure_name, reading_timestamp));
```

RabbitMQ to Spark to Cassandra Code:

App.java

```
package edu.csus.cs502;

import edu.csus.cs502.config.ApplicationProperties;

import java.io.IOException;
import java.util.Properties;

/**
 * Created by prachiutkhede on 4/9/17.
 */
public class App {
    public static void main(String[] args) throws IOException {
        IntelligentHomeApp intelligentHomeApp = new IntelligentHomeApp();

intelligentHomeApp.run(ApplicationProperties.getApplicationProperties("/application.pr
operties"));
    }
}
```

IntelligentHomeApp.java

```
package edu.csus.cs502;

import edu.csus.cs502.config.RabbitMQConfig;
import edu.csus.cs502.consumer.Consumer;
import edu.csus.cs502.consumer.rabbitmq.RabbitMQConsumer;
import edu.csus.cs502.model.HomeMessage;
import edu.csus.cs502.parser.MessageParser;
import edu.csus.cs502.parser.rabbitmq.RabbitMQMessageParser;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.Duration;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;

import java.util.Properties;
```

```
import static com.datastax.spark.connector.japi.CassandraJavaUtil.javaFunctions;
import static com.datastax.spark.connector.japi.CassandraJavaUtil.mapToRow;

/**
 * Created by prachiutkhede on 4/9/17.
 */
public class IntelligentHomeApp {
   public void run(Properties properties) {
      SparkConf sparkConf = new
SparkConf().setAppName(properties.getProperty("spark.app.name"))
                                 .setMaster(properties.getProperty("spark.master"))
                                 .set("spark.cassandra.connection.host",
properties.getProperty("spark.cassandra.connection.host"));

      JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(properties.getProperty("spark.streaming.batch.interval"))));

      Consumer rabbitMQConsumer = new RabbitMQConsumer();
      MessageParser messageParser = new RabbitMQMessageParser();


      JavaDStream<HomeMessage> messages = rabbitMQConsumer.createStream(jssc,
                                                HomeMessage.class,

RabbitMQConfig.getConnectionParams(properties),
                                                messageParser.getMessageParser())
                                          .filter(message -> message != null);

      messages.foreachRDD(rdd -> {
         javaFunctions(rdd)
               .writerBuilder(properties.getProperty("cassandra.keyspace"),
                     properties.getProperty("cassandra.table"),
                     mapToRow(HomeMessage.class, HomeMessage.PAIRS))
               .saveToCassandra();
      });

      jssc.start();
      jssc.awaitTermination();
   }
}
```

RabbitMQConsumer.java

```
package edu.csus.cs502.consumer.rabbitmq;

import com.rabbitmq.client.QueueingConsumer;
import edu.csus.cs502.consumer.Consumer;
import org.apache.spark.streaming.api.java.JavaReceiverInputDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.StreamingContext;


import java.util.Map;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.streaming.rabbitmq.RabbitMQUtils;

/**
 * Created by prachiutkhede on 4/9/17.
 */
public class RabbitMQConsumer implements Consumer {
   @Override
   public <R> JavaReceiverInputDStream<R> createStream(JavaStreamingContext jsc,
                                        Class<R> typeClass,
                                        Map<String, String> params,
                                        Function<byte[], R> messageHandler) {
      return RabbitMQUtils.createJavaStream(jsc, typeClass, params, messageHandler);
   }
}
```

ApplicationProperty.java

```
package edu.csus.cs502.config;

import java.io.IOException;
import java.util.Properties;

/**
 * Created by prachiutkhede on 9/17/17.
 */
public class ApplicationProperties {
   public static Properties getApplicationProperties(String fileName) throws IOException
{
      Properties properties = new Properties();
```

```
        properties.load(ApplicationProperties.class.getResourceAsStream(fileName));
        System.out.println("Properties: " + properties);
        return properties;
    }
}
```

RabbitMQConfig.java

```
package edu.csus.cs502.config;

import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

/**
 * Created by prachiutkhede on 4/9/17.
 */
public class RabbitMQConfig {
    public static Map<String, String> getConnectionParams(Properties properties) {
        Map<String, String> connectionParams = new HashMap<>();

        connectionParams.put("hosts", properties.getProperty("rabbitmq.hosts"));
        connectionParams.put("queueName",
properties.getProperty("rabbitmq.queueName"));
        connectionParams.put("vHost", properties.getProperty("rabbitmq.vHost"));
        connectionParams.put("userName", properties.getProperty("rabbitmq.userName"));
        connectionParams.put("password", properties.getProperty("rabbitmq.password"));

        return connectionParams;
    }
}
```

HomeMessage.java

```
package edu.csus.cs502.model;

import org.codehaus.jackson.annotate.JsonProperty;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
```

```
/**
 * Created by prachiutkhede on 4/9/17.
 */
public class HomeMessage implements Serializable {
    public final static Map<String, String> PAIRS = new HashMap<>();
    static {
        PAIRS.put("homeId", "home_id");
        PAIRS.put("zoneId", "zone_id");
        PAIRS.put("measureName", "measure_name");
        PAIRS.put("measureNumericValue", "measure_numeric_value");
        PAIRS.put("measureStringValue", "measure_string_value");
        PAIRS.put("readingTimestamp", "reading_timestamp");
    }

    @JsonProperty("home_id")
    private String homeId;

    @JsonProperty("zone_id")
    private String zoneId;

    @JsonProperty("measure_name")
    private String measureName;

    @JsonProperty("measure_numeric_value")
    private Double measureNumericValue;

    @JsonProperty("measure_string_value")
    private String measureStringValue;

    @JsonProperty("reading_timestamp")
    private Long readingTimestamp;

    public HomeMessage() {
    }

    public String getHomeId() {
        return homeId;
    }

    public void setHomeId(String homeId) {
        this.homeId = homeId;
    }
```

```
public String getZoneId() {
    return zoneId;
}

public void setZoneId(String zoneId) {
    this.zoneId = zoneId;
}

public String getMeasureName() {
    return measureName;
}

public void setMeasureName(String measureName) {
    this.measureName = measureName;
}

public Double getMeasureNumericValue() {
    return measureNumericValue;
}

public void setMeasureNumericValue(Double measureNumericValue) {
    this.measureNumericValue = measureNumericValue;
}

public String getMeasureStringValue() {
    return measureStringValue;
}

public void setMeasureStringValue(String measureStringValue) {
    this.measureStringValue = measureStringValue;
}

public Long getReadingTimestamp() {
    return readingTimestamp;
}

public void setReadingTimestamp(Long readingTimestamp) {
    this.readingTimestamp = readingTimestamp;
}

@Override
public String toString() {
```

```
        return "HomeMessage{" + "homeId='" + homeId + '\'' + ", zoneId='" + zoneId + '\'' +
", measureName='" + measureName + '\'' + ", measureNumericValue=" +
measureNumericValue + ", measureStringValue='"
                + measureStringValue + '\'' + ", readingTimestamp=" + readingTimestamp +
'}';
    }
```

HomeMessageBuilder.java

```
    public static final class HomeMessageBuilder {

        private String homeId;
        private String zoneId;
        private String measureName;
        private Double measureNumericValue;
        private String measureStringValue;
        private Long readingTimestamp;

        private HomeMessageBuilder() {
        }

        public static HomeMessageBuilder aHomeMessage() {
            return new HomeMessageBuilder();
        }

        public HomeMessageBuilder withHomeId(String homeId) {
            this.homeId = homeId;
            return this;
        }

        public HomeMessageBuilder withZoneId(String zoneId) {
            this.zoneId = zoneId;
            return this;
        }

        public HomeMessageBuilder withMeasureName(String measureName) {
            this.measureName = measureName;
            return this;
        }

        public HomeMessageBuilder withMeasureNumericValue(Double
```

```
measureNumericValue) {
            this.measureNumericValue = measureNumericValue;
            return this;
        }

        public HomeMessageBuilder withMeasureStringValue(String measureStringValue)
{
            this.measureStringValue = measureStringValue;
            return this;
        }

        public HomeMessageBuilder withReadingTimestamp(Long readingTimestamp) {
            this.readingTimestamp = readingTimestamp;
            return this;
        }

        public HomeMessage build() {
            HomeMessage homeMessage = new HomeMessage();
            homeMessage.setHomeId(homeId);
            homeMessage.setZoneId(zoneId);
            homeMessage.setMeasureName(measureName);
            homeMessage.setMeasureNumericValue(measureNumericValue);
            homeMessage.setMeasureStringValue(measureStringValue);
            homeMessage.setReadingTimestamp(readingTimestamp);
            return homeMessage;
        }
    }
}
```

Resources – Application.properties

```
# Spark properties
spark.master=local[*]
spark.app.name=IntelligentHomes
spark.streaming.batch.interval=1000

# RabbitMQ Properties
rabbitmq.hosts=localhost
rabbitmq.queueName=intelligent-homes-queue
rabbitmq.vHost=/
rabbitmq.userName=test
rabbitmq.password=test
```

```
spark.cassandra.connection.host=127.0.0.1
cassandra.keyspace=intelligent_homes
cassandra.table=readings
```

Spring Boot WebService Code :

CassandraConfig.java

```
package edu.csus.homes.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.data.cassandra.config.CassandraClusterFactoryBean;
import org.springframework.data.cassandra.config.CassandraSessionFactoryBean;
import org.springframework.data.cassandra.config.SchemaAction;
import org.springframework.data.cassandra.convert.CassandraConverter;
import org.springframework.data.cassandra.convert.MappingCassandraConverter;
import org.springframework.data.cassandra.core.CassandraOperations;
import org.springframework.data.cassandra.core.CassandraTemplate;
import org.springframework.data.cassandra.mapping.BasicCassandraMappingContext;
import org.springframework.data.cassandra.mapping.CassandraMappingContext;
import
org.springframework.data.cassandra.repository.config.EnableCassandraRepositories;

@Configuration
@PropertySource(value = { "classpath:application.properties" })
@EnableCassandraRepositories(basePackages = { "edu.csus.homes" })
public class CassandraConfig {

   private static final Logger LOG = LoggerFactory.getLogger(CassandraConfig.class);
```

```
    @Autowired
    private Environment env;

    @Bean
    public CassandraClusterFactoryBean cluster() {

        CassandraClusterFactoryBean cluster = new CassandraClusterFactoryBean();
        cluster.setContactPoints(env.getProperty("cassandra.contactpoints"));
        cluster.setPort(Integer.parseInt(env.getProperty("cassandra.port")));

        return cluster;
    }

    @Bean
    public CassandraMappingContext mappingContext() {
        return new BasicCassandraMappingContext();
    }

    @Bean
    public CassandraConverter converter() {
        return new MappingCassandraConverter(mappingContext());
    }

    @Bean
    public CassandraSessionFactoryBean session() throws Exception {

        CassandraSessionFactoryBean session = new CassandraSessionFactoryBean();
        session.setCluster(cluster().getObject());
        session.setKeyspaceName(env.getProperty("cassandra.keyspace"));
        session.setConverter(converter());
        session.setSchemaAction(SchemaAction.NONE);

        return session;
    }

    @Bean
    public CassandraOperations cassandraTemplate() throws Exception {
        return new CassandraTemplate(session().getObject());
    }
}
```

MeasureReadingDTO.java

```
package edu.csus.homes.dto;

import java.util.Date;

public class MeasureReadingsDTO {
   private Date readingTimestamp;
   private Double measureNumericValue;

   public MeasureReadingsDTO(Date readingTimestamp, Double
measureNumericValue) {
      this.readingTimestamp = readingTimestamp;
      this.measureNumericValue = measureNumericValue;
   }

   public Date getReadingTimestamp() {
      return readingTimestamp;
   }

   public void setReadingTimestamp(Date readingTimestamp) {
      this.readingTimestamp = readingTimestamp;
   }

   public Double getMeasureNumericValue() {
      return measureNumericValue;
   }

   public void setMeasureNumericValue(Double measureNumericValue) {
      this.measureNumericValue = measureNumericValue;
   }

   @Override
   public String toString() {
      return "MeasureReadingsDTO{" +
            "readingTimestamp=" + readingTimestamp +
            ", measureNumericValue=" + measureNumericValue +
            '}';
   }
}
```

Readings.java

```
package edu.csus.homes.entity;

import org.springframework.data.cassandra.mapping.Column;
import org.springframework.data.cassandra.mapping.PrimaryKey;
import org.springframework.data.cassandra.mapping.PrimaryKeyColumn;
import org.springframework.data.cassandra.mapping.Table;

import java.util.Date;

/**
 * Created by prachiutkhede on 10/14/17.
 */
@Table(value = "readings")
public class Readings {

    @PrimaryKey
    private ReadingsPrimaryKey pk;

    @Column(value = "measure_numeric_value")
    Double measureNumericValue;

    @Column(value = "measure_string_value")
    String measureStringValue;

    public Readings(ReadingsPrimaryKey pk, Double measureNumericValue, String
measureStringValue) {
        this.pk = pk;
        this.measureNumericValue = measureNumericValue;
        this.measureStringValue = measureStringValue;
    }

    public ReadingsPrimaryKey getPk() {
        return pk;
    }

    public void setPk(ReadingsPrimaryKey pk) {
        this.pk = pk;
    }
```

```
    public Double getMeasureNumericValue() {
        return measureNumericValue;
    }

    public void setMeasureNumericValue(Double measureNumericValue) {
        this.measureNumericValue = measureNumericValue;
    }

    public String getMeasureStringValue() {
        return measureStringValue;
    }

    public void setMeasureStringValue(String measureStringValue) {
        this.measureStringValue = measureStringValue;
    }

    @Override
    public String toString() {
        return "Readings{" +
                "pk=" + pk +
                ", measureNumericValue=" + measureNumericValue +
                ", measureStringValue='" + measureStringValue + '\'' +
                '}';
    }
}
```

ReadingsPrimaryKey.java

```
package edu.csus.homes.entity;

import org.springframework.cassandra.core.PrimaryKeyType;
import org.springframework.data.cassandra.mapping.PrimaryKeyClass;
import org.springframework.data.cassandra.mapping.PrimaryKeyColumn;

import java.io.Serializable;
import java.util.Date;

@PrimaryKeyClass
public class ReadingsPrimaryKey implements Serializable {

    @PrimaryKeyColumn(name = "home_id", ordinal = 0, type =
PrimaryKeyType.PARTITIONED)
```

```
    private String homeId;

    @PrimaryKeyColumn(name = "zone_id", ordinal = 1, type =
PrimaryKeyType.PARTITIONED)
    private String zoneId;

    @PrimaryKeyColumn(name = "measure_name", ordinal = 2, type =
PrimaryKeyType.PARTITIONED)
    private String measureName;

    @PrimaryKeyColumn(name = "reading_timestamp", ordinal = 3, type =
PrimaryKeyType.PARTITIONED)
    private Date readingTimestamp;

  public ReadingsPrimaryKey(String homeId, String zoneId, String measureName, Date
readingTimestamp) {
        this.homeId = homeId;
        this.zoneId = zoneId;
        this.measureName = measureName;
        this.readingTimestamp = readingTimestamp;
    }

    public String getHomeId() {
        return homeId;
    }

    public void setHomeId(String homeId) {
        this.homeId = homeId;
    }

    public String getZoneId() {
        return zoneId;
    }

    public void setZoneId(String zoneId) {
        this.zoneId = zoneId;
    }

    public String getMeasureName() {
        return measureName;
    }

    public void setMeasureName(String measureName) {
```

```
        this.measureName = measureName;
    }

    public Date getReadingTimestamp() {
        return readingTimestamp;
    }

    public void setReadingTimestamp(Date readingTimestamp) {
        this.readingTimestamp = readingTimestamp;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof ReadingsPrimaryKey)) return false;

        ReadingsPrimaryKey that = (ReadingsPrimaryKey) o;

        if (!homeId.equals(that.homeId)) return false;
        if (!zoneId.equals(that.zoneId)) return false;
        if (!measureName.equals(that.measureName)) return false;
        return readingTimestamp.equals(that.readingTimestamp);
    }

    @Override
    public int hashCode() {
        int result = homeId.hashCode();
        result = 31 * result + zoneId.hashCode();
        result = 31 * result + measureName.hashCode();
        result = 31 * result + readingTimestamp.hashCode();
        return result;
    }
}
```

CassandraQueries.java

```
package edu.csus.homes.query;

public class CassandraQueries {
    public static final String READINGS_QUERY = "SELECT * FROM readings WHERE
home_id='%s' AND zone_id='%s' AND measure_name='%s'";
}
```

ReadingRepository.java

```
package edu.csus.homes.repository;

import edu.csus.homes.entity.Readings;
import edu.csus.homes.entity.ReadingsPrimaryKey;
import org.springframework.data.repository.CrudRepository;

public interface ReadingsRepository extends CrudRepository<Readings,
ReadingsPrimaryKey> {
}
```

App.java

```
package edu.csus.homes;

/**
 * Hello world!
 *
 */
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class App {
   public static void main(String[] args) {
      SpringApplication.run(App.class, args);
   }
}
```

CassandraDataFetcher.java

```
package edu.csus.homes.rest;

import edu.csus.homes.dto.MeasureReadingsDTO;
import edu.csus.homes.entity.Readings;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.cassandra.core.CassandraOperations;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
```

```
import java.util.List;

import static edu.csus.homes.query.CassandraQueries.READINGS_QUERY;
import static org.springframework.web.bind.annotation.RequestMethod.GET;

/**
 * Created by prachiutkhede on 9/5/17.
 */
@RestController
public class CassandraDataFetcher {

   @Autowired
   private CassandraOperations cassandraOperations;

   @CrossOrigin
   @RequestMapping(method=GET, value =
"/readings/{homeId}/{zoneId}/{measureName}")
   public List<MeasureReadingsDTO> getTimeSeriesData(@PathVariable String
homeId,
                                     @PathVariable String zoneId,
                                     @PathVariable String measureName) {

      List<MeasureReadingsDTO> readings = new ArrayList<>();

      String query = String.format(READINGS_QUERY, homeId, zoneId, measureName);

      cassandraOperations.select(query, Readings.class)
                  .stream()
                  .forEach(r -> readings.add(new
MeasureReadingsDTO(r.getPk().getReadingTimestamp(),
r.getMeasureNumericValue())));

      return readings;
   }
}
```

Web Service to Web UI

```
<!DOCTYPE HTML>
<html>
<head>
   <script type="text/javascript" src="https://canvasjs.com/assets/script/jquery-
```

```
                    1.11.1.min.js"></script>
<script type="text/javascript"
                    src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/series-label.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>

<script type="text/javascript">
                    function temperatureData (label,number) {
                       $('#chartContainer').highcharts({
                       chart: {
                       type: 'line'
                       },
                       title: {
                       text: 'Room Temperature'
                       },
                       xAxis: {
                       categories : label,
                       title: {
                          text : 'Time'
                       }
                       },
                       yAxis: {
                       title: {
                       text: 'Temperature'
                       }
                       },
                       series: [{
                       name: 'temperature',
                       data: number
                       }]
                       });
                    }

                    $(document).ready(setInterval(function() {
                       $.ajax({
                          url: 'http://localhost:8080/readings/HOME-
            1/LIVING_ROOM/temperature',
                          type: 'GET',
                          async: true,
                          dataType: "json",
                          success: function (data) {
                             var labels = [], number=[]
```

```
                        for(var i = 0; i < data.length; i++) {
                            labels.push( data[i]['readingTimestamp']);
                            number.push(data[i].measureNumericValue);
                        }
                        temperatureData(labels,number);
                    }
                });
            }, 5000));

        </script>

</head>
<body>
<div id="chartContainer" style="height: 400px; width: 80%;"></div>
</body>
</html>
```

BIBLIOGRAPHY

[1] IoT. "Home - IEEE Internet of Things", [Online]. Available: http://iot.ieee.org/. [Accessed: 28- Nov- 2017].

[2] Rohan, "Home Automation System Market Worth 79.57 Billion USD by 2022", [Online]. Available: https://www.thestreet.com/story/14076591/1/home-automation-system-market-worth-7957-billion-usd-by-2022.html. [Accessed: 28- Nov- 2017].

[3] Raspberry Pi. "Raspberry Pi - Teach, Learn, and Make with Raspberry Pi", [Online]. Available: http://www.raspberrypi.org/. [Accessed: 28- Nov- 2017].

[4] RabbitMQ. "RabbitMQ - Messaging that just works", [Online]. Available: http://www.rabbitmq.com. [Accessed: 29- Nov- 2017].

[5] Apache Spark. "Apache Spark™ - Lightning-Fast Cluster Computing", [Online]. Available: http://spark.apache.org/. [Accessed: 29- Nov- 2017].

[6] Apache Cassandra. "What is Apache Cassandra | DataStax Academy: Free Cassandra Tutorials and Training", [Online]. Available: https://academy.datastax.com/planet-cassandra/what-is-apache-cassandra. [Accessed: 29- Nov- 2017].

[7] Spring Boot. "Spring Boot", [Online]. Available: https://projects.spring.io/spring-boot/. [Accessed: 29- Nov- 2017].

[8] Highcharts. "Interactive JavaScript charts for your webpage | Highcharts", [Online]. Available: http://www.highcharts.com. [Accessed: 29- Nov- 2017].

[9] GeeksforGeeks. "Producer-Consumer solution using threads in Java - GeeksforGeeks", [Online]. Available: http://www.geeksforgeeks.org/producer-consumer-solution-using-threads-java/. [Accessed: 30- Nov- 2017].