Project Report

AWS Iot Sensor App

By

Aashika Vasra Thirukkonda Ramesh
Vishakha Supekar

Git Url: https://github.com/VishakhaSupekar04/AWS_IoT_Sensor_App.git

1.Abstract/Executive Summary:

The aim of the project is to combine the domains of IoT and cloud and make an effective analysis of the data gathered. The project used the Cortex M4 microcontroller - nucleo F401RE and the sensor shield IKS01A2 as a project set up. The sensor shield sensed the temperature data using its HTS221 sensor and the data was sent to the computer using I2C serial communication. The results of the temperature data was projected in putty, via serial communication. The results from putty was stored in a text file. This is the IoT part.

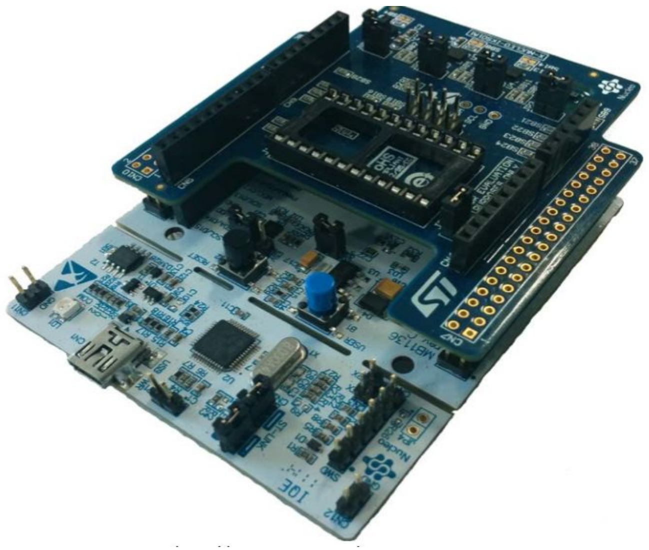
For the cloud side, we used DynamoDB to store all the information into the cloud. An Amazon AWS free student account was created, and code was written in Java to upload the data from the text file to the AWS DynamoDB via socket communication. On successful data upload, the next part was data visualization. So, a mobile application was created using Xamarin to fetch the data from our AWS account and project the data in the form of a graph. Other analysis like the max and min temperature recorded and the average temperature collected was also show via the mobile application emulator and visualization clearly portrayed the temperature changes in the environment.

2. Introduction:

Internet of Things is everywhere in today's scenario. The number of connected devices is more and so is the data generated every second. This has led to the birth of a new domain called the big data. With so many connected devices and big data, where will the humans store them? Database? File? No, the answer to this question is the cloud. Large data centers are available to store and process information. The IoT devices started to store their results in cloud leading to better solutions. Our project is just to try to give an actual glimpse of how this is done. How an IoT device stores its information in a cloud. We wanted to provide a snapshot of what a real world IoT device does with its captured information. The aim of the project is to capture a real-time sensor data and upload it to the cloud and provide data analysis. The data analysis was provided using a mobile application using Xamarin which constructed a graph to highlight the results of the data gathered.

3. Conceptual Design of the system:

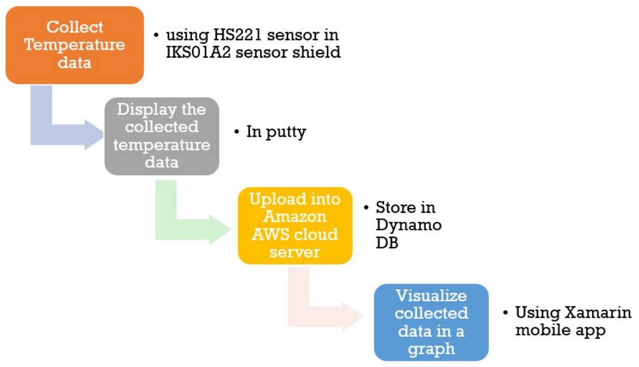
The system uses the cortex M4 Microcontroller nucleo-401RE and the sensor shield IKAS01A2 mounted over it.



The software required for the project includes Spring Tool Suite (STS) and Xamarin. The hardware part was to collect the data from the surroundings while the software part enabled to connect to the AWS cloud and then upload the data into DynamoDB in the cloud. In Xamarin, code was written to extract the data from our DB in our AWS account and then project it as a graph. A refresh button was added in the app to periodically update the new incoming information.

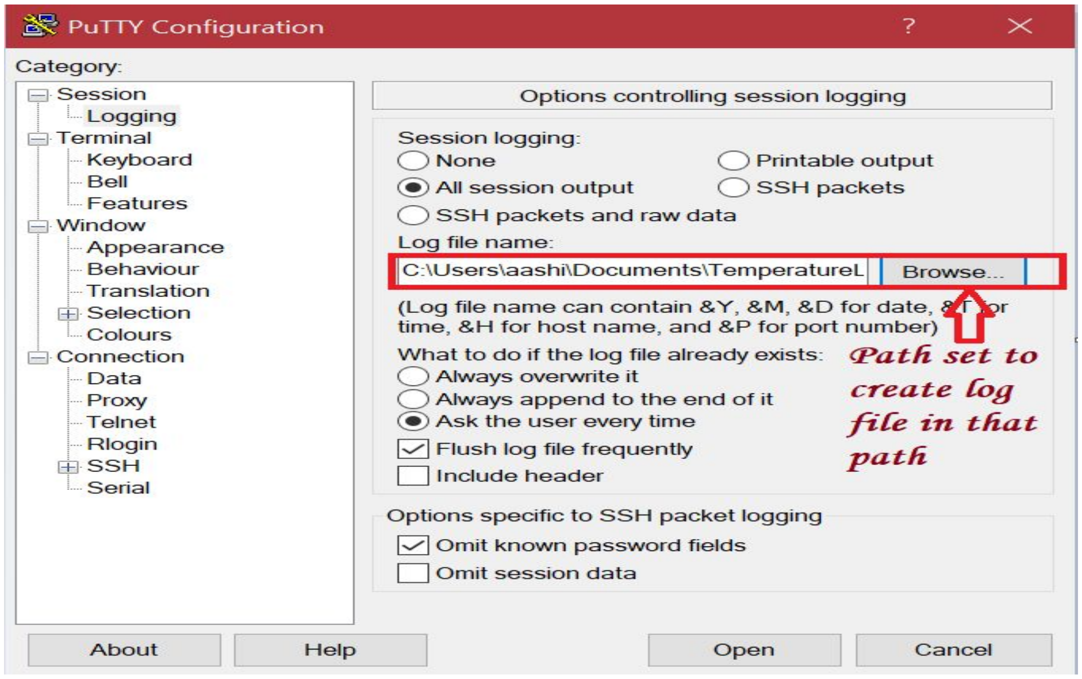
4. Detailed design and implementation of the code:

The following are the steps followed to get accurate results:


Collect Temperature data
• using HS221 sensor in IKS01A2 sensor shield
Display the collected temperature data
• In putty
Upload into Amazon AWS cloud server
• Store in Dynamo DB
Visualize collected data in a graph
• Using Xamarin mobile app

First the data was collected using the sensor. The sensor shield has HTS221 sensor which senses temperature and humidity. For our project, only temperature data is taken into consideration.

Hence only temperature value is fetched. The data was displayed in putty - a free open-source terminal emulator, which provides SSH, Telnet, Rlogin and raw socket connection. Putty also supports a serial port. We make use of Putty's serial port communication to show our data on a computer console. But before displaying our data from sensor on putty, the session output of putty has to be collected in a text file to upload it into AWS. So, we set the session log as follows:



Session logging: ( ) None ( ) Printable output [● All session output] ( ) SSH packets ( ) SSH packets and raw data

Log file name: C:\Users\aashi\Documents\TemperatureL

(Log file name can contain &Y, &M, &D for date, &T for time, &H for host name, and &P for port number)

What to do if the log file already exists: ( ) Always overwrite it ( ) Always append to the end of it [● Ask the user every time]

[✓] Flush log file frequently

[ ] Include header

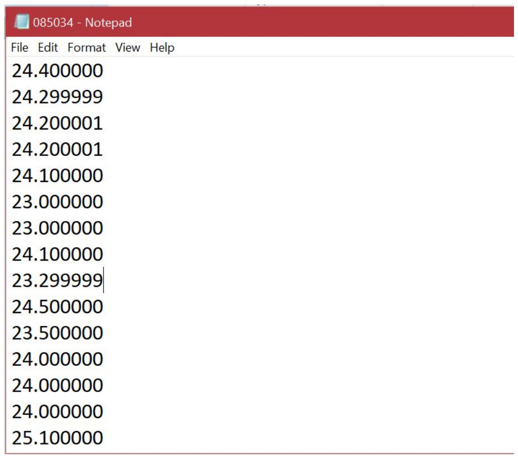
Path set to create log file in that path

Options specific to SSH packet logging

[✓] Omit known password fields

[ ] Omit session data

Now after the data is received and the log file is generated on the local system, we must upload the log file data into AWS DynamoDB by starting the server to client connectivity.

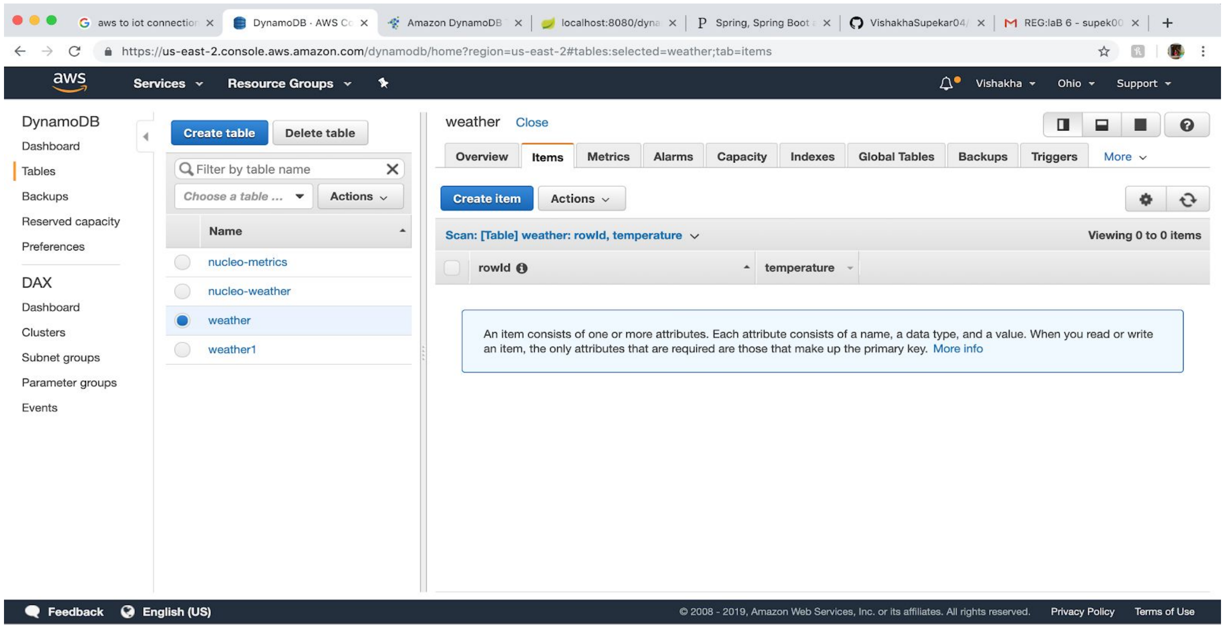


```
24.400000
24.299999
24.200001
24.200001
24.100000
23.000000
23.000000
24.100000
23.299999
24.500000
23.500000
24.000000
24.000000
24.000000
25.100000
```

Now the data has been uploaded into the DynamoDB in AWS. So, the data has to be traced as a graph and then data analysis has to be performed. For this, a mobile application was made using

Xamarin which fetches the code from AWS and then draws a graph and then derives maximum, minimum and average temperature recorded. The following is the code for the xamarin mobile app. The output of this app is showed in a mobile emulator provided by Xamarin IDE.

4.1 DynamoDB table data

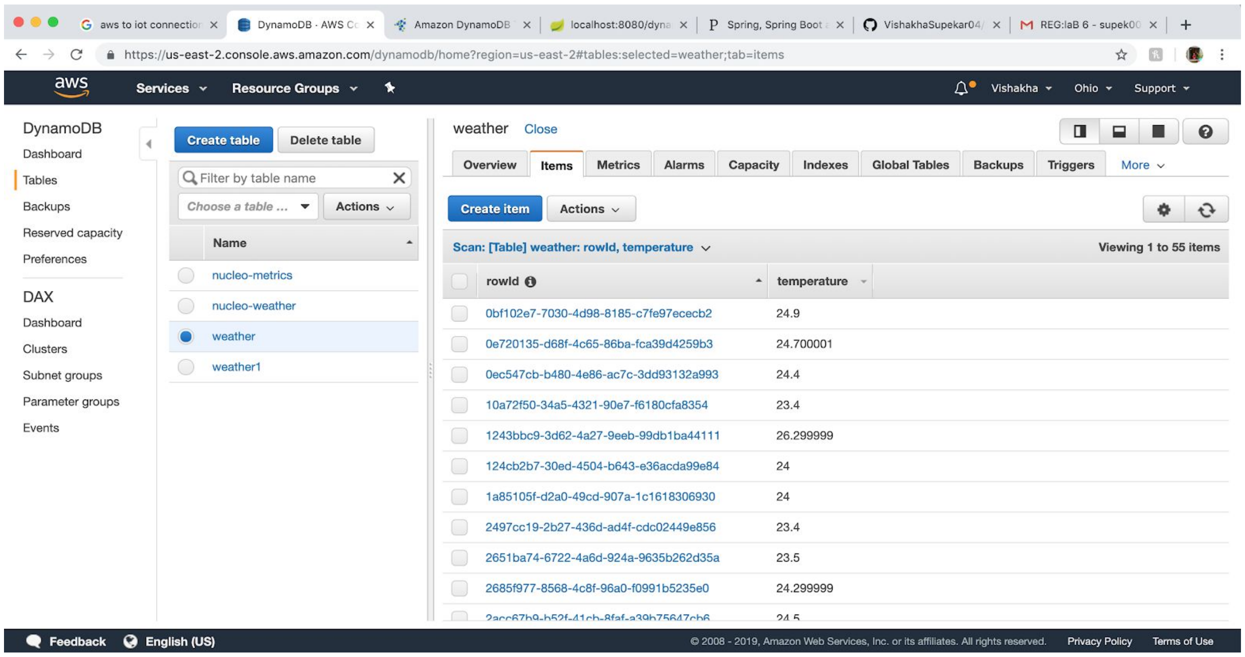


Scan: [Table] weather: rowId, temperature — Viewing 0 to 0 items

An item consists of one or more attributes. Each attribute consists of a name, a data type, and a value. When you read or write an item, the only attributes that are required are those that make up the primary key. More info

Name
nucleo-metrics
nucleo-weather
weather
weather1

a. Initially table is empty



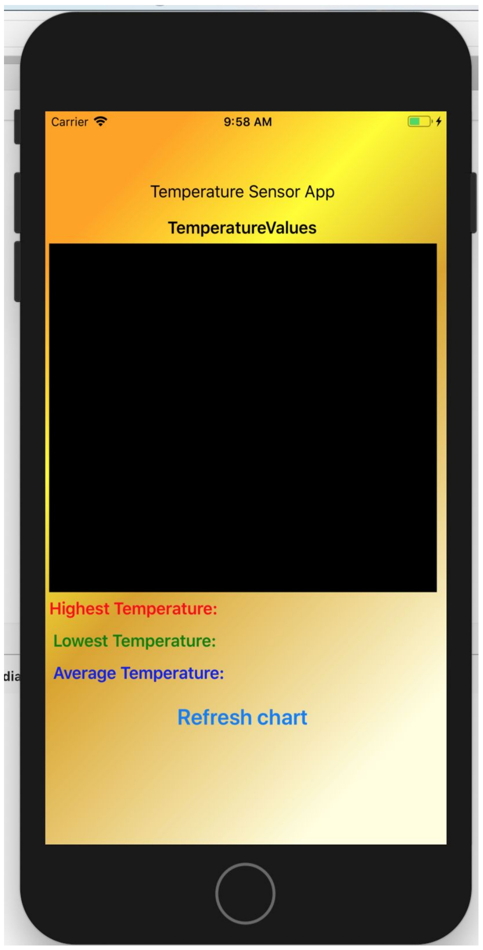
Scan: [Table] weather: rowId, temperature — Viewing 1 to 55 items

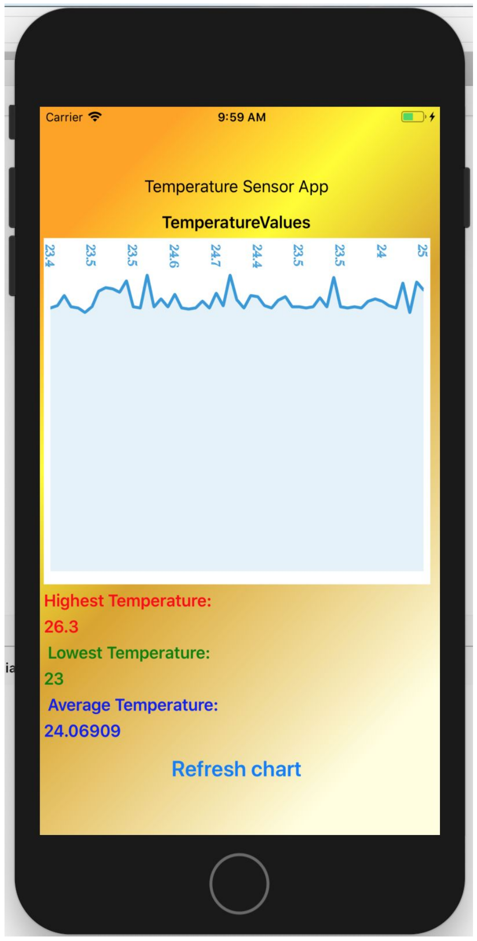
rowId	temperature
0bf102e7-7030-4d98-8185-c7fe97ececb2	24.9
0e720135-d68f-4c65-86ba-fca39d4259b3	24.700001
0ec547cb-b480-4e86-ac7c-3dd93132a993	24.4
10a72f50-34a5-4321-90e7-f6180cfa8354	23.4
1243bbc9-3d62-4a27-9eeb-99db1ba44111	26.299999
124cb2b7-30ed-4504-b643-e36acda99e84	24
1a85105f-d2a0-49cd-907a-1c1618306930	24
2497cc19-2b27-436d-ad4f-cdc02449e856	23.4
2651ba74-6722-4a6d-924a-9635b262d35a	23.5
2685f977-8568-4c8f-96a0-f0991b5235e0	24.299999
2acc67b9-b52f-41cb-8faf-a39b75647cb6	24.5

b. Data is populated in the table after log file is generated

Table Schema
rowId - auto generated Hash key value which is the Primary key of the table
temperature- sensor temperature values

4.2 Mobile application graph generated:





a. Before clicking Refresh button *b. After clicking Refresh button*

5. Main challenges of implementation:

The connectivity between the sensed information and the cloud was initially planned to be done using bluetooth. But it wasn't possible due to the lack of apps that connect bluetooth and the cloud. Then we decided to use Wifi shield to connect. But the sensor shield board was deprecated and hence we decided to write the output in the file. But from the mbed code, it was very difficult to write into a file directly. No file was generated since the board occupied a separate drive and the file wasn't getting generated. Finally, we decided to use the copy the output from the console into a text file and convert it into JSON format and upload it into AWS, which finally worked.

6. Testing:

The program was tested for different values of temperature recorded. The sensor was taken to different places to record different temperature values to record different values of temperatures. The graph for different locations fluctuated as per the temperature data recorded and the graph changed for every click of the refresh button to update newly added data. New graphs appeared for every set of values.

7. Summary:

Our project was an attempt to show where the data from an IoT sensor was stored, processed and analysed. The data was successfully displayed on a console, copied in a text file and uploaded into the cloud. From the cloud to see better results, we constructed a graph. Thus, from a sensor the data is sent to cloud for analysis is what our project depicts. Though it may be simple to say the data is transferred, there is a lot of background work involved to transfer just one type of data - temperature data to cloud and it is still a challenging task to collect different types of data collected by different sensors and parsing them into JSON format and then uploading into the cloud.

8. References and Bibliography:

The following sites were referred to finish the project:

1. https://os.mbed.com/teams/ST/code/HTS221/docs/tip/
2. https://os.mbed.com/teams/ST/code/HelloWorld_IKS01A2/file/175f561f1a71/main.cpp/
3. https://docs.aws.amazon.com/aws-technical-content/latest/aws-vpc-connectivity-options/introduction.html
4. https://www.youtube.com/watch?v=Q9hLQfBBbhU
5. https://www.tutorialspoint.com/xamarin/

9. Code Listings:

Demo video: Please click download to downlaod the video
https://github.com/VishakhaSupekar04/AWS_IoT_Sensor_App/blob/master/SensorAppDemo.mov

The code to collect temperature sensor data display it in putty as follows:

```
#include "mbed.h"
#include "XNucleoIKS01A2.h"

DigitalOut led1(LED1, 1);
bool flag=false;

Ticker t1;
Ticker t2;

// Initialize variables
/* Instantiate the expansion board */
static XNucleoIKS01A2 *mems_expansion_board = XNucleoIKS01A2::instance(D14, D15, D4,
D5);

/* Retrieve the composing elements of the expansion board */
static HTS221Sensor *hum_temp = mems_expansion_board->ht_sensor;

// Handler for the aliveness LED; to be called every 0.5s
void blink()
{
        led1=!led1;
}
// Handler for the measurements update; rise a flag every 3 seconds
void updateFlag()
{
        flag =!flag;  //toggle flag to true
}

/* Simple main function */
int main()
{
        uint8_t id;
        float value1;
```

```
 /* Enable all sensors */
 hum_temp->enable();

 /* Attach a function to be called by the Ticker objects at a specific interval in seconds */
 t1.attach(&blink,0.5);
 t2.attach(&updateFlag,3);

 hum_temp->read_id(&id);
 while(1) {

         while(flag)
         {
         hum_temp->get_temperature(&value1);// Collect temperature data
         printf("\n\r%2f", value1);//Print it on screen in putty
         flag = false;
         }
}
}
```

Now code for uploading the file into DB is written in JAVA Spring Tool Suite. We have 4 files - DynamoDBConfig, WebController, DynamoDBRepository and Weather.

DynamoDBConfig.java:

```
*********************
package com.springboot.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;

@Configuration
public class DynamoDBConfig {

        @Value("${amazon.access.key}")
        private String awsAccessKey;

        @Value("${amazon.access.secret-key}")
```

```
	private String awsSecretKey;

	@Value("${amazon.region}")
	private String awsRegion;

	@Value("${amazon.end-point.url}")
	private String awsDynamoDBEndPoint;

	@Bean
	public DynamoDBMapper mapper() {
		return new DynamoDBMapper(amazonDynamoDBConfig());
	}

	public AmazonDynamoDB amazonDynamoDBConfig() {
		return AmazonDynamoDBClientBuilder.standard()
				.withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(awsDynamoDBEndPoint,awsRegion))
				.withCredentials(new AWSStaticCredentialsProvider(new
BasicAWSCredentials(awsAccessKey,awsSecretKey)))
				.build();
	}
}
```

DynamoDBRepository.java:

```
************************
package com.springboot.repository;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBScanExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.PaginatedScanList;
import com.springboot.model.Weather;


@Repository
public class DynamoDbRepository {

	//private static final Logger LOGGER =
LoggerFactory.getLogger(DynamoDbRepository.class);
```

```
	@Autowired
	private DynamoDBMapper mapper;

	public void insertIntoDynamoDB(Weather weather) {
		mapper.save(weather);
		}

	public ArrayList<Weather> getAll() {
		PaginatedScanList<Weather> results = mapper.scan(Weather.class, new
DynamoDBScanExpression());
		ArrayList<Weather> temps = new ArrayList<Weather>();
		results.forEach(w -> temps.add(w));
		return temps;
	}
}
```

WebController.java:

```
******************
package com.springboot.controller;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.springboot.model.Weather;
import com.springboot.repository.DynamoDbRepository;

@RestController
@RequestMapping("/dynamoDb")
public class WebController {

	@Autowired
	private DynamoDbRepository repository;

	private void readFile(String fileName) throws IOException {
```

```
            FileReader fr = new FileReader(fileName);
            BufferedReader br = new BufferedReader(fr);

            String str;

            while((str=br.readLine())!=null)
            {
                    if(!str.isEmpty()) {
                            Weather w = new Weather();
                            w.setTemperature(Double.parseDouble(str));
                            repository.insertIntoDynamoDB(w);
                    }
            }

            br.close();
    }


    private void readDir() throws IOException {
            File folder = new File("C:/Users/aashi/Documents/TemperatureLogs/");
            File[] listOfFiles = folder.listFiles();

            for (File file : listOfFiles) {
               if (file.isFile()) {

                   this.readFile(file.getAbsolutePath());
               }
            }
    }

    @PostMapping
    public String insertIntoDynamoDB(@RequestBody Weather weather) {
            repository.insertIntoDynamoDB(weather);
            return "Successfully inserted into DynamoDB table";
    }

    @GetMapping
    @RequestMapping("/all")
    public List<Weather> getAll() {
            return repository.getAll();
    }

    @GetMapping
    @RequestMapping("/processData")
    public String processData() throws IOException {
```

```
		this.readDir();
		return "Successfully Processed Data";
	}
}
```

Weather.java:

```
***********
package com.springboot.model;

import java.io.Serializable;

import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAutoGeneratedKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName = "weather")
public class Weather implements Serializable{

	private static final long serialVersionUID =1L;

	private String rowId;
	private double temperature;

	@DynamoDBHashKey(attributeName ="rowId")
	@DynamoDBAutoGeneratedKey
	public String getRowId() {
		return rowId;
	}

	public void setRowId(String rowId)
	{
		this.rowId=rowId;
	}

	@DynamoDBRangeKey
	public double getTemperature() {
		return temperature;
	}

	public void setTemperature(double temperature)
	{
		this.temperature=temperature;
	}

}
```