Anthony, Hamza, Matt, Phong, Sang
Project Overview and Documentation
Technical Software Project

IoT: Convenience for you

Abstract

The Internet of Things are a collection of devices that consumers uses to monitor real world objects or events through the internet. These are not seen as a typical device like a computer or a phone, they are thermostats, smart assistants, appliances and many others. In this project, we will be taking a look into how these devices are created and how to create your own using the guide that we will provide laying out the steps that we took to create our application. We will provide a skeleton code and documentation on how to build you own Internet of Things or IoT device using a Raspberry Pi 3 B+, AWS S3 bucket, Webpack, Codeship, node.js, shadow, MQTT protocol, and a temperature and humidity sensor. With this paper you will have a clearly laid out, step by step guide for the process needed to create your own IoT device. This guide will be a stripped out version of our codebase to allow for easy implementation for a wide variety of different sensors along with different implementations using different deployment techniques.

Motivation

Building your own IoT device offers you a chance to create something interesting, and at the same time it gives you a great opportunity to learn about this exciting emerging field. Technology is always changing at a rapid pace, and by going through this documentation you will learn how the latest tools and concepts are used to implement an IoT device. We have already done the research needed to understand how the different tools work together and what all is needed to make the hardware talk with the software to give the user an easy to use application. Developing IoT devices has rapidly grown in the recent years due to the increasing demand for “smart” household items. By following this guide, an individual can gain the necessary knowledge to begin creating their own IoT devices that suit their own personal desires and needs, without having to do all of the in depth research needed to build a framework and transmission protocol to share data between devices.

Introduction

Recently, with the growing popularity for smart home devices there has been a huge demand created for internet connected devices. The Internet of Things, are internet connected devices which gives their users the ability to control or to react to events in real time remotely from anywhere in the world. This include the ability to monitor their homes, adjust their thermostat, control lighting, and many other home related functionalities. Due to the growing demand for IoT device, one may want to create a more personalize IoT device to better suit their needs. In order to create an IoT device is currently a very hard task to accomplish because of the lack of documentation and resources available. There are many approach to create an IoT device and each individual creating a new device will take a slightly different approach. In this paper we have outlined the procedure to implement an IoT device using a temperature and humidity sensor integrated with a Raspberry Pi 3 B+ and continuously displaying data on a web application using AWS a secure cloud platform created by Amazon to deploy our application. We have created a list of documentation that will go over important concepts and tools such as Webpack, Codeship, AWS S3 bucket, MQTT protocol, shadow, and node.js to create your own IoT device. After reading this documentation, you will have a better understanding about how an IoT device works and give you the necessary skill to create other IoT devices.

Methods

What did you do to find the answer?

Include screenshots and code here.

When looking at creating an IoT device it can seem like a daunting task and you may not have any idea about where to start. For this we will break down creating an IoT device into a few different parts. We will be taking a look at the Raspberry Pi, MQTT Transmission Protocol, AWS and S3 Bucket and final our Front End application.

Raspberry Pi

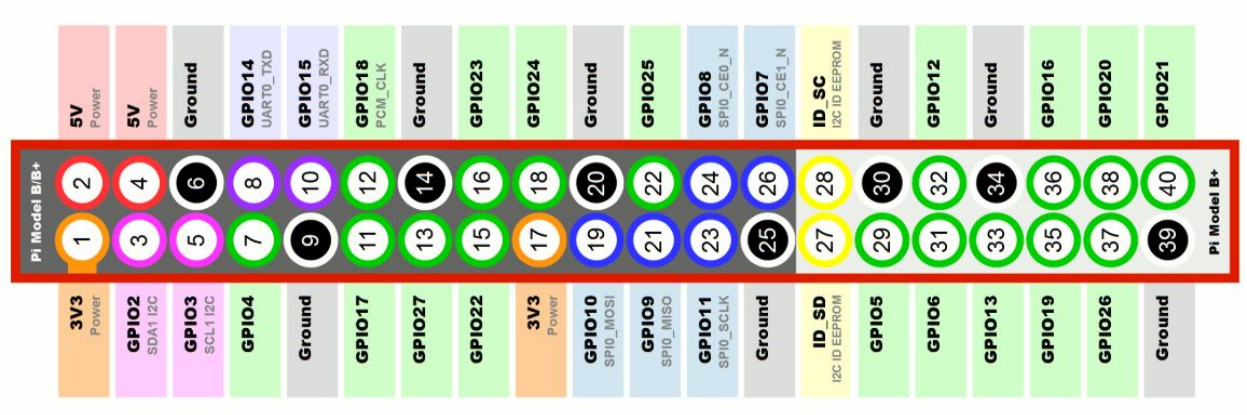
First off for this IoT project we will be using the Raspberry Pi 3 B+. If you are creating your own IoT device any version of the Raspberry Pi will work so long as you have either a Ethernet or WiFi connection capabilities on the Pi. For this project we started off with a brand new Raspberry Pi so ours needed some setup to get started. If you have already done the setup and your Raspberry Pi already boots up you can skip down to the start of installing software.

For this project on it is not necessary to have Raspbian the Raspberry Pi operating system installed, just the basic terminal boot will work, however Raspbian makes running applications a bit easier to follow so that is what we will be doing. To do this we are following the guide at thepi.io which gives a great outline of how to install Raspbian and get your Pi working.

Now that our Pi is booting up into Raspbian OS we can start installing software. For this project we will be using JavaScript and more specifically node.js to handle interfacing with our sensor on the Raspberry Pi. This can be done fairly easily by pulling up the terminal window and running the following command.

```
pi on raspberrypi in ~
> sudo apt-get install node
```

This when run will install node.js onto your Pi for us to connect to our sensor. For this project we will be using the temperature and humidity sensor. For this we will need to connect the sensor to our Pi. The temperature sensor that we have chosen uses the pins provided on the sensor itself to connect onto the Raspberry Pi using the GPIO board. Depending on what sensor you choose to implement the wiring of the sensor will be different so you will need to pull up a wiring diagram for your specific sensor and see how it will interface with the Pi's GPIO pins.



Now that we have our sensor all wired up we will need to write some JavaScript code that will allow us to monitor the sensor. This code will be running locally on the Pi and will allow us to retrieve the state of the sensor and get data in real time. This code will interface with the MQTT Protocol to relay the data to the AWS S3 bucket.

For our application the code is relatively straightforward and only has one method for the on state of the sensor.

```
device.on('connect', function() {
    console.log("device connected");
    device.subscribe('test_2');
    device.register(deviceId, {
        enableVersioning: false
    }, () => {
        console.log("thingshadow registered");
        setInterval(function () {
            dht11.read(11, 4, function(err, temperature, humidity) {
                if (!err) {
                    shadow.state.reported.temperature = temperature.toFixed(1);
                    shadow.state.reported.humidity = humidity.toFixed(1);
                    console.log('temp: ' + temperature.toFixed(1) + '°C, ' +
                        'humidity: ' + humidity.toFixed(1) + '%'
                    );
                    device.update(deviceId, shadow, () => {
                        console.log(arguments);
                    });
                }
            });
        }, 2000);
    })

});
```

This code will report the data to MQTT and the AWS Shadow protocol to change the state of the sensor remotely.

MQTT Transmission Protocol

When the data from the sensor is sent over the MQTT protocol will cleanup the data and tell the AWS web page what to display. This is done by creating a model that is continuously updating the view when new data is sent over. The view is made up from different components that control each individual sensor. The web page that we created on AWS was created dynamically so that it would allow for implementation of multiple sensors at once.

```
// Triggered when a component is removed
componentManager.onComponentRemove = (id) => {
    console.log('componentRemoved', id);
    sdk.client.unsubscribe(`$aws/things/${id}/shadow/update/accepted`)
};

//Triggered when something is changed on the component itself like a button press or state change
componentManager.onComponentStateChange = (component) => {
    sdk.client.publish(
        `$aws/things/${component.model.id}/shadow/update`,
        JSON.stringify({state: { desired: component.model.state.desired}})
    );
};
```

When the model detects that there was a change in any of the components, it will send a request to that specific component to be redrawn with the updated data.

When the component receives that the model has been changed it will check that the sensor is still active. The component will also render the screen again displaying any updates to the data along with displaying a checkbox with the current state of the sensor being active or inactive.

```
this.model.onStateDesiredChange = (desiredState) => {
    if(desiredState && desiredState.samplerState && desiredState.samplerState === 'active'){
        let samplerState = desiredState.samplerState === 'active';
        $(this.compiledTemplate).find("#sensorStateCheckBox").prop('checked', samplerState);
    }
}
```

```
onRenderComplete(){
    $(this.compiledTemplate).find("button").click(() => {
        this.remove();
    });

    if(this.model.state.desired.samplerState){
        let samplerState = this.model.state.desired.samplerState === 'active';
        $(this.compiledTemplate).find("#sensorStateCheckBox").prop('checked', samplerState);
    }

    $(this.compiledTemplate).find("#sensorStateCheckBox").change(() => {
        let state = $(this.compiledTemplate).find("#sensorStateCheckBox").prop('checked') ? 'active': 'inactive';
        this.model.updateDesiredState({
            samplerState: state
        });
        if(this.onStateChange && typeof this.onStateChange === 'function'){
            this.onStateChange();
        }
    });
```

```
var ctx = document.getElementById("myChart").getContext('2d');
var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        datasets:[
            {
                label: "Temperature",
                data: this.model.timeseries.temperature,
                fill: false,
                backgroundColor: "#FF0000",
                borderColor: "#FF0000",
            }, {
                label: "Humidity",
                data: this.model.timeseries.humidity,
                fill: false,
                backgroundColor: "#0000FF",
                borderColor: "#0000FF",
            }
        ]

    },
```

```
    options: {
        responsive: true,
        scales: {
            yAxes: [{
                display: true,
                type: 'linear',
                scaleLabel: {
                    display:true,
                    labelString: 'Value'
                },
                ticks: {
                    suggestedMax: 35,
                    suggestedMin: 0,
                }
            }],
            xAxes: [{
                type: 'time',
                distribution: 'series',
                display:true,
                scaleLabel: {
                    display: true,
                    labelString: 'Date'
                }
            }]
        }
    }
});
```

For this sensor we are using a graph to display the changes in data over time and this requires us to setup the chart. This is done using the figures above to display and update the data accordingly with a fixed scale on the y axes for displaying the current temperature with a continuously updating x axes scale with the time for each update.

The model will also send a request to the AWS Shadow telling it that the state has changed and to update the data stored in the S3 bucket. This is done by converting the data into a JSON stream and sending the data across using the MQTT protocol.

This is important because shadow is what keeps track of the state of the sensor and allows us to stop the sensor from transmitting data using the web page and the start/stop button.

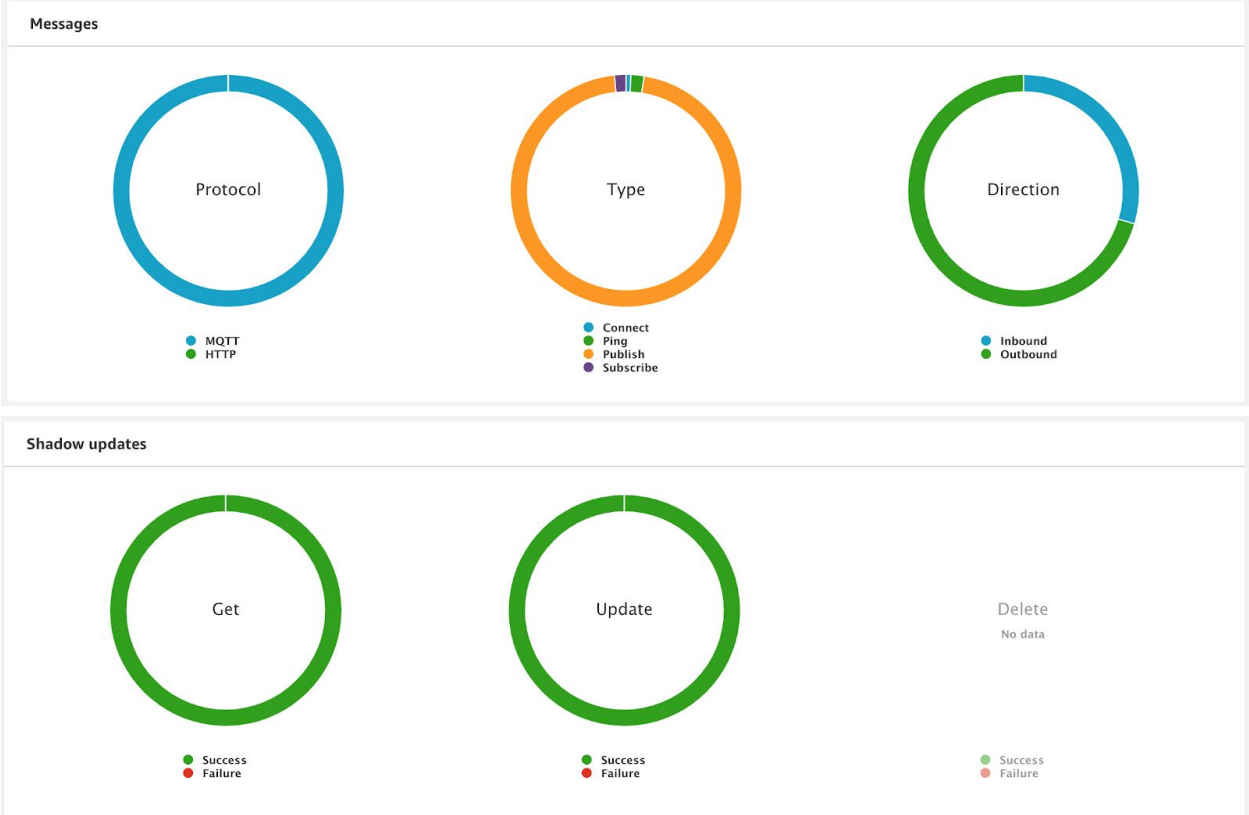
```
// MQTT message handler
sdk.client.on('message', (topic, data) => {
   //TODO: Filter message based on the deviceType and update component appropriately
    let _topic = new Topic(topic);
    let payload = JSON.parse(data.toString());

    if(_topic.type  === 'shadow'){
        console.log("client:", JSON.stringify(payload.state));
        if(payload.state.reported){
            componentManager.components[_topic.deviceId].model.updateReportedState(payload.state.reported);
        }
        if(payload.state.desired){
            componentManager.components[_topic.deviceId].model.updateDesiredState(payload.state.desired);
        }
    }

});
```

AWS and S3 Bucket

Once the data is sent over to Amazon Web Services, S3 Bucket handles all of the deployment for our data and will package and deploy the site. S3 Bucket is a simple service that provides object storage through a web interface, and when linked to the other AWS tools can provide a very powerful web platform. It is used to host our web page in order for us to see the data that the pi is sending. We use Shadow on AWS that will keep track of the JSON object that stores the active state of the sensor and the current temperature and humidity. This Shadow object is updated every 2 seconds and displayed on the page with the current sensor data. AWS allows for many security options making it a very popular tool for developers.

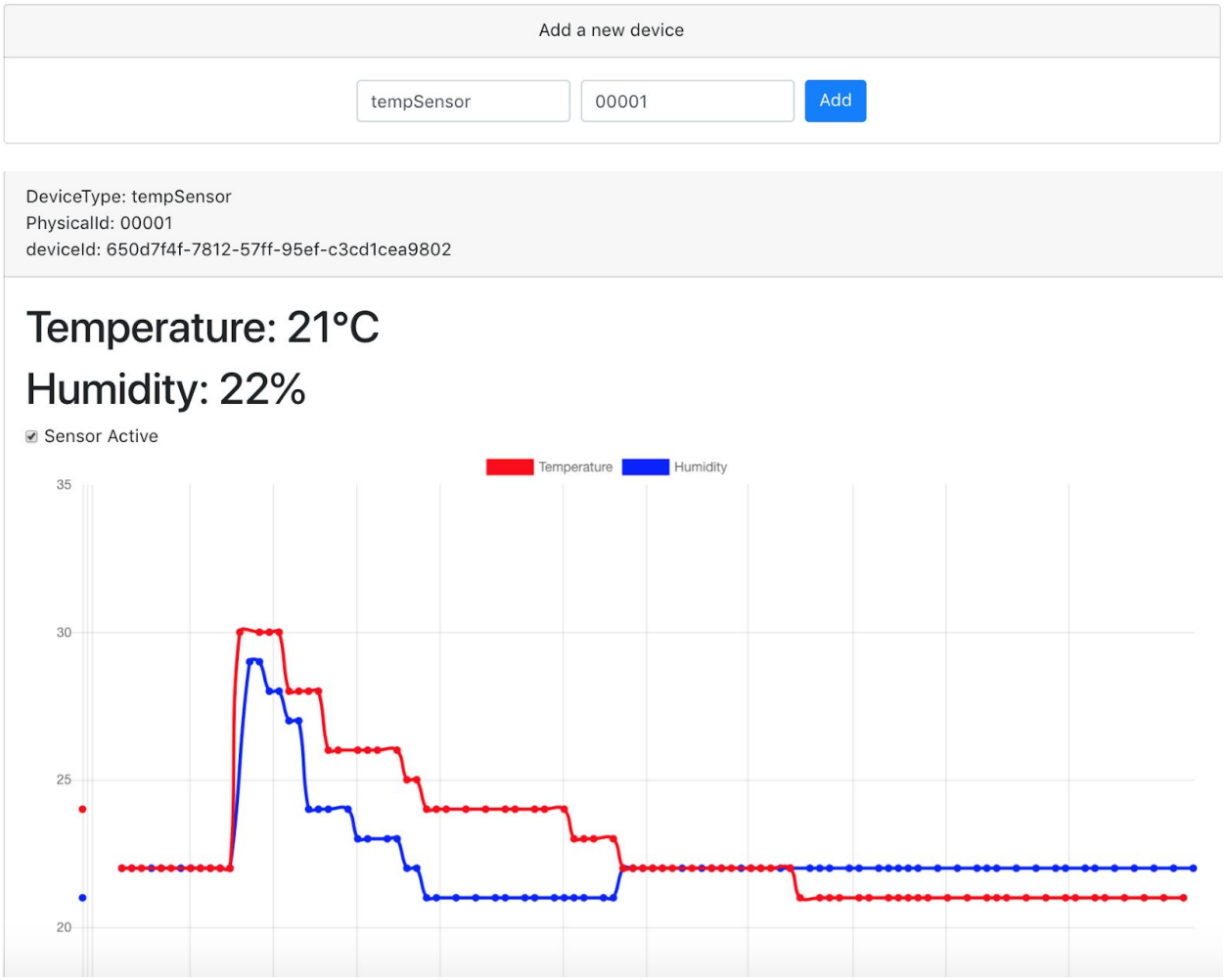


Front End Application

Our front end application is written in JavaScript using Webpack framework. This application as mentioned before utilizes a component style allowing for dynamically adjusting the page as more items are added.

The component structure allows the devices that are dynamically added to be attached and removed using a easy to use device card that will create a new card with the correct sensor display based on the name of the sensor.

The webpage features a start/stop button that can remotely connect or disconnect the sensor from AWS allowing for a seamless use of the page without needing to mess with the Pi client.



Bibliography

Amazon IoT directory

https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

MQTT

https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

Shadows

https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html

https://docs.aws.amazon.com/iot/latest/developerguide/using-device-shadows.html

Helpful Raspberry Pi sensor tutorial

https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp

Raspberry Pi Setup

https://thejackalofjavascript.com/getting-started-raspberry-pi-node-js/

https://www.raspberrypi.org/help/noobs-setup/2/

https://thepi.io/how-to-install-raspbian-on-the-raspberry-pi/

https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html

Our Project

https://trello.com/cs4360/home

https://docs.google.com/presentation/d/1UCq72PSr2_-q7QVv0EQIFGf0pkvkjIaCX8QLrvjdqcI/edit?usp=sharing

https://docs.google.com/document/d/1IoSiRngUIoMUghNniKwwm639y7Rl0hJoBdnvtRTgHqU/edit?usp=sharing

https://s3.amazonaws.com/iot-interface/index.html

GitHub

https://github.com/hamzakhokhar/mqtt-client-interface

https://github.com/hamzakhokhar/iot-web-interface

https://github.com/hamzakhokhar/pi-client