

DT582D: Internet of Things

Description of Lab Tasks and Lab Environment

(by Qinghua Wang)

Table of Contents

Lab Equipments	1
Cloud Infrastructure	1
Task Description	2
Task Implementation	2
Groups	3
Lab Network	3
Appendix 1: Node.js	
Appendix 2: Camera module on Pi	
Appendix 3: Control HS100 with Pi	
Appendix 4: Test Nordic Thingy52 on Pi	
Appendix 5: PIR sensor on Pi	
Appendix 6: Firebase	
Appendix 7: Enable Web Bluetooth API in Chrome	

Lab Equipments:

PC or laptop	x1 per student (<i>not provided by the school</i>)
Raspberry Pi	x1 per group
Monitor, HDMI cable, Keyboard and Mouse (accessory to Pi)	
	x1 per group
Nordic Thingy52 (sensors)	x1 per group
TP-link HS100 smart plug	x1 per group

-----The following equipment are shared -----

Philips Hue Light

Netio4 smart plug

Bluetooth 4.0 adapter

ASUS WiFi Router

Touch Screen (accessory to Pi)

Camera (accessory to Pi)

PIR sensor (accessory to Pi)

Cloud Infrastructure:

Google Firebase will be used in this lab. A google account is required.

IFTTT will also be used. An account can be easily registered.

Task Description:

The task is to build a smart house demonstration of the Internet of Things. In this project, you get a set of sensors (e.g. Nordic Thingy52 has integrated 52 different types of sensors), and you need to use the data from the sensors to aid decision making in the smart house implementation. Situation awareness, command, and control should be able to be performed both locally and remotely.

Task Implementation:

1. Turn on/off a smart plug wirelessly – A taste of IoT
Description: using the button at the Nordic Thingy52 to turn on/off a HS100 smart plug.
Implementation:
Step 1: Install Node.js at Pi
Step 2: Connect Nordic Thingy52 to Pi and read the sensor data. In this case, the data is a button press.
Step 3: Connect your Pi to HS100 using the example code from the Internet.
Step 4: Command HS100 through Pi according to the button press event.
2. Turn on/off a smart plug remotely via IFTTT
Description: you need to report a sensor event to IFTTT, and use the Webhooks module in IFTTT to command a remote device
In this task, you can choose to use the Netio4 plug instead of a HS100, as Netio 4 allows direct access from the Internet while the control of HS100 must be done via a local gateway (e.g. Pi).
3. Web access on sensor data
Description: Build a local web server at your Pi (e.g. using Node.js). Display all dynamic sensor data on a web page. Use SSH for remote access of the web page. You can refer to the demo web implementation of the Nordic Thingy52.
4. Interactive web page
Description: Extend the web page implementation from task 3 with the functions of command & control. For example, it should allow a user to turn on/off a smart

plug from the web page. Connect a touch screen module to your Pi to allow users to command & control via the touch screen.

5. Interactive web via the cloud

Description: Store sensor data to Google Firebase. Host your web page in the google cloud. Use Cloud functions to automatically update the real-time contents in the web page. An expectation is to display graphics instead of numbers pretty much like what you see in the Nordic Thingy52 app. The web page should also be interactive.

6. An automated IoT implementation – Part I

Description:

In a smart room, automatically turn on the element (via a smart plug) when the temperature goes below 22 Celsius. Report the event to a local web server. Meanwhile, take a video clip (using the Pi camera) as a proof that the action has been taken. Update the video clip to the web.

Reference: a web app demo from Nordic:

<https://github.com/NordicPlayground/webapp-nordic-thingy> You can implement this solution in together with SSH and remote desktop (using VNC server) for remote access.

7. An automated IoT implementation – Part II

Description:

In a smart room, automatically turn on a Philips Hue light when a person comes in. A person can be detected using a PIR sensor. Report the event to the Google Firebase. Meanwhile, take a video clip (using the Pi camera) as a proof that the action has been taken. Update the video clip to the cloud hosted web.

Reference: a demo implementation using Amazon AWS cloud:

<https://github.com/Klika-Tech/nordic-aws-iot-demo#back-end>

Groups:

Groups shall be formed during the first lab session.

Some of the lab activities shall be performed by all groups (in particular tasks 1-3).

The teachers may distribute the other tasks among the groups.

Lab Network: IoT_Lab

When you implement the labs, it is recommended that you connect to our own lab SSID (i.e. IoT_Lab) for all WiFi connections (including Pi, TP link smart plugs, etc.). For the TP link smart plug, only 2.4 GHz WLAN works.

The password to connect to the SSID IoT_Lab is: HKR_IoT_Labb.

After you have connected to the same SSID, you can send and receive signals from those smart IoT devices which should be configured to be connected to the same local area network.

When you connect to the IoT_Lab, most of your network access will be limited to the local area network. But you will be able to perform some internet access, such as visiting IFTTT, Firebase, Google, GitHub which are needed for this lab.

If you want to access other websites, you have to switch to Eduroam or HKR Guest for internet access, and then switch back to IoT_Lab to do the lab tasks.

The following are our network configurations.

LAN Gateway/Firewall: 192.168.230.1

Switch/DHCP server: 192.168.230.2

Pi Web/SSH/VNC Server: 192.168.230.100

HS100-01 smart plug: 192.168.230.201

HS110-02 smart plug: 192.168.230.202 (power consumption can be read)

HS100-03 smart plug: 192.168.230.203

HS110-04 smart plug: 192.168.230.204 (power consumption can be read)

Netio 4 smart plug: 192.168.230.210 (can be visited directly from the internet)

External IP (Firewall): 194.47.34.210

Port-Forwarding on the firewall:

```
194.47.34.210:8080 ----> 192.168.230.100:80
194.47.34.210:4443 ----> 192.168.230.100:443
194.47.34.210:8081 ----> 192.168.230.210:80 (obsolete and disabled)
194.47.34.210:4444 ----> 192.168.230.210:443 (used by Netio4)
194.47.34.210:4445 ----> 192.168.230.101:5000 (used by students' Pi
server for whatever service)
```

The websites that the students can visit are whitelisted in the firewall and they are:

1. Ifttt.com, platform.ifttt.com, maker.ifttt.com
2. Google.com, gmail.com, firebase.google.com, mail.google.com, (xxx).firebaseapp.com, (xxx).firebaseio.com
3. Github.com

In addition, the teacher can have VPN access (Server: *hkfw-iot.hkr.se*) to the LAN using pre-installed HKR credentials. While in the LAN, it is possible to have SSH and VNC access to Pi servers. Otherwise, SSH and VNC are blocked by the firewall for external access.

Appendix 1: Node.js

<https://nodejs.org>

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

Appendix 2: Camera module on Pi

<https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>

What you will make

Get started with the Raspberry Pi Camera Module, using Python and picamera. You'll take still pictures, record video, and apply image effects.

What you will learn

By following the 'Getting started with picamera' resource, you will learn:

- How to connect the Camera Module to the Raspberry Pi
- How to use Python to control the Camera Module
- How to use `start_preview()` and `stop_preview()` to control the camera preview
- How to take still pictures with `capture()`
- How to record video with `start_recording()` and `stop_recording()`
- How to play back video with `omxplayer`
- How to alter the brightness and contrast
- How to apply image effects and exposure modes

This resource covers elements from the following strands of the [Raspberry Pi Digital Making Curriculum](#):

- [Use basic programming constructs to create simple programs](#)
- [Use basic digital, analogue, and electromechanical components](#)

What you will need

Hardware

- Raspberry Pi Camera Module

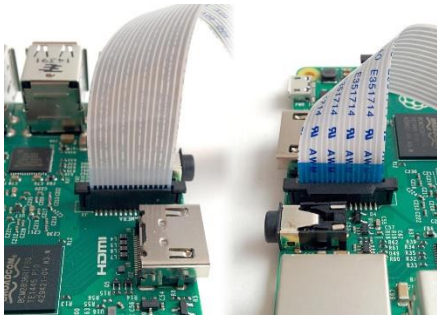
Getting started with picamera

The Camera Module is a great accessory for the Raspberry Pi, allowing users to take still pictures and record video in full HD.

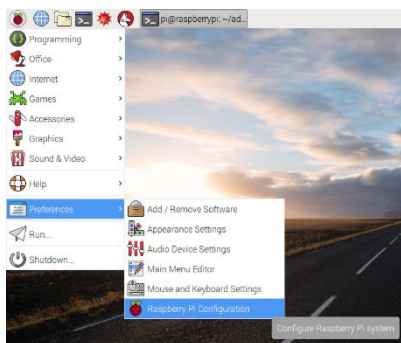
Connect the Camera Module

First of all, with the Pi switched off, you'll need to connect the Camera Module to the Raspberry Pi's camera port, then start up the Pi and ensure the software is enabled.

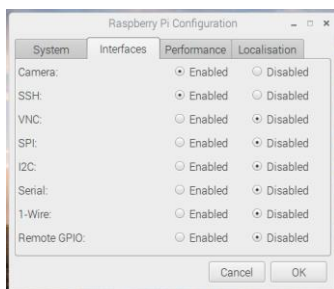
- Locate the camera port and connect the camera:



- Start up the Pi.
- Open the Raspberry Pi Configuration Tool from the main menu:



- Ensure the camera software is enabled:

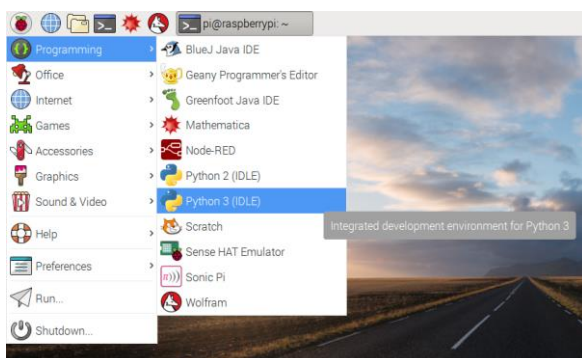


If it's not enabled, enable it and reboot your Pi to begin.

Camera preview

Now your camera is connected and the software is enabled, you can get started by trying out the camera preview.

- Open Python 3 from the main menu:



- Open a new file and save it as `camera.py`. It's important that you do not save it as `picamera.py`.
- Enter the following code:

```

○ from picamera import PiCamera
○ from time import sleep
○
○ camera = PiCamera()
○
○ camera.start_preview()
○ sleep(10)
○ camera.stop_preview()

```

- Save with Ctrl + S and run with F5. The camera preview should be shown for 10 seconds, and then close. Move the camera around to preview what the camera sees.

The live camera preview should fill the screen like so:



Note that the camera preview only works when a monitor is connected to the Pi, so remote access (such as SSH and VNC) will not allow you to see the camera preview

- If your preview was upside-down, you can rotate it with the following code:

```

○ camera.rotation = 180
○ camera.start_preview()
○ sleep(10)
○ camera.stop_preview()

```

You can rotate the image by 90, 180, or 270 degrees, or you can set it to 0 to reset.

- You can alter the transparency of the camera preview by setting an alpha level:

```

○ from picamera import PiCamera
○ from time import sleep
○
○ camera = PiCamera()
○
○ camera.start_preview(alpha=200)
○ sleep(10)
○ camera.stop_preview()

```

alpha can be any value between 0 and 255.

Still pictures

The most common use for the Camera Module is taking still pictures.

- Amend your code to reduce the `sleep` and add a `camera.capture()` line:

```

○ camera.start_preview()
○ sleep(5)
○ camera.capture('/home/pi/Desktop/image.jpg')
○ camera.stop_preview()

```

It's important to sleep for at least 2 seconds before capturing, to give the sensor time to set its light levels.

- Run the code and you'll see the camera preview open for 5 seconds before capturing a still picture. You'll see the preview adjust to a different resolution momentarily as the picture is taken.
- You'll see your photo on the Desktop. Double-click the file icon to open it:



- Now try adding a loop to take five pictures in a row:

```
○ camera.start_preview()
○ for i in range(5):
○     sleep(5)
○     camera.capture('/home/pi/Desktop/image%s.jpg' % i)
○ camera.stop_preview()
```

The variable `i` contains the current iteration number, from 0 to 4, so the images will be saved as `image0.jpg`, `image1.jpg`, and so on.

- Run the code again and hold the camera in position. It will take one picture every five seconds.
- Once the fifth picture is taken, the preview will close. Now look at the images on your Desktop and you'll see five new pictures.

Recording video

Now you've used the camera to take still pictures, you can move on to recording video.

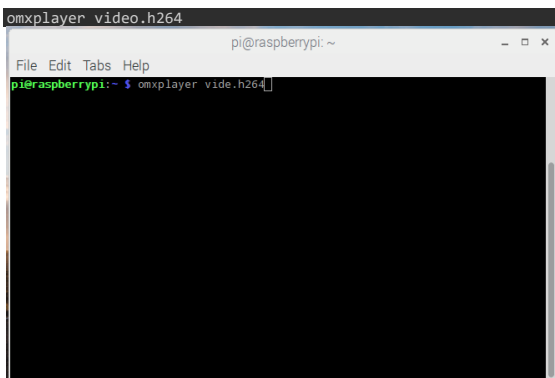
- Amend your code to replace `capture()` with `start_recording()` and `stop_recording()`:

```
○ camera.start_preview()
○ camera.start_recording('/home/pi/video.h264')
○ sleep(10)
○ camera.stop_recording()
○ camera.stop_preview()
```

- Run the code; it will record 10 seconds of video and then close the preview.
- To play the video, you'll need to open a terminal window by clicking the black monitor icon in the taskbar:



- Type the following command and press Enter to play the video:



- The video should play. It may actually play at a faster speed than what has been recorded, due to `omxplayer`'s fast frame rate.

More information: <https://picamera.readthedocs.io/en/release-1.13/>

<https://www.npmjs.com/package/pi-camera>

Appendix 3: Control HS100 with Pi

<https://bloggerbrothers.com/2016/12/19/raspberry-pi-for-controlling-tp-link-power/>

Raspberry PI for controlling TP-Link POWER

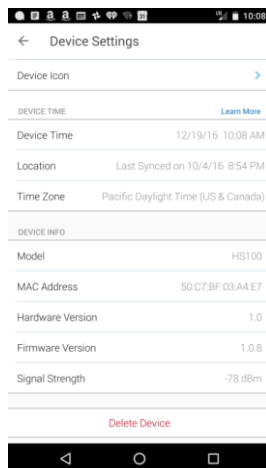
December 19, 2016 [mamacker](#)



The [TP-Link HS100](#) has a nifty little library that allows devices on the same network to control them using a tiny amount of code. This article assumes you know how to setup a raspberry pi, and that you have access to its linux terminal, either through SSH or locally via a keyboard and mouse.

The first step is to make sure your raspberry pi is setup on the same network as the outlet itself. Then, login to your router to discover the TP-Link outlet IP addresses.

It can be difficult to tell **which** device is the outlet. I figured it out by using the “Kasa” app, and looking at the MAC addresses of the outlets here:



Note the MAC address then login to your router, and find the device with the matching MAC. Note its IP address.

Now to control the device, all you need to do is install NodeJS.

Installing NodeJS:

See my other article for that... then come back.

[Installing NodeJS on a Raspberry PI](#)

Installing the hs100-api library...

Then install the library: hs100-api

```
git clone https://github.com/plasticrake/hs100-api.git
```

Then... since you know the IP address, this is all the NodeJs code you need!

```
const Hs100Api = require('./hs100-api');

const client = new Hs100Api.Client();

const lightplug = client.getPlug({host: 'YOUR PLUG IP HERE'});

lightplug.getInfo().then(console.log); // Makes sure the plug is found.

var last = true;

setInterval(() => {

  last = !last;

  lightplug.setPowerState(last);

}, 1000);
```

If everything goes well – that light, the one plugged into the tp-link, should be blinking on and off!

These tp-links can be hard to find at Christmas though – so check out [the same program for the Wemo Outlets!](#)

Now, you'll want to keep that blink program running forever! So check out – [running processes forever on your raspberry pi!](#)

Appendix 4: Test Nordic Thingy52 on Pi

Nordic Thingy:52 Node.js library on Raspberry Pi

Welcome to the Nordic Thingy:52 Node.js library example. Please see <http://www.nordicsemi.com/thingy> for the latest Nordic Thingy:52 news and software releases.

This library is using [noble-device](#) and [noble](#) to handle the Bluetooth connection.

Prerequisites

1. A Raspberry Pi with built in Bluetooth or a Raspberry Pi and a Bluetooth USB dongle.

Note that the internet radio and microphone example might not work to well with the built in Bluetooth adapter due to bandwidth limitations. It is therefore recommended to use an external Bluetooth USB dongle when using this example.

2. The [Raspbian Jessie](#) operating system image.
3. Git, Node.js, npm and noble-device.

Setup Raspbian

1. [Install](#) Raspbian on your Raspberry Pi's SD card using [Etcher](#).

To enable the SSH server on your Pi, make a new file called ssh without any extensions in the boot partition on the SD card.

2. Insert the SD card and power up the Raspberry Pi. Then log in using the default username and password. It's highly recommended to change the default password using the `passwd` command.
3. Update the package manager: `sudo apt-get update`.
4. Add the latest version of Node.js to package manager: `curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -`
5. Install dependencies: `sudo apt-get install bluetooth bluez libbluetooth-dev libudev-dev git nodejs`

Install using GitHub

1. Clone the repository: `git clone https://github.com/NordicPlayground/Nordic-Thingy52-Nodejs.git`
2. Go into the Nordic-Thingy52-Nodejs folder. `cd /Nordic-Thingy52-Nodejs`
3. Install [noble-device](#): `npm install noble-device`
4. Find examples `cd examples`.

Install using npm

1. Install package: `npm install thingy52`
2. Find examples `cd node_modules/thingy52/examples`.

Run the examples

3. Check if the example you want to run has other required npm packages by opening the example. As we can see below the `radio.js` script requires `icecast`, `lame` and `util`.

```
var Thingy = require('../index');
var icecast = require("icecast");
var lame = require("lame");
var util = require('util');
```

4. Install required npm packages: `npm install <package name>`
5. Run example: `sudo node <example_name>.js`

Thingy:52 Node.js Raspberry Pi Interface

The following text are available at:

<https://devzone.nordicsemi.com/b/blog/posts/walkthrough-of-nordic-thingy52-nodejs-raspberry-pi>

This blog post is a follow up to the [Nordic Thingy:52 node.js/Raspberry Pi demos](#) blog post, and will describe how you can run and what to expect from each of the Node.js examples that have been provided in the GitHub repository. It will also describe an additional example of how you can use the Node.js library to post a tweet based on sensor data.

If you are looking for a Python interface instead of Node.js, please have a look at the following blog post for some more

information: <https://devzone.nordicsemi.com/blogs/1162/nordic-thingy52-raspberry-pi-python-interface/>.

Raspberry Pi Setup

To setup the Raspberry Pi so that it matches the setup used for this blog post, please follow the instructions as found on this link: <https://www.raspberrypi.org/help/noobs-setup/2/>, and install the Raspbian operating system when booting the RPi. If you install Raspbian directly onto the SD card as described under "Setup Raspbian" in the Nordic Thingy:52 Node.js Github repository; <https://github.com/NordicSemiconductor/Nordic-Thingy52-Nodejs/blob/master/RASPBERRYPI.md>, you might save some time as the installation from NOOBS takes some time on first boot.

Summary of installing NOOBS: * Format SD card using tool, * download NOOBS offline and network install, * extract folder and drag them onto the SD card, * eject, * insert SD into RPi, * boot up and install Raspbian, * set up keyboard, time and date, etc through Preferences, Raspberry Pi Configuration, * change password, * and I would suggest enabling SSH as this makes developing a lot easier when you don't have a separate monitor/keyboard setup.

When Raspbian has been setup, follow the last steps as explained under "Setup Raspbian" in the Nordic Thingy:52 Node.js github repository; "apt-get update, curl latest version of Node.js, apt-get extra packages".

Node.js Install using GitHub

Once the Raspbian OS is setup, you are ready to get started testing the Node.js examples. In order to make it easier to alter and add our own examples, I suggest following the "Install using Github" approach as listed in the RASPBERRYPI.md; "git clone, cd repo, npm install noble-device, cd examples".

Running the Node.js examples

As mentioned in the instructions on GitHub, some of the examples requires extra npm packages to be installed before they can be run. I will go through running all the examples and the packages needed should be listed as part of that walkthrough.

Please note that to run the examples, you should use the "sudo" command.

BLE Issues on the RPi

If you some time during the example testing experience any problems with the Bluetooth connection on the Raspberry Pi, and the Thingy seems to be working just fine (it still connects to the iPhone Thingy app, for example), the problem might be that the BLE on the RPi has gotten stuck in an unknown state. I have found that the best way to check if the BLE interface on the RPi is still up and running is to simply call "\$ sudo hcitool lescan" and see that it scans and finds the Thingy advertisement (it should

say "LE SCAN ...", and then list all the BLE devices it finds. "Thingy" should be one of them). If this command does not work, the BLE on the RPi has probably stopped working somehow and I suggest restarting the RPi as that was the most reliable approach I found to get the BLE up and running again. After a restart, everything is hopefully working as it should; if not, I suggest googling your error codes to see if there is a fix available. If you do find a solution to this issue, please feel free to post it as a comment and I will include it in this blog post :)

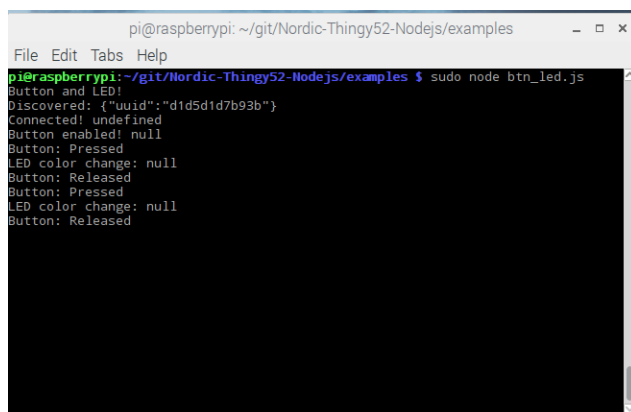
@@ Testing btn_led.js @@

This example does not require installing any additional packages (the noble-device package is needed, but this should have been installed already as part of the "Install using GitHub" steps).

Make sure your Thingy is ON and advertising (blue LED should be breathing). Enter the example folder and run

```
pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $ sudo node btn_led.js
```

The terminal should say "Button and LED!" together with some other text. When the button on top of the Thingy is pressed, the terminal will say "Button: Pressed", and it will change the color of the LED. See screen capture below for terminal output of the example (capture using 'scrot' command on RPi). Hit "Ctrl + c" to exit the execution.



```
pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples
File Edit Tabs Help
pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $ sudo node btn_led.js
Button and LED!
Discovered: {"uuid":"d1d5d1d7b93b"}
Connected! undefined
Button enabled! null
Button: Pressed
LED color change: null
Button: Released
Button: Pressed
LED color change: null
Button: Released
```

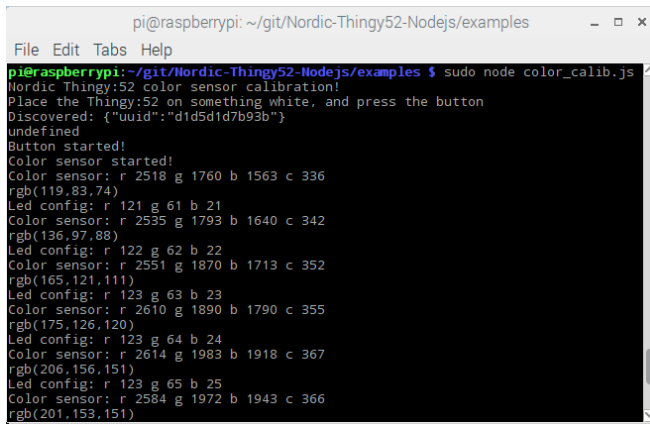
@@ Testing color_calib.js @@

This example does not require installing any additional packages to be installed.

Enter the example folder and run by typing the following

```
$ sudo node color_calib.js
```

The terminal will tell you to place the Thingy on something white and then press the button to calibrate the sensor. Once this is done, the Thingy will start pushing color sensor readings to the RPi and they can be seen in the terminal window. Screen capture of the terminal is added below.

A terminal window titled 'pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples' with a menu bar (File, Edit, Tabs, Help). The prompt is 'pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples \$'. The command 'sudo node color_calib.js' has been executed. The output shows instructions to place the Thingy on something white and press the button. It then displays a series of color sensor readings (r, g, b values) and LED configurations (r, g, b values) as the sensor is calibrated.

```
pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples $ sudo node color_calib.js
Nordic Thingy:52 color sensor calibration!
Place the Thingy:52 on something white, and press the button
Discovered: {"uuid":"d1d5d1d7b93b"}
undefined
Button started!
Color sensor started!
Color sensor: r 2518 g 1760 b 1563 c 336
rgb(119,83,74)
Led config: r 121 g 61 b 21
Color sensor: r 2535 g 1793 b 1640 c 342
rgb(136,97,88)
Led config: r 122 g 62 b 22
Color sensor: r 2551 g 1870 b 1713 c 352
rgb(165,121,111)
Led config: r 123 g 63 b 23
Color sensor: r 2610 g 1890 b 1790 c 355
rgb(175,126,120)
Led config: r 123 g 64 b 24
Color sensor: r 2614 g 1983 b 1918 c 367
rgb(206,156,151)
Led config: r 123 g 65 b 25
Color sensor: r 2584 g 1972 b 1943 c 366
rgb(201,153,151)
```

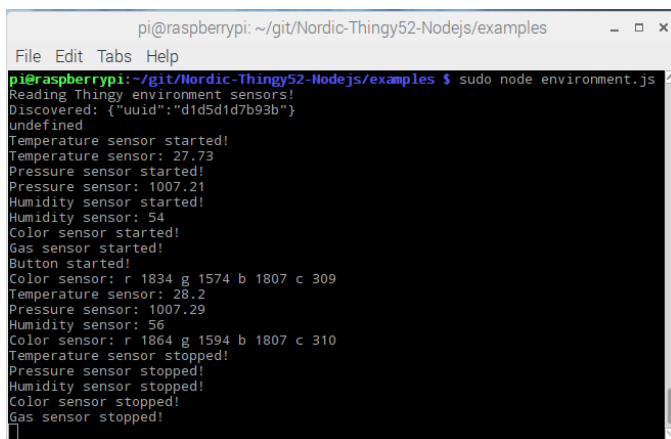
@@@ Testing environment.js @@@

This example does not require installing any additional packages to be installed.

Enter the example folder and run by typing the following

```
$ sudo node environment.js
```

The terminal will print "Reading Thingy environment sensors!", and then start printing sensor data. If you click the button on the Thingy it will stop reading and sending the data until the button is pressed again. Screen capture of the terminal is added below.

A terminal window titled 'pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples' with a menu bar (File, Edit, Tabs, Help). The prompt is 'pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples \$'. The command 'sudo node environment.js' has been executed. The output shows 'Reading Thingy environment sensors!' followed by a series of sensor readings (Temperature, Pressure, Humidity, Color, Gas) and their corresponding values. It also shows the sensor data stopping and then starting again after a button press.

```
pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples $ sudo node environment.js
Reading Thingy environment sensors!
Discovered: {"uuid":"d1d5d1d7b93b"}
undefined
Temperature sensor started!
Temperature sensor: 27.73
Pressure sensor started!
Pressure sensor: 1007.21
Humidity sensor started!
Humidity sensor: 54
Color sensor started!
Gas sensor started!
Button started!
Color sensor: r 1834 g 1574 b 1807 c 309
Temperature sensor: 28.2
Pressure sensor: 1007.29
Humidity sensor: 56
Color sensor: r 1864 g 1594 b 1807 c 310
Temperature sensor stopped!
Pressure sensor stopped!
Humidity sensor stopped!
Color sensor stopped!
Gas sensor stopped!
```

@@ Testing microphone.js @@

This example requires installing the "speaker" package using npm. In the GitHub repository there is a description on how to set up the Raspberry Pi running Raspbian and Google Assistant using the Thingy:52 as microphone input:

https://github.com/NordicSemiconductor/Nordic-Thingy52-Nodejs/blob/master/GOOGLE_ASSISTANT.md. As the speaker package will need to be patched, I suggest looking into this description for how that can be done/tested.

@@ Testing radio.js @@

Testing the radio.js example on the Raspberry Pi with the built-in BLE radio will not yield good quality audio, so this is not recommended and you should have a BLE USB Dongle if you plan to make any applications similar to this. Regardless, if you just want to make sure it works, testing the example requires 'icecast', 'lame', and 'util' packages which are installed by typing "\$ npm install [package name]".

Enter the example folder and install the three packages with the command as described above. Then run the example by typing

```
$ sudo node radio.js
```

You should now be hearing choppy audio coming from the Thingy:52. "Ctrl + c" to exit. The reason the audio is choppy is because the BLE connection between the Thingy and the RPi is not optimal. Using a BLE dongle should solve this issue.

@@ Testing firebase.js @@

This example requires the packages 'util', and 'firebase' to be installed using the 'npm install' command as described earlier.

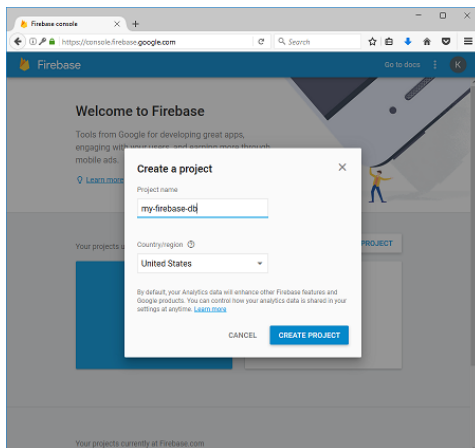
This example also requires you to have a firebase/google account in order to receive the sensor data.

If you have never heard of Firebase before, here is a quick summary: "Firebase is Google's mobile platform that helps you quickly develop high-quality apps and grow your business". It will handle the backend infrastructure, user engagement, and monetization, while you build the app for your users. In this example we will simply use it as a database for the sensor data and show how the data can be passed to the server and displayed there.

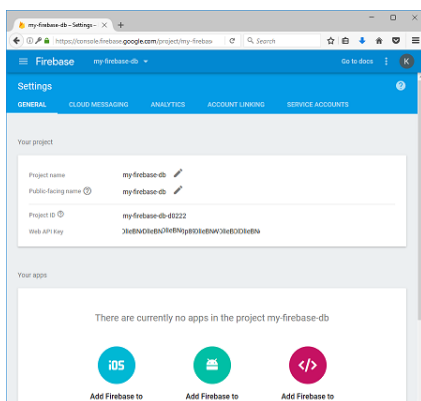
Setting up the Firebase database

There are several tutorials on setting up Firebase and getting started with it; such as <https://rominirani.com/firebase-iot-tutorial-46203a92f869>. I will describe the approach I took below which should be sufficient to get this example up and running.

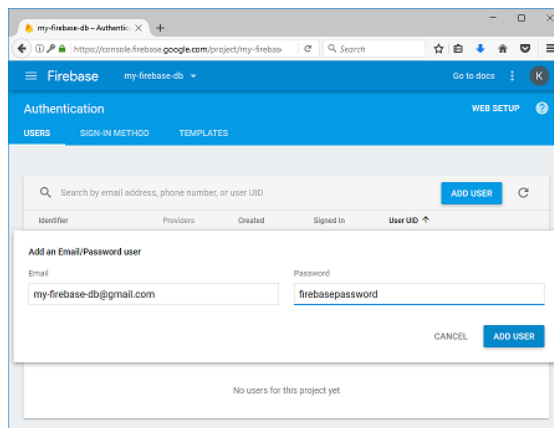
In order to get started with the Firebase database, go to <https://console.firebase.google.com> and log into your Google account if you are not already logged in. Click the "Add project" button, and you will be prompted to "Create a project". Choose a fitting project name, such as "my-firebase-db" and select your country/region.



Once the project has been created, go to "Project settings" through the gear icon in the menu. Take note of the "Project ID" as well as the "Web API Key", as we will need these later on.



To test this example, you can set up an authentication method. Go to "Authentication" in the menu, then click on "SIGN-IN METHOD". Enable "Email/Password" sign-in, and then click the "USERS" tab and "ADD USER". Add a user with any random email address and password and take note of this as we will use it in the example later on.



Now that we have set up the database and user access, we can go ahead and alter the `firebase.js` file to include the necessary information for publishing to the database. Edit the two variables in `'firebase_login'` to use the user login email and password that we just created. Then edit the four variables in `'firebase_config'` to include the project ID and Web API Key that we found in the firebase project settings.

```
var firebase_login = {  
    email : "my-firebase-db@gmail.com",  
    pass : "firebasepassword"  
};  
  
var firebase_config = {  
    apiKey: "AIzaAIZAaIzaAIzaAIzaAIza-AIzaAIzaAIzaAIza",  
    authDomain: "my-firebase-db-d0222.firebaseio.com",  
    databaseURL: "https://my-firebase-db-d0222.firebaseio.com",  
    storageBucket: "my-firebase-db-d0222.appspot.com",  
};
```

Running the example

Once this is all done, run the example by typing

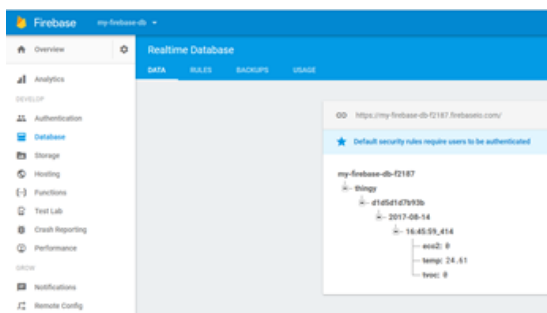
```
$ sudo node firebase.js
```

The terminal should look as shown below. Please note that it might take some time before the first data has been published to the database (The line with date and time shows when it publishes; it might take 1 second, or 1-2 minutes based on your gas sensor settings). To stop the execution, press "Ctrl + c" and you should see Firebase sign out as well as the gas sensor readings being stopped.

```
pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples
File Edit Tabs Help

pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $ sudo node firebase.js
Firebase Thingy gas sensor!
Firebase signing in as: my-firebase-db@gmail.com
Discovered: {"uuid":"d1d5d1d7b93b"}
Connected!
Gas sensor configured!
Gas sensor started!
Temperature sensor started!
Gas sensor: eCO2 0 - TVOC 0
2017-08-14/16:45:59_414
^CFirebase signing out :
Firebase signed out!
Gas sensor stopped!
Disconnected:
pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $
```

If you now open your Firebase console webpage again and click the Database option in the menu, you should see that some data has been added to the database along with a date and time for when the data was posted; please see screenshot below. This concludes the explanation of this example and shows how you would get data passed into your Firebase server.



@@ Testing ifttt_gas.js @@

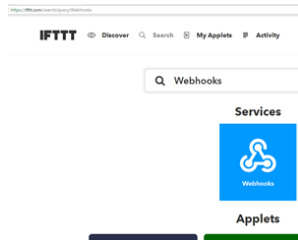
This example requires the packages 'util', and 'request' to be installed using the 'npm install' command as described earlier. This example also requires you to have a IFTTT account in order to receive and take an action based on the event.

If you have never used IFTTT before, here is a brief summary: "IFTTT is a free platform that helps you do more with all your apps and devices". IFTTT stands for 'If This Then That'. It is a free web-based service that people use to create chains of simple conditional statements, called applets. For example, an applet may send an e-mail message to someone if a user tweets using a specific hashtag. Reference, [IFTTT Wiki](#).

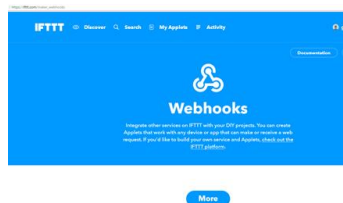
Setting up the IFTTT account and the event trigger

If you have never used IFTTT before, please read through the following steps to set up the required Webhook. If you know how to use IFTTT, set up the Webhook to receive an event (you will pass the name of the event to the Node.js application as an argument) and take an action based on that event; also note down your Webhook key (this key is passed into the Node.js application as an argument).

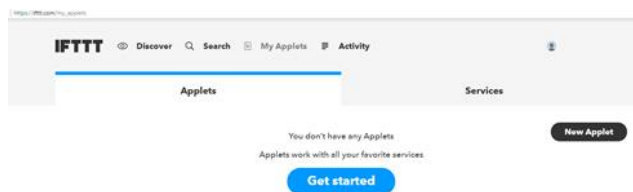
Go to <https://ifttt.com/> and Sign up, or log into your existing IFTTT account. Once logged in, click the "Search" tab at the top and search for "Webhooks". Click the blue "Webhooks" button under Services, and then click Connect to connect this service to your account.



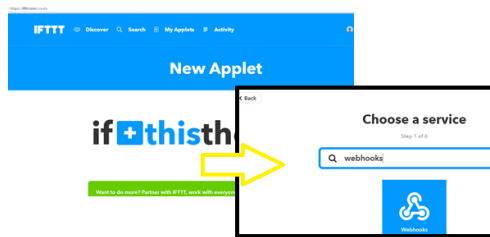
Once the Service has been connected to your account, click the Documentation button up right on the screen (see screenshot below). This should open a new tab that says: "Your key is: abcdefghijk_123456789" (your key will of course be different then mine). Take note of this Key as we will use it later on when calling the Node.js application. You can now close that Documentation tab; if you ever need to find it again, click the "My Applets" tab at the top of your IFTTT homepage, click Services, and search for Webhooks, then click the blue Webhooks button to get back to the homepage of the Webhooks service.



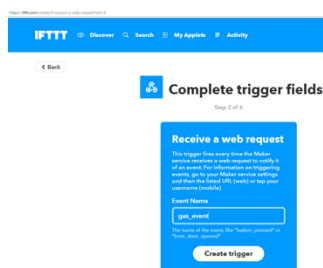
Next step is to activate an Applet that is triggered from the event that our RPi will send. Click "My Applets" and then "New Applet".



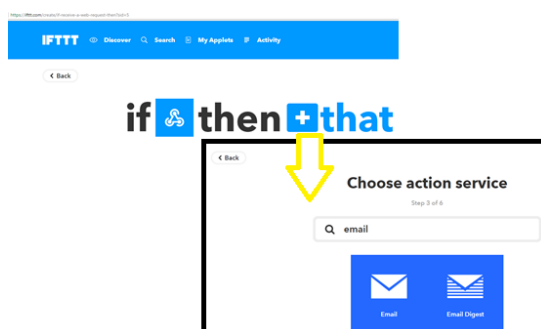
On the screen that appears, click the blue "+this" icon. This will take you to a page where you can search for services. Search for "Webhooks", and select it.



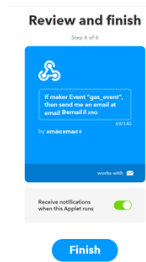
When you click the Webhooks service it will take you to a page that says Choose trigger (it will say “Connect Webhooks” if this is your first time using the Webhooks service and you did not connect it earlier from the Search menu. Click “Connect” and you should be set). Click the “Receive a web request” button, and you will be prompted to enter the Event Name that should trigger the action we are going to add later. Enter the name “gas_event” and click “Create trigger”. This event name will later be passed to the Node.js application as an argument.



We are now done setting up the trigger for this Applet. Next, click the blue “+that” button on the screen that has appeared. When you click this, you will be prompted to “Choose action service”. To make this simple, search for email and click the blue “Email” button to select this action.



On the next page, click the blue “Send me an email” button and leave it at the default entries, or change it to something funny, then click “Create Action”. On the next page, make sure the email address is correct, activate the “Receive notifications when this Applet runs” option, then click finish. Your Applet is not set up and ready to be triggered by the event that we will send from the RPi.



Running the example

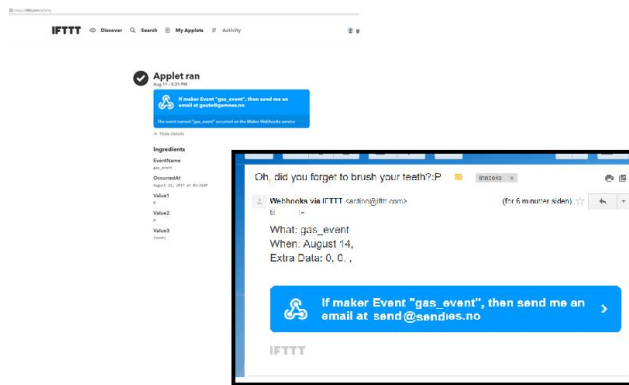
To run the example on the Raspberry Pi, we need the event name that we set up on IFTTT, as well as the Webhooks Key that we read out from the Webhooks Documentation page. Run the example by typing as described below (remember to replace the -e, and -k arguments with the corresponding name and key that you have for your setup). Note: This is where my BLE interface got stuck the first time and I had to restart the Raspberry Pi as mentioned at the top of this blog post. If you run into the same problems, try restarting the RPi and see if that solves the issue.

```
$ sudo node ifttt_gas.js -e gas_event -k abcdefghijk_123456789
```

You should not see the terminal print "IFTTT Thingy gas sensor!", ..., "Gas sensor started!". The application is then ready to send the event whenever the gas sensor is sent from the Thingy to the RPi. To make sure the Thingy is configured to send the gas data, you might want to connect via the Thingy app on an iPhone or Android and make sure the settings of the gas sensor is set so that it sends gas data every 10 seconds. If not, you might have to wait 1 minute for the gas sensor data to be passed over the BLE connection and for anything to appear in the terminal.

```
pi@raspberrypi: ~/git/Nordic-Thingy52-Nodejs/examples
File Edit Tabs Help
pi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $ sudo node ifttt_gas.js -e gas_event -k abcdefghijk_123456789
IFTTT Thingy gas sensor!
Discovered: {"uuid":"d1d5d1d7a93b"}
Connected!
Gas sensor configured!
Gas sensor started!
Gas sensor: eCO2 0 - TVOC 0
Gas data pushed to IFTTT
^Cpi@raspberrypi:~/git/Nordic-Thingy52-Nodejs/examples $ scrot
```

To make sure the event triggered the webhook on IFTTT, go to the IFTTT webpage, and click the "Activity" tab near the top. This will take you to an activity log and if you enabled the notifications on the webhook as mentioned earlier, you should see the log "Applet ran" printed on the screen as well as some details on what it did. You can then check your email and you should have gotten an email specifying what you wrote when the webhook was created. See image below.



@@ Testing motion.js @@

This examples requires packages 'console.table' to be installed.

Enter the example folder and run by typing the following

```
$ sudo node motion.js
```

The terminal will print "Reading Thingy Motion sensors!", and then start printing sensor data. If you click the button on the Thingy, it will stop reading all sensors. If clicked again, it will enable all motion sensors and send the data. This might cause disconnect if using the onboard RPi BLE as far as I have seen.

Node.js Twitter Post Based On Sensor Data

As part of a Raspberry Pi Jam in Kirkenes, Norway (<https://twitter.com/BarentsRJam>), I made an example of using the Thingy:52 and the Raspberry Pi to post a message including sensor data on Twitter when the button on the Thingy was pressed.

GitHub Clone

UPDATE: This example has now been included in the master repository, and there is no need to checkout the gamnes branch any longer.

--Note: This is not needed anylonger-- If you would like to test this example on your own, clone the repository

<https://github.com/gamnes/Nordic-Thingy52-Nodejs.git>, or simply copy the https://github.com/gamnes/Nordic-Thingy52-Nodejs/blob/master/examples/twitter_environment.js file and run it after installing the required packages. -

NPM Packages

This example requires the 'twit' package to be installed using npm. The example also requires some Twitter keys to be included in order to make posting possible. **These keys should never be shared online** as it might provide others with access to your twitter account, so please do not include these keys in your repository if you are pushing it online.

Twitter Account Applications Setup

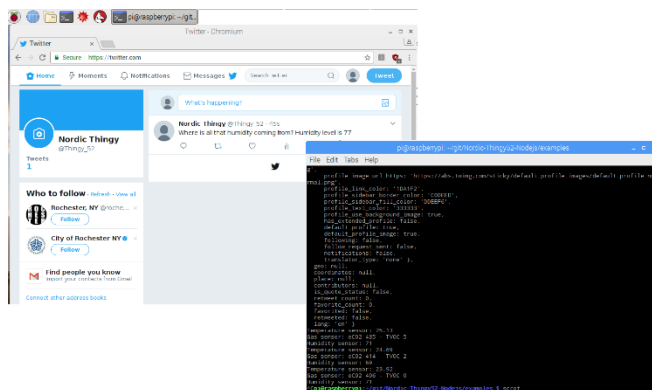
To allow posting twitter message from your RPi, your twitter account needs to be set up to allow applications to post tweets. Please follow this tutorial <https://videlais.com/2015/02/28/how-to-create-a-basic-twitterbot-in-node-js-using-twit/>, and when you are done you should be able to find your 'Consumer Key', 'Consumer Secret', 'Access Token', and 'Access Token Secret' under 'Keys and Access Tokens'. These keys will be used to test this example and they have to be added to the twitter_environment.js file. Summary of steps from tutorial: * Go to apps.twitter.com and register the app; * change permissions from read-only to read and write; * read Consumer Key and Consumer Secret, * generate Access Token and read out this as well as the Access Token Secret. * Take note of all four keys so that they can be included in the example.

Running the Example

The twitter_environment.js example is made such that the four keys noted from the previous step have to be added to the file itself for the example to run. In the file, there are four 'INSERT_KEY' strings that needs to be replaced with the four keys you should have taken note of. Add the corresponding key to the correct variable, and save the file; Then run the example by typing

```
$ sudo node twitter_environment.js
```

The example should run and start printing all the sensor data in the terminal. To post on Twitter, press the button on top of the Thingy and you should see some twitter post logging in the terminal. Based on the sensor readings, your twitter account should now have posted a message including data from one or more of the sensor. See screen capture below as an example.



Appendix 5: PIR sensor on Pi

<https://tutorials-raspberrypi.com/connect-and-control-raspberry-pi-motion-detector-pir/>

Appendix 6: Firebase

<https://firebase.google.com>

Appendix 7: Enable Web Bluetooth API in Chrome

I have successfully tested the Web Bluetooth API in Windows 10 (with Chrome Canary browser), and in Raspberry pi (with Chromium browser). What you might need to do is to enable the feature manually in the browser via typing in: `chrome://flags/#enable-experimental-web-platform-features`

In Windows, you are also required to manually connect to the selected Bluetooth device manually in your operating system settings first, then in the browser.

I also observed that not all features are available when I am using Chrome in Windows. There is also a traffic jam (with packet loss or delay) problem when multiple computers are connected to the same Bluetooth device. You might also need to restart the Bluetooth device manually before your web browser can discover it.

The Nordic Thingy 52 Web app that I have tested is React, you find the code in the following. I have also tested the Polymer version in Raspberry Pi, but it seems that there is currently no support for installing polymer-cli in 32 bits platform (and Pi is 32 bits platform).

Nordic Thingy Web apps: <https://github.com/NordicPlayground/webapp-nordic-thingy>

The following is some code to install the web app **React** for Nordic Thingy52.

Recommendations

- This web app was built using a Web Bluetooth API which aims to make it easier to start developing Web Bluetooth applications using Thingy:52. To find out more about this API, [click here](#).
- Learn about the Web Bluetooth API by reading the [Interact with Bluetooth devices on the Web](#) guide by François Beaufort.
- Learn about React by reading the official [React - getting started](#) guide.
- For an introduction in how to quickly and effortlessly create React apps, visit [Create React App](#).

Prerequisites

- **Node.js** - Install an [active LTS version of Node.js](#) (e.g. v8.11.3). The current version (10.6.0) should work, but is not officially supported.
- **Git** - If you want to clone this repository, you will have to install [Git](#). Alternatively, you can download the repository by clicking "Clone or download", and then "Download ZIP".
- **Google Chrome** - As [Google Chrome](#) is currently the only browser supporting Web Bluetooth, you will need it to use the web app.
- **Web Bluetooth polyfill for Windows 10** - If you are using Windows you will have to install a polyfill to enable Web Bluetooth. A guide with download and setup instructions can be found [here](#).

Installation instructions

1. Clone or download this repository.
2. Make sure you have all prerequisites.
3. Open a command line tool, navigate to the root folder of the repository, and download dependencies by typing:

```
npm i
```

4. To test the project, type:

```
npm start
```

Build the project

From the root folder of the project, in a command line tool, write:

```
npm run build
```

This will create a new *build* folder. Inside is a *bundled* package of the website. The build can now be hosted by any server that is able to serve static files.

Note: If you want to serve the build from a folder other than the root folder, open the package.json file and change the "homepage" field to match your desired path.

After you have built your local server files, you can follow the following instructions to start the server.

Static Server for React

For environments using *Node*, the easiest way to handle this would be to install *serve* and let it handle the rest:

```
npm install -g serve  
serve -s build
```

The last command shown above will serve your static site on the port **5000**. Like many of *serve*'s internal settings, the port can be adjusted using the `-p` or `--port` flags.

Run this command to get a full list of the options available:

```
serve -h
```

More information and other alternatives for the web server service can be found at:

<https://facebook.github.io/create-react-app/docs/deployment>