**How to guide : 3**

Control Pi GPIO pins remotely with Node.JS, Socket.IO, and an express web server

# What are we doing?

1. Creating a local web server using Node.JS and express with a HTML page to display Input/output to the raspberry Pi.
2. Creating a simple electronic circuit on the Pi with a breadboard, an LED (for output) with a 330-ohm resistor, a button (for input), with a 1K resistor, and some jumper wires to connect to the GPIO pins.
3. Using Socket.IO, creating a real time server communication between a Pi(client), a server(host), and the browser HTML page (another client). Whenever the virtual button is toggled on the page, the LED will toggle in real time too without refreshing
4. Adding functionality so whenever a button is pressed on the Pi, the state is past to the server and onto the dynamic HTML page to toggle a virtual check box
5. Deploying this server remotely on Heroku so raspberry Pi GPIO communication can be done from any network
6. Deploying code on GitHub (separate guide possibly)

The circuitry/ application used for this guide is purely for demonstration, but the possibilities are vast. I will be developing this further and using it as my base for Gesture recognition/logging of sensing data.

# Prerequisites

For the following tasks, each highlighted code is directory inputted into the terminal of The raspberry Pi, alternative terminals can be used. This gives us the full dependencies/libraries used for all aspects of the guide.

* Download node.JS onto Pi - pi@raspberrypi:~ $ sudo apt-get install NodeJS -y
* Install Express Web framework - pi@raspberrypi:~ $ sudo npm install express –save
* Install socket.io onto Pi - pi@raspberrypi:~ $ sudo npm install socket.io –save
* Registered Heroku account ( if you want to control your Pi hardware across any network and on any device with a browser and an internet connection.)

All code hosted on <https://github.com/MichaelMcFerran/ControlPiGpioRemotely> Edit client code URL at the top to suit your Heroku app that you are connected too, or comment out and use the line connecting to localhost with correct networking port the server is listening on.

var socket = require("socket.io-client")("https://ledbutton0607.herokuapp.com");

//var socket = require("socket.io-client")("http://localhost:3000"); //may need to change 3000

# How to

Part 1 Getting front end webpage and server communications working locally with socket.io

1. Create a Folder to hold the client and server code , change directory the newly created folder. For example ;

* pi@raspberrypi:~ $ mkdir ledtest09
* pi@raspberrypi:~ $ cd ledtest09

1. Create server within folder with terminal command - express server. Now change directory into server – cd server. Install all the dependencies for server with command – sudo npm install
2. Install socket.io on server that will allow for communications between Pi, server, and HTML front end in real time. Use the command – sudo npm install socketio
3. Using a text editor or inbuilt nano editor within Pi terminal, go to the view folder within server. Mine is found at - /home/pi/ledtest09/server/views
4. Create a new file called index.html.
5. To save time, copy the contents from my GitHub link at <https://github.com/MichaelMcFerran/ControlPiGpioRemotely/blob/master/server/views/index.html>
6. Now go to the servername/public/stylesheets folder within text editor(or and IDE). Mine is at /home/pi/ledtest09/server/public/stylesheets, download and replace the CSS with my file at <https://github.com/MichaelMcFerran/ControlPiGpioRemotely/blob/master/server/public/stylesheets/style.css>
7. Open index.jade from the views folder again, and replace content with - include index.html

This means that index.html will be served at root of server call on browser.

1. Now let us check server runs locally with the follow commands from the server folder within terminal. This is used often when developing/troubleshooting. sudo npm start or sudo node bin/www.

*You may get errors with the port already being in use, simply change the port listening number from within the server/bin/www file. But remember to change back to 3000 before deploying to Heroku. Here is the line where port number can be changed*

*app.set('port', process.env.PORT || 3000); //change from 3000 to work locally to match client*

If you haven’t copied my bin/www from GitHub Repo, then you may have newer version of express server running where equivalent code where port can be changed is ;

*var port = normalizePort(process.env.PORT || '3000');*

*app.set('port', port);*

1. Check code is running by visiting <http://localhost:3011/> from your Pi browser or choice. Or by using your external Pi IP address on any device with a browser on local network. You can use another terminal command to find your external IP by typing ifconfig (wired or wireless/wlan.)

Mine for this guide is located at <http://192.168.86.133:3011/>

1. Above is the current browser front end, where clicking LED toggle doesn’t trigger a Led to turn on. That’s where the client-side code comes in to communicate with server and then the HTML.
2. At any time, you can use ctrl^z to end server from terminal window, freeing up system resources during development
3. Now open the package.json file within server folder and add a line within dependencies to include socket.io if it isn’t already there. "socketio": "\*" then run npm install from server terminal window to install dependencies.
4. Let’s edit the server/bin/www file to start the socket.io connection between server and clients (HTML and Pi). Add this code to the bottom of the file and save.

*var io = require("socket.io").listen(server);*

*io.on("connection", function(socket){*

*console.log("Client Connected");*

*});*

1. To get Browser page setup to listen and emit events to the server with socket.io, open index.html, go to the bottom and make sure the following is present if you didn’t copy my index.html.

*<script src="socket.io/socket.io.js"></script>*

*//Add the following code if it isn’t already there*

*var socket = io.connect('/'); //connects to server and triggers console connected message*

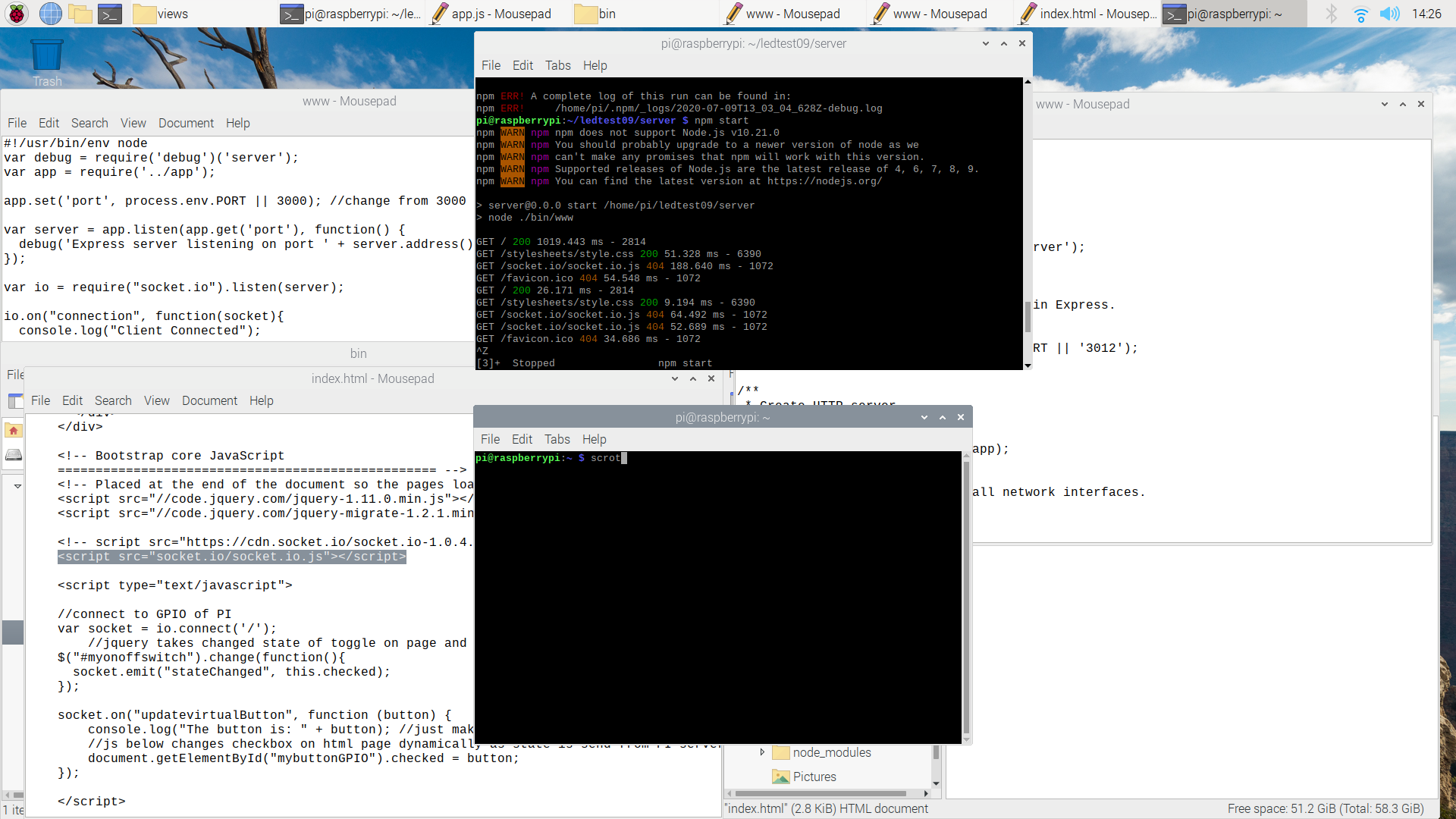
*//jQuery takes changed state of toggle on page and passes state to server*

*$("#myonoffswitch").change(function(){*

*socket.emit("stateChanged", this.checked);*

*});*

When you visit the browser page again with server running, you should get a Client connected message(console.) Now end the server.



1. So now we need to edit bin/www to display/listen to the state of the toggle as it is pressed/changed by the client. New code is the following;

*io.on("connection", function(socket){*

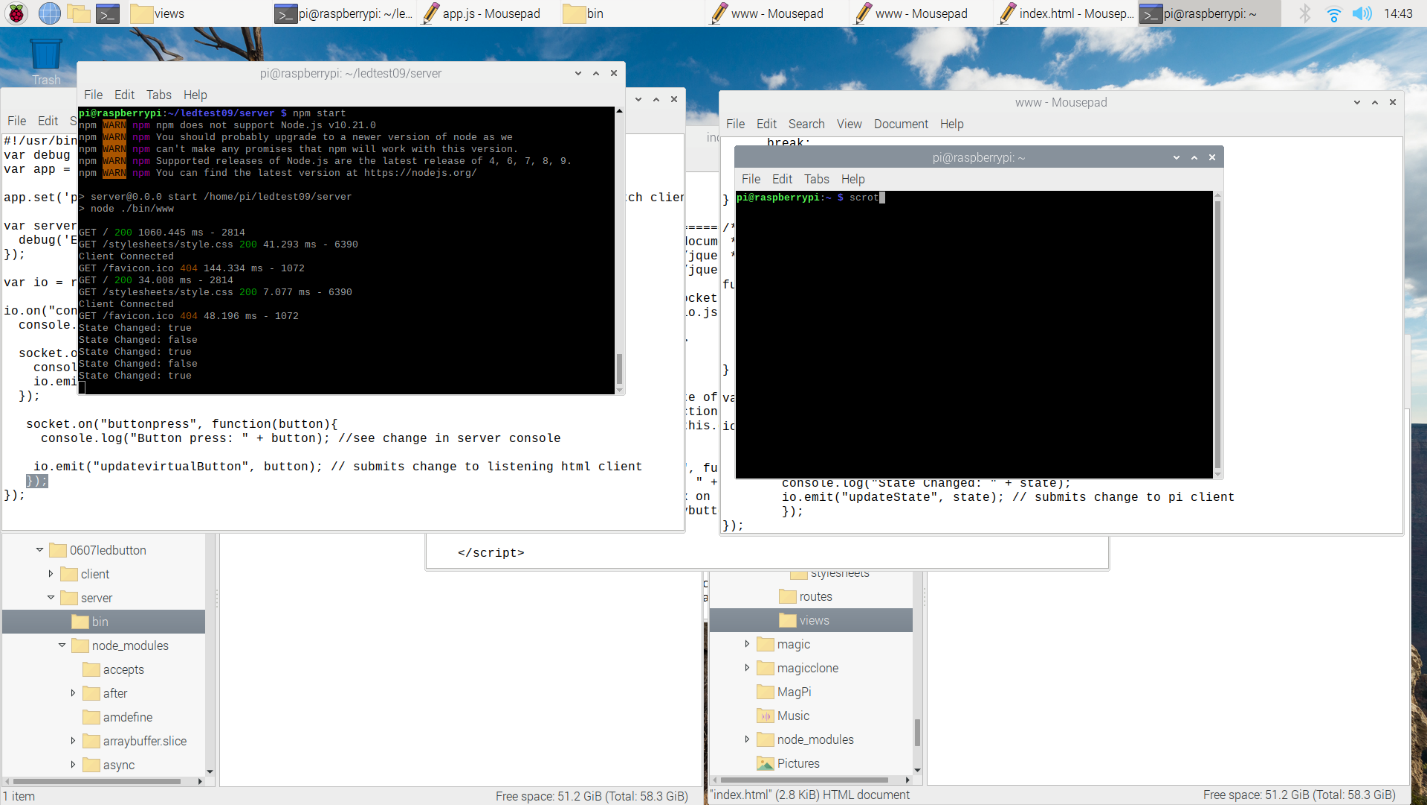
*console.log("Client Connected");*

*socket.on("stateChanged", function(state){*

*console.log("State Changed: " + state);*

*});*

*});*

1. Rerun server(editing port if running into errors, I change mine each time currently 3014.) As the button is pressed on the html side. The server will reflect the change (console illustrates the message has passed to server.)

## Part 2 - Getting Pi client communicating with server and front-end client.

Client app.js can be downloaded from <https://github.com/MichaelMcFerran/ControlPiGpioRemotely/blob/master/client/app.js>

1. End server if you wish (ctrl+z), open a new terminal window create and change directory to client folder from the project folder, then create an app.js that will run the code. Use following commands;

pi@raspberrypi:~ $ cd ledtest09

pi@raspberrypi:~/ledtest09 $ mkdir client

pi@raspberrypi:~/ledtest09 $ cd client

pi@raspberrypi:~/ledtest09/client $ touch app.js

1. Let’s get all of the dependencies out of the way and set up package.json. Type in the following commands. Note: after installing all of these libraries use command sudo reboot to eliminate issues.

* pi@raspberrypi:~/ledtest09/client $ npm init -y (sets up package.json)
* Install rpi-gpio for LED control on PI

pi@raspberrypi:~//ledtest09/client $ sudo npm install rpi-gpio

* Install gpio for Button state reading on Pi

pi@raspberrypi:~ /ledtest09/client $ sudo npm install gpio

* Install ONOFF module - pi@raspberrypi:~ /ledtest09/client $ sudo npm I -s onoff
* Install socket.io client - pi@raspberrypi:~/ledtest09/client $ npm install socket.io-client

1. Now let us edit this app.js file with your chosen text editor to connect to the server while server is running. Note that the port numbers on server/bin/www and client/app.js must match. This code is all you need to connect to server

*var socket = require("socket.io-client")("http://localhost:3014"); //change 3000 to suit www //server*

*socket.on("connect", function () {*

*console.log("Connected to server");*

*});*

1. Run client server to check it is connecting correctly. But first type npm install in console to make sure all dependencies are installed correctly. Use the following code, pi@raspberrypi:~/ledtest09/client $ node app.js

If it connects it will display “connected to server” (server must be running) and server will display “Client Connected”

1. Next we will edit the server/bin/www file to pass on a message to the Pi client (emit to all clients listening to specific event name.) This message is a function from index.html called stateChanged and passes the state of the toggle on the HTML. The server will now emit this to whatever client is listening.

*io.on("connection", function(socket){*

*console.log("Client Connected");*

*socket.on("stateChanged", function(state){*

*console.log("State Changed: " + state);*

*io.emit("updateState", state); // submits change to pi client*

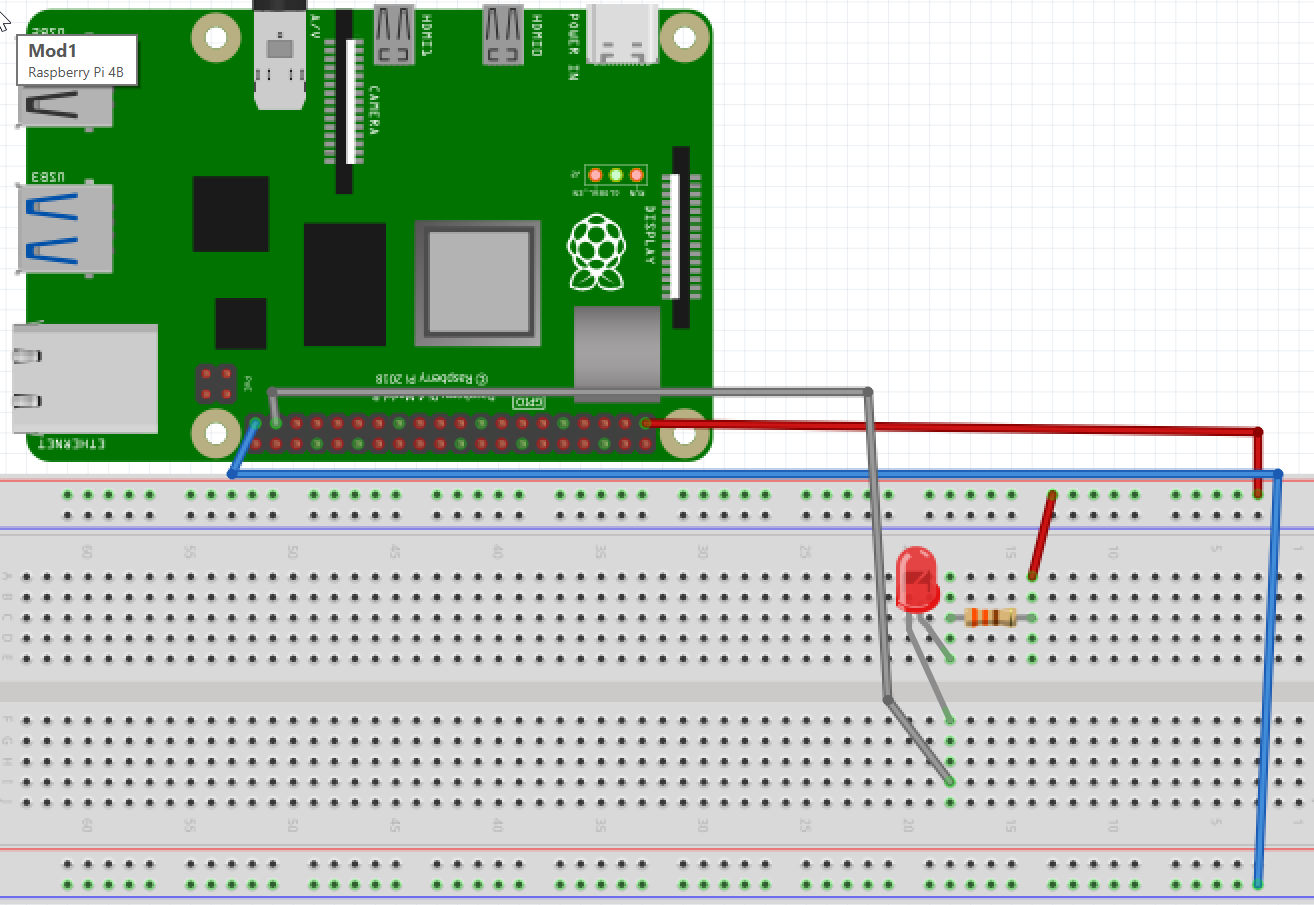
*});*

*});*

## Part 3 Allowing control of the Pi’s GPIO pins and their states. Using socket.io communication to the server and HTML front end.

1. Below is the circuit needed to toggle an LED through the HTML page that is connected to a raspberry PI. You can change the Pin but be careful connecting to constant power pins/ground by mistake and be careful of Pin numbering. The rpi-gpio module uses direct pin numbers not GPIO. LED is connected to pin number 37 which is GPIO 26. Here is a pin diagram for the PI model 4 <https://www.raspberrypi.org/documentation/usage/gpio/>

You will need;

* Raspberry Pi. I use the most recent model 4 4gb.
* Jumper Wires
* 330-ohm resistor
* Breadboard and LED make sure positive(longer side) is connected to positive/red side.

1. Now we will edit the client app.js to take the state change event from server(that was passed by html page) and use it to toggle an LED through use of rpi-gpio library.

*var socket = require("socket.io-client")("http://localhost:3014"); //change 3000 to suit www//*

*var gpio = require("rpi-gpio"); //uses direct pin numbering*

*//needed to control pin state*

*var led =gpio.setup(37, gpio.DIR\_OUT, function(){*

*gpio.write(37, true); // turns led off as in active low config*

*});*

*//this is used for when program ends, turns led off when client not connected*

*process.on("SIGINT", function(){*

*gpio.write(37, true, function(){*

*gpio.destroy(function(){*

*process.exit();*

*});*

*});*

*});*

*socket.on("connect", function () {*

*console.log("Connected to server");*

*//update change anytime there is one*

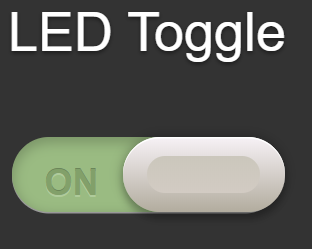
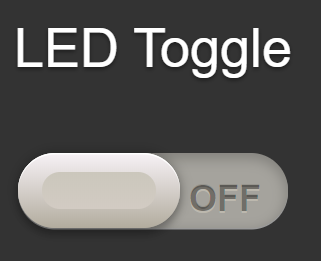
*socket.on("updateState", function (state) {*

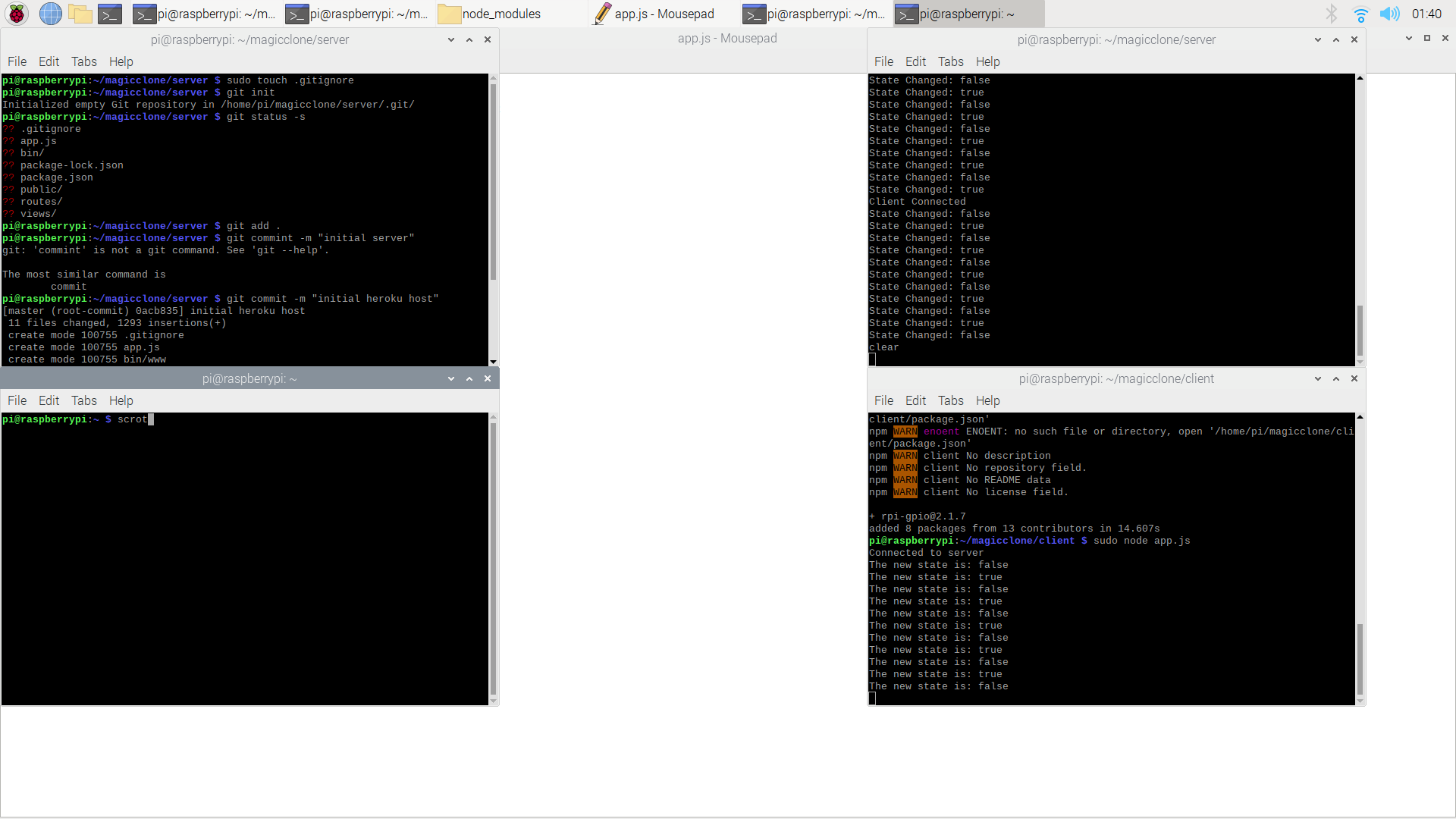
*console.log("The new state is: " + state);*

*gpio.write(37, !state);*

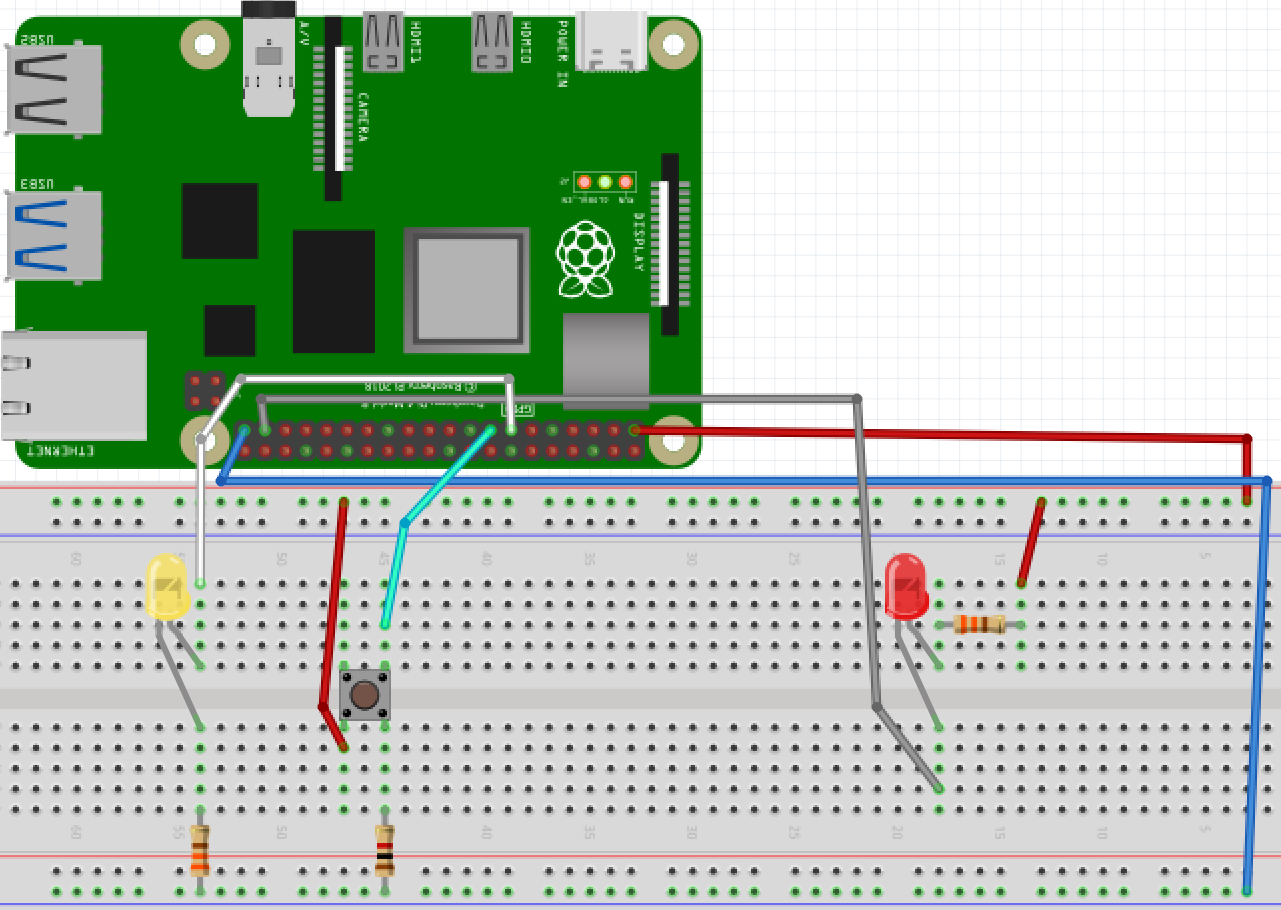
*});*

*});*

1. Save both server and client changes, rerun the server and then client(check listening ports match.) Now when we toggle the led checkbox on the HTML page, the LED on the PI will light up. Below is example of what a mouse click does while server and pi client is running. This is to show software works, the LED should toggle too or there is a circuit issue.



1. Now we will allow the Pi to toggle pin states from a physical push button, then display the pin changes in real time on the HTML page and toggle another LED for illustration. Here is the updated circuit. Remember this uses the gpio library, not rpi-gpio and uses GPIO/BCM pin numbering not the direct pin number. Push button on GPIO 22/pin 15 and LED on GPIO 27 pin13You will need ;

* More jumper wires
* A push button and LED ( last led was in active low configuration, this is provided power when input pin is turned on as result of push button state change1k ohm resistor for push button,
*  330-ohm resistor for LED

1. Now let’s edit the client/App.js file to read the states from the push button. Below is the updated code ;

*var socket = require("socket.io-client")("http://localhost:3014"); //change 3000 to suit www//*

*var gpio = require("rpi-gpio"); //uses direct pin numbering*

*//added below led control for button/led control*

*var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO as BCM numbering*

*var LEDB = new Gpio(27, 'out'); //use GPIO27/pin 15 as output*

*var pushButton = new Gpio(22, 'in', 'both'); //use GPIO 22/pin 15 as input, and 'both' button presses,*

*//needed to control pin state*

*var led =gpio.setup(37, gpio.DIR\_OUT, function(){*

*gpio.write(37, true); // turns led off as in active low config*

*});*

*//this is used for qol when program ends, turns led off when client not connected*

*process.on("SIGINT", function(){*

*LEDB.writeSync(0); // Turn LED off*

*LEDB.unexport(); // Unexport LED GPIO to free resources*

*pushButton.unexport(); // Unexport Button GPIO to free resources*

*gpio.write(37, true, function(){*

*gpio.destroy(function(){*

*process.exit();*

*});*

*});*

*});*

*//connect to server*

*socket.on("connect", function () {*

*console.log("Connected to server");*

*//update change anytime there is one*

*socket.on("updateState", function (state) {*

*console.log("The new state is: " + state);*

*gpio.write(37, !state);*

*});*

*pushButton.watch(function (err, value) { //Watch for hardware interrupts on pushButton GPIO, //specify callback function*

*if (err) { //if an error*

*console.error('There was an error', err); //output error message to console*

*return;*

*}*

*LEDB.writeSync(value); //turn LED on or off depending on the button state (0 or 1)*

*//write pin status to console*

*console.log(' Pin status is : ', value);*

*//need to change to emit when true or false*

*//Sending pin state to server but isn't its own socket.on as not triggered*

*//by a listening event but a gpio state change*

*socket.emit("buttonpress", value);*

*});*

*});*

1. Now let’s edit the server/bin/www file. This update will allow the passed state to be read by the server through the update function name “buttonpress”. Then it will pass the state onto the connected HTML page(client) with an emit and another unique title “updatevirtualButton.” Here is the except from newly edited code.

*io.on("connection", function(socket){*

*console.log("Client Connected");*

*socket.on("stateChanged", function(state){*

*console.log("State Changed: " + state);*

*io.emit("updateState", state); // submits change to pi client*

*});*

*socket.on("buttonpress", function(button){*

*console.log("Button press: " + button); //see change in server console*

*io.emit("updatevirtualButton", button); // submits change to listening html client*

*});*

*});*

1. Let’s edit the HTML to display the passed push button state by toggling the check box. Here is the only edited part of the code

*<script type="text/javascript">*

*//connect to GPIO of PI*

*var socket = io.connect('/');*

*//jQuery takes changed state of toggle on page and passes state to server*

*$("#myonoffswitch").change(function(){*

*socket.emit("stateChanged", this.checked);*

*});*

*socket.on("updatevirtualButton", function (button) {*

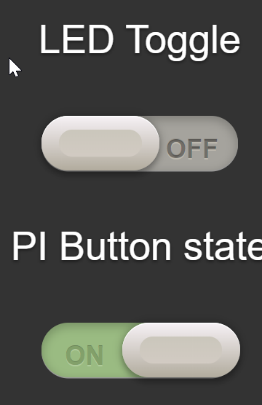
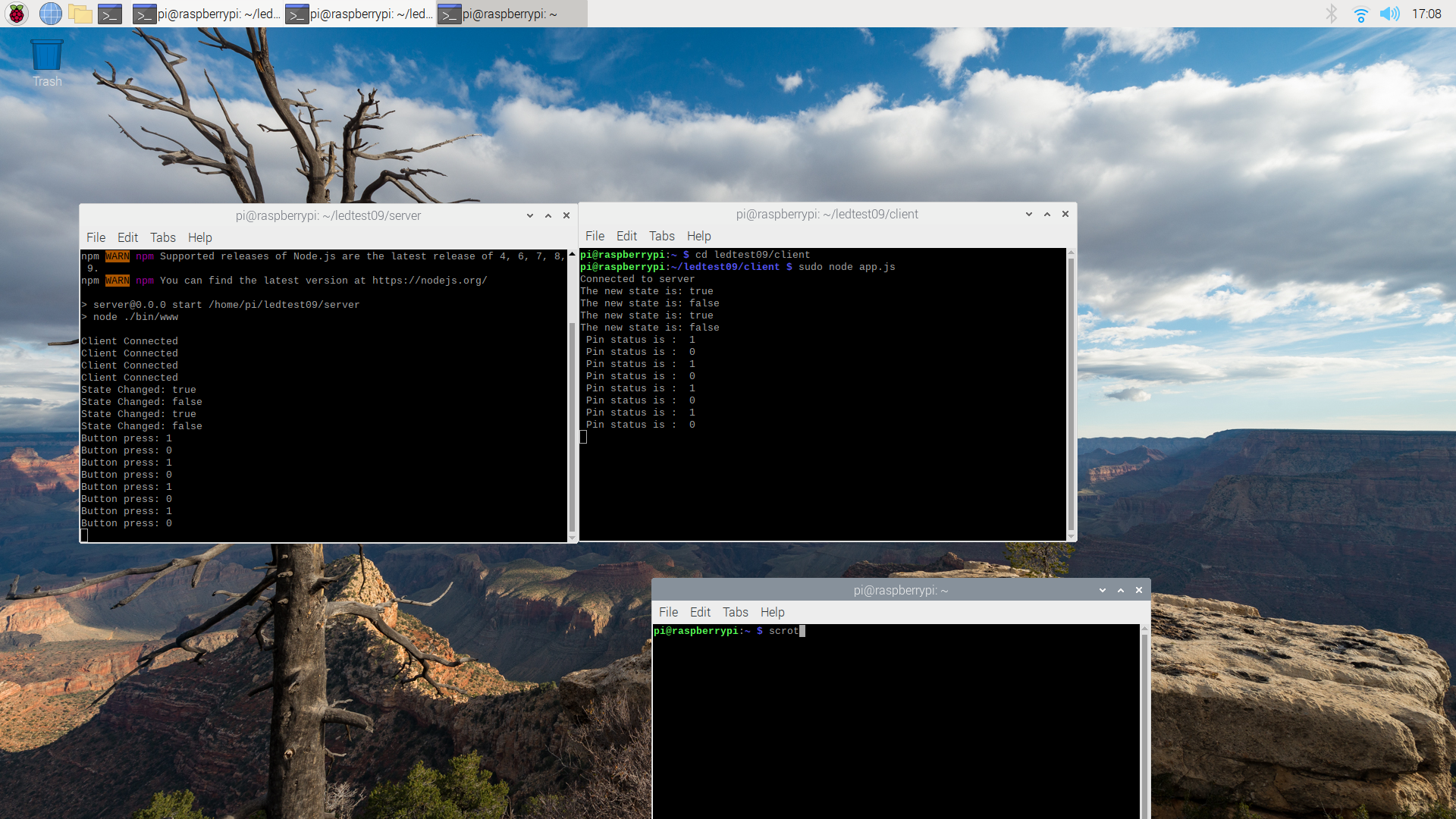
*console.log("The button is: " + button); //just makes sure state is received*

*//js below changes checkbox on html page dynamically as state is send from PI-server-HTML*

*document.getElementById("mybuttonGPIO").checked = button;*

*});*

*</script>*

1. Finally, we can restart the server and client terminals, making sure everything works correctly. Here are the terminal screenshots and HTML as Led toggle is pressed on html, and push button is physically pressed in real time.

## Part 4 Deploying application on Heroku for remote control of the server from any network

I will assume you already have a Heroku account. But essentially Heroku is a cloud platform that allows you to deploy your applications quickly. For us this will allow us to host/run our server and front-end HTML on Heroku instead of locally, with the Pi client listening to the Heroku URL instead of our local network/port. This increases security and capability of our applications; the possibilities are endless for useful IOT applications. After this we will be able to view our application front end from any device anywhere with a browser and read/write to the GPIO pin states.

1. Install the Heroku CLI (command line interface) to your Pi terminal by using the following command from your root directory.

pi@raspberrypi:~/ curl https://cli-assets.heroku.com/install.sh | sh

Or use this command( does not auto update so not recommended)

pi@raspberrypi:~/ npm install -g Heroku Followed by heroku –version to check install. Go to here for more information <https://devcenter.heroku.com/articles/heroku-cli>

1. First go to your server/bin/www file and change the port back to 3000. In my case this is the first line below, but if you copied my GitHub copy it is likely to be the second line.

*var port = normalizePort(process.env.PORT || '3000');//change port to work locally with client*

*app.set('port', process.env.PORT || 3000); //change from 3000 to work locally to match client*

1. In terminal cd to the server code only . E.g. cd ledtest09/server and follow these commands;

* pi@raspberrypi:~/ledtest09/server $ sudo touch .gitignore (used for heroku to ignore installing pi modules) may need to remove!
* pi@raspberrypi:~/ledtest09/server $ git init (initialise repository to deploy)
* pi@raspberrypi:~/ledtest09/server $ git status -s (checks .gitignore removes packages that don’t need deployed)
* pi@raspberrypi:~/ledtest09/server $ git add . Adds every file in folder to repository
* pi@raspberrypi:~/ledtest09/server $ git commit -m”initial commit” gives committed file changes to repo a title (useful for version control in git)
* pi@raspberrypi:~/ledtest09/server $ heroku login then press any key to open a browser and login to your Heroku account. Exit and come back to terminal when logged in
* pi@raspberrypi:~/ledtest09/server $ heroku create controlgpiotutorial where controlgpio is my app URL, change to your own unique one or leave blank for Heroku to generate.

Additionally, you can change to a Heroku application your already have with the command heroku git:remote -a nameofyourapp. For more Heroku Info on git, go here <https://devcenter.heroku.com/articles/git>

* pi@raspberrypi:~/ledtest09/server $ git push heroku master to deploy application to Heroku URL, a link will appear to visit the page.
* Note : you can remove git repo to initialise again with the command rm -rf .git\*

1. Visit the URL from a device of your choosing, and cd back to your client software. ( cd then cd yourfoldername/client)
2. Open your client/ app.js, comment out the localhost line and paste in the following code.

*// heroku, change to URL of your application*

*var socket = require("socket.io-client")("https://controlgpiotutorial.herokuapp.com");*

*//local host*

*//var socket = require("socket.io-client")("http://localhost:3000"); //change 3000 to suit server/bin/www*

1. Run client/app.js from terminal using command sudo node app.js
2. Check push button on breadboard toggles the virtual button on html and led toggle on HTML triggers the first LED to turn on/off. Any errors at this point would likely be the wrong URL used or code is still trying to listen to localhost not the URL. If it connects, terminal will display “connected to server” and all actions will work as expected.
3. Note, that the HTML will only work whenever the client software is manually triggered, but auto scripts could be used. See my first tutorial on using Barrier virtual keyboard and mouse.

Congratulations, that’s us done. Emulate and use this method to develop your IOT projects with real time server communications. Optionally, let’s look at the next tutorial, deploying our applications on GITHUB for version control management and collaboration support.