

گزارش پروژه کامپیوتر

پیاده سازی **IOT HOUSE** با استفاده از میکروکنترلر و سیستم های
نهفته

نگارش:

زهرا کیانپور

شهریور 1402

چکیده:

هدف از انجام این پروژه ، پیاده سازی IOT HOUSE با استفاده از مفاهیم سیستم های نهفته و میکروکنترلرها است . این مقصود با استفاده از دریافت اطلاعات مشخصه های محیط از سنسورها ، انتقال آن به میکروکنترلر و بارگذاری آن ها در اینترنت انجام گردیده است .

از آنجایی که همه ی این اطلاعات را نمیتوان تنها از یک ناحیه ی خاص جمع آوری کرد ، نیاز است که سنسورها در نواحی مختلف از محیط کارگذاشته شوند و اطلاعات را به یک دستگاه مرکزی ارسال کنند . از آنجایی که انتقال اطلاعات از طریق سیم فیزیکی چندان استاندارد و قابل استناد نیست، همت بر این گذاشته شد که سنسورهایی که نیاز به انتقال از راه دور دارند ، با استفاده از پروتکل ارتباطی بیسیم ، اطلاعات را انتقال دهند . در اینجا به انتقال اطلاعات مربوط به وضعیت باز یا بسته بودن درب به شکل بیسیم بسنده کردیم . (افزودن مابقی سنسورها برای انتقال اطلاعات از طریق بیسیم دشواری اضافه ای ندارد . چالش اصلی این پروژه صرفا افزودن این امکانات به یک سنسور از راه دور بوده و افزایش میزان اطلاعات قابل انتقال ، تاثیری در ادامه ی این روند ندارد .)

EMBEDDED SYSTEMS, IOT HOUSE , MICRO CONTROLLERS ,ESP8266 ,
ESP-NOW , LINEAR ACTUATOR , CLIENT MODE IN ESP8266

7.....	فصل اول: مقدمه
9.....	فصل دوم: پیش زمینه و سابقه تحقیق
10.....	فصل سوم: شرح مفاهیم پایه
21.....	فصل چهارم: معرفی طرح و یا الگوریتم پیشنهادی با جزئیات کامل
30.....	فصل پنجم: معرفی روش پیاده سازی و تنظیمات سیستم
45.....	نتیجه گیری و پیشنهادات
46.....	منابع

فصل اول

مقدمه:

اینروز ها بحث اینترنت اشیا (Internet of Things) یا همان IoT یکی از موضوعات پر طرفدار در زمینه مهندسی است. با گسترش تکنولوژی و وجود تلفن های همراه و شبکه های ارتباطی ، شما میتوانید با استفاده از تلفن همراه یا لپتاپ از طریق اینترنت از هر کجای دنیا وسایل خود را کنترل کنید و اطلاعات سنسور های خود را دریافت کنید .

اینترنت اشیاء تکنولوژی جدیدی است که از طریق حسگرها و اتصال وسایل مختلف به اینترنت، آنها را هوشمند می‌سازد. هوشمندسازی وسایل علاوه بر افزایش بازدهی و کاهش هزینه‌ها آسایش و راحتی کاربر را نیز افزایش می‌دهد. با ظهور سیستم های هوشمند و استفاده از داده ها و هوش مصنوعی، پیش بینی های در مورد اینترنت اشیا درست از آب در آمده است. اینترنت اشیا انقلاب صنعتی چهارم شده است و با موفقیت تکنولوژی و تولید مسکونی را تغییر داده است.

وب سرور نرم افزار، سخت افزار یا ترکیبی از هر دو است که حاوی فایل های مورد نیاز برای پردازش و ارائه صفحات وب است. برای نمایش نتایج نهایی بر روی صفحه ی وب سلسه مراتب متعددی باید طی شود . هدف اولیه ی این پروژه ارائه ی سیستم خانه ای مبتنی بر اینترنت اشیا میباشد که به مرور با پر و بال دادن به نیازمندی های این پروژه به مفهومی نهایی از شاخصه های موردنیاز که این پروژه موظف به پیاده سازی آن بود رسیدیم . در انتها ، این پروژه با دریافت اطلاعات از سنسورهایی اعم از سنسور حرکت ، دما ،رطوبت ، شدت نور ، میزان آلاینده ی موجود در هوا و اعلام وضعیت باز و بسته بودن در ، آنها را بر روی وب سرور نمایش میدهد و بر اساس المان هایی که خودمان آنها را معیار قرار میدهیم ، تصمیم گیری میکند و نتیجه را به شکل سیگنال خروجی تولید میکند. در اینجا المان مورد نظر ما دما است و سیگنال خروجی بر یک جک برقی متصل به پنجره اعمال میشود که با توجه به وضعیت دما ، نسبت به باز و بسته بودن پنجره تصمیم گیری میشود . همانطور که بیان شد معیار این تصمیم گیری هرکدام از اطلاعات مربوط به سنسورها میتواند باشد و تاثیری بر قالب کلی و طراحی شده ی این پروژه ندارد . در طول انجام پروژه طبق پیاده سازی های انجام شده مشخص شد که امکان نگه داشتن همه ی سنسور ها در یکجا وجود ندارد علی الخصوص سنسور درب که نیازمند

قرار گرفتن بر روی درب دارد و نگه داشتن باقی تجهیزات سخت افزاری در آن منطقه امری دشوار به نظر می‌رسد . به همین ترتیب تصمیم بر این شد که باقی تجهیزات اعم از سنسور ها و میکروکنترلر اصلی در مکانی دیگر نگه داری شوند و سنسور در به شکل بیسیم اطلاعات را ارسال کند. در این گزارش در ابتدا به شرح مفاهیم اولیه ، نحوه ی تکامل پروژه ، چرایی استفاده از پروتکل ها ، معرفی تجهیزات سخت افزاری ، محدودیت ها و چالش ها و تشریح کد پروژه که به چهار قسمت مختلف تقسیم می‌شود می‌پردازیم .

فصل دوم: پیش زمینه و سابقه تحقیق

در ابتدا به علت علاقه به سخت افزار بیشتر در این زمینه تحقیق کرده و تلاش برای انجام پروژه هایی در این زمینه داشتم . بعد از گذراندن درس میکروکنترلر ها و آزمایشگاه این درس ، علاقه به سخت افزار به سمت سیستم های نهفته جهتگیری داده شد . در این مسیر فرصت آشنایی با انواع میکروکنترلرها ، انواع پروتکل های ارتباطی و مفاهیم مختلف و بروز مرتبط با آنها و انجام پروژه های مرتبط با آن را داشتم و در دوره ی یکساله مشغول به تدریس مباحث های متنوع تحت عنوان درس آزمایشگاه میکروکنترلر بودم . هرچند تا به آن موقع این مباحث را در قالب ارتباط بیسیم دنبال نکرده بودم و این پروژه این شانس را فراهم کرد تا با چالش های جدیدی مواجه شده و بتوانم علم و مهارت خود در این زمینه را گسترش دهم .

فصل سوم : شرح مفاهیم پایه

وب سرور نرم افزار، سخت افزار یا ترکیبی از هر دو است که حاوی فایل های مورد نیاز برای پردازش و ارائه صفحات وب است. کلاینت وب هر دستگاهی است که می تواند یک درخواست **HTTP/Web** را به یک وب سرور ارسال کند. پروتکل **HTTP** یا **Hypertext Transfer Protocol** یک پروتکل منحصر به فرد است که وب سرور و سرویس گیرنده وب برای برقراری ارتباط از آن استفاده می کنند.

برای درک بهتر، فکر کنید می خواهید وارد سایت شوید. ابتدا آدرس را در مرورگر خود وارد میکنید، چند ثانیه بعد صفحه اصلی مشاهده میشود.

در این مثال، کامپیوتر شما یک سرویس گیرنده وب است. رایانه شما با استفاده از یک مرورگر وب، مثلاً **Chrome** یا **Firefox**، یک درخواست وب ارسال می کند. مرورگر وب درخواست را به سرور وب که میزبان است می فرستد. سپس داده های مورد نیاز برای نمایش صفحه اصلی در کامپیوتر شما دریافت میشود. وب سروری که یک وب سایت را میزبانی می کند، معمولاً یک رایانه هدفمند است که حجم عظیمی از داده ها را ذخیره می کند. آنها آدرس های **IP** منحصر به فردی نیز دارند.

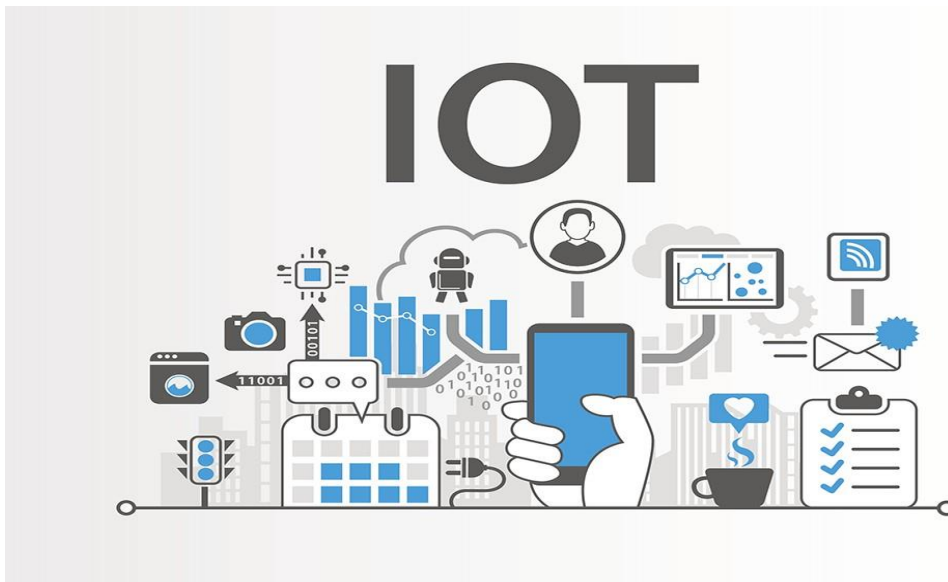
در این پروژه ما به سرور های بزرگ که مخصوص سایت ها هستند نیازی نخواهیم داشت. میتوانیم به سادگی یک آردوینو را به یک ماژول **ESP8266** متصل کنیم و یک وب سرور ساده اما کار آمد بسازیم. با یک وب سرور آردوینو میتوانیم یک صفحه وب ایجاد کنیم که داده های سنسور ها را نمایش دهد یا کنترل پایه ها را از طریق آن انجام دهیم. وب سروری که از طریق اینترنت در هر نقطه ای از جهان قابل دسترسی باشد، سرور جهانی نامیده می شود. همچنین وب سروری که فقط در شبکه محلی (**LAN**) قابل دسترسی است، سرور محلی نامیده می شود.

برای دریافت داده از سرورهای وب، کلاینت ها (درخواست کننده – کاربر) از درخواست های **HTTP** استفاده می کنند. چندین نوع درخواست **HTTP** وجود دارد، اما برای ایجاد یک سرور آردوینو فقط باید دو مورد را یاد بگیریم. این درخواست ها **HTTP GET** و **HTTP POST** نامیده می شوند.

- **HTTP GET** یک درخواست وب است که داده ها را از یک مرورگر وب دریافت می کند. هیچ چیز روی سرور تغییر نمی کند. فقط داده ها را از دریافت میکند و میخواند.

- **HTTP POST** یک درخواست وب است که داده ها را به سرور منتقل می کند. یعنی چیز جدیدی را به سرور اضافه می کند.

یک مثال معمولی از درخواست **GET** ، مرور ساده یک وب سایت است. از سوی دیگر، درخواست های **POST** در تایپ متن در یک صفحه وب، به عنوان مثال، نام کاربری و رمز عبور استفاده می شود.



ESP8266

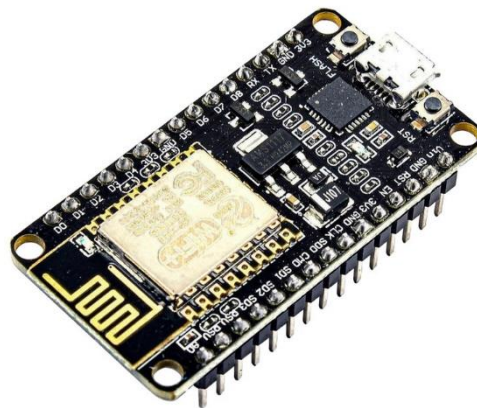
ماژول **ESP8266** دو ویژگی بسیار مهم دارد. ابعاد کوچک و قیمت اقتصادی آن بین تمامی کاربران زبان زد شده است. یکی از ابزارهای بسیار مهم جهت ورود به پروژه های اینترنت اشیا، ماژول های وای فای سری **ESP** هستند. به کمک این ماژول ها می توان پروژه های هوشمند سازی بسیاری را انجام داد. **ESP8266** یک ماژول وای فای قدرتمند و کوچک است که برای اتصال دستگاه ها به شبکه های بی سیم طراحی شده است. این برد کوچک اما قدرتمند، دارای قابلیت های بسیاری است که آن را برای کاربردهای اینترنت اشیا (**IoT**) بسیار مناسب می کند **ESP8266** دارای یک پردازنده قوی و حافظه مناسب است که امکان برنامه نویسی مستقیم بر روی برد را فراهم می کند. این برد با قیمت

مناسب و قابلیت‌های بسیار، به عنوان یکی از محبوب‌ترین بردهای توسعه اینترنت اشیا شناخته می‌شود. از طریق پورت **USB** می‌توان برنامه‌های توسعه داده شده را به **ESP8266** ارسال کرده و توسط برنامه نویسی آن‌ها را اجرا کرد. با استفاده از این برد، می‌توان تجربه‌های آموزشی در زمینه برنامه نویسی وای فای و اینترنت اشیا را به راحتی به دست آورد و همچنین پروژه‌های متنوعی را در این حوزه پیاده سازی کرد. با توجه به توانایی‌های بالای **ESP8266** در اتصال به شبکه بی سیم و پشتیبانی از پروتکل‌های مختلف مانند **TCP/IP**، **HTTP** و **MQTT**، این برد در اتصال دستگاه‌های هوشمند، کنترل خانه هوشمند، مانیتورینگ سیستم‌ها، رصد هواشناسی و بسیاری از کاربردهای دیگر مفید است. به طور خلاصه، **ESP8266** یک ابزار قدرتمند برای ایجاد دستگاه‌های متصل به اینترنت و ساخت پروژه‌های **IoT** است که از آن می‌توان به راحتی بهره برد. **ESP8266** یک ماژول میکروکنترلر وای فای با قابلیت برنامه نویسی مستقل است که توسط شرکت **Espressif Systems** توسعه داده شده است. این ماژول، امکان اتصال دستگاه‌ها به شبکه‌های بی‌سیم را فراهم می‌کند و از معماری کوچک و قدرتمند خود برای ایجاد برنامه‌های قابل اجرا بر روی خود استفاده می‌کند. این برد دارای چیپ **Wi-Fi** با استاندارد **802.11 b/g/n** است که به دستگاه‌ها امکان اتصال به شبکه‌های بی‌سیم را می‌دهد.

ESP8266 به طور معمول از طریق پورت **USB** یا پین‌های دیجیتال و آنالوگ به کامپیوتر یا سایر دستگاه‌ها متصل می‌شود. برنامه نویسی آن به وسیله زبان **Arduino** و با استفاده از محیط توسعه **Arduino IDE** انجام می‌شود. همچنین، برای برنامه نویسی پیشرفته‌تر، می‌توان از زبان‌های برنامه نویسی مانند **MicroPython** و **Lua** نیز استفاده کرد. علاوه بر اتصال به شبکه‌های بی‌سیم، **ESP8266** همچنین دارای پورت‌های **I2C**، **UART** و **SPI** است که به کاربر امکان اتصال به سنسورها، ماژول‌ها و دستگاه‌های جانبی را می‌دهد. این قابلیت‌ها، تعامل با دستگاه‌های خارجی را برای پروژه‌های مختلف بسیار آسان می‌کند. **ESP8266** همچنین دارای امکانات امنیتی است. این برد از پروتکل‌های رمزنگاری **WPA/WPA2** و **WEP** برای اتصال امن به شبکه‌های بی‌سیم استفاده می‌کند و قابلیت احراز هویت و رمزگذاری را فراهم می‌کند. همچنین، ماژول دارای یک آنتن داخلی است که برد سیگنال را بهبود می‌بخشد. با توجه به جامعه فعالیت برنامه‌نویسان و سازمان‌های توسعه‌دهنده در حوزه **ESP8266**، محیط توسعه این برد بسیار پویا و پشتیبانی شده است. وجود کتابخانه‌های غنی و ابزارهای توسعه متنوع، به برنامه‌نویسان اجازه می‌دهد تا به سرعت و با راحتی پروژه‌های خود را پیاده‌سازی کنند. در نتیجه، **ESP8266** به عنوان یک برد قدرتمند و چند منظوره، برای

توسعه‌دهندگان و سازندگان پروژه‌های IoT یک انتخاب بسیار مناسب است. امکانات و قابلیت‌های آن، همراه با هزینه مناسب و انعطاف‌پذیری بالا، آن را به یکی از بردهای محبوب و مورد علاقه در دنیای IoT تبدیل کرده است.

- دارای مدهای مختلف توان مصرفی پایین
- دارای بازه ولتاژ تغذیه بین ۲٫۵۶ الی ۳٫۶ ولت DC
- قابلیت اتصال به تلفن‌های همراه جهت تبادل داده
- قابلیت اتصال به اینترنت جهت ارسال و دریافت داده‌ها
- قابلیت قرارگیری در مد **STA** جهت اتصال مازول به مودم‌ها
- قابلیت تعریف در حالت سرور جهت دریافت و مدیریت داده‌های ارسالی
- دارای حافظه فلش و فرکانس کاری بالا جهت اجرای برنامه‌های سنگین
- قابلیت برنامه‌نویسی به کمک دستورات آردوینو و اجرای برنامه‌های آردوینو
- قابلیت برنامه‌نویسی با دستورات آردوینو و بدون نیاز به هیچ میکروکنترلر اضافی
- قابلیت قرارگیری در مد نقطه دسترسی **AP** جهت اتصال سایر دستگاه‌ها به مازول

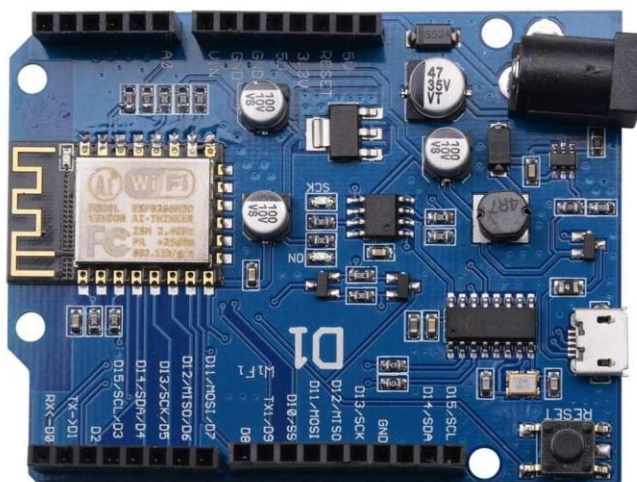


شکل 1 - برد NODE MCU

WEMOS

ماژول **Wemos D1** یک برد آردوینو با تراشه وای فای **ESP8266** می باشد که در واقع بسیار شبیه به ماژول **NodeMCU** است با این تفاوت که اندازه آن کوچکتر و دارای نسخه جدیدی از ماژول وای فای **F12** است. این ویژگی به علاوه مصرف کم انرژی این برد باعث شده تا بتوان از آن در پروژه های اینترنت اشیا بهره برد. به این دلیل که این ماژول با آردوینو سازگار است، شما می توانید کدهای نوشته شده برای آردوینو را روی این برد اجرا کنید. روی این برد یک مبدل **USB-UART** نیز وجود دارد و تمام چیزی که شما برای شروع برنامه ریزی به آن نیاز دارید، یک کابل **USB** ساده است.

ماژول **Wemos D1**، از **GPIO 9** با **PWM**، **I2C**، **SPI** و ارتباط تک سیم پشتیبانی می کند. سه روش برای استفاده از **Wemos** وجود دارد؛ **AT command** ها، زبان برنامه نویسی **LUA** به همراه فریمور **NODEMCU** و در آخر، آسانترین و محبوب ترین روش یعنی؛ آردوینو **IDE**.



شکل 2 – برد **WEMOS D1 R2**

معرفی سنسور ها

سنسور حرکت

ماژول های تشخیص حرکت، یکی از پرکاربردترین ماژول ها در تشخیص حضور افراد در مکان های مختلف است. این ماژول ها که بر مبنای امواج مادون قرمز فعالیت می کنند، می توانند حضور افراد در محیط را تشخیص دهند. ماژول

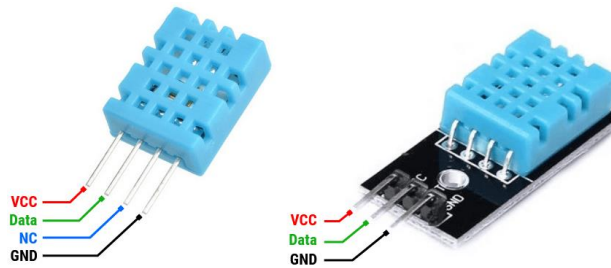
مادون قرمز SR602. یک ماژول کاربردی جهت تشخیص حرکت است. به کمک این ماژول می توانید به سادگی و بدون نیاز به هیچ ابزار جانبی، حضور و یا حرکت افراد در محیط را تشخیص دهید. راه اندازی و کار با این ماژول بسیار ساده است. کافایت تغذیه آن متصل شود. سپس در صورت تشخیص حضور افراد، پایه خروجی آن تغییر وضعیت و ولتاژ خواهد داد. بنابراین به سادگی می توانید این ماژول را به بردهای **ESP** و سایر میکروکنترلرها نظیر **ARM** متصل کنید. همچنین، مطابق اطلاعات دیتاشیت، این ماژول می تواند حضور افراد را در فاصله 3 الی 5 متر، با زمان پاسخ دهی بسیار سریع، تشخیص دهد.



شکل 3 – سنسور حرکت مادون قرمز مدل SR602

سنسور دما و رطوبت

سنسور **DHT11**، یک سنسور دما و رطوبت دیجیتال است. این سنسور قادر است میزان رطوبت بین 20 تا 80 درصد را با دقت 5٪ و دما را در بازه ی 0 تا 50 درجه ی سلسیوس، با دقت 2 درجه اندازه گیری کند. فرکانس نمونه برداری این سنسور 1 هرتز است. در نتیجه باید بین دوبار خواندن متوالی سنسور حداقل یک ثانیه زمان وجود داشته باشد، در غیر این صورت، مقادیر خوانده شده اشتباه خواهد بود.



شکل 4 – سنسور دما و رطوبت مدل DHT11

سنسور گاز

MQ2 یک ماژول گاز مقاوم برای تشخیص غلظت **LPG** ، دود، الکل، پروپان، هیدروژن، متان و کربن منواکسید در هوا می‌باشد. اگر قصد ایجاد سیستم نظارت بر کیفیت هوا در محیط داخلی را دارید؛ ماژول سنسور گاز **MQ2** یک انتخاب عالی برای این کار است.

MQ2 یکی از سنسورهای متداول در سری سنسورهای گازی **MQ** است. این سنسور از نوع اکسید نیمه هادی (**MOS**) بوده و به عنوان مقاومت شیمیایی نیز شناخته می‌شود زیرا فرآیند تشخیص را با تغییر مقاومت احساس شده مواد در هنگام تماس گاز با آن‌ها سنجش می‌کند. با استفاده از یک شبکه تقسیم ولتاژ ساده ، می‌توان غلظت گاز را تشخیص داد. این سنسور در واقع داخل دولا به استیل ضد زنگ که شبکه ضد انفجار نیز نامید می‌شود، محصور شده است. این لایه‌ها تضمین می‌کنند که عنقصر بخاری داخل سنسور منفجر نخواهد شد و هیچ خطری را برای کاربران در پی نخواهد داشت.



شکل 5 – سنسور گاز و آلاینده های هوا مدل MQ2

سنسور نور TEMT6000

این سنسور یک سنسور آنالوگ با عملکردی مشابه ترانزیستور NPN و حساسیتی مطابق با حساسیت چشم انسان است. این سنسور عکس العمل مناسبی نسبت به کاهش و افزایش وضعیت روشنایی دارد و نسبت به کوچکترین تغییری در محدوده ی وسیع روشنایی پاسخ می دهد. توجه داشته باشید که این سنسور نسبت به نور IR و UV واکنش مناسبی نشان نمی دهد.



شکل 6 – سنسور دما مدل TEMT6000

سنسور درب

سنسور مغناطیسی تشخیص وضعیت درب با بدنه تمام فلزی و ضد آب قابلیت نصب سریع بر روی انواع درب و پنجره را دارد.

کاربردهای این محصول در درهای کشویی، درهای پارکینگ، درهای بادبزی، پنجره های کشویی، پنجره تاشو می باشد. این سنسور با بدنه ای از آلیاژ فلز روی، می تواند به عنوان یک سنسور ضد سرقت ایده آل برای گاراژ منزل، انبار شرکت، فروشگاه ها باشد.

در صورت بسته بودن درب و نزدیک بودن ۲ بخش سنسور به هم، خروجی سنسور وصل (اتصال کوتاه) و در صورت باز بودن درب و فاصله داشتن دو یخس سنسور از هم، خروجی سنسور قطع (اتصال باز) می شود



شکل 7 – سنسور درب

کنترل کننده موتور

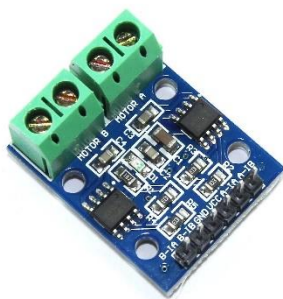
یک کنترل کننده موتور یک دستگاه الکترونیکی است (معمولا به شکل یه مدار بدون پوشش و محفظه است) که به عنوان یک دستگاه واسطه بین میکروکنترلر، یک منبع تغذیه یا باتری و موتورها عمل می کند.

اگرچه میکروکنترلر (مغز ربات) سرعت و جهت موتور را مشخص میکند، اما به دلیل محدودیت زیاد در تغذیه خروجی (جریان و ولتاژ) نمی تواند آن ها را مستقیما هدایت کند. از طرف دیگر درایور موتور میتواند جریان را در ولتاژ مورد نظر فراهم کند اما نمیتواند تصمیم بگیرد که موتور تا چه میزان سریع بچرخد.

بنابراین، میکروکنترلر و کنترل کننده موتور باید باهم کار کنند تا موتور به طور مناسبی حرکت کند. معمولا میکروکنترلر میتواند از طریق یک روش ارتباطی ساده مانند **UART(serial)** یا **PWM** به کنترل کننده موتور دستورالعملی برای چگونگی تغذیه موتور بدهد. هم چنین برخی از کنترل کننده های موتور را می توان به صورت دستی و با استفاده از ولتاژ آنالوگ کنترل کرد (معمولا با یک پتانسیومتر ایجاد می شود)

اندازه و وزن فیزیکی کنترل کننده موتور می تواند بسیار متفاوت باشد، از یک دستگاه کوچکتر از نوک انگشتان برای کنترل یک ربات کوچک **sumo** تا یک کنترل کننده سنگین وزن چند کیلوگرمی. اندازه و وزن کنترل کننده ربات معمولا کمینه تاثیر را روی ربات دارد، تا زمانی که شما با ربات های خیلی کوچک و یا هواپیماهای بدون سرنشین کار می کنید که در این حالت کوچکترین وزن ها هم تاثیرگذار خواهد بود. اندازه کنترل کننده موتور معمولا به حداکثر جریانی که میتواند فراهم کند وابسته است. جریان بیشتر به معنی استفاده از سیم هایی با قطر بزرگتر است.

از آنجا که انواع مختلفی از محرک ها یا عملگرها وجود دارد، انواع گوناگونی از کنترل کننده های موتور نیز وجود دارد .
کنترل کننده موتور های براش **DC** (با جاروبک) دسته ی خاصی از این کنترل کننده هاست که در این پروژا از آن بهره
بردیم . این کنترل کننده ها مختص موتورهای **DC** ، موتورهای **DC** گیربکس دار و بیشتر موتورها یا عملگرهای خطی
است. در این پروژه از کنترل کننده ی مدل **L9110** استفاده میکنیم .



شکل 8 – کنترل کننده ی موتور مدل **L9110**

IDE آردوینو

برای آسان تر کردن کدنویسی ، آردوینو از یک محیط توسعه یکپارچه رسمی (**IDE**) استفاده می کند که نوشتن، کامپایل
و آپلود کد در برد را ساده می کند. برای برنامه نویسی آردوینو که به زبان **C** یا **C++** نوشته می شود، می توان از پلتفرم های
دیگری که مشابه **Arduino IDE** رایگان، **Open Source** و آسان هستند، استفاده کرد؛ ولی معمولاً کاربران، برای
برنامه ریزی میکروکنترلر آردوینو **IDE** را انتخاب می کنند.



شکل 9 – **ARDUINO IDE**

SPIFFS

SPIFFS این قابلیت را به شما می دهد تا همانطور که به سیستم فایلی رایانه خود دسترسی دارید به حافظه فلش میکرو کنترلر دسترسی داشته باشید اما با سادگی و محدودیت بیشتر. می توانید فایلها را بخوانید، بنویسید، ببندید و یا پاک کنید **SPIFFS**. از پوشه ها پشتیبانی نمی کند و کلیه فایلها در کنار هم ذخیره خواهند شد.

در صورتیکه از قابلیت **SPIFFS** مربوط به برد **ESP8266** استفاده می کنید روشهای سودمند ذیل بهره مند شوید:

- استفاده جهت ذخیره فایلهای پیکربندی و تنظیمات.
- ذخیره سازی دائمی داده ها.
- در صورت کوچک بودن داده ها شما را از استفاده میکرو **SD** ها بی نیاز خواهد کرد.
- هنگام ایجاد سرور وب می توانید فایلهای **html** و **css** صفحات وب را در آن ذخیره کرده و فرخوانی کنید.
- می توانید فایل مربوط به تصاویر، اشکال و ایکن ها را در آن ذخیره کنید.

به همین منظور برای **IDE** یک پلاگین ساخته شده است که این امکان را می دهد تا فایلها را به حافظه **SPIFFS** آپلود کنیم.

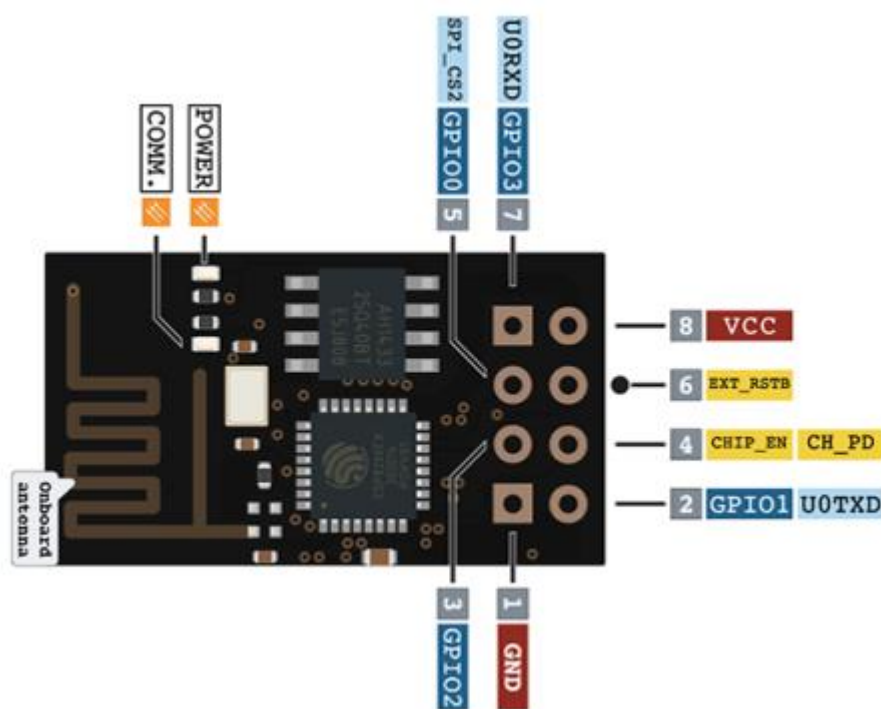
فصل چهارم: معرفی طرح و یا الگوریتم پیشنهادی با جزئیات کامل

در ابتدا طرح اولیه ی پروژه پیاده سازی یک خانه ی هوشمند بود . هماهنگی که میدانیم خانه ی هوشمند نیازی به دسترسی به اینترنت ندارد و تنها با دریافت اطلاعات از محیط قادر به تصمیم گیری و تولید سیگنال خروجی است .

پیاده سازی خانه ی هوشمند از طریق میکروکنترلر های مختلفی قابل انجام است که بهینه ترین نوع آن **ARDUINO UNO** است . اما بعد از ارائه ی این طرح که مبتنی بر آردوینو **UNO** فعالیت میکرد ، بنا شد که این سیستم قادر به اتصال به اینترنت باشد و بتواند اطلاعات لازم را از طریق سنسورهای متصل به میکروکنترلر دریافت کرده و آن را از طریق اینترنت و صفحه ی وب برای کاربر به نمایش در آورد. برای انجام این گام راه های گوناگونی وجود داشت . هم از نظر نرم افزاری و هم از نظر سخت افزاری . از نظر نرم افزاری این امکان را داشتیم که از اپلیکیشن های خاص منظوره در این زمینه استفاده کنیم که برای راه اندازی آنها تقریباً به هیچ علم برنامه نویسی نیاز نبود . از معروف ترین این نرم افزار ها ، نرم افزار **BLINK** است که براحتی قابل اجرا بوده و هم بر روی تلفن همراه و هم سیستم عامل ویندوز قابل نصب است . جدا از اینکه استفاده از این نرم افزار در این پروژه از ارزش علمی آن میکاست ، در نهایت به علت تحریم آپدیت جدید آن در ایران ، قادر به استفاده از این اپلیکیشن نبودیم . به همین علت **BLINK** و نرم افزار های مشابه به آن خط خوردند . هرچند که این روش چندان هم مورد علاقه ی بنده نبود . پس در آخر به این نتیجه رسیدیم که این قسمت را به شکل دستی و با پیاده سازی صفحه ای مبتنی بر **HTML** و **CSS** انجام دهیم که در فصل چهارم همین گزارش در مورد نحوه ی پیاده سازی آن به تفصیل صحبت خواهیم کرد . برای قسمت سخت افزاری این قدم از پروژه دچار اولین آزمون و خطا شدیم . از آنجا که در قدم اول پروژه که پیاده سازی خانه ی هوشمند بود ، با آردوینو **UNO** فعالیت کرده بودیم و تا قسمت مطلوبی از پروژه با این میکروکنترلر پیش رفته بود ، تمایل داشتیم که با همین میکروکنترلر فعالیت

کنم . اما وقتی که قرار بر این شد که از اتصال به اینترنت استفاده کنیم ، دوراهی نحوه ی فراهم کردن این امکانات به وجود آمد . راه اول استفاده از ماژول و راه دوم استفاده از برد جدید بود .

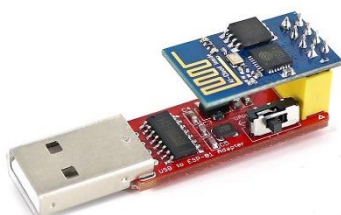
در راه اول تنها لازم است یک ماژول **ESP8266** تهیه کنیم و آن را به برد آردوینوی خود متصل کنیم . این کار به منزله اضافه کردن امکانات جدید به میکروکنترلی است که در ابتدا از آنها بهره ای نبرده است .
روش دوم تهیه ی یک برد مبتنی بر **ESP8266** است که این برد خود دارای تراشه ی مورد نیاز است و نیازی به یک ماژول برای افزودن امکان ایجاد برقراری ارتباط مبتنی بر اینترنت ندارد.



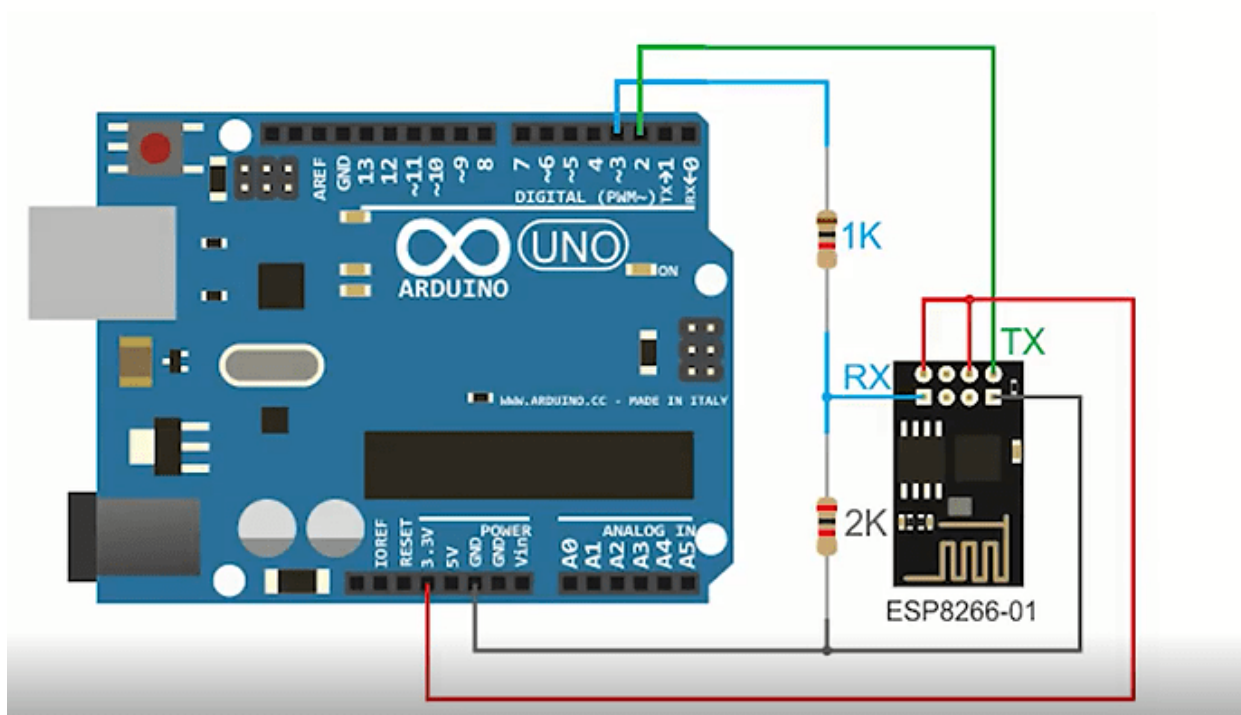
شکل 10- شماتیک ماژول ESP8266 مدل 01S

در ابتدا به علت داشتن آردوینو **UNO** و همچنین هزینه ی کم ماژول **ESP8266** راه حل اول گزینه ی مناسبی بنظر آمد هرچند که پروگرام کردن این ماژول به سادگی پروگرام کردن یک برد نبود زیرا این ماژول پورتهای برای اتصال مستقیم به کامپیوتر ندارد و باید از مبدل های مناسب استفاده کرد تا کد مربوط به اتصال به وای فای را در آن بارگذاری کرد و سپس با استفاده از پروتکل ارتباطی **UART** آن را به آردوینو **UNO** متصل کرد . در اصل با اتصال این ماژول و

آردوینو ، ماژول قادر خواهد بود دستورها را از برد آردینو دریافت کرده و با کمک امکان اتصال به اینترنت آن را اجرایی کند و همچنین دیتاهای دریافتی را برای آردوینو ارسال کند .



شکل 11 – مبدل سریال به USB مخصوص ماژول های ESP8266



شکل 12 – نحوه ی اتصال ESP8266-01 به برد آردوینو UNO

هرچند که طی تحقیقات انجام شده قبل از تهیه ی این ماژول ، بنظر میرسید که نحوه ی عملکرد این ماژول نباید تفاوتی با بردهای مبتنی بر آن داشته باشد و باید عملکرد کاملی داشته باشد ، اما در عمل چنین اتفاقی رخ نداد. آنتن این ماژول به شدت ضعیف بود و حتی قادر به ایجاد یک ارتباط پایدار با یک مودم را نداشت. ناکارآمدی و ضعف این ماژول آنجا آشکارتر شد که گاهی آپلود کد را نمیپذیرفت. به این شکل که یک کد یکسان را گاهی اجرا میکرد یا گاهی نمیکرد بدون اینکه از نظر سخت افزاری تغییری ایجاد شود یا دچار قطع و وصلی شده باشد. به همین علت این روش از مسیر پروژه حذف شد و به برد های مبتنی بر **ESP8266** روی آوردیم .

در این زمینه بردهای مختلفی در بازار موجود است و همه در نوع خود بهترین عملکردها را نشان داده اند . با پرس و جو از افراد باتجربه و متخصص به دو انتخاب نهایی برد **NODEMCU** و برد **WEMOS D1** رسیدیم . از آنجایی که این پروژه از سنسورهای زیادی بهره میبرد به همین علت نیاز به بردی داریم که پین های ورودی و خروجی زیادی داشته باشد به همین خاطر در این بین برد **WEMOS D1 R2** به علت دارا بودن پین های ورودی و خروجی بیشتر برگزیده شد . مسیر پروژه ازین قسمت تقریبا هموار شد . دریافت ورودی از سنسور ها و ایجاد یک صفحه ی وب و بارگذاری اطلاعات روی این صفحه اولین کاری بود که باید انجام میشد. این کار از طریق اتصال به یک وای فای ، نوشتن کد **HTML** و مشتقات آن ، ایجاد یک **local IP** و بارگذاری کد طراحی وب و اطلاعات سنسور ها بر روی این **IP** ، انجام میپذیرد . بعد از آن با استفاده از اطلاعات ورودی و پردازش آنها نسبت به باز و بسته بودن آن تصمیم گیر شده و مطابق تصمیم نهایی ، سیگنال نهایی مبنی بر باز یا بسته بودن پنجره تولید شده و به عنوان خروجی میکروکنترلر پردازشگر ، بر روی یک کنترل کننده ی موتور اعمال میشود . این کنترل کننده که در فصل قبل آن را معرفی کردیم ، با توجه به سیگنالی که دریافت میکند ، پنجره را به واسطه ی یک **linear actuator** باز و بسته میکند . نحوه ی پیاده سازی این قسمت در فصل بعد به تفصیل توضیح داده خواهد شد . اما این پروژه در این قسمت تمام نشد و با اضافه شدن سنسور در به پروژه ، با چالش های جدیدی مواجه شدیم . همانطور که در قسمت مقدمه بیان شد ، نگهداری تجهیزات در نزدیکی درب امری دشوار است و از طرفی دیگر اگر بخواهیم برای باز و بسته کردن پنجره اقدام کنیم نمیتوانیم هم به درب نزدیک باشیم و هم به پنجره . پس میتوانیم با نگهداری همه ی تجهیزات در نزدیکی پنجره ، اطلاعات سنسور درب را از راه دور دریافت کنیم . برای این اتصال دو انتخاب داریم :

1. استفاده از سیم رابط برای انتقال سیگنال

2. استفاده از ارتباط بیسیم

پس از مشورت های انجام شده ، روش اول کنار گذاشته شد . زیرا از طریق ارتباط سیمی ، به علت اینکه به سیم بلند نیاز داریم ، محتمل است حین انتقال اطلاعات ، سیم تحت تاثیر نویزهای محیطی نتواند اطلاعات را بدرستی انتقال دهد و از طرفی در صورت افزایش تعداد درب ها ، تعداد پین های ورودی درگیر سنسور افزایش میابد و موجب میشود که یک میکروکنترلر توانایی مدیریت همه ی درب ها را نداشته باشد . به همین خاطر روش دوم برگزیده شد. این روش با وجود ایجاد کیفیت در انتقال داده ها ، چالش های جدیدی را در طراحی مدار بوجود آورد. برای اتصال بیسیم درب به یک برد ویموس نمیتوان خود سنسور را به شکل مستقیم متصل کرد بلکه نیاز است این سنسور به یک مازول یا میکروکنترلر دیگر متصل شده ، اطلاعات آن پردازش شده و سپس از طریق پروتکل های بیسیم به میکروکنترلر دیگر دیتا را ارسال کند . مازول ها بخاطر تجربه ی ناموفق-و کمی تلخ سی که در ابتدای پروژه داشتند در همان ابتدا از لیست انتخاب ها خط خوردند . (هرچند که بعد ها بعلت پیچیدگی پروژه این مازول نیزتست شد و بعد از شکست در اجرای کدها ، مشخص شد که این دسته از مازول ها به طور کلی توانایی بعمل رساندن پروتکل های ارتباطی بیسیم را ندارند .) حال که یک برد دیگر تهیه کرده ایم و میخواهیم به یک برد دیگر متصل کنیم ، باید نوع پروتکل ارتباطی را مشخص کنیم . بعد از تحقیق های طولانی لیستی از پروتکل ها و ابزار ها یافت شد که هر کدام را بررسی میکنیم. اولین ابزاری که توجه ها را به خود معطوف کرد ، آنتن رادیویی **ZigBee** بود . این آنتن در اصل یک مازول است که قابلیت ارتباط بیسیم با استفاده از امواج رادیویی را برای هر دو میکروکنترلر فعال میکند . با این حال از آنجا که برای فراهم کردن شرایط مناسب برای پیاده سازی این روش نیاز است دو مازول تهیه کنیم ، موجب میشود که این روش به صرفه نباشد . از طرفی این قطعه کمیاب بوده و برای یافتن مراکز فروش آن اندکی به دشواری افتادم و این مسئله بیانگر این است که این مازول یا چندان مورد آزمون و خطا قرار نگرفته و یا چندان پرترفدار نیست در حالی که روش های ارتباطی دیگری وجود دارد که دارای سرعت و امنیت مطلوب بوده و نیازی به پذیرفتن چالش های بیهوده نداریم . دو پروتکل ارتباطی دیگری که میان کاربران مورد استقبال واقع شده بودند ، پروتکل های **ESP-NOW** و **ESP-MESH** میباشند که هرکدام به نوبه ی خود دارای نقاط و ضعفی هستند . **ESP-MESH** و **ESP-NOW** هر دو پروتکل های ارتباطی برای

تراشه‌های **ESP8266** و **ESP32** هستند، اما با هدف‌ها و ویژگی‌های متفاوتی عمل می‌کنند. **ESP-MESH** برای ایجاد شبکه‌های مش مورد استفاده قرار می‌گیرد. این شبکه‌ها شامل گره‌های متصل به یکدیگر هستند که برای ارتباط و همکاری در انتقال داده‌ها و کنترل دستگاه‌ها با یکدیگر همکاری می‌کنند. به عبارتی نوعی شبکه مبتنی بر مش تولید می‌شود.

ESP-MESH امکاناتی مانند تشخیص گره‌ها، توانایی انتقال داده‌ها از طریق چندین گره مش و هماهنگی خودکار برای راهبری داده‌ها را فراهم می‌کند. این پروتکل امکان جمع‌آوری داده‌ها، کنترل و مدیریت دستگاه‌های متصل را در شبکه‌های مش فراهم می‌کند.

از طرفی دیگر **ESP-NOW** برای ارتباط مستقیم بین دو دستگاه **ESP8266** و **ESP32** به کار می‌رود. این پروتکل امکان ارسال داده‌ها بدون نیاز به یک شبکه مش یا نقطه دسترسی را فراهم می‌کند. **ESP-NOW** توانایی ارسال و دریافت داده‌ها با سرعت بالا و به طور مستقیم بین دستگاه‌های **ESP** را داراست. این پروتکل برای ارتباط دستگاه‌هایی که در محدوده نزدیک به یکدیگر هستند و نیاز به ارتباط مستقیم دارند، مفید است.

بنابراین، **ESP-MESH** برای ایجاد شبکه‌های مش چندگانه و هماهنگی بین گره‌ها، و **ESP-NOW** برای ارتباط مستقیم بین دو دستگاه **ESP** یا بیشتر از دو دستگاه استفاده می‌شود. هر یک از این پروتکل‌ها ویژگی‌ها و کاربردهای خاص خود را دارند و بسته به نیازهای پروژه می‌توانند انتخاب شوند و با توجه به نیازمندی که در پروژه داریم بایستی سناریو مناسبی پیاده سازی کنیم. در این قسمت ما نیازی نداریم که چندین دستگاه با یکدیگر به شکل یک شبکه صحبت کنند در اصل نیاز داریم که همه ی دستگاه‌ها صرفاً با یک دستگاه مرکزی صحبت کنند. با در نظر گرفتن این نکته و همچنین توضیحاتی که راجب این دو پروتکل داده شد، از **ESP-NOW** استفاده میکنیم. این پروتکل توسط دو میکروکنترلر ویموس تست شد و به درستی وظایف انتقال داده به صورت بیسیم را انجام داد. در این جا بنظر میرسد که همه چیز درست است و تنها کافیسیت کدها را ادغام کنیم. اما هنگام پیاده سازی قسمت نهایی به مشکلی اساسی برخوردیم. میکروکنترلر پردازنده هر دو وظیفه را به شکل همزمان انجام نمیداد! یعنی توانایی برقرار ارتباط با میکروی دیگر را داشت و دیتا را نیز دریافت میکرد و همینطور به شکل جداگانه توانایی ایجاد **IP** و یک صفحه ی وب را داشت اما وقتی زمان انجام هر دو وظیفه به شکل همزمان میرسید نه میتوانست به میکروی دیگر وصل شود و نه صفحه ی وب

را راه اندازی کند . با تحقیقاتی که در پی بروز این مشکل انجام شد ، مشخص شد که هر میکروکنترلر مبتنی بر تراشه ی **esp8266** تنها دارای یک آنتن است و از این آنتن تنها میتواند برای یک کار استفاده کند . زیرا مد های کاری آن در این دو حالت متفاوت است . یکی از امکاناتی که وجود داشت استفاده از وایفای آسنکرون بود. این روش موجب میشود که میکروکنترلر دائما مد خود را سوییچ کند و بدین ترتیب در واحد زمان تنها در یک مد فعالیت کند و بلافاصله آنرا عوض کند . قبلا در پدیده های مختلفی شاهد چنین تکنیکی بوده ایم . تکنیکی که سرعت سوییچ به نحوی بالا باشد که کاربر متوجه نشود و احساس کند که وظایف مورد نظر بطور همزمان در حال انجام است در حالی که در واقعیت اینچنین نیست . (مانند کپیدها) اما متاسفانه چنین روشی برای این نحوه از ارتباط بیسیم ممکن نیست. هرچند که انجام عمل سوییچینگ غیرممکن نیست اما انجام این کار به شکل وایرلس نیازمند زمان بیشتری است . و طبق تست های انجام شده سریع تر از 5 دقیقه نخواهد بود که خود ضعف بزرگی به شمار میآید. طبق تحقیقاتی که در پی پیدایش این مشکل انجام شد ، مشخص شد کاربرانی که خود درگیر این مشکل بوده اند ارتباط سیمی را برگزیده اند . هر چند که این مسئله بر پیچیدگی طراحی مدار افزود اما این روش از سریعترین و قابل اطمینان ترین روشها بشمار میرود. این روش به شرح زیر میباشد :

در هر صورت میکروکنترلر متصل به در باید اطلاعات را به شکل بیسیم ارسال کند و در اینکه پروتکل **ESP-NOW** بهترین انتخاب برای این نیاز پروژه است شکی نیست . . از طرفی هم نیاز داریم که میکروکنترلر مقصد حتما صفحه ی وبی مطابق نیازمندی پروژه تولید کند اما نمیتواند اینکار را همزمان با دریافت دیتای سنسور درب انجام دهد . به همین خاطر میتوانیم از یک میکروکنترلر کمکی استفاده کنیم . به این ترتیب که میکروکنترلر متصل به سایر سنسور ها دیتا را به شکل بیسیم از میکرووی متصل به سنسور درب دریافت کرده و سپس دیتای سنسورهای خود و سنسور درب را از طریق پروتکل های وابسته به سیم را به یک میکرووی کمکی ارسال کند و این میکرووی کمکی صفحه ی وب را ایجاد کند و اطلاعات دریافتی سنسور ها از طریق سیم را در آن نمایش دهد. در اینجا چالش جدیدی که به وجود می آید این است که بهترین پروتکل ارتباطی از طریق سیم که مناسب این پروژه باشد ، کدام است . از میان پروتکل های بسیاری که وجود دارد ، پروتکل ارتباطی **I2C** از پروتکل های معروفی است که بین مابقی پروتکل ها دارای بیشترین سرعت در انتقال داده میباشد. اما در حین پیاده سازی این روش مشخص شد که هیچ یک از میکروکنترلر های مبتنی بر

ESP8266 قادر به ایفای نقش در مد **SLAVE** در این پروتکل نیستند. این مشکلی بود که تا بعد از تهیه ی برد سوم و بارگذاری کدهای سمپل مربوطه در آن خود را نشان نداد. پیش از تهیه ی برد سوم، دیتاشیت آن بررسی شد و بنظر میرسید که در آن پین های مربوط به این پروتکل به خوبی پیاده سازی شده است. اما پس از جست و جو بین نظرات کاربران مشخص شد که این نوع از میکروکنترلر ها هرگز توانایی ایفای نقش **SLAVE** در این پروتکل را ندارند. اگر بخواهیم توضیح کوتاهی درباره ی نقش **SLAVE** دهیم، باید گفت که پروتکل **I2C** مبتنی بر ایفای نقش یک مستر و چند اسلیو بنا شده است. این روش از 4 سیم ارتباط بهره برده است و همیشه فقط یک عدد مستر و یک یا چندین اسلیو خواهیم داشت و نحوه ی برقراری ارتباط به این شکل است که شروع کننده و تمام کننده ی این ارتباط تنها مستر است و همچنین این مستر است که انتخاب میکند از کدام اسلیو دیتا دریافت کند یا به آن ارسال کند که این کار از طریق آدرسدهی اسلیو ها انجام میپذیرد. با خط خوردن این روش از لیست راه حل های ممکن دو انتخاب بعدی پروتکل های **SPI** و **UART** باقی ماندند. مشخص شد که روش ارتباطی **SPI** نیز به همین دلیل توانایی برقراری ارتباط را ندارد زیرا این روش نیز نیازمند ایفای نقش های مستر و اسلیو است. در آخر پروتکلی که باقی ماند پروتکل **UART** است که از طریق پین های مربوط به سریال قابل انجام است. بر خلاف دو روش قبل که تنها یک سیک برای انتقال دو طرفه ی دیتا لازم بود، در اینجا به دو سیم یک طرفه برای انتقال دیتا نیازمندیم. اما باید در نظر بگیریم که این پروتکل از پین های سریال استفاده میکند که خود یک نقطه ضعف برای این روش حساب میشود. همانطور که میدانیم هیچکدام از **IDE** های آردوینو توانایی انجام عملیات دیباگ را ندارند و بهترین و ساده ترین روش دیباگ کردن کد ها و گرفتن خروجی از عملیات میکروکنترلرها، استفاده از مانیتور سریال است. اما اگر بخواهیم از **UART** استفاده کنیم باید پین های مربوط به سریال را بکار بگیریم که این امر موجب عدم توانایی در استفاده از سریال در نقش دیباگر میشود. خوشبختانه برای حل این مشکل یک کتابخانه موسوم به **SerialSoftware** بوجود آمده که این امکان را میدهد که دو پین دیگر را بجای پین های **Rx** و **Tx** مقداردهی اولیه کرده و از آنها برای پیاده سازی این پروتکل استفاده کنیم. بدین ترتیب طراحی نهایی مدار ما به این شکل شد که میکروکنترلر متصل به درب اطلاعات مربوط به درب را از به شکل بیسیم و با استفاده از **ESP-NOW** به میکروی شامل مابقی سنسور ها میفرستند و این میکرو همه ی اطلاعات مربوط به سنسور ها را از طریق پروتکل ارتباطی **UART** به میکروی اصلی و پردازشگر میفرستد که این میکرو وظیفه ی

جمع آوری دیتای همه ی سنسور ها ، ایجاد صفحه ی وب ، ارسال اطلاعات به وب و نمایش آنها و در نهایت تصمیم گیری نسبت به سیگنال خروجی مربوط به جک برقی یا همان **LINEAR ACTUATOR** را دارد . لازم به ذکر است کد مربوط به وب در یک فایل **HTML** جداگانه نوشته میشود . سپس این فایل با استفاده از **SPIFFS** بر روی میکرو قابل اجرا میشود . **SPIFFS** در واقع ابزاری برای بارگذاری فایل ها بر روی حافظه ی میکروکنترلر است . این کتابخانه با داشتن متود های مخصوص این اجازه را میدهد که در هنگام نیاز فایل های مورد نیاز را از روی حافظه خوانده و بکار بگیریم .

فصل پنجم: معرفی روش پیاده سازی و تنظیمات سیستم

کدهای این پروژه به چهار فایل تقسیم میشود . فایل اول شامل کد مربوط به دریافت اطلاعات از سنسور و ارسال آن از طریق **ESP-NOW** میباشد .

بواسطه ی خط زیر **MAC ID** میکروی مقصد را تعریف میکنیم . ارتباط در **ESP-NOW** به صورت آدرسدهی **MAC ID** انجام میشود در حالی که در **ESP-MESH** این روش با اختصاص شماره ی گره به هر میکرو انجام میشود .

```
uint8_t broadcastAddress[] = {0xA0, 0x20, 0xA6, 0x10, 0x44, 0x42};
```

دریافت مک آدرس میکروی مقصد با استفاده از متود `WiFi.macAddress()` موجود در کتابخانه ی **ESP8266WiFi** انجام میشود.

قطعه کد زیر ساختار کلی پیامی که قرار است ارسال شود را نشان میدهد. در اصل ما یک **struct** را ارسال میکنیم که شامل وضعیت باز یا بسته بودن در میباشد . و از طرفی دیگر یک **ID** که بیانگر این است که کدام میکروکنترلر این دیتا را ارسال کرده است . این قسمت برای حالتی که بیش از یک درب داشته باشیم کاربرد دارد .

```
typedef struct struct_message {  
    int id;  
    int doorstate;  
} struct_message;
```

این تابع برای بیان وضعیت آخرین پکت ارسال شده کاربرد داشته و جنبه ی دیباگ و ارور یابی دارد .

```
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
```

خط زیر برای قرار دادن میکروکنترلر در مد ایستگاه وایفای است . خط زیر آن برای این است که مد موردنظر فعال شود .

```
WiFi.mode(WIFI_STA);  
WiFi.disconnect();
```

متود **esp_now_init** یا مقدار صفر یا یک را برمیگرداند. این متود برای بررسی این است که آیا این پروتکل در هنگام شروع بکار میکرو فعال شده است یا خیر .

```
if (esp_now_init() != 0) {  
    Serial.println("Error initializing ESP-NOW");  
    return;  
}  
if (esp_now_init() == 0) {  
    Serial.println("SUCCESS initializing ESP-NOW");  
    return;  
}
```

پس از راه اندازی موفقیت آمیز **ESP-NOW**، تابع **callback** را که هنگام ارسال پیام فراخوانی می شود، مشخص میکنیم . در این مورد، ما برای تابع **OnDataSent ()** که قبلا ایجاد شده است را به عنوان تابع **callback** مشخص کنیم. و سپس مطابق کد زیر به یک دستگاه مجهز به **ESP-NOW** دیگر متصل میکنیم تا آماده ی انتقال پیام شود .

```
esp_now_register_send_cb(OnDataSent);  
esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
```

esp_now_add_peer آرگومان های زیر را به ترتیب می پذیرد: مک آدرس، نقش، کانال وای فای، کلید و طول کلید.

در **loop** ، ما هر 2 ثانیه یک پیام از طریق **ESP-NOW** ارسال می کنیم که میتوانیم این مقدار را تغییر دهیم . در این قسمت میکروکنترلر دائما مقدار سنسور را میخواند و سپس مقدار خوانده شده را ارسال میکند .

```
int sensorValue = digitalRead(sensorPin); // read the sensor value  
Serial.println(sensorValue);
```

لازم بذکر است که دیتای ارسالی یک **struct** است. در اینجا مقادیری را که می خواهیم به داخل ساختار ارسال کنیم اختصاص می دهیم و در پایان دیتا را به شکل زیر ارسال میکنیم :

```
esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
```

کد میکروکنترلر اول در اینجا به پایان میرسد .

قسمت دوم که مورد بررسی قرار میگیرد مربوط به میکروکنترلر دوم است که شامل دریافت دیتا از طریق **ESP-NOW**، دریافت دیتاهای مابقی سنسور ها ، جمع آوری آن ها و در نهایت ارسال آنها از طریق پروتکل **UART** به میکروکنترلر سوم است .

پس از فراخوانی کتابخانه های مورد نظر به شکل زیر برای پروتکل **UART** دو پین خروجی معرفی میکنیم . همانطور که گفته شد **SerialSoftware** توانایی ایجاد پین های مخصوص سریال جدا از پینهای اصلی تعریف شده بر روی برد دارد .

```
// RX, TX pins for software serial
SoftwareSerial mySerial(D2, D3);
```

پس از آن برای سنسور **DHT** که وظیفه ی دریافت دما و رطوبت را دارد ، تایپ و پین های مورد نظر ورودی را مشخص میکنیم.

```
// Set DHT pin:
#define DHTPIN D4
// Set DHT type
#define DHTTYPE DHT11 // DHT 11
// Initialize DHT sensor for normal 16mhz
DHT dht = DHT(DHTPIN, DHTTYPE);
```

بعد از معرفی مابقی متغیر های مربوط به سنسور ها ، ساختاری مشابه ساختاری که در کد میکرو ی ارسال کننده ی دیتا از طریق **ESP-NOW** نوشته بودیم را وارد میکنیم . همیشه باید دقت داشته باشیم که ساختار دیتا باید در میکرو ی مبدا و میکرو ی مقصد به شکل یکسان نوشته شود .

```
typedef struct struct_message {
    int id;
    int doorstate ;
} struct_message;
```

یک تابع **callback** ایجاد میکنیم که زمانی که **ESP8266** داده ها را از طریق **ESP-NOW** دریافت می کند، فراخوانی می شود. این تابع **onDataRecv()** نامیده می شود و باید چندین پارامتر را به شرح زیر بپذیرد:

```
void onDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len)
```

محتوای **incomingData** را در متغیر **myData** کپی می کنیم :

```
memcpy(&myData, incomingData, sizeof(myData));
```

اکنون، ساختار **myData** شامل چندین متغیر با مقادیر ارسال شده توسط فرستنده است.

در قسمت **setup**، سرعت سریال اصلی میکرو و سریال انتزاعی که خودمان ایجاد کرده ایم را مقداردهی میکنیم.

```
Serial.begin(115200);  
mySerial.begin(9600); // Initialize the software serial port
```

در ادامه ی این قسمت پین های مربوط به سنسور ها را مشخص کرده و به میکروکنترلر اعلام میکنیم که این پینها به عنوان ورودی باید عمل کنند. مابقی قسمت **setup** مشابه با کد بخش اول است اعم مشخص کردن تابع **callback**، مشخص کردن مد وایفای و چک کردن فعال شدن **ESP-NOW**. در ضمن در این قسمت یک وقفه ی یک ثانیه ای قرار میدهیم تا سنسور ها فرصت راه اندازی داشته باشند در غیر اینصورت سنسور ها قادر به رد و بدل دیتای درست نخواهند بود .

در قسمت **loop**، دائما ورودی سنسور ها را میخوانیم و روی متغیر های مربوطه ی آن مینویسیم . سپس این مقادیر را از طریق پروتکل **UART**، انتقال میدهیم .در کد دو نوع استفاده از متد **println()** داریم . یکی از آنها توسط **Serial** فراخوانده شده است که این همان سریال اصلی میکروکنترلر است که جهت نمایش دیتاها بر روی مانیتور اصلی سریال است و دیگری **mySerial** است که با استفاده از این متود دیتا ها را از طریق **UART** به میکروکنترلر مقصد ارسال میکند .

```
Serial.println(movement);  
mySerial.println(movement);
```

در سومین بخش از کدهای این پروژه با میکروکنترلر اجرا کننده ی وب سروکار داریم . برای ساخت وب سرور از کتابخانه **ESPAsyncWebServer** استفاده می کنیم که راه آسانی برای ساخت یک وب سرور آسنکرون ارائه می دهد. ساخت یک وب سرور آسنکرون دارای مزایای متعددی است که مهم ترین آن مدیریت بیش از یک اتصال به طور همزمان است .

در این قسمت با فراخوانی کتابخانه های مورد نیاز شروع کرده و سپس مانند کد قسمت قبل با استفاده از

SerialSoftware پین های جدید برای دریافت داده از طریق **UART** را مشخص میکنیم.

```
// RX, TX pins for software serial
SoftwareSerial mySerial(D2, D3);
```

در ادامه نام وایفای و رمز عبور وای فای که میخواهیم با استفاده از آن **local IP** تولید کرده و صفحه ی وب را بر روی آن اجرا کنیم را وارد و متغیر هارا تعریف و یک شی **AsyncWebServer** در پورت 80 ایجاد میکنیم .

```
// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

const char* ssid = "poca";
const char* password = "12345678";
```

سپس برای هر متغیر که نیاز به نمایش در وب دارد ، تابعی تعریف میکنیم که خروجی مناسب را به شکل **string** برگرداند .

```
String getTemperature()
String getHumidity()
String getLight()
String getGas()
String getMovement()
String getDoor()
```

بعد از آن همه ی این توابع را در تابعی کلی تر به نام **processor** مورد استفاده قرار میدهیم . در اصل این تابع دریافت متغیرها و جایگذاری آنها در صفحه ی وب را مدیریت میکند .


```
String processor(const String& var){
  Serial.println(var);
  if (var == "TEMPERATURE"){
    | return getTemperature();
  }
  else if (var == "HUMIDITY"){
    | return getHumidity();
  }
  else if (var == "LIGHT"){
    | return getLight();
  }
  else if (var == "DOOR"){
    | return getDoor();
  }
  else if (var == "GAS"){
    | return getGas();
  }
  else if (var == "MOVEMENT"){
    | return getMovement();
  }
  }

  return String();
}
```

در قسمت **setup**، مانند قبل سرعت سریالها را مشخص کرده و پین های ورودی و خروجی را نیز مشخص میکنیم.

```
// Serial port for debugging purposes
Serial.begin(115200);          // Initialize the hardware serial port
mySerial.begin(9600);         // Initialize the software serial port

pinMode(A0, INPUT);
```

مطابق کد زیر اتصال **SPIFFS** را بررسی میکنیم و در صورت خطا پیام خطا را در مانیتور سریال پرینت میکنیم.

```
if(!SPIFFS.begin()){
  Serial.println("An Error has occurred while mounting SPIFFS");
  return;
}
```

در قطعه کد بعد سعی میکنیم اتصال وایفای را ایجاد کنیم و برای اینکار از اسم و رمز وایفای موجود در محل استفاده میکنیم و همچنین حلقه ای برای کانکشن وایفای ایجاد میکنیم که تا زمانی که اتصال رخ ندهد میکروکنترلر شروع بکار نکند و در صورت ایجاد اتصال شماره **IP** ی که میکروکنترلر تولید کرده است را در سریال پرینت میکنیم تا با وارد کردن آن در مرورگر به صفحه ی وب دسترسی داشته باشیم .

```
// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi..");
}
// Print ESP8266 Local IP Address
Serial.println(WiFi.localIP());
```

در قسمت بعد با استفاده از **server.on** فایل **html** مربوط به وب را که از قبل بر روی حافظه ی میکروکنترلر بارگذاری کرده ایم را میخوانیم. **Server.on** یک کالبد فانکشن را ست می کند. این متود در قسمت **setup** از برنامه فراخوانده میشود و موجب میشود کالبد فانکشن هرگاه رویداد خاصی رخ دهد اجرا شود. این اتفاق زمانی قابل رخ دادن است که یک **URL** خاص با متود **GET** درخواست می شود. هرگاه که درخواستی به وب سرور ارسال می شود، لیستی از رویدادهای ثبت شده توسط **"on"** برای موردی که با درخواست مطابقت دارد بررسی می شود، سپس تابع مرتبط اجرا می شود. و همه این کارها به صورت آسنکرون در پس زمینه در یک رشته دیگر انجام می شود.

```
// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor);
});
```

و بعد از این قدم با استفاده از همین متود مقادیر سنسورها را به صفحه ی وب ارسال میکنیم .
در قسمت **loop** از پروژه ، هر یک ثانیه دیتاها را از طریق **UART** با استفاده از متود **parseInt** و سریالی که از طریق کتابخانه ی **SerialSoftware** بوجود آورده ایم ، میخوانیم . این عمل با استفاده از متود **available** انجام میپذیرد . کار اصلی این متود چک کردن بافر مربوط به **uart** است که از طریق سیم **rx** دیتا دریافت میکند . در صورتی که دیتای جدیدی رسیده باشد و در بافر قرار گرفته باشد این متد مقدار 1 و در غیر این صورت صفر را برمیگرداند. در صورتی که مقداری که این متود در هنگام فراخوانی برمیگرداند یک باشد ، یعنی دیتای جدید دریافت شده پس با استفاده از **parseInt** اقدام به خواندن اطلاعات میکنیم .

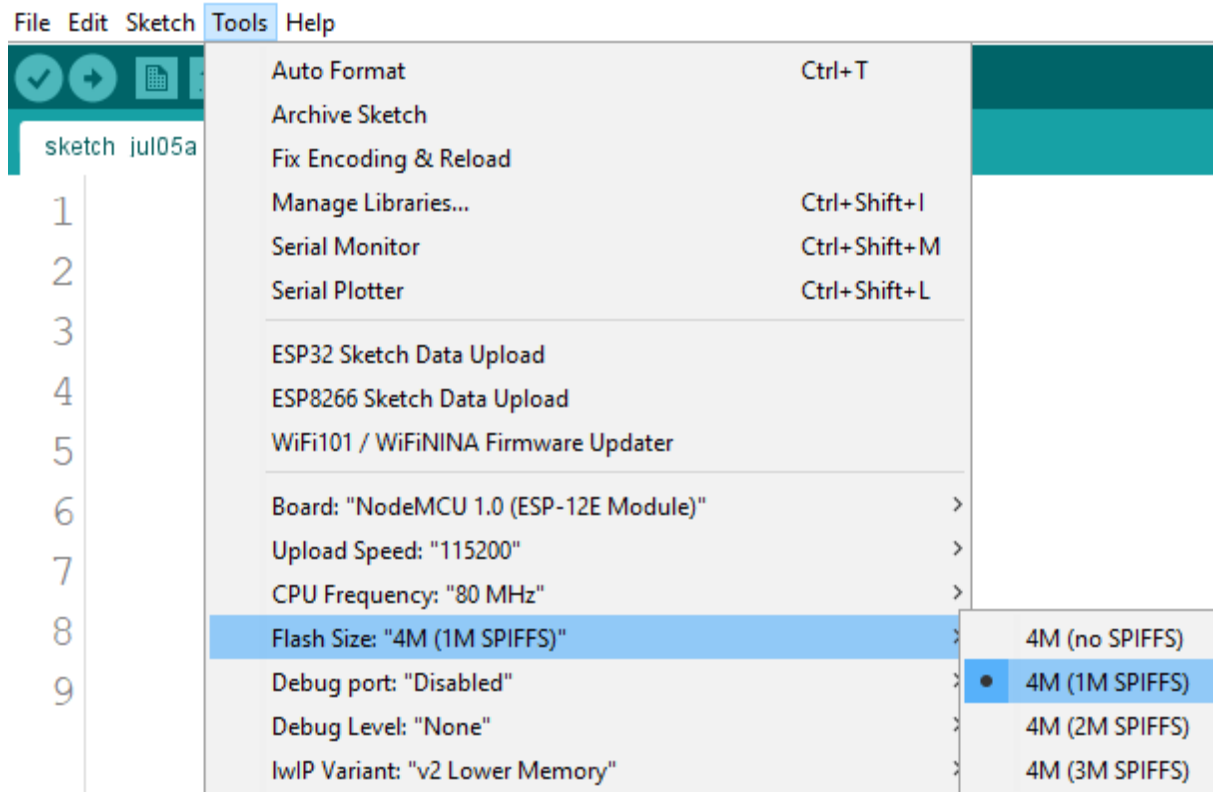
```
if (mySerial.available()) {
    movement = mySerial.parseInt(); // Integer
    door = mySerial.parseInt();
    h= mySerial.parseInt();
    t= mySerial.parseInt();
    Light = mySerial.parseInt();
}
```

برای خاموش و روشن کردن **Linear Actuator** باید یک سیگنال خروجی تولید کنیم که نحوه ی مقداردهی آن بر اساس یک یا چند سنسور باشد . نحوه ی تصمیم گیری و اعمال این تصمیمات امر دشواری نیست . میتوان معیار را

میزان را آلاینده های موجود در فضا یا میزان رطوبت یا میزان نور و دما قرار داد . ما در اینجا معیار تصمیم گیری خود را دما میگذاریم که اگر از حدی دما بیشتر بود این دستگاه اقدام به بستن پنجره کند یا در صورت کمتر بودن از مقدار مشخصی میتوانیم اقدام به باز کردن پنجره کنیم . برای این کار به دو پین خروجی برای کنترل آن نیاز داریم . در ابتدا شماره پینهای متصل جهت اعمال سیگنال را مشخص میکنیم و سپس با توجه به میزان دما نسبت به باز یا بسته بودن آن تصمیم گرفته و بر خروجی ها اعمال میکنیم .

```
if (t > open_door_temp) {  
  
    digitalWrite(relay_1, HIGH);  
    digitalWrite(relay_2, LOW);  
  
}  
  
else if (t < close_door_temp) {  
    digitalWrite(relay_1, LOW);  
    digitalWrite(relay_2, HIGH);  
}
```

در آخر کد **html** را با استفاده از **SPIFFS** در حافظه ی میکروکنترلر بارگذاری میکنیم . در ابتدا داخل پوشه مربوط به اسکچ یک پوشه به نام **data** ایجاد میکنیم . داخل پوشه **data** مکانی است که فایلهایی که باید به **ESP8266** ارسال شود قرار می گیرند..در محیط **IDE** و درمنوی **Tools** بسته به اندازه فایلهای خود مقدار حافظه **SPIFFS** را انتخاب میکنیم.



برای ارسال فایل کافیسست در محیط IDE بر روی **Tools > ESP8266 Sketch Data Upload** کلیک کنید.

در صورتیکه همه چیز به درستی انجام شده باشد در پنجره دیباگ پیامی به شکل زیر نمایش میدهد .

```

SPIFFS Image Uploaded
Compressed 2072576 bytes to 83023 bytes...
Writing at 0x00200000... (16 %)
Writing at 0x00204000... (33 %)
Writing at 0x00208000... (50 %)
Writing at 0x0020c000... (66 %)
Writing at 0x00210000... (83 %)
Writing at 0x00214000... (100 %)
Wrote 2072576 bytes (83023 compressed) at 0x00200000 in 12.4 seconds (effective 1333.6 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
  
```

کد **HTML** ی که در فولدر دیتا قرار داده شده است شامل کد **css** و **html** و **script** میباشد . به طور خلاصه، **css**

یک زبان طراحی است که باعث می شود یک وبسایت جذاب تر به نظر برسد. در حالی که **html** تا حد زیادی محتوای

متنی را تعیین می کند، **CSS** ساختار بصری، چیدمان و زیبایی شناسی را تعیین می کند. **html** یک زبان نشانه گذاری است و **CSS** یک زبان شیوه نامه است.

با استفاده از **CSS** کارتهایی طراحی میکنیم که اطلاعات هر سنسور را نمایش دهد و نحوه ی چیدمان کارتها را نیز مدیریت کند. برای کاهش پیچیدگی کد **CSS** را در فایل **html** نوشته ایم تا میکرو لازم باشد فقط یک فایل را فرا بخواند. کد **CSS** در سکشن **style** نوشته میشود.

```
<style>
html {font-family: Arial; display: inline-block; text-align: center;}
p { font-size: 1.2rem;}
body { margin: 0;}
.topnav { overflow: hidden; background-color: #2f4468; color: white; font-size: 1.7rem; }
.content { padding: 20px; }
.card { background-color: white; box-shadow: 2px 2px 12px 5px rgba(140,140,140,.5); border-radius: 20px }
.cards { max-width: 900px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit,
minmax(250px, 1fr)); }
.reading { font-size: 1.8rem;color: #133C88; }
.packet { color: #bebebe; }
.card.info { color: #16A3D0; margin: 0; }
.card.info2 { color: #16A3D0;max-width: 900px; margin: 0 auto; display: grid; grid-gap: 0rem; grid-template-columns:
repeat(auto-fit, minmax(900px, 1fr)); }
```

بطور کلی کد **html** از بخش های زیر تشکیل شده است. در بخش **head** مشخصات کلی و کد **CSS** نوشته شده و در قسمت **body** بدنه ی وبسایت و کد اسکریپت آن نوشته شده که مربوط به دریافت اطلاعات از میکروکنترلر و جایگذاری آن در کد وب است.

```
...<html> == $0
  <head>...</head>
  <body>
    <div class="topnav">...</div>
    <div class="content">...</div>
    <script>...</script>
  </body>
</html>
```

در قسمت **topnav**، هدر وبسایت نوشته میشود و در قسمت **content** بدنه ی اصلی وبسایت است که شامل کارتهای سنسورهاست که به شکل زیر نوشته میشود.

```

▼<div class="content">
  ▼<div class="cards"> grid
    ▶<div class="card info">...</div>
    ▶<div class="card info">...</div>
    ▶<div class="card info">...</div>
    ▶<div class="card info">...</div>
    ▶<div class="card info">...</div>
    ▶<div class="card info">...</div>
    ▶<div class="card info2">...</div> grid
  </div>

```

ساختار کلی هر کارت به شکل زیر نوشته میشود . در ابتدا تصویر مربوط به هر کارت نمایش داده میشود . یکی از مرسوم ترین روش نمایش تصویر قرار دادن آن در فولدر کد **html** و استفاده از آن در کد هست . این روش با استفاده از میکروکنترلر قابل پیاده سازی است اما لازم است همانطور که فایل **html** را با استفاده از **SPIFFS** در حافظه ی میکروکنترلر بارگذاری کرده ایم تصاویر را هم بارگذاری کنیم که این موجب اشغال شدن حافظه ی میکروکنترلر و همچنین کاهش سرعت بارگذاری صفحه ی وب میشود . به همین خاطر به جای انجام این کار از **Base64** استفاده میکنیم. **Base64** روشی برای رمزگذاری داده های باینری در متن **ASCII** است. در زمینه توسعه وب، تصاویر **Base64** به تصاویری اطلاق می شود که به عنوان یک رشته **Base64** کدگذاری شده اند. این رشته را می توان مستقیماً در کد **HTML** یک صفحه وب جاسازی کرد و بدون نیاز به فایل های تصویری جداگانه به صورت تصویر نمایش داد. نمایش تصاویر **Base64** در **HTML** چندین مزیت دارد :

زمان بارگذاری سریع تر صفحه: با قرار دادن مستقیم داده های تصویر در کد **html** ، می توانید نیاز به درخواست های **HTTP** جداگانه برای بازایی فایل های تصویر را از بین ببرید. این می تواند منجر به زمان بارگذاری صفحه سریعتر شود، به خصوص برای تصاویر کوچکتر.

افزایش امنیت: با رمزگذاری داده های تصویر در یک رشته **Base64**، می توانید داده های باینری واقعی را از کاربرانی که کد منبع صفحه وب شما را مشاهده می کنند پنهان کنید. این می تواند برای جلوگیری از دسترسی غیرمجاز به داده های حساس مفید باشد.

سازگاری با انواع مرورگرها : تصاویر **Base64** را می توان در تمام مرورگرها مختلف، صرف نظر از فرمت تصویر، نمایش داد. این می تواند به ویژه برای سازگاری بین مرورگرها مفید باشد، زیرا مرورگرهای مختلف ممکن است از فرمت های مختلف تصویر پشتیبانی متفاوتی داشته باشند.

توسعه ساده: در برخی موارد، رمزگذاری تصاویر به عنوان رشته های **Base64** در طول توسعه آسان تر است و سپس داده ها را مستقیماً در کد **html** خود قرار می دهید، نه اینکه نیاز به مدیریت فایل های تصویری جداگانه داشته باشید.

برای تبدیل تصویر به این فرمت از یک رمزگذار آنلاین **Base64** برای تبدیل داده های تصویر باینری به رشته **Base64** استفاده کردیم. نتیجه رشته ای از کاراکترها است که می توانند به راحتی در کد **HTML** جاسازی شوند.

```
<div class="card info">
  
  <h4>TEMPERATURE</h4>
  <p>
    <span class="reading" color: #133c88;>
      <span id="temperature"></span>
      " °C"
    </span>
  </p>
</div>
```

همانطور که در کد بالا مشخص است برای نوشتن مقادیر سنسور ها از **ID** مربوط به آن استفاده میکنیم . این **ID** در قسمت **script** مقدار دهی میشود. عنصر **script** برای جاسازی کد یا داده های ارسالی استفاده می شود. برای اسکریپت های کلاسیک، چونکه انجام دستورات وب به شکل آسنکرون اجرا میشود ، اسکریپت به صورت موازی اقدام به دریافت دیتا ها و ارسال آن به صفحه ی وب میکند یعنی در صورت در دسترس بودن دیتاهای ارسال بلافاصله اقدام به ارزیابی میکند .

```

▼<script>
    setInterval(function ( ) {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("temperature").innerHTML = this.responseText;
            }
        };
        xhttp.open("GET", "/temperature", true);
        xhttp.send();
    }, 10000 ) ;

```

در کد بالا ابتدا یک شی جدید از **XMLHttpRequest** ایجاد می کنیم که برای ارسال درخواست های **HTTP** به سرور استفاده می شود. سپس یک کنترلر به متود **onreadystatechange** از **XMLHttpRequest** اختصاص داده می شود. هر زمان که وضعیت **Request** تغییر کند، این تابع فراخوانی می شود. در این مورد، بررسی می شود که اگر **readyState** برابر با 4 است (که نشان می دهد درخواست کامل شده است) و در عین حال مقدار وضعیت برابر با 200 است (نشان دهنده پاسخ موفقیت آمیز)، دیتای دریافتی را بازبینی می کند و محتوای عنصری که ID آن برابر با **temperature** را برابر با این مقدار قرار می دهد.

خط زیر درخواست **AJAX** را آماده می کند. متد **HTTP** را روی **GET** و نقطه پایانی **URL** را به عنوان **"temperature"** مشخص می کند. مقدار پارامتر سوم را **true** می گذاریم که بیانگر آسنکرون بود کانکشن است و سپس درخواست **AJAX** را به سرور ارسال می کنیم.

```

xhttp.open("GET", "/temperature", true);
xhttp.send();

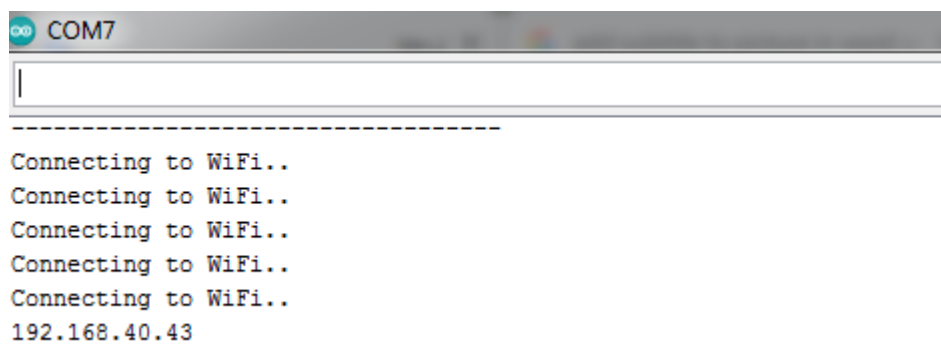
```

در نهایت، تابع به نحوی بسته می شود، که فاصله زمانی را بر حسب میلی ثانیه مشخص می کند. در این حالت، این فاصله روی 10000 میلی ثانیه (10 ثانیه) تنظیم شده است، بنابراین کل فرآیند هر 10 ثانیه تکرار می شود.

به طور خلاصه، این کد یک کار تکراری را انجام میدهد که هر 10 ثانیه یک درخواست **AJAX** را به `"temperature"` ارسال می کند. هنگامی که یک پاسخ موفقیت آمیز دریافت می شود، محتوای یک عنصر **HTML** با شناسه `"temperature"` را با دیتای دریافتی آپدیت می کند.

فصل پنجم: نتیجه گیری

پس از انجام و پیاده سازی قسمت های قبل ، اکنون با آپلود کد نهایی ، مانیتور سریال را باز میکنیم تا **local IP** را تولید کرده و آن را مانند شکل زیر نمایش دهد .



```
COM7
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
192.168.40.43
.
```

پس از وارد کردن **IP** بالا در مرورگر ، شاهد نتیجه ی نهایی خواهیم بود . همچنین در حین اجرای کد با استفاده از سریال و متود `println()` ، همه ی دیتا ها را پرینت میکنیم تا در صورت وجود تناقض میان داده های دریافتی و داده های موجود در وب ، متوجه آن شویم .

```
COM7
|
-----
door 0
gas :8.00
h:30
t:28
light :7
move; 0
-----
door 0
gas :15.00
h:30
t:28
light :6
move; 0
-----
door 0
gas :8.00
h:30
t:28
light :6
move; 0
-----
door 0
gas :15.00
h:30
t:28
light :5
move; 0
-----
door 0
gas :15.00
h:30
t:28
light :4
move; 0
-----
```

تصویر بالا آن چیزی است که در صفحه ی سریال مانیتور به نمایش گذاشته میشود و تصویر زیر نتایج نهایی صفحه ی وب است .

IOT HOUSE DASHBOARD



TEMPERATURE

27 °C



GAS POPULATION

15.00 %



MOVEMENT DETECTION

No movement
detected



LIGHT INTENSITY

6 %



DOOR SITUATION

Door is closed



HUMIDITY

32 %

نتیجه گیری و پیشنهادات:

آنچه که در انتهای این پروژه بیشتر به چشم میخورد ، به نتیجه رسیدن آن است که در حین انجام آن گاهی امری ناممکن بنظر می آمد . چالش اصلی پروژه طراحی نحوه ی پیاده سازی آن بود که به شکیبایی و تحقیق و آزمون و خطای بسیار نیاز داشت . در طی این پروژه با وجود پیش زمینه ی ذهنی از اینترنت اشیا و سابقه ی تحقیق و ارائه ی مباحث اولیه ی آن ، متوجه شدم آنطور که بر خلاف آنچه که بنظر می آید ، این رشته بسیار نوپاست و جای پیشرفت بسیاری دارد و شاید به همین دلیل است که این رشته امروزه به این اندازه قابل بحث و برجسته واقع شده است . محدودیت هایی که به ناگاه در حین پروژه خودنمایی میکردند ، ابدا قابل قبول نبودند و انتظار میرفت که چنین محدودیت هایی تا به الان در میکروکنترلر ها برطرف شده باشد . این محدودیت ها موجب شد که متوجه اهمیت طراحی بردها چه از نظر نرم افزاری و چه از نظر سخت افزاری بشوم و علاقه مند شدم تا بتوانم در آینده ای نزدیک تلاشی هرچند کوچک در راستای حل این مشکلات و رفع محدودیت ها داشته باشم .

این پروژه با توجه به بکار گیری مباحث جدید روز پتانسیل های بسیاری برای گسترش و افزودن امکانات بیشتر دارد . بطور مثال میتوان بجای یک وبسایت یک اپلیکیشن ساخت یا میتوان رویداد های هشدار آمیز را ایمیل کرد به تلفن همراه یا حتی اطلاعات را در دیتابیس ذخیره کرد . میتوان سیستم امنیتی اضافه کرد و حتی میتوان با استفاده از کارتخوان های مغناطیسی ورود و خروج ها را ثبت کنیم . براحتی میتوان سیستم آبیاری گلها را پیاده سازی کرد یا حتی با تهیه ی دوربین های مقرون به صرفه ، رویداد های خانه را به تلفن همراه ارسال کرد . همچنین میتوان با استفاده از ماژول های دریافت صدا با دریافت عبارات دستوری از کاربر ، دستورات لازم را اجرای کنیم . و این تنها گوشه ای از امکانات وسیعی است که میتوان با استفاده از اینترنت اشیا برای یک خانه در نظر گرفت. حال با تغییر محیط مورد هدف ، تصور کنید که با تغییر نیازمندی ها چه امکانات متفاوتی را میتوان برای آن محیط عرضه کرد . این مکان یک مدرسه ،

بیمارستان ، ماشین و یا حتی معدن باشد . (در مورد این امکانات به تفصیل در کتاب **IoT Fundamentals:**

Networking Technologies, Protocols, and Use Cases for the Internet of Things 1st

Edition مورد بحث و بررسی قرار گرفته است .

فهرست منابع:

سورس کد پیاده سازی شده برای **NODEMCU** در **ARDUINO IDE**

https://github.com/esp8266/Arduino/blob/master/variants/nodemcu/pins_arduino.h

سورس کد پیاده سازی شده برای **WEMOS** در **ARDUINO IDE**

https://github.com/esp8266/Arduino/blob/master/variants/d1_mini/pins_arduino.h#L32

آشنایی با **SPIFFS**

<http://freeelec.ir/?p=257>

<https://arduino-esp8266.readthedocs.io/en/latest/filesystem.html>

آشنایی با **ESP-NOW**

https://www.youtube.com/watch?v=Ydi0M3Xd_vs

<https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/>

<https://randomnerdtutorials.com/esp-now-two-way-communication-esp8266-nodemcu/>

آشنایی با **ESP-MESH**

<https://digispark.ir/getting-started-with-esp-mesh-and-arduino/>

<https://randomnerdtutorials.com/esp-mesh-esp32-esp8266-painlessmesh/>

آشنایی با **SOFTWARESERIAL**

<https://docs.arduino.cc/learn/built-in-libraries/software-serial>

<https://github.com/PaulStoffregen/SoftwareSerial>

آشنایی با **ASYNC WEB SERVER**

<https://techtutorialsx.com/2018/01/01/esp8266-arduino-asynchronous-http-web-server/>

<https://randomnerdtutorials.com/esp8266-nodemcu-async-web-server-espasynwebserver-library/>

<https://virgool.io/uiRobotics/esp8-cm814qewmemq>

فروشگاه تامین کننده ی قطعات

<https://aftabrayaneh.com/>

مشکل برقراری ارتباط با اینترنت در حین استفاده از **ESP-NOW**

<https://github.com/espressif/arduino-esp32/issues/878>

مشکل ارتباط 12C بین دو برد مبتنی بر **ESP8266**

<https://github.com/esp8266/Arduino/issues/5762>

مقایسه ی **ESP-NOW** و **ESP-MESH**

https://daneshjookit.com/blog/128_esp-wifi-range-esp-now-and-esp-mesh-protocol.html

آشنایی با اینترنت اشیا و برد های مبتنی بر ESP32 و ESP8266

<https://digispark.ir/>

ماژول سنسور حرکت

<https://thecaferobot.com/store/sr602-pyroelectric-infrared-motion-detector-sensor-module>

سنسور درب

<https://radkit.ir/product/%D8%B3%D9%86%D8%B3%D9%88%D8%B1-%D9%85%D8%BA%D9%86%D8%A7%D8%B7%DB%8C%D8%B3%DB%8C-%D8%AA%D8%B4%D8%AE%DB%8C%D8%B5-%D9%88%D8%B6%D8%B9%DB%8C%D8%AA-%D8%AF%D8%B1%D8%A8/>

سنسور نور

<https://eshop.eca.ir/%D8%B3%D8%A7%DB%8C%D8%B1-%D8%B3%D9%86%D8%B3%D9%88%D8%B1%D9%87%D8%A7/5829-%D9%85%D8%A7%DA%98%D9%88%D9%84-%D8%B3%D9%86%D8%B3%D9%88%D8%B1-%D9%86%D9%88%D8%B1-temt6000.html#:~:text=%D8%B3%D9%86%D8%B3%D9%88%D8%B1%20%D9%86%D9%88%D8%B1%20%D9%85%D8%AD%DB%8C%D8%B7%20TEMP600%20%D8%A7%DB%8C%D9%86,%D9%88%D8%B3%DB%8C%D8%B9%20%D8%B1%D9%88%D8%B4%D9%86%D8%A7%DB%8C%DB%8C%20%D9%BE%D8%A7%D8%B3%D8%AE%20%D9%85%DB%8C%20%D8%AF%D9%87%D8%AF.>

سنسور دما و رطوبت

<https://thecaferobot.com/learn/interfacing-dht11-temperature-humidity-sensor-arduino/#:~:text=%D8%B3%D9%86%D8%B3%D9%88%D8%B1%20DHT11%D8%8C%20%DB%8C%DA%A9%20%D8%B3%D9%86%D8%B3%D9%88%D8%B1%20%D8%AF%D9%85%D8%A7,2%20%D8%AF%D8%B1%D8%AC%D9%87%20%D8%A7%D9%86%D8%AF%D8%A7%D8%B2%D9%87%20%DA%AF%DB%8C%D8%B1%DB%8C%20%DA%A9%D9%86%D8%AF.>

سنسور گاز

<https://daneshjookit.com/module/sensor/%D8%B3%D9%86%D8%B3%D9%88%D8%B1-%DA%AF%D8%A7%D8%B2/1651-%D8%B3%D9%86%D8%B3%D9%88%D8%B1-%D8%AA%D8%B4%D8%AE%DB%8C%D8%B5-%DA%AF%D8%A7%D8%B2-mq2.html>

غیر قابل استناد ترین منبع

<https://poe.com/ChatGPT>