

دانشگاه زنجان

دانشکده فنی و مهندسی

پایان نامه کارشناسی

گروه کامپیوتر

مانیتورینگ و کنترل گلخانه

Greenhouse Monitoring and Controlling

تحقيق و نگارش :

امین شاهباقی

فرهاد جمالی

استاد راهنما :

دکتر علی امیری

شهریور ۱۴۰۱

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

فرم نهایی پروژه کارشناسی - گروه کامپیوتر دانشگاه زنجان

امین شاهباقی، فرهاد جمالی	نام و نام خانوادگی دانشجو
۹۷۴۶۳۱۱۸، ۹۷۴۶۳۱۳۵	شماره دانشجویی
مانیتور و کنترل گلخانه	عنوان پروژه
	نمره نهایی به عدد
	نمره نهایی به حروف
۱۴۰۱ شهریور	تاریخ دفاع
	نام و امضای استاد راهنما
	نام و امضای مدیر گروه

فهرست مطالب

۱.....	چکیده
۲.....	فصل اول) مقدمه
۳.....	۱-۱) شرح مسئله
۴.....	۱-۲) اهداف طراحی
	بخش اول : نرم افزار
۵.....	فصل دوم) پایگاه داده
۶.....	۲-۱) انتخاب پایگاه داده
۷.....	۲-۲) نحوه ارتباط با پایگاه داده
۸.....	۲-۲-۱) مزیت استفاده از ORM
۹.....	۲-۲-۲) ایجاد جداول در پایگاه داده
۱۰.....	۴-۲) مدل حساب کاربران
۱۱.....	۵-۲) مدل ذخیره اطلاعات سنسورها
۱۲.....	۵-۲-۱) مدل SensorType
۱۳.....	۵-۲-۲) مدل Sensor
۱۴.....	۵-۲-۳) مدل SensorValue
۱۵.....	۶-۲) مدل ذخیره اطلاعات دستگاهها
۱۶.....	۶-۲-۱) مدل DeviceType
	۶-۲-۲) مدل Device
۱۷.....	۶-۲-۳) مدل DeviceValues
۱۸.....	۷-۲) مدل اعلان ها
۱۹.....	۷-۲-۱) مدل SensorTypeRange
۲۰.....	۷-۲-۲) مدل AlarmMessage
۲۱.....	۸-۲) خلاصه
۲۲.....	فصل سوم) سرور (بکاند)
۲۳.....	۱-۳) تعاریف اولیه در جنگو

۱۶.....	۱-۱-۳) اپ‌ها
۱۷.....	۲-۱-۳) مدل‌ها
۱۷.....	۳-۱-۳) ویوها و URL‌ها
۱۷.....	۴-۱-۳) سریالایزرها (Serializer)
۱۷.....	۵-۱-۳) سیگنال‌ها
۱۸.....	۲-۲-۳) بررسی مدل‌ها
۱۸.....	۱-۲-۳) مدل‌های موجود در اپ account
۱۹.....	۲-۲-۳) مدل‌های موجود در اپ greenhouse
۲۲.....	۳-۳) ویوها
۲۲.....	۱-۳-۳) بررسی نحوه ورود و خروج کاربران در اپ account
۲۵.....	۲-۳-۳) بررسی ویوها در اپ greenhouse
۲۵.....	۱-۲-۳-۳) SensorValueInfoView
۲۶.....	۲-۲-۳-۳) DeviceValueInfoView
۲۷.....	۳-۲-۳-۳) AlarmMessageView
۲۸.....	۴-۲-۳-۳) AlarmMessageIsSeenView
۲۸.....	۵-۲-۳-۳) CountSensorDeviceView
۲۸.....	۶-۲-۳-۳) SensorTypeRangeView
۲۹.....	۳-۳-۳) بررسی سریالایزر
۳۰.....	۴-۳-۳) مسیرهای دسترسی به ویوها
۳۱.....	۴-۴-۳) به لحظه بودن داده‌ها
۳۱.....	۱-۴-۳) بررسی سیگنال‌ها
۳۳.....	۵-۳) خلاصه
۳۴.....	فصل چهارم) وب اپلیکیشن (فرانت‌اند)
۳۵.....	۱-۴) انتخاب فناوری فرانت‌اند
۳۵.....	۲-۴) مفاهیم پایه در ری‌اکت
۳۶.....	۱-۲-۴) کامپوننت‌ها (components)
۳۶.....	۲-۲-۴) وضعیت (state)
۳۶.....	۳-۲-۴) ویژگی (props)

۳۶.....	چیست؟ DOM (۴-۲-۴)
۳۷.....	مازی DOM (۵-۲-۴)
۳۸.....	context (۶-۲-۴)
۳۸.....	react-router-dom (۷-۲-۴)
۳۹.....	صفحه ورود (۳-۴)
۴۰.....	(۴-۴) ایجاد مسیر (route) های محافظت شده
۴۲.....	صفحه home (۵-۴)
۴۴.....	Dashboard کامپونت (۱-۵-۴)
۴۵.....	Analytics کامپونت (۲-۵-۴)
۴۶.....	HumdDash کامپونت (۳-۵-۴)
۴۷.....	TempDash کامپونت (۴-۵-۴)
۴۷.....	SensorSummary کامپونت (۵-۵-۴)
۴۸.....	Profile کامپونت (۶-۵-۴)
۴۹.....	صفحه ثبت اطلاعات دما (۶-۴)
۵۱.....	CustomDataTable کامپونت (۱-۶-۴)
۵۲.....	صفحه تنظیم محدوده هشدار برای مقادیر سنسورها (۷-۴)
۵۳.....	CounterRange کامپونت (۱-۷-۴)
۵۴.....	صفحه مشاهده هشدارها (۸-۴)
۵۶.....	خلاصه (۹-۴)

بخش دوم : سخت افزار

۵۸.....	فصل پنجم) آشنایی با قطعات
۵۹.....	۱-۵) شاکله و بستر آماده سازی
۶۰.....	۲-۵) برد رزبری پای (Raspberry Pi)
۶۱.....	۳-۵) سنسورها
۶۱.....	۱-۳-۵) سنسور DHT11
۶۲.....	۲-۳-۵) سنسور BH1750
۶۳.....	۴-۵) قطعات قابل کنترل
۶۴.....	۱-۴-۵) فن

۶۵.....	۲-۴-۵) پمپ آب
۶۶.....	۵-۵) مازول رله
۶۷.....	۶-۵) خلاصه
۶۸.....	فصل ششم) روند پیاده‌سازی
۶۹.....	۱-۶) پیاده‌سازی سنسور DHT11
۷۰.....	۲-۶) پیاده‌سازی سنسور BH1750
۷۰.....	۳-۶) پیاده‌سازی فن
۷۲.....	۴-۶) پیاده‌سازی پمپ آب
۷۲.....	۵-۶) خلاصه
۷۳.....	فصل هفتم) برنامه‌نویسی سخت‌افزار
۷۴.....	۱-۷) نمای کلی برنامه
۷۵.....	۲-۷) جزئیات هر برنامه
۷۵.....	۱-۲-۷) برنامه main
۷۶.....	۲-۲-۷) برنامه dht11_sensor
۷۷.....	۳-۲-۷) برنامه bh1750_sensor
۷۷.....	۴-۲-۷) برنامه device_ctrl
۷۹.....	۳-۷) خلاصه
۸۰.....	پیشنهادها
۸۱.....	فهرست منابع
۸۲.....	Abstract

فهرست اشکال و جداول

فصل دوم

۶.....	۱-۲) لوگو دیتابیس PostgreSQL
۷.....	۲-۲) مثالی از عملکرد ORM
۸.....	۳-۲) نمودار Erd مدل اکانت
۹.....	۴-۲) جدول SensorType و فیلدهای آن
۱۰.....	۵-۲) جدول Sensor
۱۰.....	۶-۲) جدول SensorValue
۱۰.....	۷-۲) نمودار Erd مدل سنسورها
۱۱.....	۸-۲) جدول DeviceType
۱۱.....	۹-۲) جدول Device
۱۲.....	۱۰-۲) جدول DeviceValue
۱۲.....	۱۱-۲) نمودار Erd قطعات قابل کنترل
۱۳.....	۱۲-۲) جدول محدوده هشدار
۱۳.....	۱۳-۲) جدول AlarmMessage
۱۴.....	۱۴-۲) نمودار Erd مدل‌های گلخانه

فصل سوم

۱۶.....	۱-۳) الگو و معماری داخلی فریمورک جنگو
۱۸.....	۲-۳) فایل مدل‌ها در هر دو اپ برنامه
۱۹.....	۳-۳) کدهای نوشته شده برای مدل Role
۲۰.....	۴-۳) کد مدل SensorType
۲۰.....	۵-۳) کد مدل Point
۲۰.....	۶-۳) کد مدل Sensor
۲۱.....	۷-۳) کد مدل SensorValue
۲۱.....	۸-۳) کد مدل DeviceType
۲۱.....	۹-۳) کد مدل‌های DeviceValue و Device
۲۲.....	۱۰-۳) کد مدل SensorTypeRange

۲۲.....	AlarmMessage کد مدل
۲۳.....	۱۲-۳) اضافه کردن تنظیمات احراز هویت به برنامه
۲۴.....	۱۳-۳) کدهای مربوط به پیاده‌سازی احراز هویت بر اساس token
۲۵.....	۱۴-۳) کد مربوط ارسال و دریافت مقادیر سنسورها
۲۶.....	۱۵-۳) تست ویو گرفتن مقادیر سنسورها در نرم‌افزار پستمن
۲۷.....	۱۶-۳) کد مربوط ارسال و دریافت وضعیت قطعات
۲۸.....	۱۷-۳) کد ویو مدیریت اعلان‌ها
۲۹.....	۱۸-۳) کد هشدارهای دیده نشده
۳۰.....	۱۹-۳) کد شمارش تعداد سنسورها و قطعات
۳۱.....	۲۰-۳) کد مربوط به محدوده هشدار بر اساس هر نوع سنسور
۳۲.....	۲۱-۳) کد سریالایزر مربوط به محدوده هشدار برای هر نوع سنسور
۳۳.....	۲۲-۳) کد تمام مسیرهای سیستم در اپ greenhouse
۳۴.....	۲۳-۳) کد مربوط به asgi.py که حالت اجرا برنامه را به صورت async قرار می‌دهد
۳۵.....	۲۴-۳) کد سیگنال سنسور جهت ارسال به قسمت کلاینت
۳۶.....	۲۵-۳) کد تابع ایجاد هشدار
۳۷.....	۲۶-۳) کد سیگنال مربوط به تغییر وضعیت فن و پمپ

فصل چهارم)

۳۸.....	۴-۴) لогو کتابخانه ری‌اکت
۳۹.....	۴-۴) نمودار درختی DOM
۴۰.....	۴-۴) Virtual DOM and DOM
۴۱.....	۴-۴) صفحه ورود به سیستم
۴۲.....	۴-۴) نمایش پیام خطأ در صورت اقدام به ورود با اطلاعات اشتباہ
۴۳.....	۴-۴) کد سمت فرانت برای احراز هویت به هنگام ورود
۴۴.....	۴-۴) کد مربوط به بررسی توکن
۴۵.....	۴-۴) کد بخش PrivateRoute برای جلوگیری از ورود غیرمجاز به بخش‌های مختلف سیستم
۴۶.....	۴-۴) مسیرهای داخلی سمت فرانت
۴۷.....	۴-۴) نمودار درختی کامپوننت‌های ساخته شده برای سیستم
۴۸.....	۴-۴) صفحه Home برنامه که به صورت یک داشبورد پیاده‌سازی شده است

۴۴.....	(۱۲-۴) کد کامپونت Home
۴۵.....	(۱۳-۴) کد کامپونت Dashboard
۴۵.....	(۱۴-۴) سوئیچ‌های برنامه که در وضعیت خاموش قرار دارند.
۴۵.....	(۱۵-۴) یکی از سوئیچ‌ها در وضعیت روشن
۴۶.....	(۱۶-۴) نمودار رطوبت هوا مربوط به کامپونت HumdDash
۴۶.....	(۱۷-۴) کد ریکوئست به سرور برای دریافت اطلاعات رطوبت هوا
۴۷.....	(۱۸-۴) نمودار دما مربوط به کامپونت TempDash
۴۸.....	(۱۹-۴) تصویر اجرایی از آخرین داده از هر سنسور
۴۸.....	(۲۰-۴) تصویر اجرایی از اطلاعات تعداد سنسورها و سوئیچ‌ها و هشدارها
۴۹.....	(۲۱-۴) کد بخش Profile
۵۰.....	(۲۲-۴) صفحه ثبت اطلاعات دما
۵۰.....	(۲۳-۴) کد صفحه اطلاعات دما
۵۱.....	(۲۴-۴) کد مربوط به CustomDataTable
۵۲.....	(۲۵-۴) صفحه تنظیم محدوده هشدار سنسورها
۵۳.....	(۲۶-۴) کد صفحه محدوده هشدارها
۵۳.....	(۲۷-۴) پیغام موفقیت در ثبت محدوده جدید هشدار برای سنسور
۵۴.....	(۲۸-۴) کد بخش ویرایش و بهروزرسانی محدوده هشدار
۵۵.....	(۲۹-۴) صفحه مشاهده هشدارها
۵۵.....	(۳۰-۴) نمایش جدولی هشدارها

فصل پنجم)

۵۹.....	(۱-۵) نمونه قفسه کتاب
۵۹.....	(۲-۵) نمایی از یک گلخانه مدرن
۵۹.....	(۳-۵) ردیف‌های گلخانه
۶۰.....	(۴-۵) تصویر برد رزبری‌پای چهار مدل B
۶۱.....	(۵-۵) اسامی و نوع پایه‌های GPIO
۶۱.....	(۶-۵) سنسور DHT11
۶۲.....	(۷-۵) سنسور BH1750
۶۳.....	(۸-۵) عملکرد پایه‌های سنسور BH1750

۶۴.....	۹-۵ فن ۱۲ ولت دی سی
۶۵.....	۱۰-۵ پمپ آب ۱۲ ولت
۶۶.....	۱۱-۵ شلنگ برای انتقال آب به سمت گیاه
۶۶.....	۱۲-۵ مازول رله ۲ کاناله

فصل ششم)

۶۹.....	۱-۶ اتصال سنسور DHT11 به برد
۷۰.....	۲-۶ اتصال سنسور BH1750 به برد
۷۱.....	۳-۶ پیاده سازی فن
۷۲.....	۴-۶ پیاده سازی پمپ آب

فصل هفتم)

۷۴.....	۱-۷ شماتیک کلی برنامه سمت سخت افزار
۷۵.....	۲-۷ برنامه Main که به گونه ای hub برنامه های دیگر است
۷۶.....	۳-۷ کد ارسال دما و رطوبت هوا
۷۷.....	۴-۷ کد ارسال شدت نور
۷۸.....	۵-۷ کد برنامه کنترل قطعات

چکیده

انسان همواره به دنبال مراقبت و نظارت از سرمایه‌های خود در طول سالیان مختلف بوده است، واژه سرمایه می‌تواند معانی مختلفی را در برگیرد و بسته به زمان می‌تواند شامل بسیاری از مادیات شود. از آن زمان تاکنون انسان‌ها برای پاسبانی از سرمایه‌های خود روش‌های گوناگونی را امتحان کرده‌اند؛ روش‌هایی که بعضًا نیازمند منابع انسانی بسیار، هزینه‌های گزارف چه در بحث مادی و چه در زمان، بدین ترتیب انسان همواره به دنبال روشی مناسب و جایگزین بوده است.

با ورود به عصر الکترونیک و کامپیوتر که منجر به رسیدن به درجات بالا در زمینه‌های مختلف پژوهشی و پژوهشی شد، صنعت نیز بی‌تفاوت نماند. صاحبان کسب‌وکارها با استفاده از این مزیت موفق به ابداع روشی شدند که سختی‌های روش‌های سنتی را نداشت و بیشتر متکی به تکنولوژی بود. بدین گونه گلخانه‌ها، گاوداری‌ها، پرورش ماهی و اتاق سرور با مجهر شدن به سنسورهای مختلف و راههای ارتباطی از طریق اینترنت، موفق به پیدا کردن روشی از مانیتور(نظرارت) و کنترل شدند که می‌توانست به صورت دقیق و با کمترین خطأ وضعیت حال حاضر سرمایه یک انسان را از طریق پارامترهای مختلف به نمایش بگذارد.

یکی از مثال‌های متدالوی که با این روش بسیار تطابق پیداکرده است، گلخانه است. به‌طوری‌که گلخانه‌های قدیمی و سنتی نیازمند منابع انسانی زیاد برای کنترل نظیر: آبرسانی، چرخش هوا و تهویه، ورود نور بوده است. امروزه افراد از به‌وسیله رایانه و تلفن هوشمند خود می‌توانند بر گلخانه خود نظارت داشته و حتی بر آن کنترل نیز داشته باشند.

در این پژوهه سعی شده است تا فرایند نظارت و کنترل یک گلخانه در مقیاس کوچک شبیه‌سازی و به نمایش گذاشته شود.

کلمات کلیدی : سرمایه، مانیتور، نظارت، کنترل، گلخانه

فصل اول)

مقدمه

۱-۱) شرح مسئله

روش‌های سنتی در موضوع نظارت و کنترل به مرور با روش‌های نوین کامپیوتری جایگزین شده‌اند، در این پژوهه نیز به دنبال ارائه روشی مدرن و کارآمد در زمینه نگهداری از گلخانه‌ها هستیم. روشی که در آن افراد می‌توانند تعداد کارگرانی که برای کنترل محیط نیاز هست را کاهش دهند و دیگر برای کنترل، مدیریت و حفظ گلخانه به کارگران زیادی نیاز نداشته باشند. گلخانه‌های مدرن هوشمند امروزی خودکار هستند و به افراد اجازه می‌دهند گلخانه‌ها را راحت‌تر مدیریت کنند و محیط آن را به کمک یک دکمه مدیریت نمایند. این نوع گلخانه‌ها از نیمه‌خودکار تا تمام‌خودکار متغیر خواهند بود.

۱-۲) اهداف طراحی

با رواج اینترنت اشیا انقلاب بزرگی در بسیاری از صنایع شکل گرفت. کشاورزی یکی از مهم‌ترین صنایعی است که با پیدایش اینترنت اشیا مورد تغییر و تحول بسیار زیادی قرار گرفت. در طراحی این سیستم سعی شده است تا با شبیه‌سازی یک نقطه از گلخانه و قرار دادن سنسور نور، دما و رطوبت هوا، این پارامترها را مانیتور می‌کنیم. یک فن و پمپ آب نیز قرار می‌دهیم که به صورت دستی آن‌ها را کنترل می‌کنیم. مانیتور و کنترل کردن دستگاه‌ها(فن و پمپ آب) از طریق وبسایت انجام می‌شود.

بخش اول

نرم افزار

فصل دوم)

پایگاه داده

۲-۱) انتخاب پایگاه داده

مدل داده‌ای و رابطه‌هایی که بین موجودیت‌ها در سیستم مانیتور و کنترل گلخانه وجود دارد، با ساختار پایگاه داده‌های رابطه‌ای متناسب است به همین دلیل از دیتابیس^۱ PostgreSQL استفاده می‌کنیم. PostgreSQL یک سیستم مدیریت پایگاه داده رابطه‌ای رایگان و متن‌باز به حساب می‌آید که مبتنی بر SQL است.



شکل ۲-۱) لوگو دیتابیس PostgreSQL

۲-۲) نحوه ارتباط با پایگاه داده

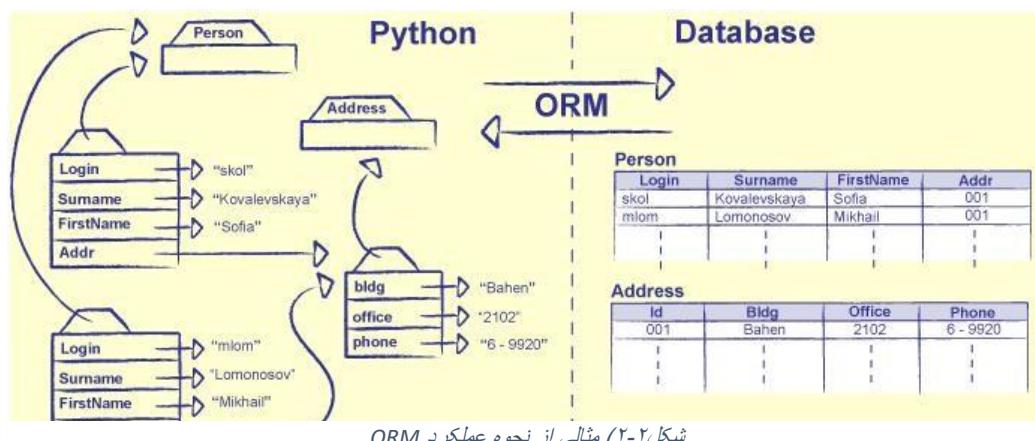
¹ Database

² Structured Query Language

برای متصل شدن به پایگاه داده و انجام عملیات درج، حذف، بهروزرسانی و بازیابی اطلاعات از ORM^۳ استفاده می‌کنیم.

۱-۲-۲) مزیت استفاده از ORM

- در صورت عدم استفاده از ORM مجبور هستیم برای ارتباط با پایگاه داده کوئری‌هایی به زبان SQL بنویسیم و به نحوی مستقیماً نرمافزار خود را به پایگاه داده متصل کنیم.
- اگر بخواهیم پایگاه داده را عوض کنیم مجبور هستیم کوئری‌های خود را عوض کنیم زیرا در بعضی موارد سینتکس^۴ پایگاه داده‌ها باهم متفاوت می‌باشد.
- زمانی که کد را با سینتکس ORM به کار می‌بریم خودش مسئولیت ارتباط با پایگاه داده و ذخیره و بازیابی اطلاعات را دارد و در پس‌زمینه دستورات ما را به کوئری‌های^۵ پایگاه داده‌ای که در تنظیمات تنظیم می‌کنیم تبدیل می‌کند.
- وقتی که می‌خواهیم پایگاه داده نرمافزار را تغییر دهیم فقط کافی است نوع دیتابیس را در تنظیمات تغییر دهیم.



شکل ۲-۲) مثالی از نحوه عملکرد ORM

³ Object-relational mapping

⁴ Syntax

⁵ Query

۳-۲) ایجاد جداول در پایگاه داده

جداول در پایگاه داده پس از طی مراحلی در سمت سرور به صورت زیر ایجاد می‌شود:
پس از نوشتن کد مربوط به مدل‌ها که در فایل `models.py` هر آپ⁶ موجود است؛ ابتدا با اجرا کردن دستور `python manage.py makemigrations` در کنسول های هر آپ ایجاد می‌شود و پس از اجرا کردن دستور `python manage.py migrate` جداول‌ها در پایگاه داده ایجاد می‌شود.

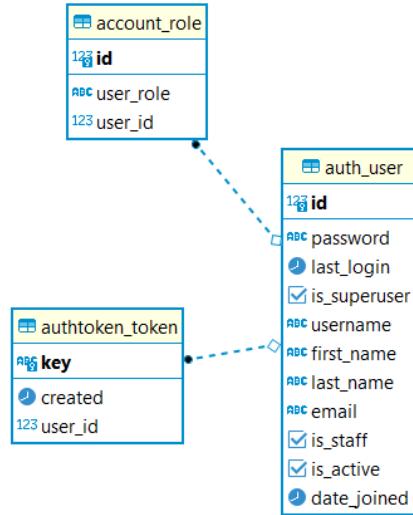
۴-۲) مدل حساب کاربران

برای بحث ورود و خروج از مدل `User` پیش‌فرض جنگو⁷ استفاده شده است و برای هر بار ورود و خروج یک توکن به هر فرد اختصاص می‌یابد. (در فصل سرور بیشتر دربارهٔ توکن⁸ صحبت خواهیم کرد) برای اعمال محدودیت دسترسی برای کاربران، نقش تعیین می‌شود.

⁶ App

⁷ Django, Python-based web framework

⁸ Token



شکل ۲-۳) نمودار ERD برای مدل اکانت و بوزر

ارتباط user با role یک به چند است یعنی هر user می‌تواند چند نقش داشته باشد. ارتباط user با token یک به یک است یعنی هر user می‌تواند یک توکن داشته باشد.

۲-۵) مدل ذخیره اطلاعات سنسورها

در گلخانه نقاطی را داریم که در آنجا سنسور قرار دارد. سنسورها اطلاعات را جمع‌آوری می‌کنند. برای ذخیره این اطلاعات نیازمند ۳ جدول هستیم که در ادامه به بررسی هر یک می‌پردازیم.

۱-۵-۲) مدل SensorType

این مدل شامل عنوان و توضیح مختصری درباره نوع سنسور می‌باشد.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123id	1	bigserial			[v]	nextv...	
ABCtitle	2	varchar(30)		default	[v]		
ABCdescription	3	varchar(255)		default	[]		

شکل ۲-۴) جدول SensorType و فیلد های مربوط به آن

Sensor (۲-۵-۲) مدل

این مدل شامل شناسه، نام ، کلید خارجی^۹ به point و کلید خارجی به SensorType می باشد. در واقع هر سنسوری که در سیستم وجود دارد در اینجا ذخیره می شود.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123id	1	bigserial			[v]	nextv...	
ABCsensor_id	2	varchar(7)		default	[v]		
ABCname	3	varchar(50)		default	[v]		
123point_id	4	int8			[v]		
123sensor_type_id	5	int8			[v]		

شکل ۲-۵) جدول Sensor و فیلد های مربوطه

SensorValue (۳-۵-۲) مدل

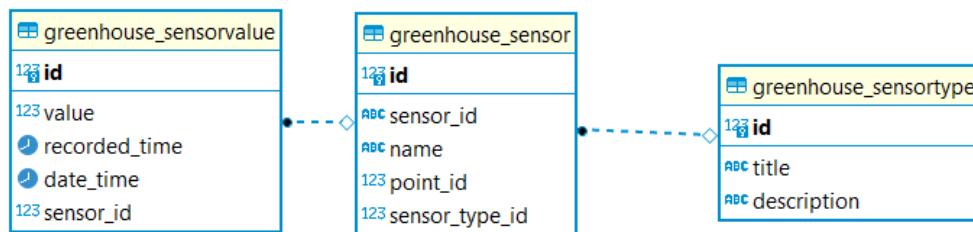
این مدل شامل مقدار دریافت شده از سنسور، تاریخ دریافت ، زمان دریافت و کلید خارجی به sensor می باشد.

⁹ Foreign Key

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123 id	1	bigserial			[v]	nextv...	
123 value	2	float8			[v]		
⌚ recorded_time	3	time			[v]		
⌚ date_time	4	date			[v]		
123 sensor_id	5	int8			[v]		

شکل ۲-۶) جدول *SensorValue* و فیلد های مربوطه

درنهایت نمودار ERD مربوط به این سه جدول به صورت زیر است.



شکل ۲-۷) نمودار ERD مدل سنسورها

۶-۲) مدل ذخیره اطلاعات قطعات قابل کنترل

همانند مدل قبلی برای ذخیره اطلاعات قطعات نیازمند ذخیره اطلاعات در ۳ جدول هستیم.

۱-۶-۲) DeviceType مدل

این مدل شامل عنوان و توضیح مختصری درباره نوع دستگاه می باشد.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123id	1	bigserial			[v]	nextv...	
ABCtitle	2	varchar(30)		<u>default</u>	[v]		
ABCdescription	3	varchar(255)		<u>default</u>	[]		

شکل ۱۰-۲) جدول DeviceType و فیلدهای مریبوطه

Device (۲-۶-۲) مدل

این مدل شامل شناسه، نام، کلید خارجی به point و کلید خارجی به deviceType می‌باشد. درواقع هر قطعه‌ی قابل کنترلی که در سیستم وجود دارد در اینجا ذخیره می‌شود.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123id	1	bigserial			[v]	nextv...	
ABCdevice_id	2	varchar(7)		<u>default</u>	[v]		
ABCname	3	varchar(50)		<u>default</u>	[v]		
123device_type_id	4	int8			[v]		
123point_id	5	int8			[v]		

شکل ۹-۲) جدول Device و فیلدهای مریبوطه

DeviceValue (۳-۶-۲) مدل

این مدل شامل وضعیت روشن یا خاموش بودن قطعه، تاریخ آخرین تغییر، زمان آخرین تغییر و کلید خارجی به device می‌باشد.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123id	1	bigserial			[v]	nextv...	
✓status	2	bool			[v]		
⌚recorded_time	3	time			[v]		
⌚date_time	4	date			[v]		
123device_id	5	int8			[v]		

شکل ۱۰-۲) جدول DeviceValue و فیلدهای مریبوطه

نمودار ERD حاصل از این ۳ جدول به صورت زیر است.



شکل ۱۱-۲) نمودار ERD مربوط به مدل قطعات قابل کنترل

۷-۲) مدل اعلان‌ها

برای اینکه بتوانیم بر روی مقادیر دریافت شده از سنسورها کنترل داشته باشیم نیازمند این هستیم که یک محدوده مشخص برای هر نوع سنسور مشخص کنیم و در صورتی که مقدار دریافت شده از سنسورها از محدوده معین خارج شد هشدار بدھیم. برای این منظور به ۲ جدول دیگر نیازمندیم.

SensorTypeRange (۱-۷-۲) مدل

این مدل شامل کمینه مقدار تعیین شده، بیشینه مقدار تعیین شده و کلید خارجی به sensorType می باشد.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123 id	1	bigserial			[v]	nextv...	
123 min_range	2	int4			[v]		
123 max_range	3	int4			[v]		
123 sensor_type_id	4	int8			[v]		

شکل ۱۲-۲) جدول محدوده هشدار برای هر نوع سنسور

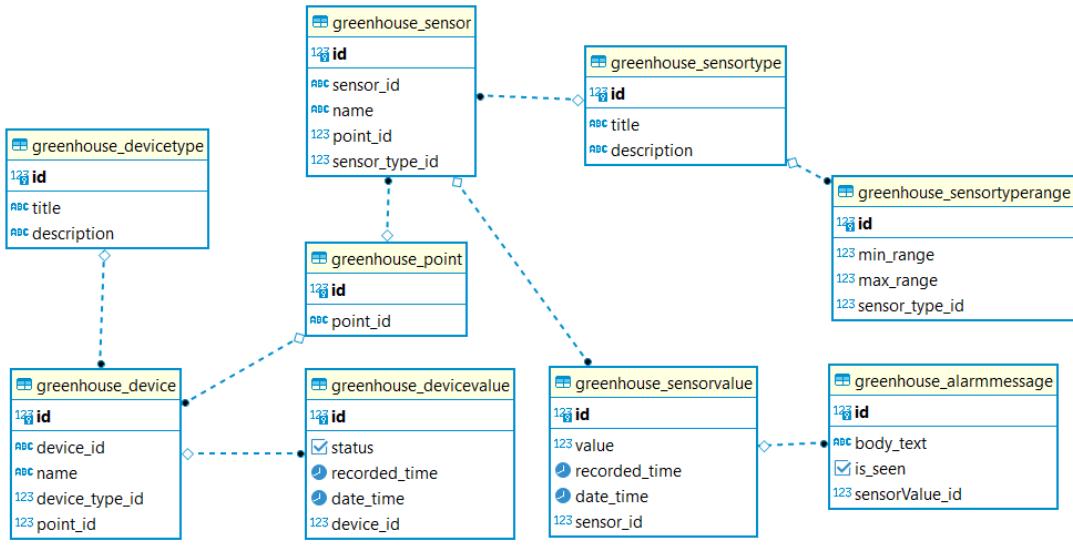
AlarmMessage (۲-۷-۲) مدل

این مدل شامل متن هشدار، وضعیت دیده شدن اعلان و کلید خارجی به SensorValue می باشد.

Column Name	#	Data type	Identity	Collation	Not Null	Default	Comment
123 id	1	bigserial			[v]	nextv...	
abc body_text	2	varchar(255)		default	[v]		
checkbox is_seen	6	bool			[v]		
123 sensorValue_id	7	int8			[v]		

شکل ۱۳-۲) جدول AlarmMessage و فیلد های مربوطه

نمودار ERD مربوط به اپ گلخانه به صورت زیر می باشد.



شکل ۲-۱۳) نمودار ERD مربوط به مدل‌های سمت گلخانه

۸-۲) خلاصه فصل

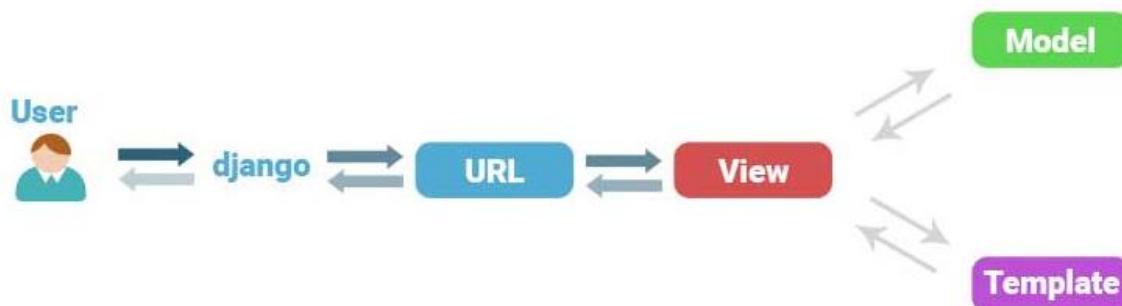
در این فصل ابتدا به بررسی علت انتخاب ، نحوه ارتباط و ایجاد جدول‌ها در پایگاه داده صحبت کردیم و سپس نحوه ارتباط مدل‌ها و جدول‌ها با یکدیگر و اینکه هر جدول شامل چه فیلد‌هایی است را موردنبررسی قراردادیم و در انتهای ساختار کلی نمودار رابطه‌ای موجودیت‌ها را در قالب یک شکل مشاهده کردیم.

فصل سوم

سرور (بکاند)

۱-۳) تعاریف اولیه در جنگو

برای پیاده‌سازی سمت سرور این پروژه از فریم‌ورک^۱ جنگو استفاده شده است که محبوب‌ترین فریم‌ورک بکاند در زبان پایتون^۲ است. یکی از دلایل محبوبیت جنگو، ایجاد یک ساختار منظم و مشخص برای برنامه‌نویس است تا از پیچیدگی‌های بیهوده جلوگیری کند و روند توسعه را ساده سازد. اگرچه برای درک سازوکار با هر زبان و فریم‌ورکی نیاز به آشنایی با مفاهیم و ادبیات مفهومی آن فناوری داریم.



شکل ۱-۳) الگو و معماری داخلی فریم‌ورک جنگو

۱-۱-۳) اپ‌ها (apps)

هر پروژه جنگو از تعدادی قطعات کوچک‌تر به نام app تشکیل شده که هر یک از این قطعات وظیفه‌ی مشخصی را بر عهده خواهد داشت. اپ‌هایی نیز موجودند که در بسیاری از پروژه‌ها مشترک‌اند و به عنوان *third party*, در پروژه‌ی جنگو به کار می‌روند.

¹ Framework

¹

² Python, Programming Language

²

(models) مدل‌ها (۲-۱-۳)

برای ذخیره اشیاء در پایگاه داده از مدل استفاده می‌کنیم. مدل‌ها در جنگو کلاس‌های به خصوصی هستند که در آن، اشیائی که می‌خواهیم در پایگاه داده باشند، پیاده‌سازی می‌شوند. مدل‌ها در فایلی به نام `models.py` پیاده‌سازی می‌شوند و ORM جنگو که در فصل دوم توضیحاتی داده شد از طریق همین مدل‌ها جداول و روابط دیتابیسی پروژه را ایجاد می‌کند.

URL‌ها و ویوها (۳-۱-۳)

برای اتصال ویوهای برنامه به مسیرهای قابل دسترسی، از URL‌ها استفاده می‌شود. منظور از ویو برنامه، تمامی کدهایی که برای کنترل ریکوئست‌ها^۱ ارسال و دریافت اطلاعات میان کلاینت^۲ و سرور نوشته می‌شود است و در فایلی به نام `views.py` پیاده‌سازی می‌شود؛ ویوها به نوعی منطق مربوط به هر آدرس در فایل `urls.py` است.

(serializers) سریالایزرها (۴-۱-۳)

کاربرد سریالایزرها تبدیل کوئری‌ست‌ها به فرمت json است و عملیات اعتبار سنجی بر روی داده‌ها را زمانی که از سمت کلاینت به صورت یک ریکوئست فرستاده می‌شود انجام می‌دهند.

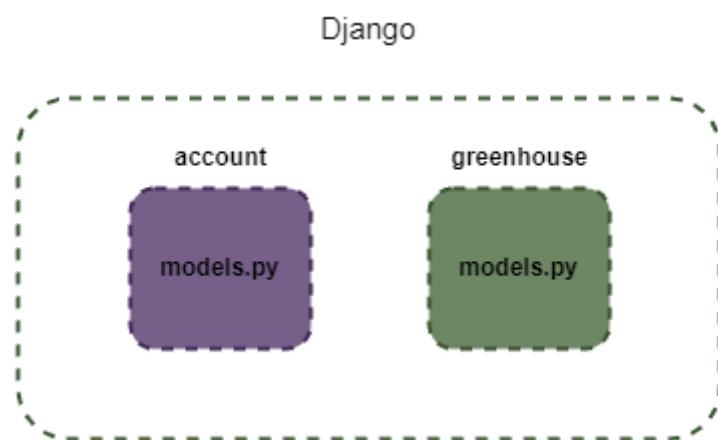
(signals) سیگنال‌ها (۵-۱-۳)

با تعریف کردن سیگنال‌ها زمانی که رخدادی^۳ در برنامه ایجاد شود مطلع می‌شویم. این رخداد می‌تواند از زمینه‌های مختلفی داشته باشد به طور مثال رخداد ایجاد نمونه از مدل برنامه، که این امکان را داریم که به صورت ضمنی کارهایی که بعد از تغییر مدنظر داریم که بسته به هدف برنامه کارهای مختلفی انجام داد.

¹ Uniform Resource Locator	3
¹ Http Request	4
¹ Client	5
¹ Event	6

۲-۳) بررسی مدل‌های پروژه

در این قسمت نگاهی دقیق به نحوه پیاده‌سازی مدل‌های این پروژه می‌اندازیم و مدل‌های هر دو اپ account و greenhouse را با تصویری از کد مدل‌ها بررسی می‌نماییم.



شکل ۲-۳) فایل مدل‌ها در هر دو اپ برنامه

۱-۲-۳) مدل‌های اپ account

در جنگو مدلی پیش‌فرض به نام User قرار دارد که این مدل ویژگی‌هایی نظیر نام، نامخانوادگی، یوزرنیم^{۱۷}، پسورد^{۱۸}، ایمیل و ... به صورت از پیش آمده برای سهولت توسعه بکار گرفته می‌شود و دیگر نیازی نیست چنین مدلی دوباره پیاده‌سازی شود. در کنار مدل آمده User مدل دیگری به نام Role پیاده‌سازی شده است که این مدل برای سطح دسترسی به قسمت‌های مختلف برنامه را به کاربران برنامه را می‌دهد.

¹ Username
¹ Password

7

8

هر user می‌تواند چند نقش (Role) داشته باشد. طبق الگوی oop هر کلاس داری ویژگی‌ها و متدهایی است. ویژگی‌ها در خطوط جدا از هم به این صورت که در ابتدا اسم آن ویژگی و در ادامه نوع فیلد آن و در انتهای در داخل پرانتز پارامترهای مربوط به آن ویژگی نوشته می‌شوند. متدهای str در واقع وظیفه نمایش هر شئ موجود در کلاس بهصورت رشته‌ای را بر عهده دارد. این کلاس بعداً توسط ORM به جدول‌های پایگاه داده تبدیل می‌شوند.

```
● ● ●

from django.db import models
from django.contrib.auth.models import User

class Role(models.Model):
    class UserRoleChoice(models.TextChoices):
        normal = 'N', 'normal'
        admin = 'A', 'admin'

    user_role = models.CharField(max_length=3, choices=UserRoleChoice.choices,
                                 default=UserRoleChoice.normal)
    user = models.ForeignKey(User, on_delete=models.DO_NOTHING)

    def __str__(self):
        return f"user_role: {self.user_role}, record: {self.pk}, ( user: {self.user.username} )"
```

شکل ۳-۳) کدهای نوشته شده برای مدل Role

کاربران در این برنامه می‌توانند دو نقش داشته باشند یک نقش معمولی و یک نقش ادمین:^{۱۹}

۲-۲-۳) مدل‌های اپ greenhouse

همان‌طور که در فصل قبل به ساختار جدول‌ها اشاره شد برای ذخیره‌سازی اطلاعات سنسورها، دستگاه‌ها و هشدارها و اعلان‌ها مدل‌هایی را در جنگو پیاده‌سازی می‌کنیم. تا بتوانیم فیلد‌هایی که مدنظر داریم در پایگاه داده ساخته شوند به ویژگی‌های هر کلاس اختصاص یابد.

```

● ● ●
from django.db import models

class SensorType(models.Model):
    title = models.CharField(max_length=30) # e.g : humidity, lux, temprature
    description = models.CharField(max_length=255, null=True)

    def __str__(self):
        return f'{self.title}, record: {self.pk}'

```

شکل ۴-۳) کد مدل *SensorType*

```

class Point(models.Model):
    point_id = models.CharField(max_length=7) # e.g : PNT-1

    def __str__(self):
        return f'{self.point_id}, record: {self.pk}'

```

شکل ۵-۳) کد مدل *Point*

مدل *Point* درواقع بیانگر محدوده‌هایی در گلخانه است که در آنجا سنسورها را قرار داده‌ایم و هر گلخانه به چند نقطه تقسیم می‌شود.

```

class Sensor(models.Model):
    sensor_id = models.CharField(max_length=7) # e.g : HUM-1, LUX-1, TMP-1
    name = models.CharField(max_length=50) # e.g : DHT11, BH1750
    sensor_type = models.ForeignKey(SensorType, on_delete=models.DO_NOTHING)
    point = models.ForeignKey(Point, on_delete=models.DO_NOTHING)

    def __str__(self):
        return f'{self.name}, type: {self.sensor_type.title}, record: {self.pk} '

```

شکل ۶-۳) کد مدل *Sensor*

آیدی هر سنسور ترکیبی از نوع سنسور به صورت خلاصه شده بعلاوه خط فاصله و یک عدد که بیانگر شماره ترتیب آن سنسور از نوع خودش است.

```
class SensorValue(models.Model):
    sensor = models.ForeignKey(Sensor, related_name='sensor_values', on_delete=models.DO_NOTHING)
    value = models.FloatField()
    recorded_time = models.TimeField()
    date_time = models.DateField()

    def __str__(self):
        return f"value: {self.value}, record: {self.pk} (sensor_id: {self.sensor})"
```

شکل ۷-۳) کد مدل SensorValue

مقادیر سنسورها در این مدل مدیریت می شود و کلید خارجی سنسور که مشخص می کند این مقدار متعلق به کدام یک از سنسورها می باشد.

```
class DeviceType(models.Model):
    title = models.CharField(max_length=30) # e.g : fan, pump
    description = models.CharField(max_length=255, null=True)

    def __str__(self):
        return f"{self.title}, record: {self.pk}"
```

شکل ۸-۳) کد مدل DeviceType

نوع قطعه کنترلی را مشخص می کند.

```
class Device(models.Model):
    device_id = models.CharField(max_length=7) # e.g : FAN-1, WPMP-1      need this?
    name = models.CharField(max_length=50)
    device_type = models.ForeignKey(DeviceType, on_delete=models.DO_NOTHING)
    point = models.ForeignKey(Point, on_delete=models.DO_NOTHING)

    def __str__(self):
        return f"{self.name}, record: {self.pk}"

class DeviceValue(models.Model):
    device = models.ForeignKey(Device, on_delete=models.DO_NOTHING)
    status = models.BooleanField()
    recorded_time = models.TimeField()
    date_time = models.DateField()

    def __str__(self):
        return f"status: {self.status}, record: {self.pk} ( device_id: {self.device} )"
```

شکل ۹-۳) کد مدل های Device و DeviceValue

این مدل‌ها مانند مدل‌های Sensor و SensorType پیاده‌سازی شده‌اند و بجای مقدار value فیلد status پیاده‌سازی شده‌اند و جایگزین شده است.

```
class SensorTypeRange(models.Model):
    min_range = models.IntegerField()
    max_range = models.IntegerField()
    sensor_type = models.OneToOneField(SensorType, on_delete=models.DO_NOTHING)

    def __str__(self):
        return f"record: {self.pk}, min_range: {self.min_range}, max_range: {self.max_range} (
sensor_type: {self.sensor_type})"
```

شکل ۱۰-۳) کد مدل SensorTypeRange

این مدل مرز و آستانه تحمل گلخانه بر حسب پارامتر مربوطه می‌باشد و برای هر سنسور مقدار بیشینه و کمینه‌ای در نظر گرفته شده است تا در صورت عبور از این مقادیر، هشدار و اعلانی ایجاد شود.

```
class AlarmMessage(models.Model):
    body_text = models.CharField(max_length=255)
    sensorValue = models.OneToOneField(SensorValue, on_delete=models.DO_NOTHING)
    is_seen = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.body_text[:12]}...", record: {self.pk}"
```

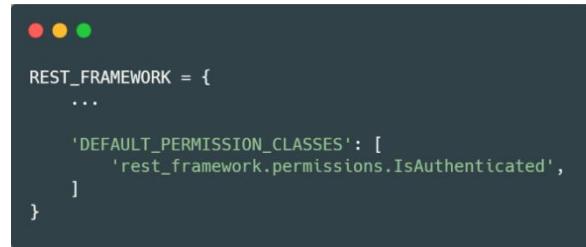
شکل ۱۱-۳) کد مدل AlarmMessage

۳-۳) بررسی ویوهای پروژه

نحوه نوشتمن ویوها در جنگو به دو صورت است:

روش اول به صورت class based views(CBV) و روشن دوم به صورت function based views(FBV)

در این پروژه از CBV استفاده می‌کنیم زیرا خواندن کد در CBV واضح‌تر و راحت‌تر است. همچنین برای تمامی ویوها محدودیت دسترسی اعمال می‌کنیم که کاربر باید حتماً احراز هویت شده باشد. برای این منظور مقادیر زیر را در تنظیمات اضافه می‌کنیم.



```
REST_FRAMEWORK = {
    ...
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}
```

شکل ۱۲-۳) اضافه کردن تنظیمات احراز هویت به برنامه

۱-۳-۳) بررسی نحوه ورود و خروج کاربران در اپ account

برای ورود به ازای هر فرد زمانی که کاربر نام کاربری و رمز عبور خود را وارد می‌کند؛ در صورت درست بودن اطلاعات یک توکن برای هر فرد ساخته می‌شود و برای خروج هم توکن فرد منقضی شده و عملاً پاک می‌شود.

دو روش کلی و پرکاربرد اعتبارسنجی سمت سرور، برای برنامه‌های سمت کاربر وب وجود دارند:

- روش اول Cookie-Based Authentication که پرکاربردترین روش بوده و در این حالت به ازای هر درخواست، یک کوکی جهت اعتبارسنجی کاربر به سمت سرور ارسال می‌شود (و برعکس).
- روش دوم Token-Based Authentication که بر مبنای ارسال یک توکن امضا شده به سرور، به ازای هر درخواست است.

برای استفاده از توکن ObtainAuthToken را از rest_framework.authtoken.views ایمپورت می‌کنیم.

```

from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.generics import GenericAPIView
from rest_framework.permissions import IsAuthenticated

from rest_framework.authtoken.views import ObtainAuthToken
from rest_framework.authtoken.models import Token
from .models import Role


class LogoutAPIView(GenericAPIView):

    permission_classes = (IsAuthenticated, )

    def post(self, request):
        request.user.auth_token.delete()
        return Response(data={'message': f"Bye {request.user.username}!"})


class CustomObtainAuthTokenView(ObtainAuthToken):

    def post(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data['user']
        token, created = Token.objects.get_or_create(user=user)
        roles = Role.objects.select_related('user_id').filter(user_id=user.id).values_list('user_role',
flat=True)
        return Response({'token': token.key, 'roles': roles})

```

شکل ۳-۱۳) کدهای مربوط به پیادهسازی احراز هویت بر اساس token

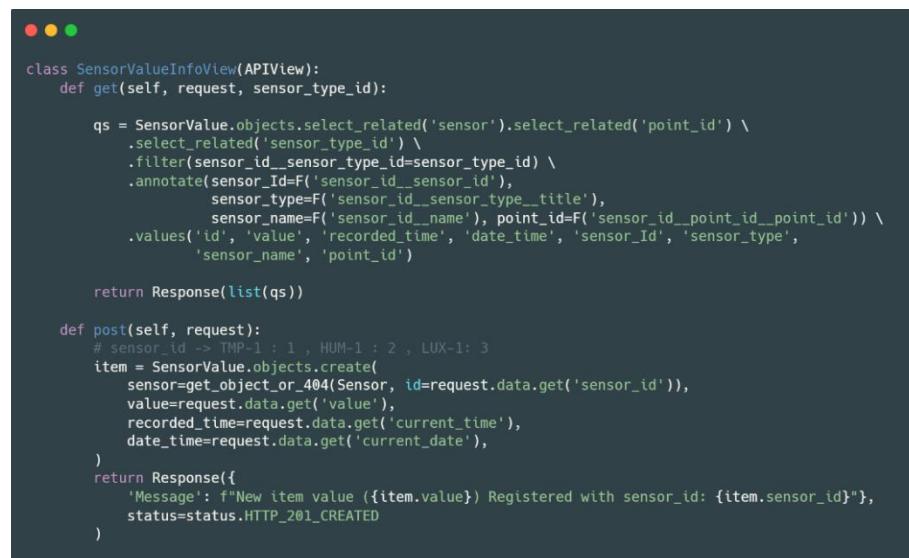
کلاس LogoutAPIView برای زمانی است که کاربر قصد خروج از سیستم را دارد و توکن هنگام خروج پاک می‌شود. کلاس CustomObtainAuthTokenView هم برای زمان ورود کاربر است و یک توکن به کاربر اختصاص می‌یابد و همچنین نقش کاربر تعیین می‌شود تا دسترسی‌هایش مشخص شود.

۳-۳-۲) بررسی ویوها در اپ greenhouse

با توجه به این‌که موضوع پروژه مانیتور و کنترل کردن می‌باشد؛ نیازمند ویوهايی هستيم که بتواند اطلاعات را بر اساس درخواست‌هايي که از سمت فرانت می‌آيد از مدل‌های مختلف جمع‌آوری کرده و پاسخ مناسب را در قالب json بازگرداند.

۳-۳-۱) SensorValueInfoView

در اين ويو اگر درخواست از نوع get باشد؛ يك کوئري به پايگاه داده می‌زند و نتيجه که اطلاعات مرتبط با سنسور شامل نام سنسور، مقدار عددی دريافت شده از سنسور، تاريخ ، زمان و نقطه قرارگيري سنسور بهصورت يك کوئري است بازگردايده می‌شود. در اينجا ازتابع values() استفاده کردیم که مقادير را در قالب json برمی‌گرداند. درصورتی که درخواست از نوع post باشد؛ مقادير موردنظر برای سنسور را در پايگاه داده ذخیره می‌کند.



```
class SensorValueInfoView(APIView):
    def get(self, request, sensor_type_id):
        qs = SensorValue.objects.select_related('sensor').select_related('point_id') \
            .select_related('sensor_type_id') \
            .filter(sensor_id__sensor_type_id=sensor_type_id) \
            .annotate(sensor_Id=F('sensor_id__sensor_type__title'),
                     sensor_name=F('sensor_id__name'), point_id=F('sensor_id__point_id__point_id')) \
            .values('id', 'value', 'recorded_time', 'date_time', 'sensor_Id', 'sensor_type',
                   'sensor_name', 'point_id')

        return Response(list(qs))

    def post(self, request):
        # sensor_id -> TMP-1 : 1 , HUM-1 : 2 , LUX-1: 3
        item = SensorValue.objects.create(
            sensor=get_object_or_404(Sensor, id=request.data.get('sensor_id')),
            value=request.data.get('value'),
            recorded_time=request.data.get('current_time'),
            date_time=request.data.get('current_date'),
        )
        return Response({
            'Message': f"New item value ({item.value}) Registered with sensor_id: {item.sensor_id}",
            status=status.HTTP_201_CREATED
        })
```

شکل ۳-۱۴) کد مربوط ارسال و دریافت مقادیر سنسورها

برای اینکه گفته‌های بالا بیشتر معلوم باشد در نرم‌افزار پستمن یک درخواست به ویو SensorValueInfoView می‌زنیم و پاسخ آن را به صورت زیر می‌گیریم:

The screenshot shows a Postman interface with the following details:

- URL:** `http://127.0.0.1:8000/api/SensorValueInfo/1`
- Method:** GET
- Headers:**
 - Authorization: token 68503333b84dbc711a48ad968289fed9d698c690
 - Key
 - Value
 - Description
- Body (Pretty):**

```

1 {
2   "id": 31,
3   "value": 22.0,
4   "recorded_time": "20:25:34",
5   "date_time": "2021-12-23",
6   "sensor_Id": "TMP-1",
7   "sensor_type": "Temperature",
8   "sensor_name": "DHT11",
9   "point_id": "PNT-1"
10 },
11 {
12   "id": 32,
13 }
```
- Response Status:** 200 OK
- Response Time:** 130 ms
- Response Size:** 2.98 KB

شکل ۱۵-۳) تست ویو گرفتن مقادیر سنسورها در نرم‌افزار پستمن

DeviceValueInfoView (۲-۲-۳-۳)

در این ویو مشابه حالت قبل اگر درخواست از نوع get باشد، یک کوئری به پایگاه داده می‌زند و نتیجه که اطلاعات مرتبط با سنسور شامل نام سنسور، حالت روشن یا خاموش بودن دستگاه، تاریخ، زمان و نقطه قرارگیری دستگاه به صورت یک کوئری است بازگردانده می‌شود.

در اینجا از تابع `values()` استفاده کردیم که مقادیر را در قالب json بر می‌گرداند. در صورتی که درخواست از نوع put باشد، وضعیت دستگاه عوض می‌شود و بعد در پایگاه داده ذخیره می‌کند.

```

● ● ●

class DeviceValueInfoView(APIView):
    def get(self, request, device_type_id):
        qs = DeviceValue.objects.select_related('device').select_related('point_id') \
            .select_related('device_type_id') \
            .filter(device_id__device_type_id=device_type_id) \
            .annotate(device_Id=F('device_id__device_id'),
                     device_type=F('device_id__device_type__title'),
                     device_name=F('device_id__name'),
                     point_id=F('device_id__point_id__point_id')) \
            .values('id', 'status', 'recorded_time', 'date_time', 'device_Id', 'device_type',
                   'device_name', 'point_id')

        return Response(list(qs))

    def put(self, request, device_id):
        # device_id -> FAN-1 : 1 , WMPM-1 : 2
        item = get_object_or_404(DeviceValue, device=device_id)
        item.status = False if item.status == True else True
        item.date_time = str(JalaliDate.today())
        item.recorded_time = (datetime.now()).strftime('%H:%M:%S')
        item.save()
        return Response({'message': 'updated successfully!', 'status': item.status, 'device_id': item.device_id})

```

شکل ۳-۱۶) کد مربوط ارسال و دریافت وضعیت قطعات

AlarmMessageView (۳-۲-۳-۳)

اگر درخواست get باشد اطلاعات مربوط به هشدارهای ایجادشده را برمی‌گرداند. در صورتی که درخواست put باشد یعنی هشدارها دیده شده و مقدار is_seen به True تغییر می‌یابد.

```

● ● ●

class AlarmMessageView(APIView):
    def get(self, request):
        qs = AlarmMessage.objects.select_related('sensorValue'). \
            annotate(sensor_id=F('sensorValue_id__sensor_id'),
                    value=F('sensorValue_id__value'),
                    recorded_time=F('sensorValue_id__recorded_time'),
                    date_time=F('sensorValue_id__date_time')).values()

        return Response(list(qs))

    def put(self, request):
        AlarmMessage.objects.filter(is_seen=False).update(is_seen=True)
        return Response({'message': 'updated successfully!'})

```

شکل ۳-۱۷) کد ویر مدیریت اعلان‌ها

AlarmMessageIsSeenView (۴-۲-۳-۳)

این ویو هشدارهایی که دیده نشده است را برمی‌گرداند.



```
class AlarmMessageIsSeenView(APIView):
    def get(self, request):
        qs = AlarmMessage.objects.select_related('sensorValue').filter(is_seen=False) \
            .annotate(value=F('sensorValue__value'),
                     recorded_time=F('sensorValue__recorded_time'),
                     date_time=F('sensorValue__date_time')).values()
        return Response(list(qs))
```

شکل ۱۸-۳) کد هشدارهای دیده نشده

CountSensorDeviceView (۵-۲-۳-۳)

شمارش تعداد سنسورها و قطعات قابل کنترل بر عهده این ویو است.



```
class CountSensorDeviceView(APIView):
    def get(self, request):
        device_count = Device.objects.all().count()
        sensor_count = Sensor.objects.all().count()

        return Response({'device_count': device_count, 'sensor_count': sensor_count})
```

شکل ۱۹-۳) کد شمارش تعداد سنسورها و قطعات

SensorTypeRangeView (۶-۲-۳-۳)

اگر درخواست از نوع `get` باشد برای هر نوع از سنسورها مقدار بازه کمینه و بیشینه را برمی‌گرداند. در صورتی که درخواست از نوع `put` باشد ابتدا مقادیر فرستاده شده را به سریالایزر می‌فرستد تا عمل صحت سنجی را برای مقادیر انجام دهد در صورت معتبر بودن اطلاعات آن را ذخیره می‌کند.

```

● ● ●

class SensorTypeRangeView(APIView):
    permission_classes = (IsAdminUser,)

    def get(self, request):
        qs = SensorTypeRange.objects.select_related('sensor_type') \
            .annotate(title=F('sensor_type__title')).values()

        return Response(list(qs))

    def put(self, request, sensor_type_range_id):
        sensor_type_range = get_object_or_404(SensorTypeRange, id=sensor_type_range_id)
        serializer = SensorTypeRangeSerializer(
            instance=sensor_type_range,
            data=request.data,
            partial=True
        )

        if serializer.is_valid():
            serializer.save()
            return Response({'message': 'updated successfully!', 'data': serializer.data})

        return Response({'message': serializer.errors})

```

شکل ۲۰-۳) کد مربوط به محدوده هشدار بر اساس هر نوع سنسور

۳-۳-۳) بررسی سریالایزر

در ویو SensorTypeRangeView از سریالایزر SensorTypeRangeSerializer استفاده کردیم زیرا می‌خواستیم هنگام ثبت داده در پایگاه داده عمل اعتبار سنجی بر روی آن‌ها اعمال کنیم. در این کلاس اول بررسی می‌کنیم که فیلد min_range کوچک‌تر از max_range باشد. دوم اینکه مقادیر min_range و max_range بزرگ‌تر از صفر باشد.

```

● ● ●

from rest_framework import serializers
from .models import SensorTypeRange

class SensorTypeRangeSerializer(serializers.ModelSerializer):
    class Meta:
        model = SensorTypeRange
        exclude = ['sensor_type']

    def validate(self, data):
        if data.get('min_range', 0) > data.get('max_range', 0):
            error = 'Maximum range should be greater than minimum range'
            raise serializers.ValidationError(error)

        return data

    def validate_min_range(self, value):
        if value < 0:
            raise serializers.ValidationError('Minimum range must be positive')

        return value

    def validate_max_range(self, value):
        if value < 0:
            raise serializers.ValidationError('Maximum range must be positive')

        return value

```

شکل ۳-۲۱) کد سریالایزر مربوط به محدوده هشدار برای هر نوع سنسور

۳-۳-۴) مسیرهای دسترسی به ویوها

این مسیرها که در فایل urls.py در اپ greenhouse قرار دارند و هر مسیر به یک ویو متصل است.

```

● ● ●

path('SensorValueInfo/<int:sensor_type_id>/', SensorValueInfoView.as_view()),
path('SensorValueInfo/', SensorValueInfoView.as_view()),

path('DeviceValueInfo/<int:device_type_id>/', DeviceValueInfoView.as_view()),
path('DeviceValueInfo/update/<int:device_id>/', DeviceValueInfoView.as_view()),

path('AlarmMessage/', AlarmMessageView.as_view()),
path('AlarmMessage/update/', AlarmMessageView.as_view()),
path('AlarmMessage/notSeen/', AlarmMessageIsSeenView.as_view()),

path('CountSensorDevice/', CountSensorDeviceView.as_view()),

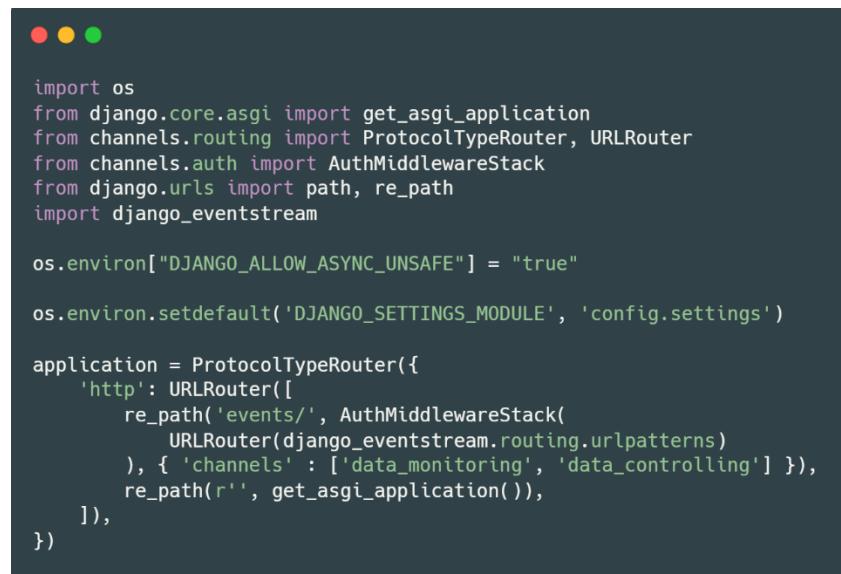
path('SensorTypeRange/', SensorTypeRangeView.as_view()),
path('SensorTypeRange/update/<int:sensor_type_range_id>/', SensorTypeRangeView.as_view()),

```

شکل ۳-۲۲) کد تمام مسیرهای سیستم در اپ greenhouse

۴-۳) به لحظه بودن داده‌ها (real-time data)

برای اینکه بتوانیم داده‌ها را به صورت لحظه‌ای داشته باشیم از channels و django_eventstream استفاده می‌کنیم. یک مسیر به نام event ایجاد می‌کنیم تا در فرانت‌اند بتوانیم زمانی که رویدادی اتفاق افتد مطلع بشویم. دو کانال با نام‌های data_controlling و data_monitoring ایجاد می‌کنیم. این تغییرات را در فایل asgi.py که در پوشه ایجاد پروژه موجود است انجام می‌دهیم.



```
import os
from django.core.asgi import get_asgi_application
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from django.urls import path, re_path
import django_eventstream

os.environ["DJANGO_ALLOW_ASYNC_UNSAFE"] = "true"

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')

application = ProtocolTypeRouter({
    'http': URLRouter([
        re_path('events/', AuthMiddlewareStack(
            URLRouter(django_eventstream.routing.urlpatterns)
        ), {'channels': ['data_monitoring', 'data_controlling']}),
        re_path(r'', get_asgi_application()),
    ]),
})
```

شکل ۳-۲۳) کد مربوط به asgi.py که حالت اجرا برنامه را به صورت async قرار می‌هد.

سپس از سیگنال برای زمانی که رویدادی اتفاق می‌افتد استفاده می‌کنیم.

۴-۴) بررسی سیگنال‌ها

زمانی که رکوردی به مدل SensorValue اضافه می‌شود با بررسی این که از کدام نوع سنسور هست عمل data_monitoring را در کانال sent_event انجام می‌دهیم.

```

@receiver(post_save, sender=SensorValue)
def send_data_sensor_value(sender, instance, created, **kwargs):
    if created:
        sensor = get_object_or_404(Sensor, id=instance.sensor_id)
        range_value = retrive_renge_val()
        if sensor.sensor_id[:3] == 'TMP':
            send_event('data_monitoring', 'temp_update', {
                'value': instance.value,
                'id': instance.id,
                'recorded_time': str(instance.recorded_time),
                'date_time': str(instance.date_time)
            })
            detect_alarm(range_value['Temperature'], instance, message="دما ")
        elif sensor.sensor_id[:3] == 'HUM':
            send_event('data_monitoring', 'hum_update', {
                'value': instance.value,
                'id': instance.id,
                'recorded_time': str(instance.recorded_time),
                'date_time': str(instance.date_time)
            })
            detect_alarm(range_value['Humidity'], instance, message=" رطوبت ")
        elif sensor.sensor_id[:3] == 'LUX':
            send_event('data_monitoring', 'lux_update', {
                'value': instance.value,
                'id': instance.id,
                'recorded_time': str(instance.recorded_time),
                'date_time': str(instance.date_time)
            })
            detect_alarm(range_value['Lux'], instance, message=" شدت نور ")

```

شکل ۳-۲۴) کد سیگنال سنسور جیت ارسال به قسمت کلاینت

تابع detect_alarm محدوده مقادیر مجاز برای مقادیر سنسورها را برمی‌گرداند تابع retrive_renge_val وظیفه تشخیص هشدار را دارد. اگر مقدار عدد سنسور کمتر و یا بیشتر از حد تعیین شده باشد، یک هشدار ایجاد می‌شود.

```

def detect_alarm(range_val, instance, message=""):

    if instance.value < range_val[0]:
        message = message + "کمتر از حد مجاز"
        AlarmMessage.objects.create(body_text=message, sensor=Sensor.objects.get(id=instance.id))

    elif instance.value > range_val[1]:
        message = message + "بیشتر از حد مجاز"
        AlarmMessage.objects.create(body_text=message,
                                     sensor=Sensor.objects.get(id=instance.sensor_id))
    return

```

شکل ۳-۲۵) کد تابع ایجاد هشدار

مشابه حالت قبل زمانی که رکوردی به مدل AlarmMessage اضافه می‌شود با بررسی این که از کدام نوع سنسور هست عمل sent_event را انجام می‌دهیم.



```
@receiver(post_save, sender=DeviceValue)
def send_device_status(sender, instance, created, update_fields, **kwargs):
    device = get_object_or_404(Device, id=instance.device_id)
    if device.device_id[:3] == 'FAN':
        send_event('data_controlling', 'fan_status', {
            'status': instance.status,
            'device_ref': instance.device_id
        })
    elif device.device_id[:4] == 'WPMP':
        send_event('data_controlling', 'pump_status', {
            'status': instance.status,
            'device_ref': instance.device_id
        })
```

شکل ۲۶-۳) کد سیگنال مربوط به تغییر وضعیت فن و پمپ

وقتی که می‌خواهیم قطعات (فن یا پمپ آب) روشن یا خاموش شود نیازمند این هستیم که سیگنالی داشته باشیم که وقتی دستگاه تغییر وضعیت بدهد عمل send_event در کانال data_controlling انجام شود.

۵-۳) خلاصه

در این فصل به بررسی دقیق‌تر مدل‌ها که در فصل پیش در مورد آن صحبت شد پرداختیم و در ادامه ساختار api‌های سمت سرور را مشاهده کردیم و با روند طراحی ویو، سریال‌ایزر و سیگنال آشنا شدیم.

² Application Programming Interface

فصل چهارم)

وب اپلیکیشن (فرانت‌اند)

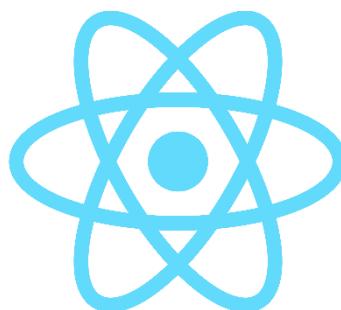
۱-۴) انتخاب فناوری فرانت‌اوند

دو الگوی طراحی کلی برای برنامه‌های تحت وب وجود دارند:

- single-page application یا SPA (برنامه‌های تکصفحه‌ای)
- multi-page application یا MPA (برنامه‌های چندصفحه‌ای)

یکی از ویژگی‌های خوب برنامه‌های تکصفحه‌ای این است که منابع برنامه (HTML و CSS و Scriptها) فقط یکبار بارگذاری می‌شوند و تنها داده‌های اصلی است که بین سرور و کلاینت رهبری می‌شود.

برای فناوری فرانت‌اوند از ری‌اکت استفاده می‌کنیم که یک کتابخانه متن‌باز جاوا اسکریپت است که برای ساختن برنامه‌های تکصفحه‌ای مورداستفاده قرار می‌گیرد. در این کتابخانه می‌توان کامپوننت‌هایی با قابلیت استفاده مجدد طراحی و ایجاد کنیم. کتابخانه ری‌اکت این قابلیت را برای برنامه نویسان می‌دهد تا برنامه‌های کاربردی تحت وب طراحی کنند که بدون بارگذاری مجدد صفحه می‌توان اطلاعات آن را تغییر داد. هدف اصلی کتابخانه ری‌اکت سریع بودن، قابل توسعه بودن و ساده بود آن است.



شکل ۱-۴) لوگو کتابخانه ری‌اکت

۴-۳) مفاهیم پایه در ری اکت

همان‌طور که در فصل سرور به مفاهیم جنگو پرداخته شد، در این قسمت هم برای درک سازوکار ری اکت به مفاهیم مورد استفاده در توسعه یک وب اپلیکیشن در ری اکت می‌پردازیم.

۱-۲-۴) کامپوننت‌ها (components)

کامپوننت‌ها اجزای سازنده هر برنامه ری اکت هستند و هر برنامه از چندین کامپوننت تشکیل شده است.

کامپوننت‌ها به دو صورت نوشته می‌شوند:

- Functional Components
- Class Components

در این پژوهه از کامپوننت‌های تابعی که در این مدت به صورت ترند در برنامه‌های ری اکتی مورد استفاده قرار گرفته‌اند، استفاده می‌کنیم.

۲-۲-۴) وضعیت (state)

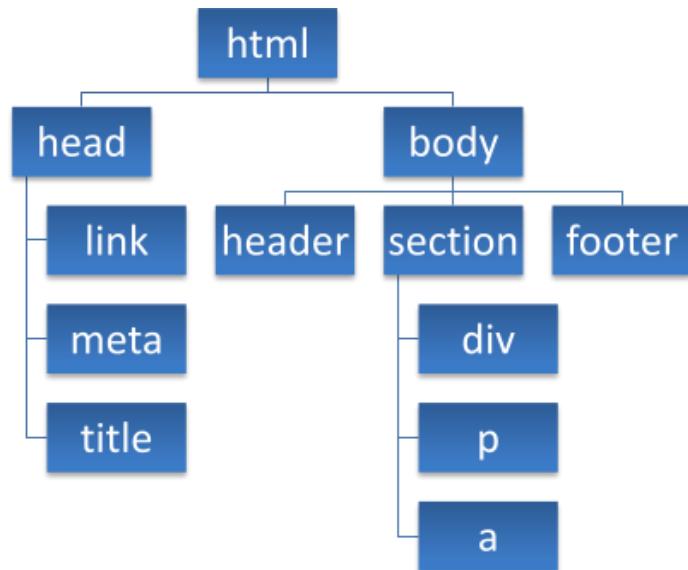
به متغیرهایی از یک کامپوننت گفته می‌شود که مستقیماً توسط خود کامپوننت مقداردهی و مدیریت می‌شوند.

۳-۲-۴) ویژگی (props)

به متغیرهایی از یک کامپوننت گفته می‌شود که توسط والد به آن انتقال پیداکرده است.

چیست؟ DOM (۴-۲-۴)

مرورگر هر صفحه‌ی HTML ای که را نمایش می‌دهد، یک لیست درختی از تمام اجزای صفحه ایجاد می‌کند که به آن DOM می‌گویند و گره‌های این درخت، اشیا آن اسناد هستند. در شکل زیر حالت درختی عناصر یک صفحه واضح‌تر دیده می‌شود.



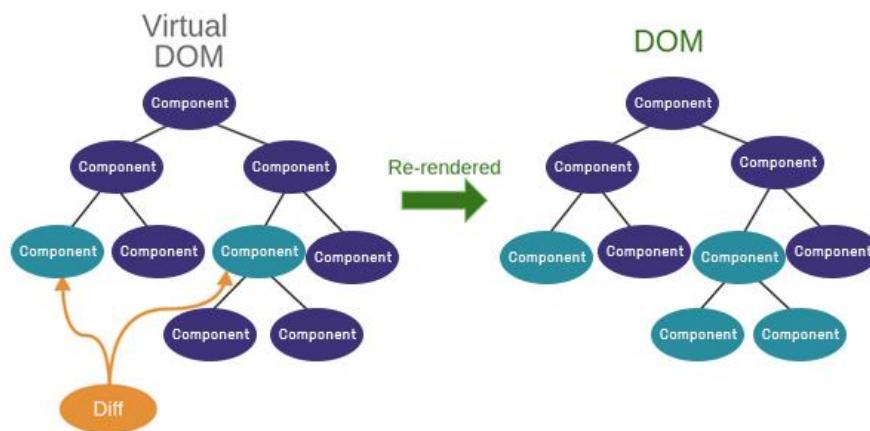
شکل ۲-۴) نمودار درختی DOM

مجازی DOM (۵-۲-۴)

ری‌اکت یک کپی از Real DOM را در حافظه‌ی خود نگه‌داری می‌کند که به آن Virtual DOM می‌گوییم. تغییرات و دست‌کاری DOM واقعی بسیار کند تر از DOM مجازی است. اگر حالت هر یک از عناصر تغییر کند، یک درخت Virtual DOM جدید ایجاد می‌شود. سپس، این درخت با درخت DOM مجازی قبلی، مقایسه می‌شود. زمانی که وضعیت یک المنت تغییر می‌کند، VDOM به جای آپدیت کردن تمام اشیا، فقط وضعیت شیء دست‌کاری شده را در Real DOM تغییر می‌دهد.

² Document Object Model ³

همان طور که در تصویر زیر می بینید در سمت چپ از virtual DOM استفاده شده است و در این حالت فقط آن کامپوننتی که تغییر کرده است ویرایش شده است ولی در سمت راست که از Virtual DOM استفاده نشده است و مستقیم DOM ویرایش شده است تمام زیرشاخه های کامپوننت، بازنویسی شده اند.



شکل (۳-۴) Virtual DOM and DOM

context (۶-۲-۴)

در یک اپلیکیشن معمولی ری اکت، داده از طریق props از بالا به پایین (والدین به فرزند) منتقل می شود. اما این کار برای انواع خاصی از props ها (برای مثال: کاربر تأیید شده فعال، تم رابط کاربری) که موردنیاز بسیاری از کامپوننت ها در یک اپلیکیشن است می تواند سنگین باشد. Context راهی را برای به اشتراک گذاری مقادیری مانند این بین کامپوننت ها بدون نیاز به انتقال prop صریحاً از هر سطح درخت فراهم می کند.

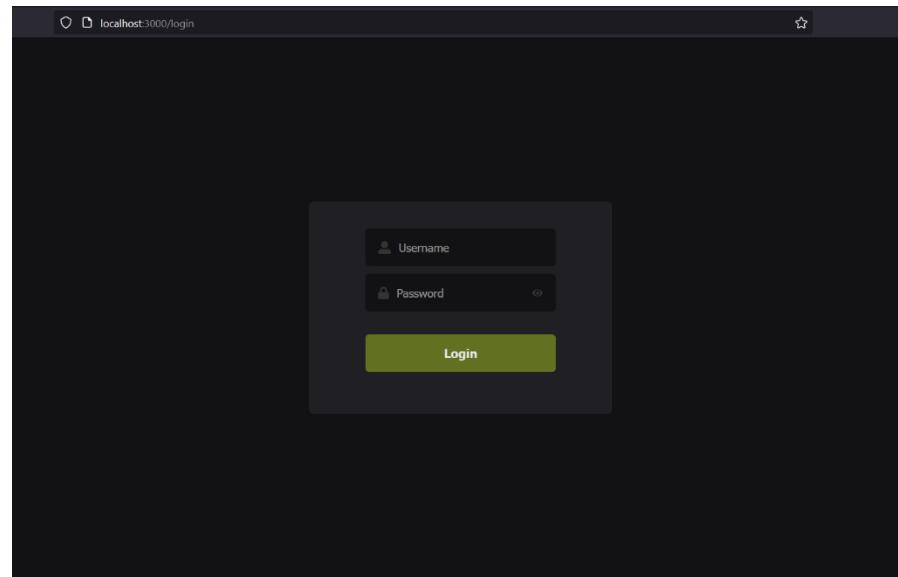
با این ویژگی می توانیم توی برنامه متغیرهایی به صورت سراسری (Global) داشته باشیم که می توان از همه جای برنامه به طور مستقیم مورد دسترسی قرار بگیرند.

react-router-dom (۷-۲-۴)

از react-router-dom برای ساختن مسیرها استفاده می کنیم. react-router-dom یک پکیج محبوب برای ایجاد مسیر محاسب محسوب می شود. که با این دستور آن را نصب می کنیم:

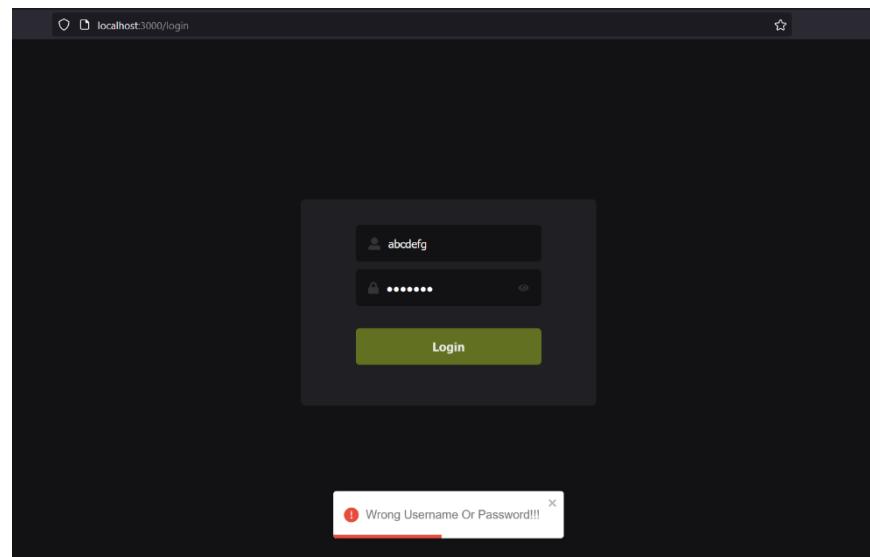
```
npm install react-router-dom
```

۳-۴) صفحه ورود



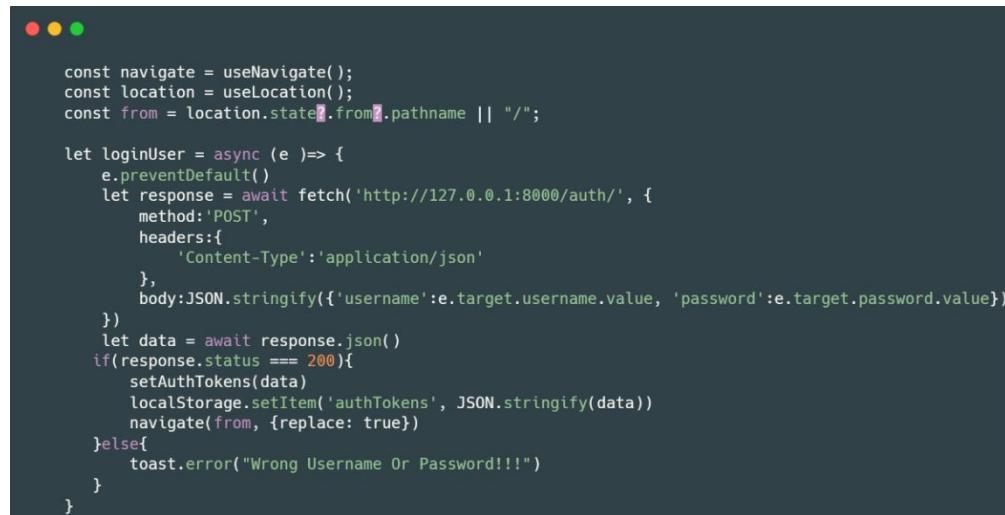
شکل ۴-۴) صفحه ورود به سیستم

کاربر با وارد کردن نام کاربری و رمز عبور به صفحه داشبورد منتقل می‌شود. در صورتی که اطلاعات ورود را اشتباه وارد کند پیغام خطا دریافت خواهد کرد.



شکل ۴-۵) نمایش پیام خطا در صورت اقدام به ورود با اطلاعات اشتباه

منطق ورود به برنامه به صورت زیر است. مقادیر نام کاربری و رمز عبور به صورت json با درخواست از نوع post به سرور فرستاده می‌شود در صورتی که سرور هویت فرد را تأیید کند، status با مقدار ۲۰۰ را برمی‌گرداند. توکن فرستاده شده از سمت سرور را ذخیره می‌کنیم و کاربر را به صفحه داشبورد منتقل می‌کنیم.



```

const navigate = useNavigate();
const location = useLocation();
const from = location.state?.from?.pathname || "/";

let loginUser = async (e) => {
  e.preventDefault()
  let response = await fetch('http://127.0.0.1:8000/auth/', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ 'username': e.target.username.value, 'password': e.target.password.value })
  })
  let data = await response.json()
  if(response.status === 200){
    setAuthTokens(data)
    localStorage.setItem('authTokens', JSON.stringify(data))
    navigate(from, {replace: true})
  }else{
    toast.error("Wrong Username Or Password!!!")
  }
}

```

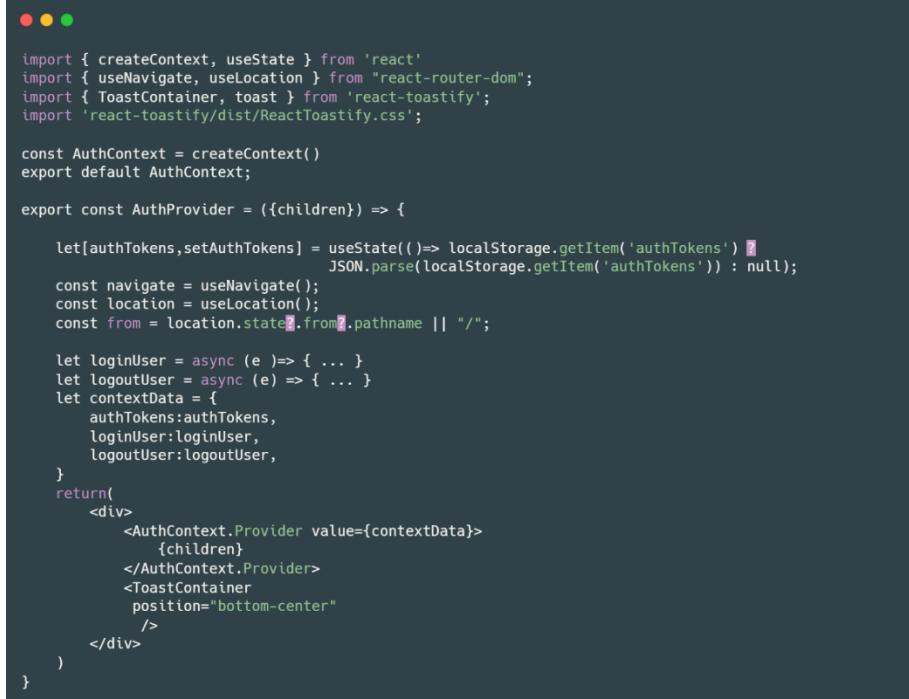
شکل ۴-۶) کد سمت فرانت برای احراز هویت به هنگام ورود

۴-۴) ایجاد مسیر (route)های محافظت شده

برای این منظور ابتدا یک پوشه با نام context در محیط پروژه می‌سازیم و در آن فایلی با عنوان AuthContext.js ایجاد می‌کنیم. سپس توکن فرستاده شده از سمت سرور و تابع ورود و خروج را در آن پیاده‌سازی می‌کنیم.

در ادامه فایل دیگری به نام PrivateRoutes.js ایجاد می‌کنیم و شرایط احراز هویت کاربر را چک می‌کنیم. اگر کاربر احراز هویت نشده باشد کاربر را به صفحه ورود منتقل می‌کنیم. در صورتی که احراز هویت شده باشد ولی دسترسی برای کاربر تعیین نشده باشد به صفحه‌ای که بیان می‌کند دسترسی ندارید منتقل می‌کنیم.

اگر کاربر قبل از وارد کردن اطلاعات ورود بخواهد با وارد کردن url به صفحه‌ی دیگری از برنامه برود؛ کاربر را به صفحه ورود منتقل می‌کنیم. بعد از وارد کردن اطلاعات ورود، به صفحه‌ی مدنظر کاربر که قبل از وارد کردن اطلاعات عبور قصد داشت آن را ببیند؛ منتقل می‌شود.



```

import { createContext, useState } from 'react'
import { useNavigate, useLocation } from "react-router-dom";
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

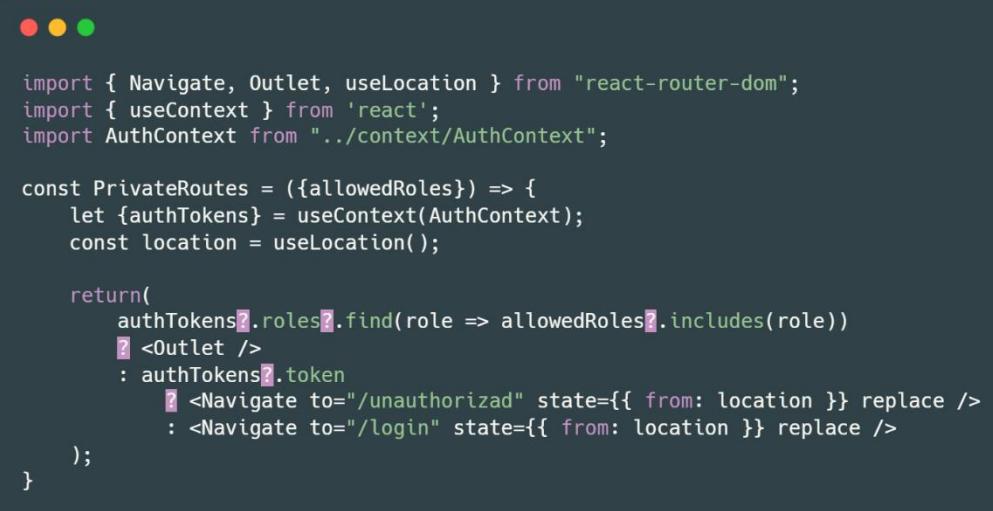
const AuthContext = createContext()
export default AuthContext;

export const AuthProvider = ({children}) => {
    let[authTokens, setAuthTokens] = useState(()=> localStorage.getItem('authTokens') ? JSON.parse(localStorage.getItem('authTokens')) : null);
    const navigate = useNavigate();
    const location = useLocation();
    const from = location.state?.from?.pathname || "/";

    let loginUser = async (e )=> { ... }
    let logoutUser = async (e) => { ... }
    let contextData = {
        authTokens:authTokens,
        loginUser:loginUser,
        logoutUser:logoutUser,
    }
    return(
        <div>
            <AuthContext.Provider value={contextData}>
                {children}
            </AuthContext.Provider>
            <ToastContainer
                position="bottom-center"
            />
        </div>
    )
}

```

شکل ۷-۴) کد مربوط به بررسی توکن



```

import { Navigate, Outlet, useLocation } from "react-router-dom";
import { useContext } from 'react';
import AuthContext from "../context/AuthContext";

const PrivateRoutes = ({allowedRoles}) => {
    let {authTokens} = useContext(AuthContext);
    const location = useLocation();

    return(
        authTokens?.roles?.find(role => allowedRoles?.includes(role))
        ? <Outlet />
        : authTokens?.token
            ? <Navigate to="/unauthorized" state={{ from: location }} replace />
            : <Navigate to="/login" state={{ from: location }} replace />
    );
}

```

شکل ۸-۴) کد بخش PrivateRoute برای جلوگیری از ورود غیرمجاز به بخش‌های مختلف سیستم

سپس روت‌هایی که در پروژه داریم را به سه دسته تقسیم می‌کنیم.

- روت‌هایی که همه به آن دسترسی دارند مثل "path="/login"
- روت‌هایی که ادمین و کاربر احراز هویت شده به آن دسترسی دارند مثل "path="/switch"
- روت‌هایی که فقط ادمین به آن دسترسی دارد مثل "path="/humidity"



```
<Router>
  <Routes>
    <Route element={<AuthProvider>} <PrivateRoutes allowedRoles={[ROLES.NormalUser, ROLES.Admin]}>
      </AuthProvider>
    <Route path="/temperature" element={<Temperature alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}>}/>
    <Route path="/humidity" element={<Humidity alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}>}/>
    <Route path="/lux" element={<Lux alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}>}/>
    <Route path="/alarm" element={<Alarm alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}> tempAlarmNotSeen={tempAlarmNotSeen}>}/>
    <Route path="/" element={<Home alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}>}/>
  </Route>

  <Route element={<AuthProvider>} <PrivateRoutes allowedRoles={[ROLES.Admin]}> </AuthProvider>>
    <Route path="/switch" element={<Switch alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}>}/>
  </Route>

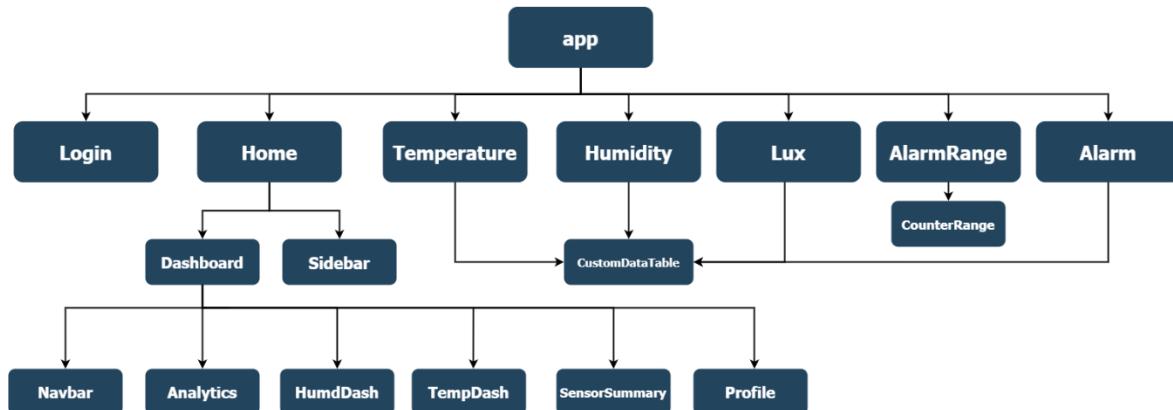
  <Route path="/login" element={<AuthProvider> <Login /> </AuthProvider>}>/>
  <Route path="/unauthorized" element={<p style={{color:'#fff'}}>unauthorized!</p>}>/>
  <Route path="/" element={<p style={{color:'#fff'}}>There's nothing here: 404!</p>}>/>
  </Routes>
</Router>
```

شکل ۹-۳) مسیرهای داخلی سمت فرانت

صفحه home (۴-۵)

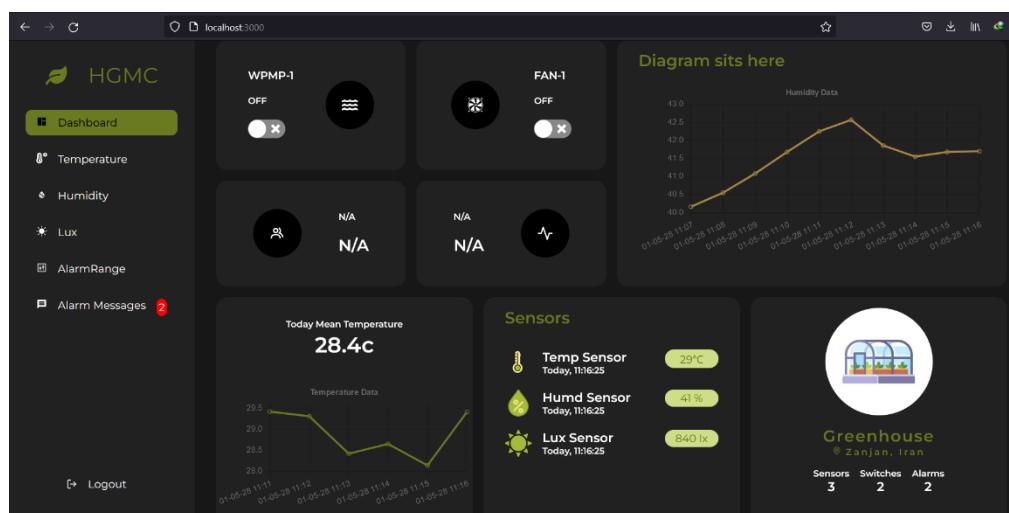
همان‌طور که می‌دانیم ری‌اکت مجموعه‌ای از کامپوننت‌ها است که در کنار هم قرار می‌گیرند و صفحه‌ای را ایجاد می‌کند. صفحه home دقیقاً به همین صورت است. از خوبی‌های این ویژگی می‌توان به این اشاره کرد که علاوه بر قابلیت استفاده مجدد، در صورتی که در فرآیند توسعه یک کامپوننت به مشکل بخوریم به صورت قطعی می‌دانیم که مشکل از کدام کامپوننت است و بین انبوی از کدها گم نمی‌شویم.

نمودار درختی زیر کل وب اپلیکیشن ما را شامل می‌شود. تمامی فرزندان app یک صفحه وب هستند. با توجه به نیازی که در هر صفحه برای طراحی وجود دارد ممکن است یک صفحه چندین فرزنده داشته باشد مثل صفحه login و یا مثل صفحه login فرزنده home باشد.



شکل ۱۰-۱) نمودار درختی کامپوننت‌های ساخته شده برای سیستم

شکل زیر صفحه home را نشان می‌دهد. در ادامه اجزاء آن را بررسی خواهیم کرد.



شکل ۱۱-۱) صفحه Home برنامه که به صورت یک داشبورد پیاده‌سازی شده است

وقتی به صفحه home می‌رویم کامپوننت Dashboard و Sidebar که شامل Home فراخوانی می‌شود.



```
import React from "react";
import styled from "styled-components";
import Dashboard from "./Dashboard";
import Sidebar from "./Sidebar";
export default function Home({alarmNotSeen,seenAlarm}) {
  return (
    <Div>
      <Sidebar alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}/>
      <Dashboard alarmNotSeen={alarmNotSeen}/>
    </Div>
  );
}
const Div = styled.div`  
  position: relative;  
`;
```

شکل ۱۲-۱۶) کد کامپوننت Home

اگر به تصویر بالا دقت کنید می‌بینید که کامپوننت Home دو مقدار را از طریق props دریافت کرده است. Alarm مربوط به هشدارهای دیده نشده و seenAlarm تابعی است که درصورتی که بر روی Message موجود در sidebar کلیک کنیم؛ یک درخواست به سرور می‌زند و هشدارهایی که مقدار فیلد آن است به True تغییر پیدا می‌کند و عملاً تعداد هشدارها صفر می‌شود.

۱-۵-۴) کامپوننت Dashboard

این کامپوننت بعد از فراخوانی چند کامپوننت دیگر را فراخوانی می‌کند.

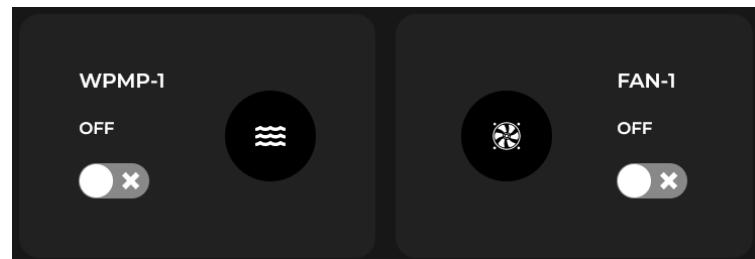
```

        return (
          <Section>
            <Navbar />
            <div className="grid">
              <div className="row__one">
                <Analytics />
                <HumdDash />
              </div>
              <div className="row__two">
                <TempDash />
                <SensorSummary />
                <Profile alarmNotSeen={alarmNotSeen}/>
              </div>
            </Section>
        );
    
```

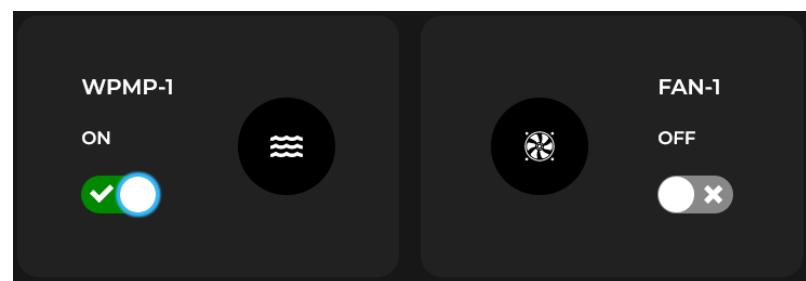
شکل ۱۳-۴) کد کامپوننت Dashboard

۱۴-۵-۴) Analytics کامپونت

این کامپونت نام و وضعیت روشن یا خاموش بودن دستگاهها را نشان می‌دهد.

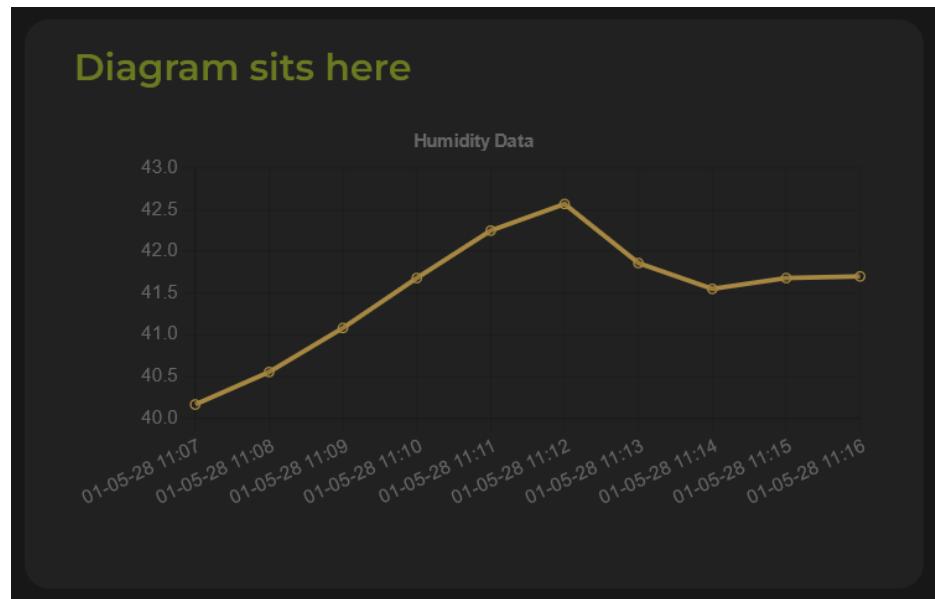


شکل ۱۴-۴) تصویر سوئیچ‌های برنامه که در وضعیت خاموش قرار دارند.



شکل ۱۵-۴) تصویر یکی از سوئیچ‌ها در وضعیت روشن

HumdDash کامپوننت (۳-۵-۴)



شکل ۱۶-۴) تصویر نمودار رطوبت هوا مربوط به کامپوننت HumdDash

این کامپوننت ۱۰ داده آخر رطوبت را بر حسب تاریخ و زمان نشان می‌دهد. در ابتدا یک درخواست از نوع GET به سرور زده می‌شود و مقادیر از پایگاه داده خوانده می‌شود.

```
const fetchHumdData = async () => {
  try {
    const url = "http://127.0.0.1:8000/api/SensorValueInfo/2/";
    const response = await fetch(url, {
      method:'GET',
      headers:{
        'Authorization': `token ${authTokens.token}`
      }
    });
    const datapoints = await response.json();
    setHumidity(datapoints);
    console.log(humidity);

  } catch (error) {
    console.log(error);
  }
};
```

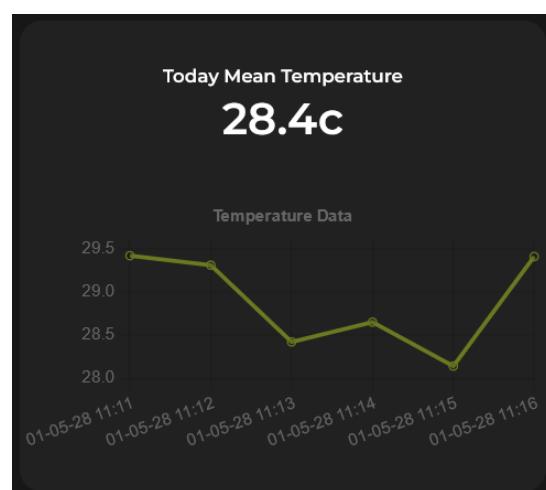
شکل ۱۷-۴) کد ریکوئست به سرور برای دریافت اطلاعات رطوبت هوا

در ادامه برای اینکه تغییرات به صورت بخط قابل مشاهده باشد از تابع زیر استفاده می‌کنیم. در این تابع اول یک شیء از EventSource می‌سازیم و بعد بر روی رویداد humd_update گوش می‌دهیم. در صورتی که مقدار رطوبت هوا جدیدی اضافه شود به مقادیر قبلی اضافه می‌شود.

برای رسم نمودار از پکیج‌های react-chartjs-2 و chart.js استفاده شده است.

۴-۵-۴) کامپوننت TempDash

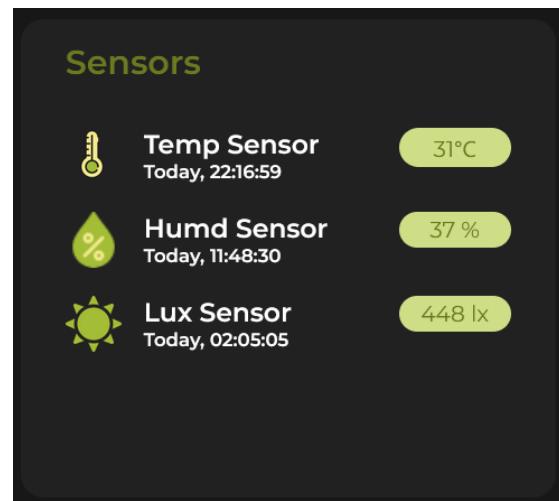
این کامپوننت هم مشابه نمودار رطوبت می‌باشد. ۶ داده دما آخر را بر حسب تاریخ و زمان نشان می‌دهد.



شکل ۱۸-۴) نمودار دما مربوط به کامپوننت TempDash

۵-۵-۴) کامپوننت SensorSummary

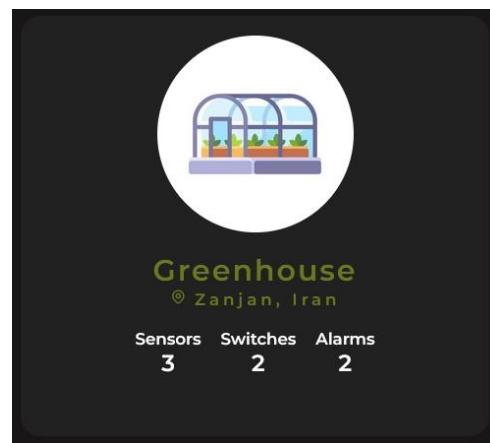
این کامپوننت آخرین مقدار دما، رطوبت هوا و شدت نور را از سنسورها دریافت کرده و نشان می‌دهد. مشابه حالت‌های قبل ابتدا به سرور یک درخواست زده می‌شود و مقادیر اولیه نشان داده می‌شود. در ادامه برای بر خط lux_update و humd_update و temp_update استفاده می‌کنیم و به رویدادهای گوش می‌دهیم.



شکل ۱۹-۴) تصویر اجرایی از آخرین داده از هر سنسور

لازم به یادآوری هست که این رویدادها در فصل قبل زمانی که می خواستیم داده‌ها لحظه‌ای باشند در داخل سیگنال تعریف شده‌اند.

Profile ۶-۵-۴) کامپوننت



شکل ۲۰-۴) تصویر اجرایی از اطلاعات تعداد سنسورها و سوئیچ‌ها و هشدارها

در این کامپوننت اطلاعاتی درباره تعداد هشدارها، کلیدهای کنترلی و نوع سنسورها را داریم. زمانی که کامپوننت ایجاد می‌شود یک درخواست با نوع GET به سرور می‌زند و مقادیر تعداد هشدارها، کلیدهای کنترلی و نوع سنسورها را دریافت می‌کند.



```
const getCount = async (url) => {
  try{
    const response = await fetch(url,{
      method:'GET',
      headers:{`Authorization`: `token ${authTokens.token}`}
    });
    const data = await response.json();
    setCount(data);
  } catch (error) {
    console.log(error);
  }
};

useEffect(() =>{
  getCount("http://127.0.0.1:8000/api/CountSensorDevice/");
}, []);
```

شکل ۲۱-۴) کد بخش Profile

۶-۴) صفحه ثبت اطلاعات دما

در این صفحه سابقه ذخیره مقادیر برای پارامتر دما به صورت جدولی ذخیره می‌شود. یک دکمه در صفحه قرار داده شده تا در صورت نیاز اطلاعات جدول را به روز کنیم. امکان جستجو بر روی این جدول وجود دارد. می‌توانیم بر اساس ستون‌های موجود در جدول مقادیر را به صورت صعودی یا نزولی مرتب کنیم.

Temprature Log					
<input type="text" value="Search here"/> Refresh Data					
id	sensor_id	temp.value	recorded_time	date_time	point
1	TMP-1	28.42	11:07:25	14/01/05/28	PNT-1
2	TMP-1	28.65	11:08:25	14/01/05/28	PNT-1
3	TMP-1	28.14	11:09:25	14/01/05/28	PNT-1
4	TMP-1	29.41	11:10:25	14/01/05/28	PNT-1
5	TMP-1	29.42	11:11:25	14/01/05/28	PNT-1
6	TMP-1	29.31	11:12:25	14/01/05/28	PNT-1
7	TMP-1	28.42	11:13:25	14/01/05/28	PNT-1
8	TMP-1	28.65	11:14:25	14/01/05/28	PNT-1
9	TMP-1	28.14	11:15:25	14/01/05/28	PNT-1
10	TMP-1	29.41	11:16:25	14/01/05/28	PNT-1

Rows per page: 10 | 1-10 of 10 | < > >>

Logout

شکل ۲۲-۴) تصویر صفحه ثبت اطلاعات دما

قابلیت رفتن به صفحه قبل یا بعد و مشخص کردن تعداد ردیف در هر صفحه از دیگر ویژگی‌های این جدول می‌باشد. برای استفاده از این جدول می‌توان از دستور زیر آن را به پروژه اضافه کرد و مطابق نیاز آن را شخصی‌سازی کرد.

```
npm install react-data-table-component
```

```

import React from 'react'
import styled from "styled-components";
import CustomDataTable from '../CustomDataTable'
import Sidebar from '../Sidebar'

const Temperature = ({alarmNotSeen,seenAlarm}) => {
  const url = "http://127.0.0.1:8000/api/SensorValueInfo/1/";
  const columns = [ ... ];
  const search_column_field = 'value';
  const title_table = 'Temprature Log';

  return (
    <Div>
      <Sidebar alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm}/>
      <CustomDataTable
        url={url}
        columns={columns}
        search_column_field={search_column_field}
        title_table={title_table}
      />
    </Div>
  )
}

export default Temperature

const Div = styled.div` 
  position: relative;
`;

```

شکل ۲۳-۴) کد صفحه اطلاعات دما

به همین ترتیب سایر صفحات مانند صفحه اطلاعات رطوبت هوا و شدت نور با همین مکانیزم پیاده‌سازی و تکرار شده‌اند.

۴-۶) کامپوننت CustomDataTable

این کامپوننت ۴ ورودی می‌گیرد. url اندپوینتی که قرار است آن را در تابع getItems استفاده کنیم. columns سرستون‌های جدول است. search_column_field فیلدی است که می‌خواهیم روی آن عمل جستجو را انجام دهیم. title_table عنوان جدول است. در تابع getItems به سرور درخواست می‌زنیم و مقادیر را در ذخیره می‌کنیم.

```
import React, { useState } from "react";
import styled from "styled-components";
import DataTable, {createTheme} from "react-data-table-component";
import { useEffect } from "react";
import useAuth from "../hooks/useAuth";

const CustomDataTable = ({url,columns,search_column_field,title_table}) => {
    createTheme('new_dark',{ ... } , 'dark');

    const [search, setSearch] =useState("");
    const [items, setItems] =useState([]);
    const [filteredItems, setfilteredItems] =useState([]);

    const [authTokens] = useAuth();

    const getItems = async () => { ... };

    useEffect(() =>{
        getItems();
    }, [ ]);

    useEffect(() =>{
        const result = items.filter(item => {
            return item[search_column_field].toString().toLowerCase().match(search.toLowerCase());
        });
        setfilteredItems(result);
    }, [search]);

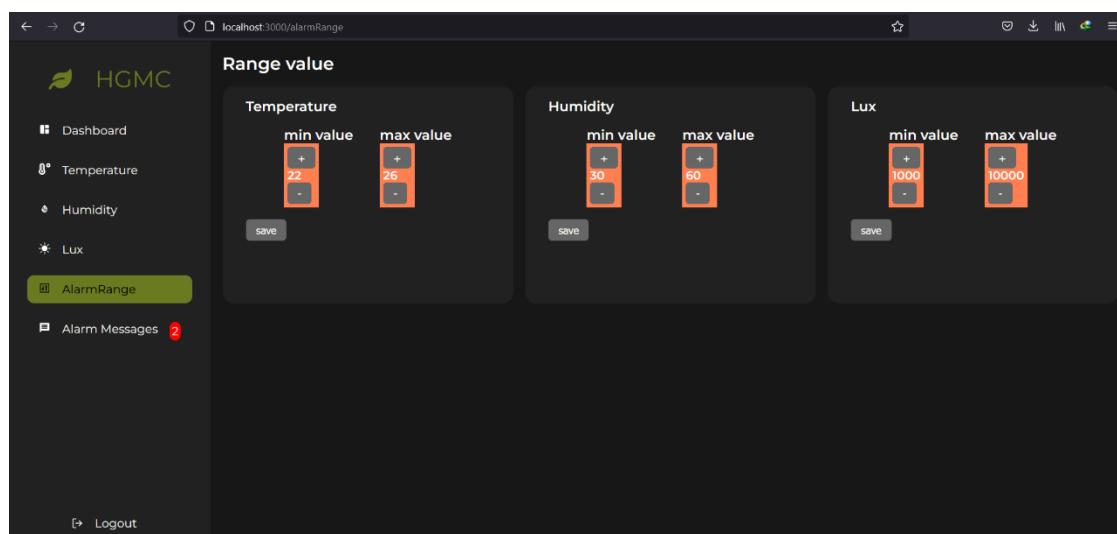
    return (
        <Section>
            <DataTable ... />
        </Section>
    );
}

export default CustomDataTable;
```

شکل ۲۴-۲) کد مربوط به CustomDataTable

از جمله شخصی‌سازی‌های دیگری که بر روی جدول اعمال کردہ‌ایم می‌توان به‌اضافه کردن زمینه تیره برای جدول، افزودن عنوان، ثابت کردن ردیف عنوان هنگام اسکرول کردن، تغییر کردن پس‌زمینه سطر جدول هنگام بردن ماوس و چپ‌چین کردن عنوان نام برد.

۷-۴) صفحه تنظیم محدوده هشدار برای مقادیر سنسورها



شکل ۲۵-۴) تصویر صفحه تنظیم محدوده هشدار سنسورها

در این صفحه این امکان وجود دارد که محدوده کمینه و بیشینه را برای هر سنسور اعمال کنیم. در صورتی که مقادیر جمع‌آوری شده توسط سنسور کمتر و یا بیشتر از حد تعیین شده باشد؛ هشداری در قسمت sidebar نشان داده خواهد شد. فقط ادمین اجازه دسترسی به این صفحه را دارد.

زمانی که کامپوننت AlarmRange ایجاد می‌شود. در تابع getSensorTypeRange مقادیری که از قبل تنظیم شده را در state ذخیره می‌کند و مقادیر کمینه و بیشینه را به فرزند خود یعنی کامپوننت CounterRange از طریق props منتقل می‌کند.

```

import React,{useState,useEffect} from'react'
import styled from "styled-components";
import { cardStyles } from "../ReusableStyles";
import CounterRange from './CounterRange'
import Sidebar from '../Sidebar'
import useAuth from '../../../../../hooks/useAuth'

const AlarmRange = ({alarmNotSeen,seenAlarm}) => {
  const [sensorTypeRange, setSensorTypeRange] = useState([]);
  const [authTokens] = useAuth();
  const getSensorTypeRange = async (url) => { ... };

  useEffect(() =>{
    getSensorTypeRange("http://127.0.0.1:8000/api/SensorTypeRange/");
  }, []);

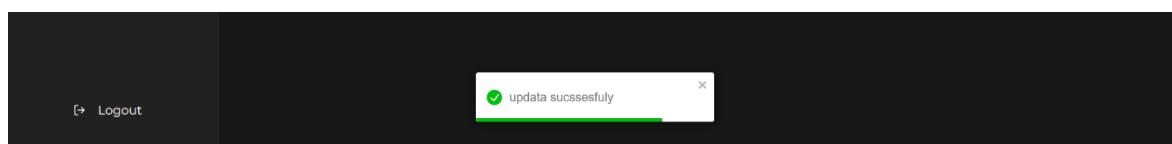
  return (
    <Div>
      <Sidebar alarmNotSeen={alarmNotSeen} seenAlarm={seenAlarm} />
      <Section>
        <div className='grid'>
          <h2>Range value</h2>
          <div className='row'>
            {sensorTypeRange.map(x=>
              <div key={x.id} className='content'>
                <h3>{x.title}</h3>
                <CounterRange min_range={x.min_range} max_range={x.max_range} id={x.id}/>
                <br><br>
                <br><br>
              </div>
            )}
          </div>
        </Section>
      </Div>
    )
}

```

شکل ۴-۲۶) کد صفحه محدوده هشدارها

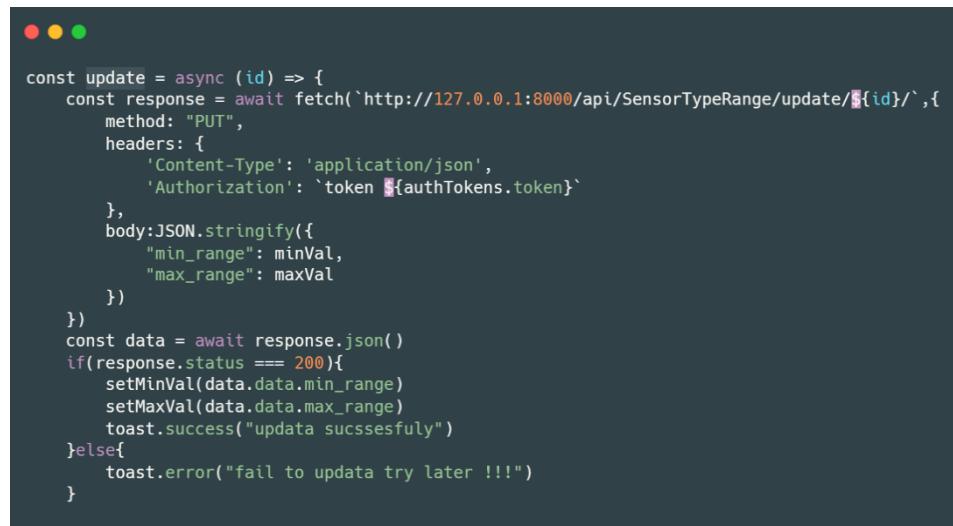
۱-۷-۴) CounterRange کامپوننت

این کامپوننت وظیفه کموزیاد کردن و بهروزرسانی مقادیر کمینه و بیشینه برای محدوده هشدار را دارد. و این مقادیر را از کامپوننت والد خود یعنی AlarmRange به دست می‌آورد. زمانی که عمل بهروزرسانی انجام شد پیغام موفقیت‌آمیز نشان می‌دهد.



شکل ۴-۲۷) پیغام موفقیت در ثبت محدوده جدید هشدار برای سنسور

منطق به روزرسانی به این صورت است که هر نوع سنسور یک شناسه منحصر به فرد دارد. با استفاده از این شناسه و مقدار کمینه و بیشینه که قابل تعیین کردن است درخواستی از نوع PUT به سرور زده می‌شود. اگر عملیات به روزرسانی موفقیت‌آمیز باشد سرور status کد ۲۰۰ برمی‌گرداند و پیغام موفقیت‌آمیز بودن به روزرسانی مشاهده می‌شود.



```

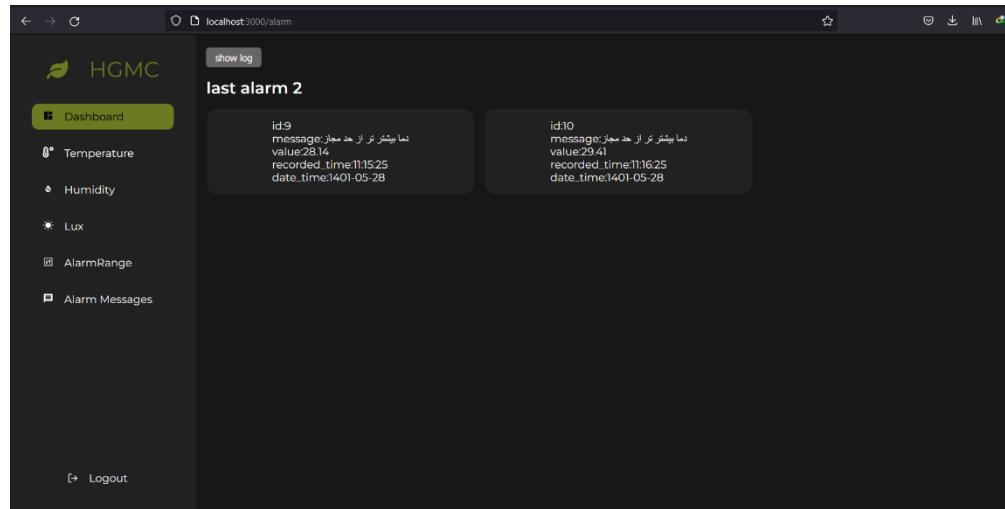
const update = async (id) => {
  const response = await fetch(`http://127.0.0.1:8000/api/SensorTypeRange/update/${id}`, {
    method: "PUT",
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `token ${authTokens.token}`
    },
    body: JSON.stringify({
      "min_range": minValue,
      "max_range": maxValue
    })
  })
  const data = await response.json()
  if(response.status === 200){
    setMinVal(data.data.min_range)
    setMaxVal(data.data.max_range)
    toast.success("update successfully")
  }else{
    toast.error("fail to update try later !!!")
  }
}

```

شکل ۴-۲۸) کد بخش ویرایش و به روزرسانی محدوده هشدار

۴-۸) صفحه مشاهده هشدارها

این صفحه هشدارهای ناشی از بالا یا پایین بودن مقدار دریافتی سنسورها، نسبت به مقدار تعیین شده در صفحه AlarmRange را به نمایش می‌گذارد. در صورتی که هشداری داشته باشیم وارد این صفحه بشویم؛ آخرین هشدارها برایمان به نمایش گذاشته می‌شود. همچنین تعداد هشدار نشان داده شده در sidebar ، محو می‌شود.



شکل ۲۹-۴) صفحه مشاهده هشدارها

در این صفحه دکمه‌ای بنام show log داریم که با کلیک بر روی آن جدول لاغ هشدارهای ایجاد شده، همانند صفحه نشان داده می‌شود.

Alarm Log						
Search here <input type="text"/> Refresh Data <input type="button"/>						
id	alarm.message	value	recorded_time	date_time	sensor_id	
11	نطونه: کم نزدیک می باشد	29	21:07:42	1401-05-28	2	
12	شدت نور کم نزدیک می باشد	990	21:12:12	1401-05-28	3	
13	شدت نور کم نزدیک می باشد	998	21:13:22	1401-05-28	3	
14	نداشتن نر نزدیک می باشد	28	21:14:34	1401-05-28	1	
15	نداشتن نر نزدیک می باشد	28	21:17:45	1401-05-28	1	
16	نداشتن نر نزدیک می باشد	28.5	21:19:53	1401-05-28	1	
17	نداشتن نر نزدیک می باشد	28.5	21:25:20	1401-05-28	1	
18	نداشتن نر نزدیک می باشد	28.8	21:30:20	1401-05-28	1	
19	نطونه: کم نزدیک می باشد	29	21:31:44	1401-05-28	2	

شکل ۳۰-۴) نمایش جدولی هشدارها

۹-۴) خلاصه

در این فصل در ابتدا با مفاهیم پایه ری اکت مثل کامپوننت‌ها آشنا شدیم. و مشاهده کردیم که چگونه با ترکیب کامپوننت‌ها می‌توان صفحه‌های وب اپلیکیشن را درست کرد. در ادامه به بررسی صفحاتی که در وب اپلیکیشن موجود بود پرداختیم و با نحوه کارکرد آن‌ها آشنا شدیم.

بخش دوم

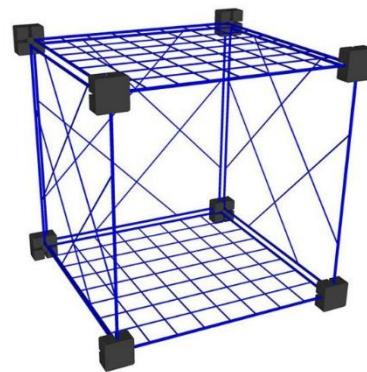
سخت افزار

فصل پنجم)

آشنايی با قطعات

۱-۵) شاکله و بستر آماده‌سازی

برای ساختار گلخانه از قفسه‌های کتاب استفاده شده است که شاکله و چارچوب پروژه را تشکیل می‌دهند.



شکل ۱-۵) نمونه قفسه کتاب

دلیل انتخاب این نوع چارچوب، وجود مشبک‌های مربعی شکل روی هر صفحه که کار نصب قطعاتی نظیر برد، رله و سایر قطعات را تسهیل می‌بخشد همچنین اتصال راحت و سریع صفحات به کمک رابط‌های اتصال، امکان توسعه بستر گیاهان بیشتر را ایجاد می‌کند و با این کار مانند ردیف‌های گلخانه در یک سوله را شبیه‌سازی می‌نماید.



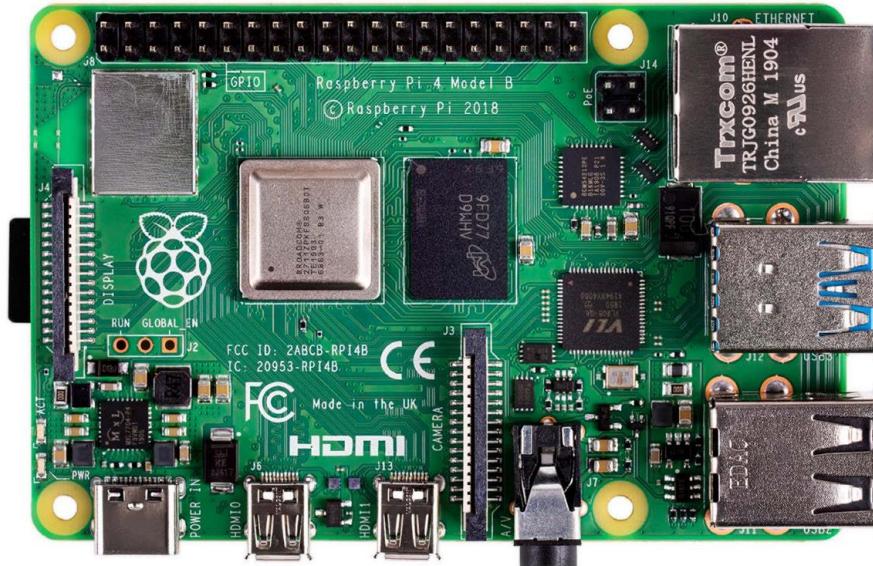
شکل ۲-۳) ردیف‌های گلخانه



شکل ۲-۴) نمایی از یک گلخانه مدرن

(۲-۵) برد رزبری پای (Raspberry Pi)

این برد که یک کامپیوتر کوچک است و مدل چهار B از خانواده بردهای رزبری پای است، هسته اصلی و مغز سختافزار این پروژه تشکیل می‌دهد. تمامی وسایل به کمک سیم‌ها و کابل‌ها به پین‌های GPIO این برد متصل‌اند و اجرای کدهای سختافزار بر عهده این برد است در ادامه به بررسی ویژگی‌های این برد می‌پردازیم.

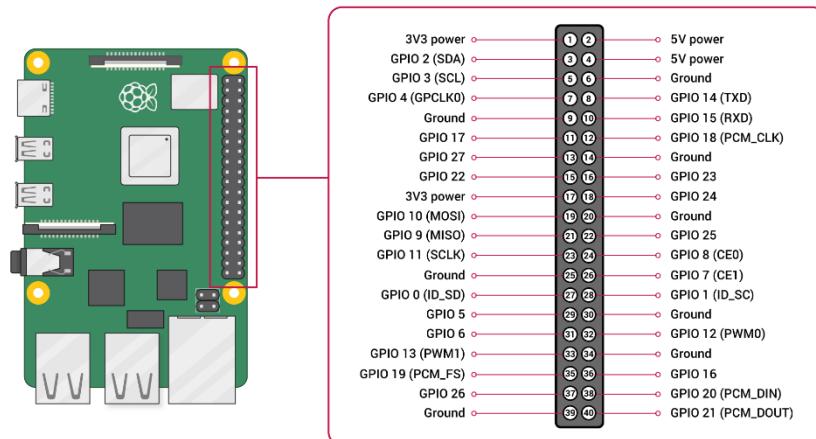


شکل ۲-۵) تصویر برد رزبری پای چهار مدل B

دلیل استفاده از این برد توان سختافزاری بالا و سهولت استفاده از آن در پروژه‌های IOT^۴ از طریق پین‌های GPIO و امکانات یک کامپیوتر واقعی در یک اندازه کوچک است که کار را برای تمامی پروژه‌های IOT و صنعتی را آسان می‌نماید.

مهم‌ترین مورد قابل ذکر در اینجا پین‌های GPIO^۵ است که این اجزه را به توسعه‌دهنده می‌دهد تا قطعات الکتریکی مانند : سنسور، فن و ... را کنترل کند که منظور از کنترل، صفر و یک کردن پین‌ها و تعریف آن‌ها به عنوان خروجی و ورودی و موارد پیچیده‌تری نظیر اجرای PWM^۶ است.

² Internet of things 4
² general-purpose input/output 5
² Pulse Width Modulation 6



شکل ۵-۵) اسامی و نوع پایه‌های GPIO

۳-۵) سنسورها

پارامترهای زیادی برای اندازه‌گیری، نظیر : میزان CO₂، میزان شدت باد، میزان سطح آب، میزان رطوبت، میزان دما و . . . در گلخانه‌ها وجود دارد. که تمامی این پارامترها به کمک سنسورها و برخی قطعات رابط قابل اندازه‌گیری است.

در این پژوهه ما به سراغ اندازه‌گیری دما، رطوبت هوا و شدت میزان نور و روشنایی رفته‌ایم و برای این پارامترها از دو سنسور استفاده کردہ‌ایم که به شرح زیر هستند.

۱-۳-۵) DHT11 سنسور

این سنسور که به صورت مازولار و غیر مازولار در بازار یافت می‌شود؛ توانایی اندازه‌گیری رطوبت هوا و دما را دارد. مدل دیگری نیز به نام DHT22 وجود دارد که دقیق و بازه بالاتری نسبت به این مدل دارد. در این پژوهه از مدل غیر مازولار استفاده شده است و مشخصات آن به شرح زیر است :



شکل ۶-۵) سنسور DHT11

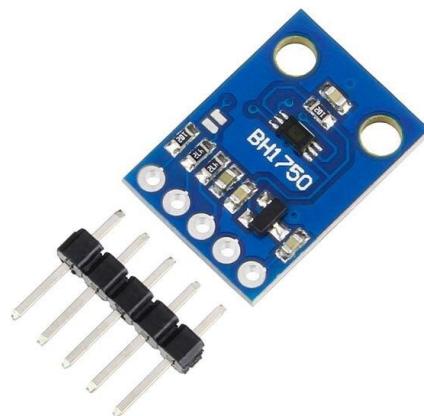
▪ ولتاژ کاری : ۳,۵ تا ۵,۵ ولت

- جریان کاری : ۳۰ میلیآمپر
- بازه اندازه‌گیری دما : ۰ تا ۵۰ درجه سانتی‌گراد
- بازه اندازه‌گیری رطوبت : ۲۰ الی ۸۰ درصد
- دقیق : حدود ۱ درجه اختلاف در دما و ۱ درصد اختلاف در رطوبت

پین‌های این سنسور از چپ به راست عبارت‌اند از : پایه Vcc، پایه Data، پایه NC، پایه GND

BH1750 سنسور ۵-۳-۲)

این سنسور برای اندازه‌گیری میزان نور گلخانه‌ها مورد استفاده قرار می‌گیرد. از آنجایی که شدت نور برای هر گلخانه‌ای حیاتی است این سنسور شدت نور را بر اساس lx یا همان لاسکس اندازه‌گیری می‌کند. لاسکس یکای شدت روشنایی در واحد SI^۲ است که به صورت شار نوری بر واحد سطح تعریف می‌شود. هر لاسکس معادل یک لومن بر مترمربع است. در علوم نورسنجی، لاسکس به عنوان مقیاسی برای سنجش شدت نوری که به وسیله چشم انسان در کم می‌شود مورد استفاده قرار می‌گیرد.



شکل ۵-۷) سنسور BH1750

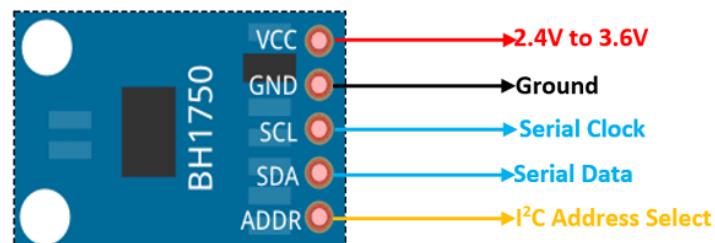
مشخصات این سنسور به شرح زیر است :

^۲ The International System of Units

- ولتاژ کاری : ۲,۴ الی ۳,۶ ولت
- جریان کاری : ۰,۱۲ میلی آمپر
- راه ارتباطی : I2C
- بازه : ۱ الی ۶۵۵۳۵ لاکس
- دارای مبدل آنالوگ به دیجیتال درونی

پین‌های این سنسور عبارت‌اند از : پین Vcc، پین GND، پین SCL، پین SDA، پین ADDR

- پین Vcc : مربوط به اتصال منبع تغذیه است.
- پین GND : همان پین زمین و گراند است.
- پین SCL : برای ایجاد پالس‌های کلاک در ارتباط I2C به کار می‌رود.
- پین SDA : برای ارسال داده از طریق ارتباط I2C به کار می‌رود.
- پین ADDR : زمانی متصل می‌شود که بیش از دو مازول وجود داشته باشد.



شکل ۴-۵) عملکرد پایه‌های سنسور BH1750

۴-۵) قطعات قابل کنترل

از آنجایی که بخش دوم عنوان پروژه، کنترلینگ می‌باشد و در گلخانه وسایلی نظری : فن و تهویه، لامپ، هیتر و ... برای اعمال برخی تغییرات وجود دارند؛ در این پروژه از دو وسیله : ۱. فن ۲. پمپ آب کوچک

استفاده شده است تا بتوان بنا بر شرایط گلخانه دستور روشن و خاموش کردن این وسایل را از سمت نرم‌افزار صادر کرد.

۱-۴-۵) فن

یک فن به ابعاد 9×9 در این پروژه به کاررفته تا زمانی که دما در گلخانه افزایش پیدا کرد با روشن کردن این وسیله دما کاهش یابد. سعی شده است با به کارگیری این فن، سیستم تهویه گلخانه‌های صنعتی شبیه‌سازی شود.



شکل ۱-۵) فن ۱۲ ولت دی‌سی

این فن به منبع تغذیه ۱۲ ولت نیاز دارد و برای راهاندازی، از آنجایی که ولتاژ خروجی رزیبری پای ۳,۳ ولت دائم و ۵ ولت به صورت سیگنال است به یک رله نیازمندیم که درباره رله جلوتر مواردی ذکر خواهد شد.

۲-۴-۵) پمپ آب کوچک

گلخانه برای حفظ سلامت گیاهان نیاز به یک سیستم آبرسانی دارد. فرقی ندارد درباره چه گلخانه‌ای صحبت می‌کنیم همواره باید مطمئن شد که گیاهان به اندازه کافی آبرسانی می‌شوند.



شکل ۵) پمپ آب ۱۲ ولت

در این پروژه از یک پمپ آب میکرو استفاده شده است که ولتاژ کاری آن بین ۱۲ ولت است و این پمپ یک موتور DC است و آب از یک دهانه وارد و از دهانه دیگر خارج می‌شود.

به طبع باید در کنار این پمپ منبع آبی وجود داشته باشد تا پمپ بتواند با استفاده از یک شلنگ آب را به مکان گیاه برای آبرسانی انتقال دهد. همچنین همانند فن برای قطع و وصل کردن این پمپ به یک رله نیازمندیم.



شکل ۱۱-۵) شلنگ برای انتقال آب به سمت گیاه

۵-۵) مازول رله

پیش‌تر درباره رله صحبت شد ولی رله چیست و چه کاربردی دارد؟! رله نوعی کلید الکترونیکی است که با عبور جریان الکتریکی از داخل آن، همانند یک کلید عمل می‌کند. در خروجی می‌تواند یک مدار دیگر را به حالت باز یا بسته تبدیل کند.



شکل ۱۲-۵) مازول رله ۲ کاناله

در این پروژه از یک مازول رله استفاده شده است و نه خود رله زیرا مازول رله مدار فرمان دارد. به عبارتی برای راهاندازی رله نیازمند طراحی مدار ولتاژ برگشت و محافظت مدار هستیم. برای این منظور بستگی به طراحی انجام گرفته، بایستی از ترانزیستور، اپتوكوپلر و دیود استفاده کنیم.

ولی با استفاده از مازول رله، Relay بدون نیاز به درگیر شدن در این بخش، می‌توانیم خیلی ساده خروجی برد را به ورودی مازول رله متصل کنیم. رله استفاده شده برای این پروژه یک مازول رله دو کاناله است.

منظور از کانال همان تعداد رله‌های به کاررفته می‌باشد و از آنجایی که دو وسیله کنترلی در این پروژه وجود دارد پس به دو عدد رله نیاز داریم که ماژول رله دو کاناله نیاز سیستم را برطرف می‌نماید.

۶-۵) خلاصه

آنچه در این فصل بررسی شد:

لوازم استفاده شده برای پیاده‌سازی سخت‌افزار پروژه :

قفسه به عنوان ساختار ▪

برد رزبری پای ▪

سنسورها ▪

▪ سنسور سنجش دما و رطوبت هوای

▪ سنسور سنجش شدت نور

▪ قطعات قابل کنترل

▪ فن

▪ پمپ آب کوچک

▪ ماژول رله

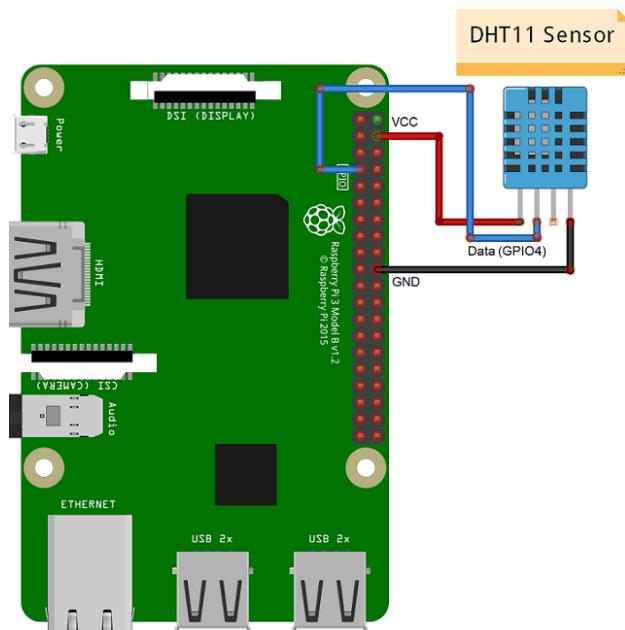
فصل ششم)

روندهای پیاده‌سازی

در این فصل به سراغ اتصال و پیاده‌سازی قطعات پروژه می‌رویم و هر کدام از قطعات را به نوبت به برد متصل کرده و باز سازوکار آن آشنا می‌شویم.

۶-۱) پیاده‌سازی سنسور DHT11

همان‌طور که در فصل قبل گفته شد سنسور DHT11 این پروژه داری ۴ پین است که هر کدام از این پین‌ها به جز پین NC که نیازی به اتصال ندارد، به پین مربوطه در قسمت GPIO برد متصل می‌شود.

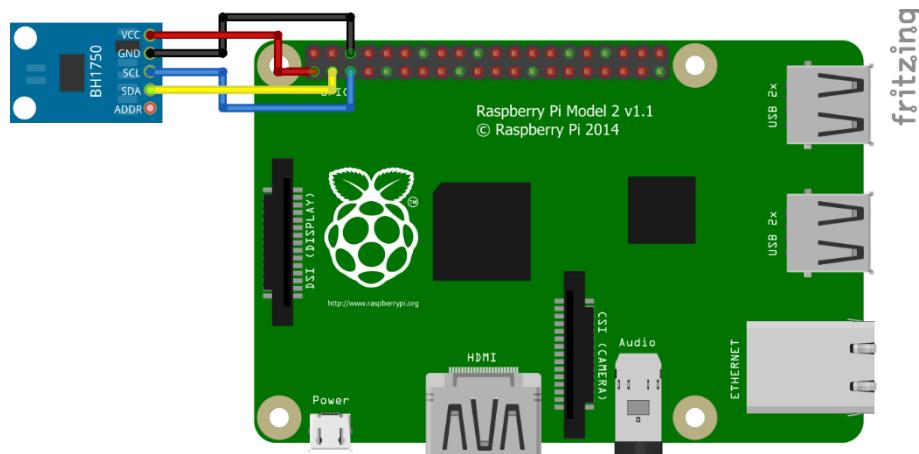


شکل ۶-۱) اتصال سنسور DHT11 به برد

پایه Vcc به یکی از پایه‌های ۵ ولت رزبری‌پای، پایه GND به یکی از پایه‌های گراند رزبری‌پای و پایه Data به یکی از پایه‌های GPIO که در شکل بالا به GPIO4 متصل شود. در این لحظه اتصالات سنسور به برد تکمیل و تنها برنامه و اسکریپتی نیاز است تا اطلاعات لازم را جمع‌آوری کند که مراحل کد را در فصل بعد بررسی خواهیم کرد.

۶-۲) پیاده‌سازی سنسور BH1750

در این سنسور به دلیل وجود تنها یک مژول نیازی به اتصال پایه ADDR، نیست. سایر پایه‌های بر اساس عنوانی که دارند به پایه‌های مربوطه در پرد متصل می‌شوند.



شکل ۲-۶) اتصال سنسور BH1750 به برد

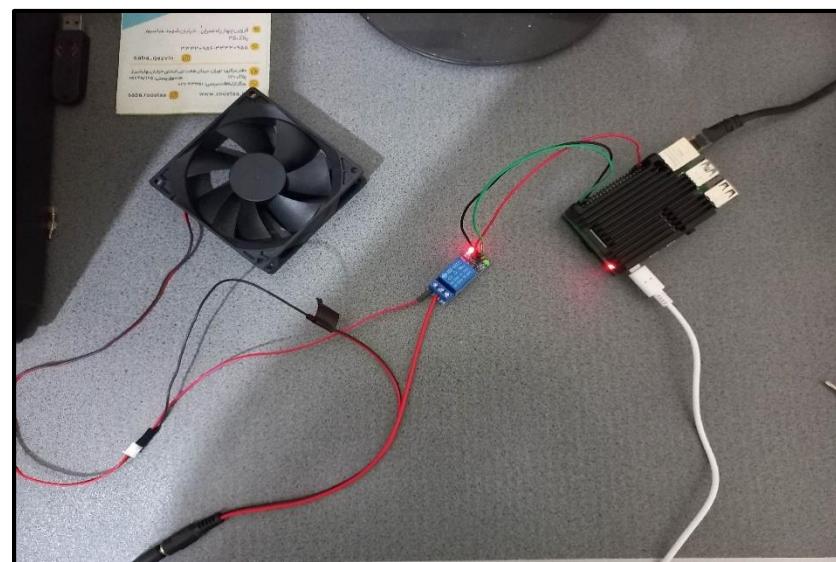
پایه VCC به دلیل نیاز به ولتاژ کمتر نسبت به DHT11 به یکی از پایه‌های ۳،۳ ولت و نه ۵ ولت متصل می‌شود، پایه GND هم به یکی از پایه‌های گراند برد و هر کدام از پایه‌های SCL و SDA به پایه‌های شماره ۳ و ۵ که با برچسبی با همین نام مشخص شده‌اند متصل می‌شوند.

مراحل بعدی نظری خواندن اطلاعات از طریق اجرای اسکرپت پنامه در فصل بعد بیگانی می‌شود.

۶-۳) پیاده‌سازی فن

برای پیاده‌سازی فن، نیازمند چند قطعه دیگر نیز هستیم؛ قطعاتی مانند: رله، آدپتور ۱۲ ولت و مبدل‌ها و اتصالاتی مانند موارد بالا. فن دارای دو سیم قرمز و مشکی است که سیم مشکی، باید به سیم مشکی فیش آدپتور

متصل شود و سیم قرمز به خروجی رله به نام NO^۳ وصل شود و سیم قرمز فیش آداتپتور باید به خروجی CO متصل شود.



شکل ۶-۳) تصویر پیاده‌سازی فن

ورودی رله نیز بر حسب لیبل‌هایی که روی پایه‌های رله نوشته شده است به پایه‌های متشابه برد وصل می‌شود یعنی پایه VCC به یکی از پایه‌های ۵ ولت، پایه GND به یکی از پایه‌های گراند برد و پایه IN یا همان D1Tا به یکی از پایه‌های GPIO برای تعریف به عنوان خروجی وصل می‌شود.

² Normally Open

9

³ Common

0

۴-۶) پیاده‌سازی پمپ آب

به دلیل ماهیت موتور DC چه در فن و چه در پمپ مراحل پیاده‌سازی پمپ هم بسیار شبیه به فن می‌باشد. برای پیاده‌سازی پمپ آب نیازمند یک رله، آداپتور ۱۲ ولتی و یک فیش تبدیل برای آداپتور هستیم.



شکل ۴-۶) تصویر پیاده‌سازی پمپ آب

پمپ که همانند فن دارای ۲ سیم قرمز و مشکی است که سیم مشکی به سیم مشکی فیش آداپتور وصل می‌شود و سیم قرمز به خروجی NO رله و سیم قرمز فیش آداپتور به خروجی CO رله وصل می‌شود.

۵-۶) خلاصه

در این فصل به اتصال و پیاده‌سازی تک تک سنسورها و قطعات به صورت جدا پرداختیم و نیمی از راه برای به دست آوردن نتیجه و خروجی از سخت‌افزار را طی کردیم.

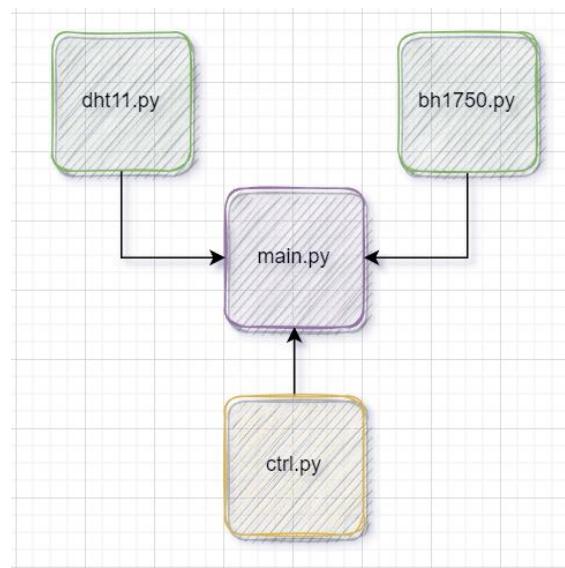
فصل هفتم)

برنامه‌نویسی سخت‌افزار

۷-۱) نمای کلی برنامه

یک گام تا اجرای نهایی و رسیدن به هدف در قسمت سخت‌افزار باقی‌مانده است و آن‌هم نوشتن برنامه‌ای است که بتواند فرمان لازم را به پایه‌های GPIO بدهد تا اطلاعات سنسورها جمع‌آوری و دستگاه‌ها آماده روشن و خاموش شدن بشوند. زبان برنامه‌نویسی به کاررفته در توسعه برنامه سخت‌افزار، زبان پایتون است. زیرا این زبان به صورت پیش‌فرض روی برد نصب است و زبان معمول برای برنامه‌نویسی در برد رزبری‌پای می‌باشد.

بدین منظور یک الگوی طراحی ترتیب داده شده است که به صورت زیر عمل می‌کند :



شکل ۷-۱) شماتیک کلی برنامه سمت سخت‌افزار

برنامه هر کدام از سنسورها در فایل یا اسکریپتی جداگانه نوشته شده است و برنامه فن و پمپ در یک فایل مشترک نوشته شده است سپس تمامی کدهای مربوط به این فایل‌ها به درون یک فایل اصلی به نام main ایمپورت می‌شود. که درنهایت این فایل اصلی همواره در حالت اجرا قرار می‌گیرد.

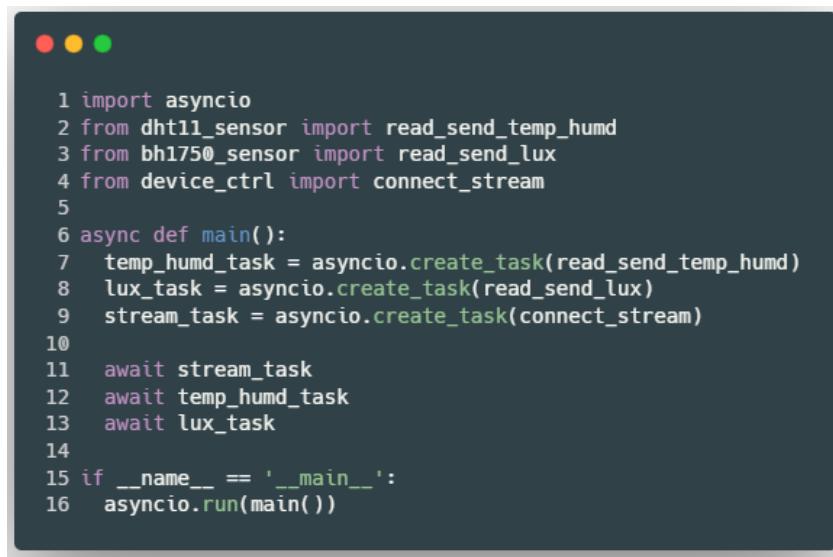
حالت اجرای این برنامه‌ها بهصورت اجرای همرونده است بهطوری که داده‌ها رأس زمان خاص جمع‌آوری و سپس به سمت سرور ارسال می‌شوند همچنین در این مدت برنامه همواره منتظر سیگنال دستگاه‌های کنترلی است تا در صورت لزوم قطعاتی مانند پمپ و فن را کنترل نماید(روشن و خاموش کند).

۲-۷) جزئیات هر برنامه

به بررسی جداگانه هر فایل می‌پردازیم تا با دستورات و نحوه اجرای آن‌ها بیشتر آشنا شویم.

۱-۲-۷) برنامه main

این برنامه وظیفه مدیریت سایر برنامه‌ها و اجرای آن‌ها بهصورت همرونده را دارد.



```
1 import asyncio
2 from dht11_sensor import read_send_temp_hum
3 from bh1750_sensor import read_send_lux
4 from device_ctrl import connect_stream
5
6 async def main():
7     temp_hum_task = asyncio.create_task(read_send_temp_hum)
8     lux_task = asyncio.create_task(read_send_lux)
9     stream_task = asyncio.create_task(connect_stream)
10
11    await stream_task
12    await temp_hum_task
13    await lux_task
14
15 if __name__ == '__main__':
16     asyncio.run(main())
```

شکل ۲-۷) برنامه Main که بهگونه‌ای hub برنامه‌های دیگر است

همان طور که مشاهده می نمایید برنامه به صورت ناهمگام اجرا می شود. زیرا اجرای برنامه در حالت همگام باعث کندی سیستم و معطل شدن CPU می شود و حتی در موقعی باعث ایجاد ناهماهنگی در سیستم می شود. اجرای ناهمگام باعث می شود برنامه پشت یک تسک خاص معطل نماند و کار سایر تسک ها را نیز انجام دهد برای رسیدن به اجرای ناهمگام در پایتون از پکیجی به نام `asyncio` استفاده کردہایم.

توابع کاری سایر برنامه ها را به فایل `main` وارد می شوند و به کمک قابلیت `task` در پکیج `asyncio` برنامه ها را به طور همزمان به اجرا در می آیند.

۲-۲-۷ برنامه dht11_sensor

این برنامه که در فایل `dht11_sensor.py` قرار دارد وظیفه خواندن و جمع آوری اطلاعات دما و رطوبت هوا و سپس ارسال به سمت سرور را دارد.

```

● ● ●

1 async def read_send_temp_humd():
2     DHT_SENSOR = Adafruit_DHT.DHT11
3     DHT_PIN = 12
4
5     async with aiohttp.ClientSession() as session:
6         while True:
7             humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)
8             if humidity is not None and temperature is not None:
9                 print("Temp = {0:0.1f}C , Humidity = {1:0.1f}%".format(temperature, humidity))
10                try:
11                    current_time = (datetime.now()).strftime("%H:%M:%S")
12                    current_date = str(JalaliDate.today())
13                    url = "http://192.168.1.100:8000/SensorValueInfo/"
14                    temp_request = await session.post(url, data={
15                        'sensor_id': 1,
16                        'value': float(temperature),
17                        'current_time': current_time,
18                        'current_date': current_date,
19                    }, ssl=False)
20
21                    humd_request = await session.post(url, data={
22                        'sensor_id': 2,
23                        'value': float(humidity),
24                        'current_time': current_time,
25                        'current_date': current_date,
26                    }, ssl=False)
27
28                    await asyncio.sleep(60)
29
30                except(Exception) as error:
31                    print("Requesting Error : ", error)
32                    await asyncio.sleep(5)
33
34            else:
35                print("Sensor failure, Check wiring!")
36                await asyncio.sleep(10)

```

شکل ۷-۳ کد ارسال دما و رطوبت هوا

برنامه سنسور دما و رطوبت هوای داخل یک تابع به نام `read_sensor_temp_humid` پیاده‌سازی شده است. با مشخص کردن پایه دیتا سنسور که در اینجا ۱۲ است و به ۱۲ GPIO اشاره می‌کند و دستور `read` در خط ۸ در تصویر بالا اطلاعات را به صورت یک تاپل خوانده و مقدار اول را داخل متغیر `humidity` و مقدار دوم را داخل متغیر `temperature` قرار می‌دهد. در ادامه اطلاعات از طریق کتابخانه `aiohttp` با یک ریکوئست از نوع `post` و شامل اطلاعاتی نظیر: آیدی سنسور، مقدار پارامتر، زمان ارسال، تاریخ ارسال است.

۳-۲-۷ bh1750_sensor برنامه

این برنامه از لحاظ پیاده‌سازی شبیه به برنامه `dht11` است و فقط برای اجرا نیاز است تا اینترفیس I²C برد رزبری‌پای فعال باشد.

```

● ● ●

1 async def read_send_lux():
2
3     i2c = board.I2C()
4     sensor = adafruit_bh1750.BH1750(i2c)
5
6     async with aiohttp.ClientSession() as session:
7         while True:
8             if sensor.lux is not None:
9                 print("%.2f Lux" % sensor.lux)
10            try:
11                current_time = (datetime.now()).strftime("%H:%M:%S")
12                current_date = str(JalaliDate.today())
13                url = "http://192.168.1.100:8000/SensorValueInfo/"
14                lux_request = await session.post(url, data={
15                    'sensor_id': 3,
16                    'value': round(sensor.lux, 2),
17                    'current_time': current_time,
18                    'current_date': current_date,
19                }, ssl=False)
20
21                await asyncio.sleep(60)
22
23            except(Exception) as error:
24                print("Requesting Error : ", error)
25                await asyncio.sleep(5)
26            else:
27                print("Sensor failure, Check wiring!")
28                await asyncio.sleep(10)

```

شکل ۳-۷) کد ارسال شدت نور

همانند برنامه قبلی این برنامه نیز اطلاعات را از طریق متغیر `sensor` خوانده و ذخیره می‌کند. سپس یک ریکوئست از نوع `post` به آدرس سرور ارسال می‌کند.

این برنامه وظیفه شنود تغییرات در یک آدرس خاص را دارد سپس بر اساس نوع رخداد، تصمیم می‌گیرد تا وسائل را کنترل نماید(روشن و خاموش نماید).

```

1  async def connect_stream():
2
3      in1 = 21
4      in2 = 26
5
6      GPIO.setwarnings(False)
7      GPIO.setmode(GPIO.BCM)
8
9      GPIO.setup(in1, GPIO.OUT, initial = GPIO.HIGH)
10     GPIO.setup(in2, GPIO.OUT, initial = GPIO.HIGH)
11
12     async with sse_client.EventSource(
13         "http://192.168.1.100:8000/events/") as event_source:
14         try:
15             async for event in event_source:
16
17                 if event.message == 'fan_status':
18                     event_data = json.loads(event.data)
19                     if event_data['status'] == True:
20                         GPIO.output(in2, GPIO.LOW)
21                         print('\nFan is turning ON !', event.data)
22                     elif event_data['status'] == False:
23                         GPIO.output(in2, GPIO.HIGH)
24                         print('\nFan is turning OFF !', event.data)
25
26                 elif event.message == 'pump_status':
27                     event_data = json.loads(event.data)
28                     if event_data['status'] == True:
29                         GPIO.output(in1, GPIO.LOW)
30                         print('\nPump is turning ON !', event.data)
31                     elif event_data['status'] == False:
32                         GPIO.output(in1, GPIO.HIGH)
33                         print('\nPump is turning OFF !', event.data)
34
35             except(ConnectionError, KeyboardInterrupt) as error:
36                 print("Stream Error: ", error)
37                 GPIO.cleanup()
38                 await asyncio.sleep(5)

```

شکل ۷-۵) کد برنامه کنترل قطعات

نحوه ارتباط سرور با برد به صورت SSE است؛ در این روش دیگر نیازی نیست تا کلاینت برای به دست آوردن اطلاعات همواره به سمت سرور ریکوئست بزند و باعث شود بار اضافی به سرور تحمیل کند در عوض زمانی که اطلاعات سمت سرور پردازش و ثبت می‌شود سرور اطلاعات را در قالب یک ریکوئست به کلاینت ارسال می‌نماید. که این روش بسیار بهینه‌تر از روش‌های اولیه است.

در این برنامه به صورت مستقیم با پایه‌های GPIO سروکار داریم، پایه‌های مربوطه که همان پایه 21 GPIO و 26 GPIO است را به صورت خروجی تعریف کرده‌ایم و مقدار اولیه ۱ را به دلیل اینکه این پایه‌ها به مژوول رله وصل‌اند و این رله از نوع Active Low است قراردادیم تا در ابتدای کار و پس از روشن نمودن برد قطعات کنترلی پروژه روشن نشوند. سپس برنامه آدرس events/ را شنود می‌کند در صورتی که رخدادهایی به نام‌های fan_status و pump_status دریافت شوند، داده‌ها از فرمت json به یک دیکشنری که از ساختمان داده‌های پایتون است تبدیل می‌شود. سپس کلید status بر حسب مقدار ارزیابی می‌شود اگر true باشد قطعه موردنظر روشن شده و کار می‌کند و بلعكس.

۳-۷) خلاصه

در این فصل روند توسعه از سمت کد نویسی دنبال شد و شماتیک کلی برنامه‌ها که شامل ۴ برنامه است مشخص شد. سپس جزئیات هر برنامه بررسی شد برنامه main که وظیفه اجرا برنامه را به صورت async دارد و سایر برنامه‌ها در قالب توابع وارد برنامه main شده و سپس در آنجا فراخوانی می‌شوند.

پیشنهادها

برای درک بهتر سمت سرور در فریمورک جنگو به لینک‌های زیر مراجعه نمایید:

- [Intro to Django](#)
- [Django REST](#)
- [Django Channels](#)

برای درک بهتر سمت فرانت در فریمورک ریاکت به لینک‌های زیر مراجعه نمایید:

- [React Overview](#)
- [ReactDOM](#)

برای درک بهتر کار با برد رزبری‌پای به لینک‌های زیر مراجعه نمایید:

- [Raspberry Pi GPIOs](#)
- [Raspberry Pi DIYs](#)

فهرست مراجع

- <https://www.djangoproject.com/>
- <https://reactjs.org/>
- <https://pypi.org/project/django-eventstream/>
- <https://pypi.org/project/aiohttp-sse-client/>
- <https://www.chartjs.org/>
- <https://tutorials-raspberrypi.com/raspberry-pi-control-relay-switch-via-gpio/>
- <https://www.thegeekpub.com/236867/using-the-dht11-temperature-sensor-with-the-raspberry-pi/>
- <https://components101.com/sensors/bh1750-ambient-light-sensor>
- <https://lastminuteengineers.com/two-channel-relay-module-arduino-tutorial/>
- <https://github.com/koolkishan/react-taxi-dashboard-with-animations>
- <https://faradars.org/courses/fvee9707-raspberry-pi-programming-using-python>

Abstract

Mankind have always sought to take care of and monitor their capital over the years, the word capital can have different meanings and depending on the time, it can include many materials. Since then, people have tried various methods to protect their capital; Methods that sometimes require a lot of human resources, exorbitant costs, both in terms of material and time, thus man has always been looking for a suitable and alternative method.

With the entry into the era of electronics and computers, which led to the achievement of high degrees in various medical and research fields, the industry did not remain indifferent.

Using this advantage, business owners succeeded in inventing a method that did not have the difficulties of traditional methods and relied more on technology, such as greenhouses, cattle farms, fish farming and server rooms by being equipped with various sensors and communication ways through The Internet managed to find a method of monitoring and control that could display the current state of a person's capital accurately and with minimal error through various parameters.

One of the common examples that is very suitable for this method is the greenhouse. So that the old and traditional greenhouses required a lot of human resources to control such as: water supply, air circulation and ventilation, light entry. Today, people can monitor and even control their greenhouses through their computers and smartphones.

In this project, an attempt has been made to simulate and display the monitoring and control process of a small-scale greenhouse.

Key words: capital, monitor, supervision, control, greenhouse



University of Zanjan

Faculty of Engineering
Bachelor's degree
Computer Engineering

Greenhouse Monitoring and Controlling

By:

Farhad Jamali
Amin Shahbaghi

Supervisor:
Dr. Ali Amiri

September 2022

