
Google
how to build an iot smart garden
The $300 Lettuce: Building a Smart Garden – I
https:/
Feb 25
cloud
Step 4: Programming Our Projec
Step 3: Compile
char ssid[] = "Wish I had Google Fiber";
#define STATIC_IP ADDRESS
Build a smart gard with these 3 DI
https://open e/17/3/arduino-g
d garden to the
GraphQL
JavaScript
write a GraphQL endpoint
framework and Node.js 8.10 Lambda. Th
needed right now, but these were all things
in the future to use third party weather and power price
GraphQL will provide more value at that point. Serverless is a really enjo
way to work with Lambda (it also supports other providers!), and you should
definitely try it if you haven't yet.
tab 'StandardFirmataV
Compile and Upload sketch to MKR1000
clone
_Garden_Wat...
based Internet of Things (IoT) smart garden ... homes,
building, smart cities and smart. healthcare as well as ...

Your Internet of Things

What is it all about?

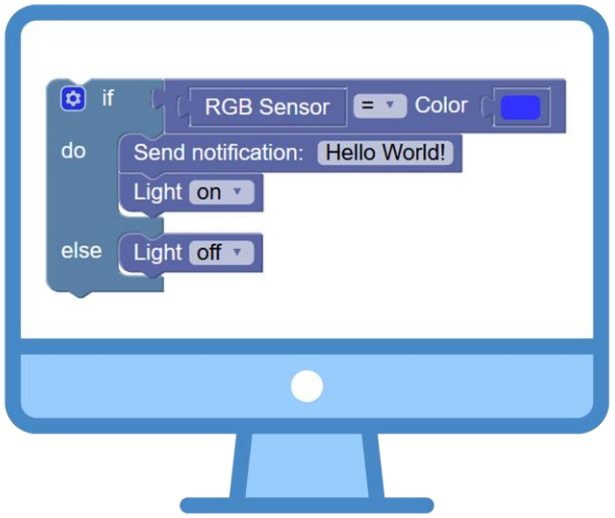
YIoT is a product that makes it possible for the general public to build IoT projects on their own, even **without previous knowledge**.

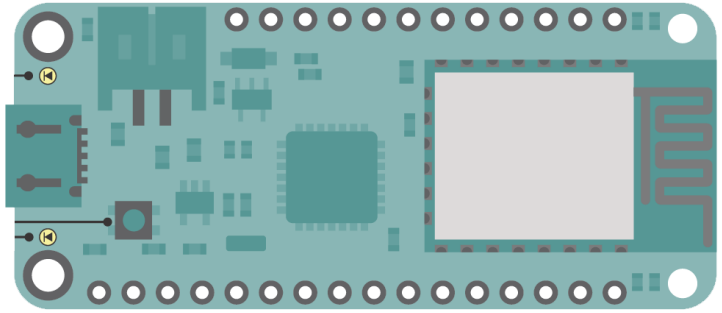
Think of a gardener who wants to build on his own a device that measures moisture in his garden. Instead of having to learn how to write a code for Arduino devices, how to prepare all of the components and how to get past the technological barrier, **WE** do it all simply for him – all he has to do is to define simple commands in natural language, plug pre-prepared devices, and, voila – the system is ready to use.

No coding!

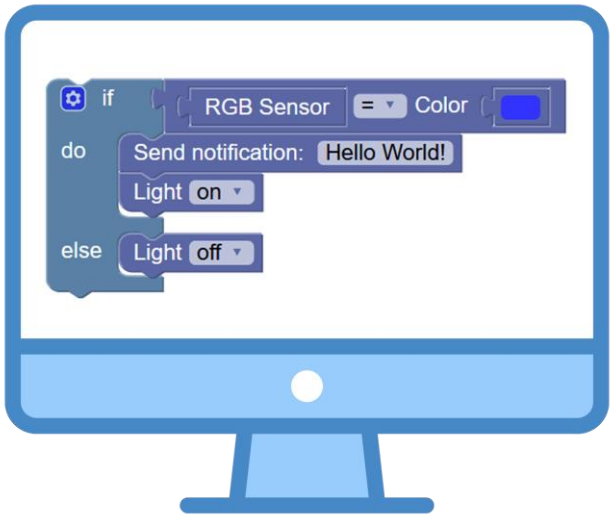
```
if (tcs.available()) {
    static uint32_t prev_ms = millis();
    TCS34725::Color color = tcs.color();
    RGB sens = RGB(color.r, color.g, color.b);
    if (se
        if
            s
                le,
                r *>("Hello World!"),
            }
            di
    } else {
        digita ce(LED_PIN, );
    }
} else {
    Serial.println("Error: TCS unavailable");
}
```

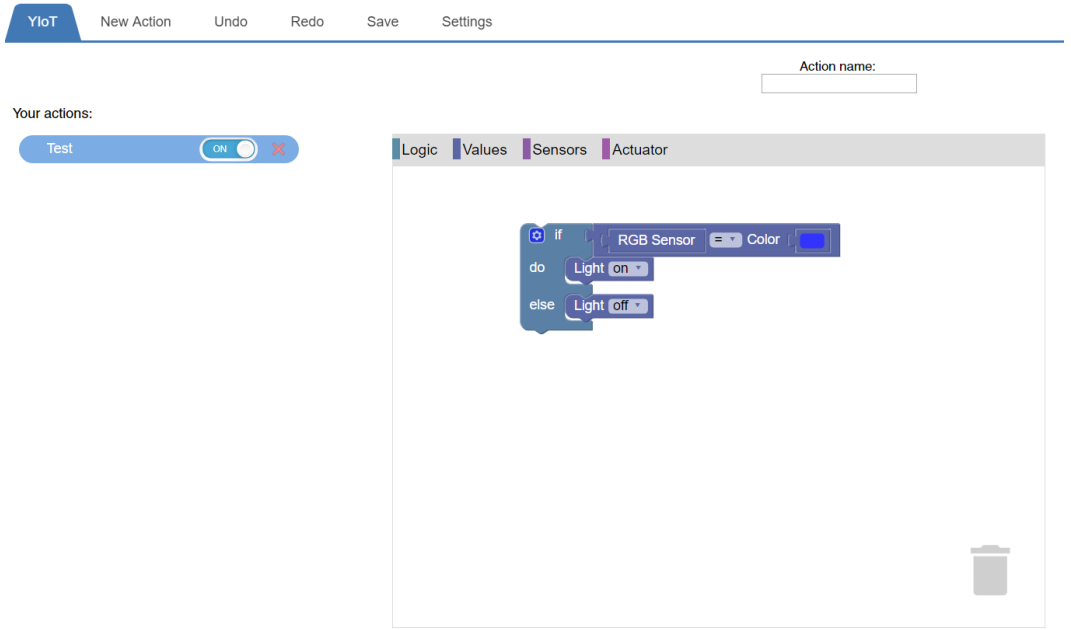
Architecture





Web App





User Web Portal

Node.JS

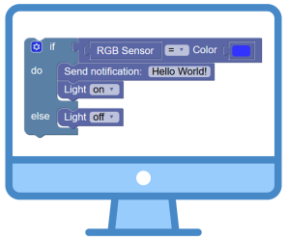
Blocky API

Device Setup

Create and Modify Actions

Web App

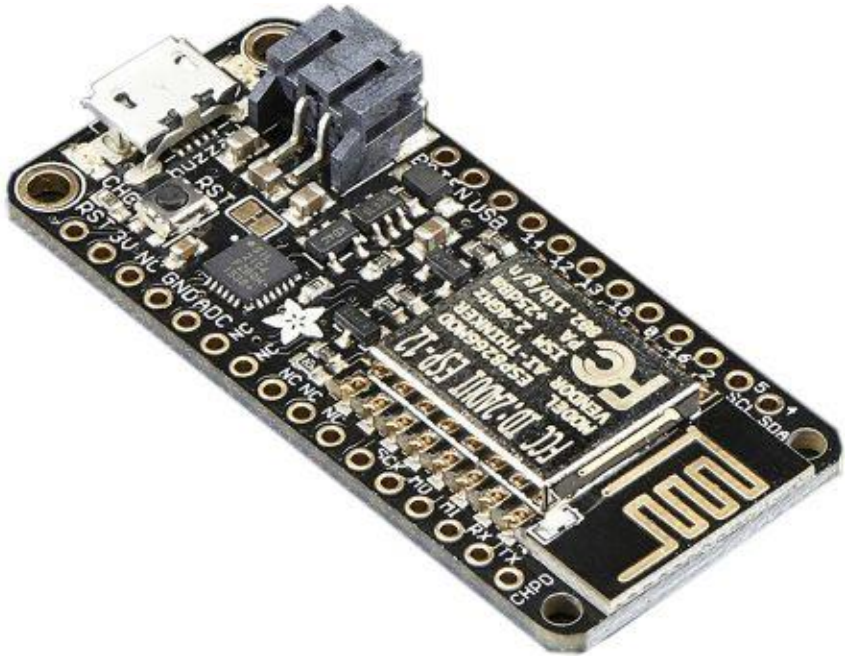
The WebApp is used for **creating**, **changing** and **configuring actions** on the Arduino devices. It is divided into two parts: the UI and the backend.

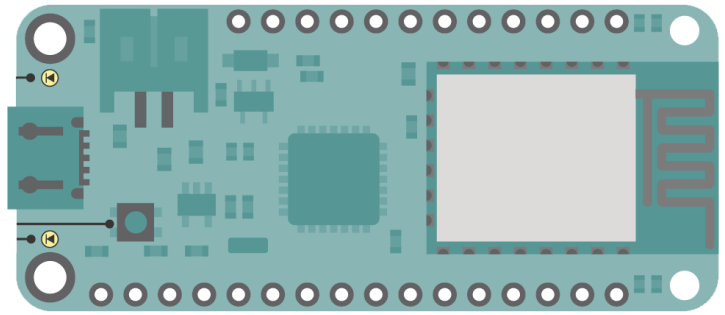


The **UI** is written in JavaScript, HTML and CSS. We use an open source API for creating actions. The actions blocks functionality is implemented with the Blockly API. The Blockly interface allows the users to write code without any knowledge in programming. In particular, the WebApp allows the user to configure settings for the Arduino such as the port numbers of the devices.

The **backend** is written in node JS. Its main purpose is to **parse** the actions that the user creates from XML representation into JSON objects, and to use the SQL server for **querying data** for the user.

Arduino Device





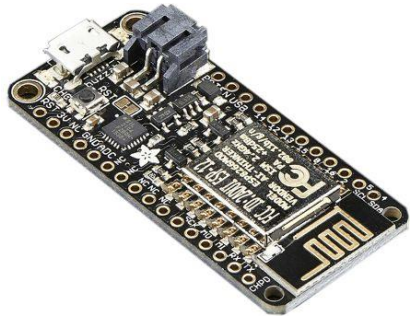
Arduino

Feather Huzzah ESP8266

C/C++

Modular code

Arduino Device



A major part of the YIoT system is the **Arduino devices**. These devices constitute the physical piece of the project – either **sensors** or **actuators**, they are connected to the internet with **WiFi** using the Adafruit Feather Huzzah ESP8266 module.

The interaction with the devices is based on **C/C++** programming languages. After we define actions in the WebApp, they are translated into **abstract syntax trees** (AST) and we run them repeatedly as they receive/send information from/to the devices.

Mobile App

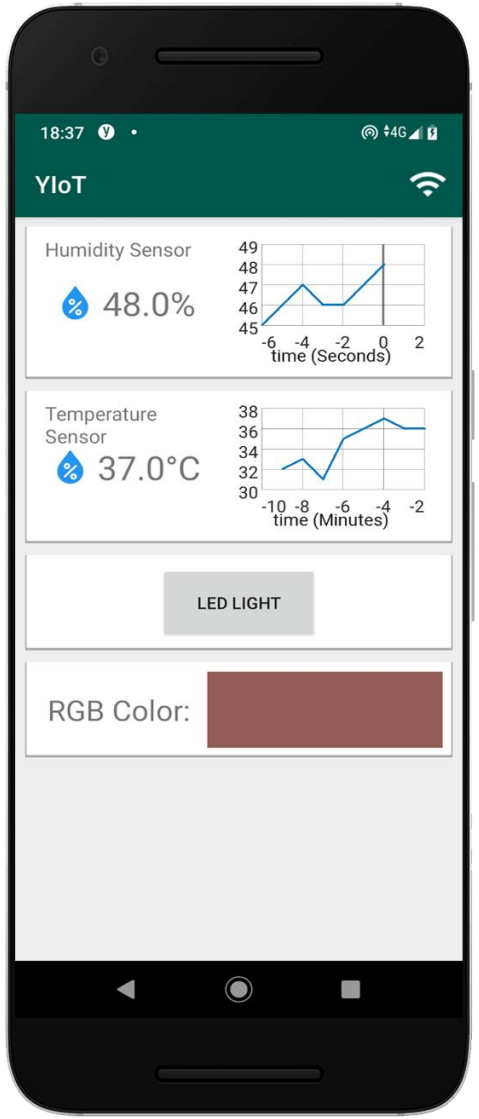
Android Companion App

Native Java

Interact with device

Displays Sensor Data

Receive Notifications from Device



Mobile App

The main purpose of the mobile application is to allow the user to **interact with devices**, by displaying the sensors' data and receiving notifications from the devices.

Written in Native Java, the mobile app is presenting **telemetry data** from the Arduino sensors in a numeric or a graph view. Data for special sensors is presented with the appropriate structure – for example, an RGB sensor data will be presented simply by the sampled color.

The mobile app also gives the ability to **control** the Arduino actuators, such as a LED device.

The application is communicating with the backend through the Azure functions.

Cloud

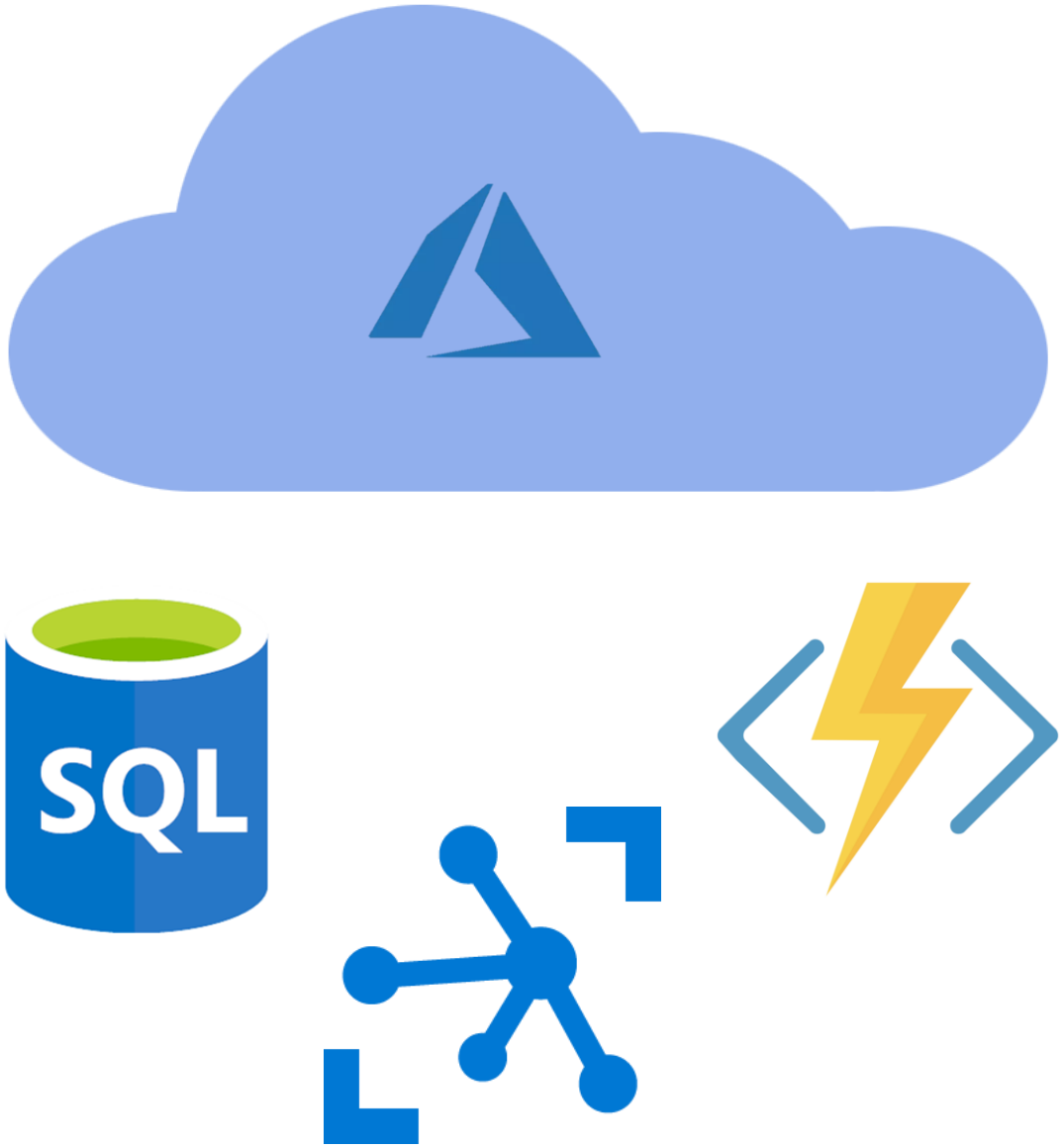
Azure Cloud Backend

IoT Hub

Notification Hub

Function App

SQL DB



Cloud

Eventually, every sub-system is connected to the heart of the project – the cloud. The cloud services are provided with **Micosoft Azure**, and include:

- **IoT Hub**: A service which allows us to establish bidirectional communication with the IoT devices.
- **Notification Hub**: a mobile push notification engine for quickly sending notifications the Android devices, using Google's GCM.
- **Function Apps**: a serverless compute service that enables us to run code on-demand. These functions constitutes the project's logic.
- **SQL Database**: The storage of out project's data, e.g. the user's actions defined on the WebApp.

Action Pipeline

User Created Blocks

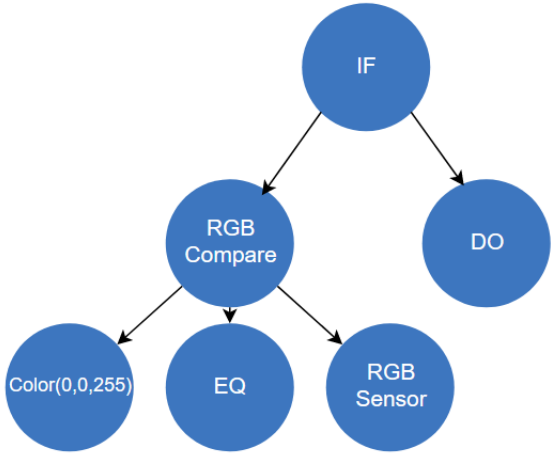
XML Representation

```
…
<value name="IF0">
        <block type="logic_compare_RGB"
id="C/;9QOfk5f1#|9cI-x9t">
          <field name="OP">EQ</field>
          <value name="A">
            <block type="RGB Sensor"
id="71*E?5`2yTPCr8nHt~q," />
          </value>
…
```

Json Representation

```
…
    "IF0": {
      "TYPE": "RGB_compare",
      "OP": "EQ",
      "OBJS_TO_COMPARE": {
        “LHS”: {
          "SENSOR_NAME": "RGB Sensor",
          "PORT": 99,
          "IS_SENSOR": true
…..
```

AST



Action Pipeline

One of the most unique properties of our project, is the ability to "**insta-compile**" code on the Arduino device via the cloud. This is done with our unique pipeline:

1. The user enters a list of actions into the WebApp by setting blocks in a logical order – this is the code that we will "compile" on the device.
2. Each action is translated in by the API into XML representation, afterwards we translate it into JSON objects.
3. The JSON objects are transmitted to the device and translated into Abstract Syntax Trees (AST) in C++ by the device itself.
4. The AST objects are being run repeatedly, each time receiving updated data from the sensors and triggering the actuators accordingly.

Thanks!

Yuval Weiss

Dan Tavori

Michael Khaitov