

Escola Senai "Anchieta"

**PÓS-GRADUAÇÃO EM SISTEMAS EMBARCADOS**

**SISTEMA OPERACIONAL EM TEMPO REAL**

**PROJETO: REGADOR INTELIGENTE**

Elaborado por:

**LEONARDO PONGILLO**

**MARCOS FLÁVIO SOARES**

Turma: 11SE

**SÃO PAULO – SP**

2022

**RESUMO**

Este trabalho de desenvolvimento apresenta um projeto unindo o Free RTOS (Real Time Operating System) e conectividade IOT (Internet of Things).

Foi desenvolvido um regador automatizado, com interface de comunicação via Wi-fi, utilizando o protocolo MQTT (Message Queue Telemetry Transport) para envio de informações para uma plataforma IOT.

O trabalho consta também com o fluxo de tarefas no sistema operacional, utilizando os conceitos de tasks, semáforos e filas, além de sensores de monitoramento de umidade e dispositivos para a ligação do regador.

**Palavras-chave:** Regador, IoT, ESP32, MQTT, Free-RTOS, Wi-fi, Monitoramento

**ABSTRACT**

This document presents a project unifying Free RTOS (Real Time Operating System) and IOT ( Internet of Things ) connectivy.

It was developed an automatized sprinkler, with communication interface via Wi-fi, using MQTT protocol (Message Queue Telemetry Transport) for sending information to an IOT platform.

The document also includes task flow in the operating system, using the concepts of *tasks*, *semaphores* and *queues*, and also humidity monitoring sensor and devices for power management.

**Keywords:** Sprinkler, IoT, ESP32, MQTT, Free-RTOS, Wi-fi, Monitoring.

**SUMÁRIO**

[INTRODUÇÃO 4](#_Toc101270140)

[CONCEITOS DE FREE-RTOS 4](#_Toc101270141)

[Protocolo MQTT 7](#_Toc101270142)

[Biblioteca WI-FI Manager 8](#_Toc101270143)

[METODOLOGIA 9](#_Toc101270144)

[Diagrama de blocos 11](#_Toc101270145)

[ESQUEMA elétrico 12](#_Toc101270146)

[Ordem inicial de execução das tarefas 13](#_Toc101270147)

[RESULTADOS E DISCUSSÃO 15](#_Toc101270148)

[CONCLUSÃO 19](#_Toc101270149)

[REFERÊNCIAS 20](#_Toc101270150)

# INTRODUÇÃO

Com a industrialização e urbanização das cidades, um tema que vem sendo cada vez mais estudado a cada ano é o desperdício e a limitação dos recursos naturais.

Dentre os recursos naturais mais abundantes do planeta, a água, na condição de potável, é também um dos recursos mais que vem diminuindo a cada dia, por vários fatores, entre eles:

* ocupação de áreas de manancial;
* desmatamento;
* degradação das nascentes
* efeitos climáticos.

Tanto no meio urbano, quanto no rural, uma forma de conter este desperdício é controlando o uso da água, de forma racional.

A irrigação é um dos grandes utilizadores da água no meio rural. Também está presente no meio urbano, em jardins públicos e privados, campos de futebol, hortas e pomares.

Por outro lado, com o surgimento de novas tecnologias, surgem novos pontos de vista sobre hábitos e costumes que não acompanharam a modernidade, com novas soluções para o que antes não era nem considerado um problema.

Por meio do desenvolvimento e prova de conceito de um regador que se comunica com um broker MQTT através da rede Wi-fi, utilizando placa de desenvolvimento ESP32 com o sistema operacional FREE-RTOS, e passando informações relevantes para serem analisadas, este trabalho tem como finalidade a diminuição do uso desnecessário de água, aliada com novos aprendizados e novas tecnologias.

## CONCEITOS DE FREE-RTOS

Desenvolvido por volta de 2003, o Free-RTOS é hoje um dos principais sistemas operacionais para microprocessadores e microcontroladores. Por ser uma ferramenta de código aberto, também é um dos mais confiáveis e fáceis de usar.

Uma das principais características deste sistema operacional é seu diagrama de estados, controlado pelo escalonador de tarefas.

O escalonador de tarefas decide qual tarefa será executada, examinando a prioridade que foi atribuída a cada uma.

A imagem a seguir mostra como funciona o fluxo de tarefas no Free-RTOS.

Diagrama

Descrição gerada automaticamente

**Figura 1:** Diagrama de transição de estado entre as tarefas. Fonte: Site FreeRTOS.org.

Uma tarefa pode fluir entre um dos seguintes estados:

**Ready** – Quando a tarefa está pronta para ser executada e está na fila;

**Running** – Quando a tarefa está sendo executada e está utilizando o processador;

**Blocked** – Quando a tarefa está esperando um evento externo ou em modo de espera, iniciado por um temporizador. Tarefas no estado de bloqueio normalmente tem um período de timeout, e são desbloqueadas depois desse período. Não podem ser selecionadas para entrar no estado de “Running”;

**Suspended** – Tarefas que são suspendidas por meio de comandos como vTaskSuspend(). Estas tarefas neste estado não tem um tempo para expirar, saindo do estado somente com o comando vTaskResume().

Num ambiente de desenvolvimento de software, a primeira tarefa a ser executada, depois da inicialização, é a função principal. Dentro dela, são criadas outras tarefas com prioridade mais alta, o que garante que ela só será executada uma vez.

Cada tarefa criada já entra em execução automaticamente e é importante equilibrar as prioridades entre as tarefas para que os recursos sejam distribuídos entre elas de acordo com a necessidade que cada uma tem de ser executada no tempo certo.

Uma simples inserção de atraso dentro da tarefa é importante para que o escalonador de tarefas a coloque em bloqueio por aquele período, possibilitando a chamada de outras tarefas e evitando a monopolização dos recursos para uma única tarefa.

O FREE-RTOS possui uma infinidade de API's que dão um toque especial ao sistema. Algumas das API's utilizadas são mostradas a seguir:

|  |  |
| --- | --- |
| API | Descrição |
| xTaskCreate | Cria uma tarefa. |
| vTaskDelete | Apaga uma tarefa |
| vTaskDelay | Introduz um atraso. |
| xQueueCreate | Cria uma fila |
| xQueuePeek | Pega o elemento do topo da fila, sem apagar. |
| xQueueSend | Envia para a fila. |
| xQueueOverwrite | Sobrescreve o topo da fila. |
| xSemaphoreTake | Obtém o semáforo. |
| xSemaphoreGive | Libera a o semáforo. |
| vSemaphoreDelete | Apaga o semáforo |
| xSemaphoreCreateBinary | Cria um semáforo binário |
| xSemaphoreGiveFromISR | Libera o semáforo. Usado numa interrupção. |
| xEventGroupCreate | Cria uma variável de event group. |
| xTaskCreatePinnedToCore | Cria tarefa para um processador fixo. |

**Tabela** 1: Algumas API’s utilizadas no projeto.

## Protocolo MQTT

O **MQTT** (Message Queue Telemetry Transport) é um protocolo utilizado para comunicação entre uma máquina e outra. Sua utilização vem crescendo a cada dia, principalmente com o crescimento da Internet das Coisas.

O protocolo permite a troca de mensagens entre dispositivos, possibilitando o envio de informações e de comandos de um para o outro.

Os dispositivos precisam ter acesso a um servidor MQTT, que intermedia a comunicação, conhecido como Broker.

Sendo assim, um dispositivo pode fazer tanto um envio de informações para o Broker, como o assinar recebimento de informações.

As informações são enviadas e recebidas através de canais, que são criados por tópico. O dispositivo que quer receber, faz a assinatura do canal naquele tópico, enquanto o dispositivo que quer enviar, simplesmente envia as informações.

O conteúdo da mensagem pode ser a leitura de um sensor, informações de estado de um interruptor, por exemplo, ou até mesmo um comando para ligar/desligar determinado periférico do dispositivo.

As informações do broker também podem ser acessadas por outros dispositivos, como computadores, que podem fazer o tratamento desses dados e inseri-los numa interface homem-máquina, de modo que facilite a manipulação, visualização e armazenamento dos dados, além de possibilitar uma maior interação do seu operador, através da inserção de comandos, com a visualização rápida das respostas, de acordo com o intervalo das mensagens enviadas e recebidas.

Um dos servidores MQTT mais conhecidos é o *Mosquitto*, que foi utilizado na neste trabalho como parte dos testes locais.

Como se trata de transferência de dados basicamente através do *stack* http, grande parte da mensagem é trafegada utilizando a formatação *Json* e a comunicação também pode ser feita de forma segura, com certificados SSL/TLS e autenticação no servidor.

publish subscribe example

**Figura 2:** Fluxo de mensagens MQTT. Fonte: Site CloudMQTT.

## Biblioteca WI-FI Manager

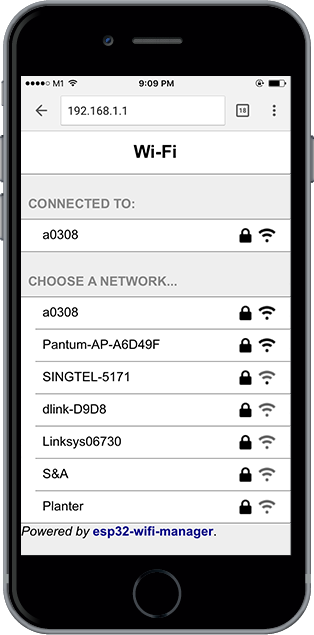
A biblioteca Wi-Fi Manager foi utilizada para facilitar a conexão do usuário, através do ESP32, à rede wi-fi, sem a necessidade da inserção da rede e credenciais do usuário no código fonte.

O usuário se conecta ao ponto de acesso wi-fi criado pelo microcontrolador e acessa o endereço padrão configurado. No caso, foi utilizado o endereço de ip: 10.10.0.1 para se ter acesso à página html do *Wi-fi manager* hospedada pelo microcontrolador.

A partir daí, o usuário seleciona a rede de acesso, entra com as credenciais, e o MCU se conecta diretamente à rede.

O *Wi-Fi Manager* foi incorporado ao projeto, como um componente externo.

Tela de celular

Descrição gerada automaticamente

**Figura 3**: Tela do Wi-Fi Manager. Fonte: Biblioteca Wi-Fi Manager.

# METODOLOGIA

Os materiais utilizados para o desenvolvimento do regador inteligente:

**-** Placa **ESP32**

|  |
| --- |
| Especificações:   * CPU: Xtensa® Dual-Core 32-bit LX6 * ROM: 448 KBytes * RAM: 520 Kbytes * Flash: 4 MB * Clock máximo: 240MHz * Wireless padrão 802.11 b/g/n * Conexão Wifi 2.4Ghz (máximo de 150 Mbps) * Antena embutida * Conector micro-usb * Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode e P2P Power Management * Modos de operação: STA/AP/STA+AP * Bluetooth BLE 4.2 * GPIO com funções de PWM, I2C, SPI * Tensão de operação: 4,5 ~ 9VTaxa de transferência: 110-460800bps * Suporta Upgrade remoto de firmware * Conversor analógico digital (ADC) * Distância entre pinos: 2,54mm * Dimensões: 49 x 25,5 x 7 mm |

**- Display 16x2**

|  |
| --- |
| Especificações:   * Cor backlight: Azul * Cor escrita: Branca * Dimensão Total: 80mm X 36mm X 12mm * Dimensão Área visível: 64,5mm X 14mm * Dimensão Caracter: 3mm X 5,02mm * Dimensão Ponto: 0,52mm X 0,54mm |

**- Modulo Relé**

|  |
| --- |
| Especificações:   * Modelo: JQC-3FF-S-Z * Tensão de operação: 5 VDC * Permite controlar cargas de até 220V AC * Corrente nominal: 71,4 mA * LED indicador de status * Pinagem: Normal Aberto, Normal Fechado e Comum * Tensão de saída: (28 VDC a 10A) ou (250VAC a 10A) ou (125VAC a 15A) * Furos de 3mm para fixação nas extremidades da placa * Tempo de resposta: 5~10ms * Dimensões: 50 mm x 37 mm x 18 mm * Peso: 30g |

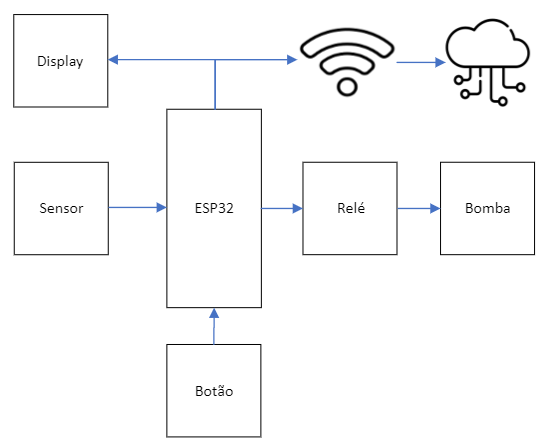
**- Bomba**

|  |
| --- |
| Dados Técnicos:   * Tensão de Operação: 5V DC * Elevação máxima: 40 a 110 cm * Vazão: 80 a 120L/h * Diâmetro externo de saída de água: 4,45 mm * Dentro interno de saída de água: 4,5 mm * Diâmetro: aproximadamente 24 mm * Comprimento: aproximadamente 25 mm |

**- Sensor de Umidade do Solo**

|  |
| --- |
| Especificações:   * Tensão de Operação: 3,3-5v * Sensibilidade ajustável via potenciômetro] * Saída Digital e Analógica * Fácil instalação * Comparador LM393 * Dimensões PCB: 3×1,5 cm * Dimensões Sonda: 6×2 cm * Comprimento Cabo: 21 cm * VCC: 3,3-5v * GND: GND * D0: Saída Digital * A0: Saída analógica |

### Diagrama de blocos



**Figura 4:** Diagrama de blocos do projeto Regador Inteligente.

### ESQUEMA elétrico

Diagrama, Esquemático

Descrição gerada automaticamente

**Figura 5:** Esquemático do projeto Regador Inteligente.

Principais softwares e bibliotecas utilizados no desenvolvimento

* MQTT
* LWIP
* FREE-RTOS
* ESP-IDF ou VS Code
* WI-FI Manager
* Linguagem C
* Windows
* Fritzing

Este trabalho foi dividido em basicamente duas etapas.

Na **primeira etapa**, foram verificados os modos de funcionamento do sensor de umidade, o display e o motor, além da plataforma mqtt que seria utilizada e o formato da mensagem. Para cada um desses periféricos, foi implementada uma tarefa.

|  |  |
| --- | --- |
| Tarefa | Prioridade |
| sensor\_display\_task | 10 |
| sensor\_umidade\_task | 10 |
| motor\_task | 10 |
| dashboard\_task | 10 |

**Tabela 2**: Tarefas iniciais e prioridades.

Para evitar complicações, as tarefas foram criadas com a mesma prioridade inicial.

Foi criada uma *Queue* do tipo Mailbox para o envio da informação do sensor de umidade para as tarefas *sensor\_display\_task* e *dashboard\_task*. Da mesma forma, foi criado outro Mailbox para o envio do estado da bomba para o dashboard, de modo que a informação possa ser visualizada.

Foi criado um semáforo para controlar a interrupção do botão, que, ao ser pressionado, liga o regador por um certo intervalo de tempo, independentemente da lógica de controle.

### Ordem inicial de execução das tarefas

Sensor\_display\_task

Pega o valor armazenado da fila, aplica a lógica para ligação do motor e envia as informações do sensor para o display. Grava a informação do estado do motor em outra fila, que será lida pela *dashboard\_task.*

Devido ao atraso inserido e à prioridade igual às outras tarefas, a próxima tarefa é criada e entra em execução, após o bloqueio da tarefa atual.

Sensor\_umidade\_task

Configura o conversor analógico-digital, lê o valor analógico e o transforma em percentual, e envia para a fila *xQueueSensor*. Também, devido ao atraso inserido e à prioridade igual, a próxima tarefa é criada e entra em execução, após o bloqueio da tarefa atual.

Motor\_task

Espera a liberação do semáforo para acionar o motor, o que significa que houve uma interrupção através do botão. Entra em estado de bloqueio devido ao atraso inserido, o que garante a continuidade do fluxo das tarefas.

Dashboard*\_task*

Captura as informações do mailbox da medida do sensor e do estado do motor e publica no tópico de umidade da plataforma MQTT. As informações são visualizadas em formato de dashboard e ficam armazenadas na plataforma, podendo ser baixadas em formato de arquivo. A comunicação efetiva ocorre somente na segunda etapa.

Na **segunda etapa**, foi inserida a conectividade ao projeto. Foi utilizada uma biblioteca do *Wi-Fi Manager* incorporada ao código, seguindo as instruções descritas na biblioteca.

Também foi inserida a biblioteca *“mqtt\_client”*, para o envio de mensagens através do protocolo mqtt.

Em ambos os casos, foram utilizadas variáveis de *“event group*” para sinalizar alguns eventos importantes, como wi-fi conectado ou mqtt.

A primeira informação necessária na função main é a inicialização da memória não volátil, necessária para o armazenamento das informações de credenciais e rede wi-fi escolhida.

A rede wi-fi é iniciada através da inicialização do wi-fi manager, o que possibilita a conexão à rede.

A inicialização do wi-fi manager implica em utilização de *queue*, *semáforo mutex*, entre outros, e na criação da tarefa “*wifi\_manager*”, que no caso está configurada com prioridade 5, não oferecendo risco às demais tarefas, até porque a biblioteca já está implementada utilizando-se o Free-RTOS.

Depois da conexão wi-fi, a rede mqtt é iniciada e configurada.

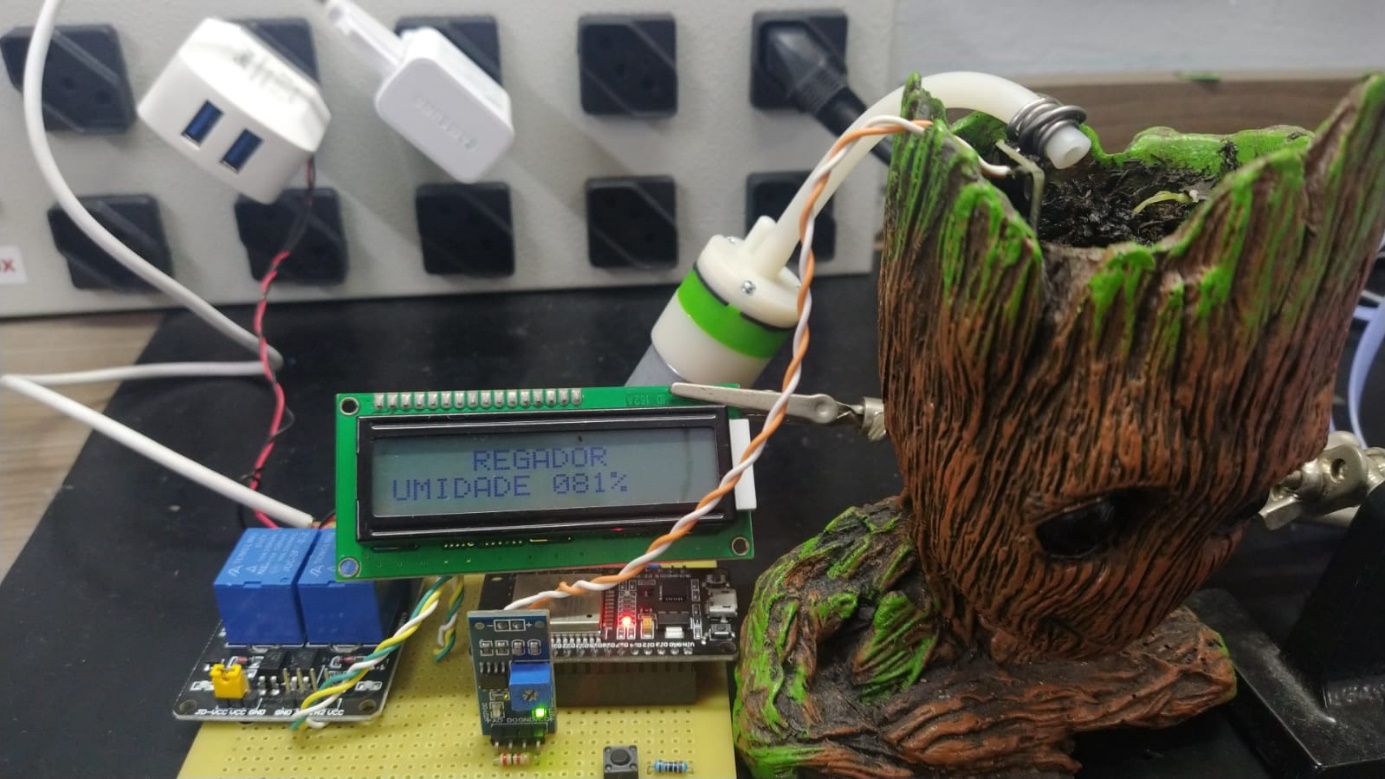
Foi utilizado o broker da *Thingspeak*, que possibilita a implementação de dashboards de acompanhamento. Em uma situação de implementação em ambiente de produção, se utilizaria um broker que faria somente papel de broker, e uma aplicação para se conectar com o broker e fazer os dashboards necessários. Nessa implementação não foram utilizadas subscrições, somente publicações no broker, devido a limitações da plataforma e conta.

O projeto envia para o broker mqtt as informações de umidade e estado do motor, e os dashboards apresentam estas informações. Há a possibilidade de armazenar as informações para estudo ou verificar o comportamento do regador ao longo do tempo.

A biblioteca *“mqtt\_client”* também cria uma tarefa*, “mqtt\_task”*, cuja prioridade padrão também é igual a 5. Isso possibilita que todas as tarefas tenham chances de serem chamadas.

# RESULTADOS E DISCUSSÃO

A imagem a seguir mostra o modelo implementado do regador inteligente. Foi utilizado relé para a ligação da bomba e o sensor conectado ao protoboard. A informação de umidade aparece no display e é transmitida para a nuvem através de uma plataforma MQTT.



**Figura6**: Modelo implementado do Regador Inteligente**.**

Na inicialização do dispositivo, é observada a troca de mensagens para a conexão wi-fi, tanto com as informações vindas do Wi-fi Manager, como com as mensagens de log inseridas no código principal.

Pode ser observada a conexão wi-fi, seguida do início dos eventos de conexão via MQTT, que dão espaço para as outras tarefas, que são executadas até que o MQTT se conecte ao broker e comece a enviar as informações.

A temporização do envio de informações à plataforma MQTT é tratada separadamente da temporização da medição do sensor de umidade, possibilitando a configuração aproximada do intervalo de envio de dados para a plataforma. Com isso, o sensor faz várias medições, mas os dados são enviados somente a cada 3 ou 4 medições.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente **Figura 7. Logs de debug do Regador Inteligente.**

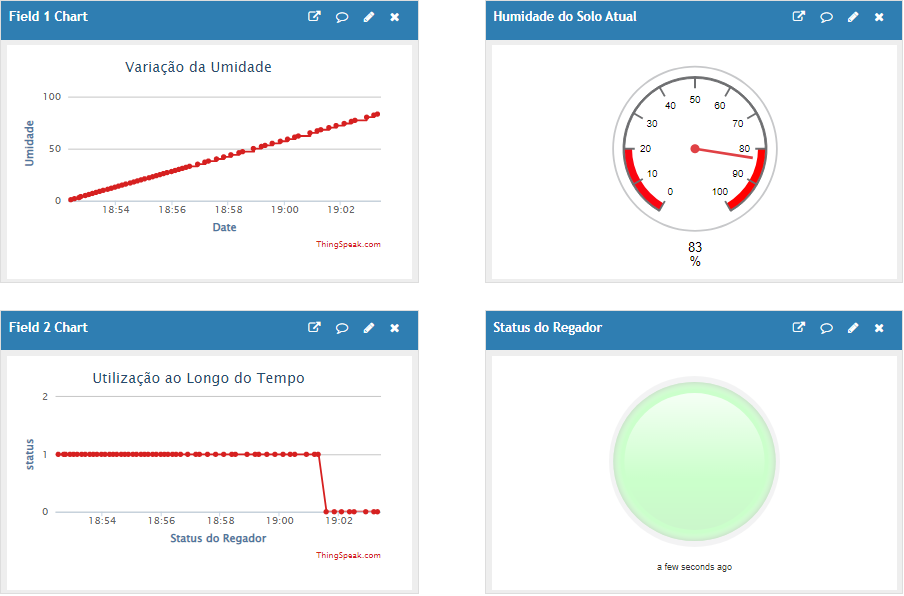
Foram criados dashboards para representar graficamente as medições, de forma que o usuário consiga ter uma informação histórica ou momentânea da variação da umidade do solo, ao mesmo tempo se tem informações sobre a utilização do regador ao longo do tempo e se está ligado ao desligado no momento da análise.



**Figura 8**: Dashboards criados na plataforma ThingSpeak.

Idealmente, para jardins, a umidade do solo se dá em torno de 70%. Foi com essa base em que foi projetado o regador, lembrando-se de que para cada local e tipo de plantação, tem-se um percentual ideal de umidade de solo diferente. Nas imagens, pode-se perceber o desligamento do regador inteligente ao atingir a umidade desejada. No caso, para efeitos de simulação, o valor da umidade continua subindo, mas num ambiente externo, esse valor se manteria em torno dos 70%, pois o regador desligaria após esse percentual. As imagens foram feitas para mostrar o desligamento após atingir a umidade desejada. Se, hipoteticamente, ocorresse uma chuva após esse período, o regador permaneceria desligado.

Existe a possibilidade da incorporação futura de uma lógica para previsão de chuva, uma vez que alguns brokers oferecem o serviço de previsão de chuva e localização.



**Figura 9**: Dashboards na plataforma ThingSpeak. Desligamento em 70 por cento.

# CONCLUSÃO

Com a implementação deste projeto, é seguro dizer que essa prática pode ter resultados muito significativos, tanto na economia de água, gastando somente aquilo que é necessário para o consumo da vegetação, como também no planejamento da irrigação em pequena e larga escala.

A utilização de um sistema operacional em tempo real dá uma maior credibilidade ao projeto, que não trabalha de forma sequencial, e sim por eventos, possibilitando a integração de uma gama maior de atividades em paralelo.

Os detalhes são muito importantes na utilização do Free-RTOS e as bibliotecas devem ser escolhidas a dedo e analisadas com muito cuidado, para se evitar uma exceção do código em algum momento e uma indesejada inicialização do sistema.

A escolha de um broker que atenda às necessidades do projeto também é muito importante, porque pode ser o diferencial que viabilizará a utilização do produto em larga escala. Importante também a utilização de um banco de dados para salvar as informações de medição, com o objetivo de se ter análises mais detalhadas, e ao mesmo tempo globais, das medições.

O aprendizado de novas tecnologias, aliado à prática e busca de solução para problemas antigos, sob um novo ponto de vista, é, sem dúvida, o que consolida o conhecimento de forma ativa e inovadora.

# REFERÊNCIAS

**Site Free-RTOS.org**. Disponível em: < <https://www.freertos.org/> >. Acesso em 18.04.2022.

**Amazon Web Services**. The Free-RTOS Reference Manual - API functions and configuration options. Amazon, 2017.

**Site CloudMQTT**. Disponível em: < <https://www.cloudmqtt.com/docs/index.html> >. Acesso em 18.04.2022.

**Site Embarcados**. Disponível em: < <https://www.embarcados.com.br/mqtt-protocolos-para-iot/> >. Acesso em 18.04.2022.

**RTOS Task States**, Site Free-RTOS.org. Disponível em: < <https://www.freertos.org/RTOS-task-states.html> >. Acesso em 17.04.2022.

**Biblioteca Wi-Fi Manager**. Disponível em: < <https://github.com/tonyp7/esp32-wifi-manager> >. Acesso em 18.04.2022.