ManRenewables

Trabalho Prático

Engenharia de Sistemas e Serviços

Engenharia Informática 2020/21

Grupo 1

Fábio Pereira Henriques – 2181359

Matias Ariel Ortiz Luna – 2182082

Gonçalo Miguel Conceição Vicente - 2172131

Leiria, junho de 2021

Índice

Lista de Figuras

1. Introdução

Nos dias de hoje, e tendo em conta as profundas alterações climáticas que tem vindo a existir ao longo dos anos, projetos relacionados com a preservação do meio ambiente são cada vez mais necessários para a sobrevivência do planeta Terra. Como grande auxílio a este tipo de projetos, existe a Internet das Coisas (*IoT – Internet of Things*), que tem vindo a ter uma enorme expansão e a tornar-se cada vez mais útil em diversas áreas.

De modo a ajudar na preservação do meio ambiente, utilizou-se as *IoT* de forma que seja possível controlar a produção e o consumo de uma instalação de energia solar. Com o uso das *IoT* é possível desenvolver este projeto a um baixo custo, permitindo assim que seja acessível a uma maior quantidade de pessoas.

Este relatório encontra-se dividido em 6 capítulos que apresentam todo o processo de planeamento, construção e desenvolvimento do projeto.

No primeiro capítulo é abordado, o objetivo do trabalho no contexto na qual a aplicação se insere, as *IoT* (*Internt of Things*). No segundo capítulo é descrito o cenário de implementação, dividindo nas ligações físicas e nas comunicações. No terceiro capítulo apresentam-se os esquemas que representam o funcionamento da aplicação. No quarto capítulo aborda-se todo o cenário de testes, onde é exposto o cenário de testes do protótipo e o cenário de testes da aplicação final. No quinto capítulo é apresentado uma análise critica ao projeto bem como uma proposta de melhoria. Por fim, no sexto capítulo, apresenta-se a conclusão.

O presente relatório insere-se no âmbito da disciplina de Engenharia de Sistemas e Serviços pertence à Licenciatura em Engenharia Informática, no ano letivo de 2020/2021.

2. Objetivos do trabalho

Com este trabalho pretendemos desenvolver uma solução que proporcione a qualquer tipo de utilizador, principalmente utilizadores particulares com orçamento baixo, uma forma de controlar a produção e consumo de uma instalação de energia solar. Com isto, é pretendido ajudar o utilizador a decidir se a quantidade de energia produzida é suficiente ou em demasia para o respetivo consumo, permitindo assim aumentar ou reduzir a produção de uma forma controlada.

A ideia inicial era utilizar isto para alimentar e controlar o consumo de diversos grupos de sensores numa solução já existente. Ao decorrer do desenvolvimento, chegamos à conclusão de que este projeto pode ser aplicado a qualquer instalação de energia solar, uma vez que apenas controla a produção e consumo através de sensores facilmente introduzidos num sistema de produção já existente.

Para o controlo, desejamos desenvolver uma aplicação web que permite ao utilizador consultar tanto a produção como o consumo atual, bem como analisar gráficos de produção e consumo por dia ou mês. Esta aplicação será disponibilizada de forma *Anytime & Anywhere*, aumentando a acessibilidade e facilidade de utilização da solução.

3. Cenário de implementação

O cenário de implementação contém uma parte física (produção de energia e controlo) e uma virtual (aplicação web na *cloud* fornecido pela AWS). Decidimos dividir o cenário em dois, com vista a facilitar a visualização do mesmo: um com as ligações físicas e outro com as comunicações.

A Figura 27 apresenta o cenário físico e a Figura 28 a respetiva legenda apenas das ligações, uma vez que todos os equipamentos já estão devidamente identificados no cenário. Já a Figura 29 apresenta a o cenário de comunicações entre o cenário físico e o serviço *Anytime & Anywhere*.

3.1. Protótipo

O protótipo construído tem por base a produção de energia através da fonte solar, o armazenamento dessa energia e o consumo da mesma por um grupo de sensores (simulado por uma ventoinha com um LED) e pelo próprio sistema de controlo. Consideramos que os equipamentos a alimentar estão ligados 24h por dia.

3.1.1. Painel solar, bateria e controlador de carga

Uma vez que o painel solar, a bateria e o controlador de carga não são equipamentos baratos (relativamente com os restantes (secções 3.1.2)) e comuns, decidimos utilizar os fornecidos pelo professor António Pereira. Mas, para confirmar que a bateria e o painel solar tinham capacidade suficiente para alimentar os diversos componentes durante um intervalo de tempo aceitável, foi necessário realizar alguns testes (secção 5.1.1). De seguida iremos apresentar as características da bateria e do painel e concluir se ambos se adequam ao protótipo.

A bateria é da marca Ultracell (Figura 1) e possui as seguintes características mais relevantes:

- Tensão: 12 V
- Capacidade: 7 AH

Figura 1 - Bateria 12V da Ultracell

Tendo o valor de consumo total dos diversos componentes do protótipo (secção 5.1.1) (240 mA / 1000 = 0.24 A). Chegamos à conclusão a vida da bateria é aproximadamente de 29 horas seguidas (7 AH / 0.24 A = 29.7 H), o que proporciona um bom fornecimento de energia mesmo quando não existe produção solar por mais de 24 horas.

Uma dúvida que surgiu foi se o painel solar tinha capacidade de produção suficiente para alimentar os componentes e carregar a bateria. Como não descobrimos as características do painel (não sabemos nem o fabricante nem o modelo), tivemos de realizar alguns testes para resolver esta questão (secção 5.1.2).

Após estes testes, conseguimos deduzir as seguintes características do painel:

- Tensão de operação: 20V
- Corrente máxima de produção: 0.41 A
- Potência: 20V * 0.41 A ≈ 8 W

Isto permite aferir que num dia de sol e com a inclinação certa do painel, é possível carregar a bateria e ainda alimentar os equipamentos sem problema. A Figura 2 apresenta uma fotografia do painel utilizado.

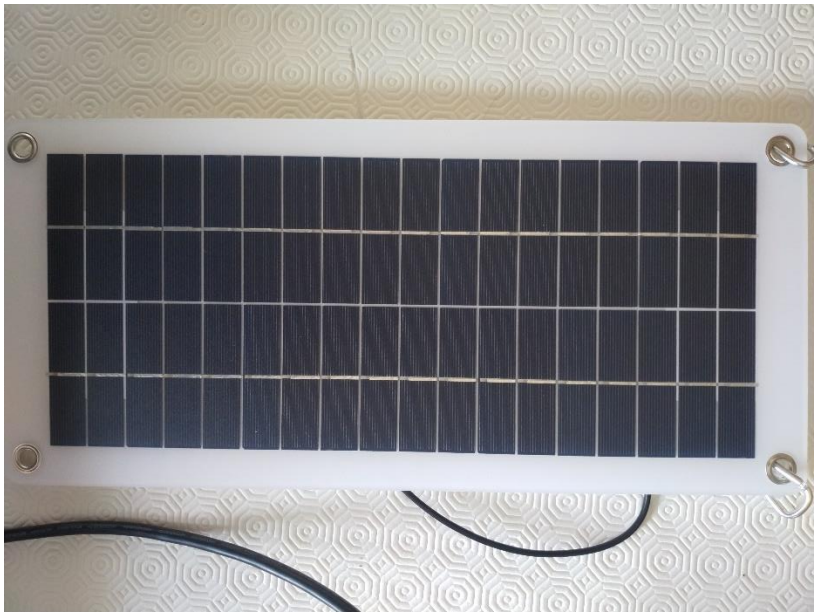


Figura 2 - Painel Solar utilizado

Para o controlador de carga não foi necessário realizar nenhum tipo de testes, uma vez que já estava configurado para a bateria e o painel em questão. O controlador utilizado é genérico. Permite carregar e alimentar outros equipamentos ao mesmo tempo, o que nos facilita a implementação. Ele é que faz a gestão de para onde direcionar a corrente consoante a necessidade. A Figura 3 apresenta o controlador utilizado.

Figura 3 - Controlador de carga

Ele possui 3 elementos de ligação (cada um com 2 cabos: positivo e negativo): ligação do painel (mais à esquerda), ligação da bateria (centro) e ligação a outros equipamentos (direita). A saída para outros equipamentos tem uma tensão de 12v, o que nos obrigou a utilizar um regulador de tensão para alimentar os diversos sensores e microcontrolador.

Também possui duas portas USB que podiam ser utilizadas para os alimentar (pois têm uma tensão de 5v), mas a corrente máxima das mesmas não é muito elevada e poderia limitar o funcionamento.

3.1.2. Sensores

Como o nosso objetivo é calcular a potência produzida e a potência consumida, decidimos logo à partida que ia ser necessário 2 sensores de corrente e 2 sensores de tensão.

Para os sensores de corrente, primeiro resolvemos utilizar o modelo ACS712 de 5A [1] para as duas medições necessárias. Logo no primeiro teste, descobrimos que este sensor não funciona muito bem quando as correntes são baixas. Mas mesmo assim decidimos utilizá-lo para medir a corrente de consumo, uma vez que no início pensávamos que a do painel era inferior (o que era falso, como descrito na secção 3.1.1). Apesar desta descoberta, mantivemos este sensor a medir o mesmo, o que correu bastante bem devido a melhorias no código.

Para medir a corrente do painel, utilizamos o sensor INA219 [2], que oferece uma medição mais precisa, bem como uma facilidade enorme no código (apenas com uma linha se consegue o pretendido). A Figura 30 mostra o esquema de ligações destes dois sensores. Já a Figura 4 é uma fotografia dos sensores no protótipo real.

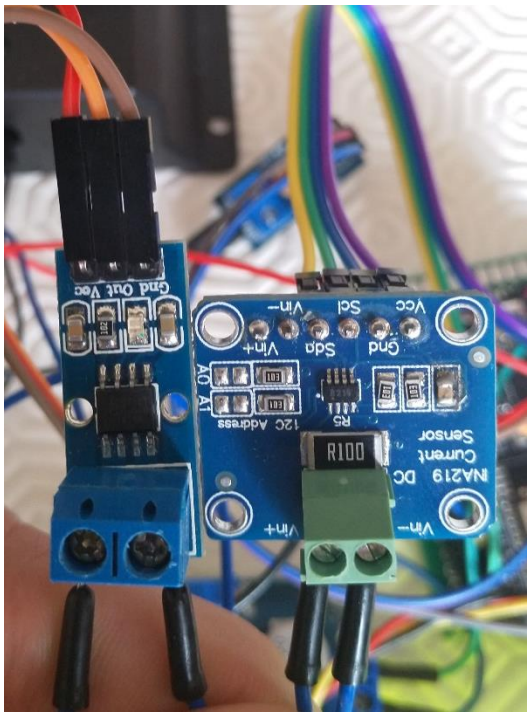


Figura 4 - Sensores de corrente no protótipo real (esquerda → ACS712 | direita → INA219)

Para medir a tensão, inicialmente propusemos utilizar 2 sensores de tensão (que basicamente são divisores de tensão de 25v para 5v). Mas após alguns testes, foi de notar que a precisão deles não é muito boa e então decidimos criar o nosso próprio sensor. De qualquer maneira utilizamos um desses sensores para medir a tensão da bateria, a fim de facilitar a montagem e mostrar que conseguimos melhorar as leituras através do código.

Apesar de o sensor devolver valores de 0v a 5v e as portas do microcontrolador utilizado apenas suportarem um máximo de 3.3v, não existe perigo em utilizá-lo, uma vez que a tensão da bateria nunca atinge um valor que faça o sensor devolver mais de 3.3v (secção 5.1.3). A Figura 5 mostra o sensor no protótipo real.

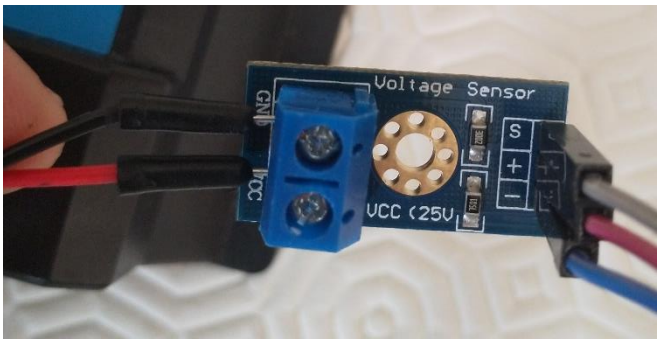


Figura 5 - Sensor de tensão 25v para 5v no protótipo real

Para criarmos o nosso sensor, desenvolvemos um divisor de tensão de 20v para 3.3v, que é a tensão aceitável nas portas do microcontrolador utilizado. Como não tínhamos resistências de muitos valores, utilizamos três: uma de 1k ohm e outra de 330ohm em série (R1) e uma de 220 ohm (R2). Chegamos a estes valores através de uma calculadora online [3] que utiliza a equação V_{out}= (V_{in}* R2) / (R1 + R2) para calcular o valor de uma das resistências (R1 ou R2) sabendo o valor da outra.

Após o calcula das resistências, procedemos a pequenos testes que correram como esperado. A Figura 6 apresenta este divisor de tensão no protótipo real. A Figura 31 representa o esquema do mesmo.

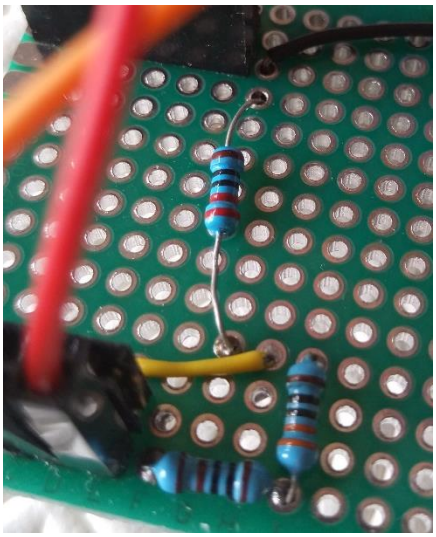


Figura 6 - Divisor de tensão de 20v para 3.3v no protótipo real

Com estes quatro sensores é possível calcular a potência consumida e a potência produzido. Através da analisa dos dados será possível determinar se a produção está de acordo com o consumo.

3.1.3. ESP32

Devido à necessidade de enviar os dados capturados localmente para uma aplicação na cloud, era preciso existir um microcontrolador que tivesse capacidade de ler diversos valores analógicos, bem como comunicar com um servidor na internet. Uma vez que esta solução permite a adição de vários microcontroladores, também é necessário existir um *gateway* que agregue a informação dos vários e a envie para a aplicação na *cloud*.

O ESP32 foi a nossa escolha para o microcontrolador, pois possui 15 portas com capacidade de leitura analógica (portas com ADC - Figura 32), baixo consumo e ainda Wi-Fi, que é a tecnologia utilizada para comunicar com o *gateway* [4].

Neste microcontrolador é realizado:

- Leitura dos sensores;
- Melhoria das leituras;
- Ligação ao *gateway*;
- Envio das leituras para a API no *gateway*.

Para tentar eliminar a variação que as leituras dos sensores possuem, realizamos a média de 2000 leituras. Este pequeno ajusto melhorou significativamente os valores lidos. Só não

efetuamos isto nas leituras do sensor INA219, pois este já tem uma boa precisão e quase variação nenhuma. A Figura 7 apresenta o excerto de código com esta melhoria.

```
for(int i=0; i<2000; i++){
  adcConsumptionCurrent += analogRead(CSC_PIN);
  adcBatteryVoltage += analogRead(VSB_PIN);
  adcSolarPanelVoltage += analogRead(VSP_PIN);
  delay(2);
}
```

Figura 7 - Excerto de código com as 2000 leituras para a melhoria das mesmas

O pedido POST de envio das leituras para o *gateway* é realizado de 4 em 4 segundos, que é o tempo que as leituras demoram. Neste pedido é enviado as quatro leituras (corrente de produção, tensão do painel, corrente de consumo e tensão da bateria) e ainda o identificador único do microcontrolador. A Figura 8 mostra exatamente o pedido realizado. O *endpoint* de envio (primeiro argumento da função "sendData") nunca é alterado, pois o microcontrolador vai se ligar ao *gateway*, que fornece a sua própria rede (secção 3.1.4).

```
String dataToSend = "{\"mcu_id\": 1, \"battery_voltage\":\"" + String(batteryVoltage) + "\",\"solar_panel_voltage\":\"" +
String(solarPanelVoltage) + "\",\"consumption_current\":\"" + String(consumptionCurrent) + "\",\"producing_current\":\"" +
String(producingCurrent) + "\",\"voltage_consumption_current\":\"" + String(voltageConsumptionCurrent) + "\"}";
sendData("http://192.168.4.1/payload", dataToSend);
```

Figura 8 - Excerto de código com o pedido POST de envio das leituras

Uma vez que o sensor de corrente ACS712 não funciona muito bem nesta utilização, é necessário enviar o valor "voltageConsumtionCurrent" para que a leitura possa ser calibrada o melhor possível. Este valor é referente à tensão que é enviada pelo sensor. Com ela conseguimos saber qual é o valor lido quando a corrente é 0 A e assim realizar corretamente o cálculo da mesma.

3.1.4. RaspberryPi 3B

O dispositivo que escolhemos para fazer de *Gateway* foi o RaspberryPi 3B, uma vez que é de baixo consumo e de baixo custo, permite correr um pequeno servidor web para receber os pedidos dos microcontroladores e depois enviá-los para o servidor com a plataforma Web na *Cloud* para que o utilizador possa visualizar os dados em tempo real.

De acordo com o esquema de implementação do projeto que pode ser visualizado na Figura 29, o RaspberryPi 3B atua como *Gateway* e *AccessPoint* de uma rede privada da qual fazem parte os microcontroladores e demais atuadores e sensores que o utilizador final possa

requerer para o seu ambiente, por tanto, foi configurado um servidor DNS que fornece de endereços IPv4 aos dispositivos de acordo com a Tabela 1

DISPOSITIVO/S	ENDEREÇO/S IPv4
Gateway RaspberryPi 3B	192.168.4.1
Microcontroladores/Atuadores/Sensores	192.168.4.2 – 192.168.4.25

Tabela 1 - Endereços IPv4 dos dispositivos da rede

Foi instalado o serviço *Hostap*d, cria uma rede privada por meio da interface de rede wlan0 e da placa de rede LAN que tem o dispositivo, o nome da rede é "*SolarNetwork*" e a palavra-passe de acesso é "*12345678*".

```
pi@raspberrypi:~ $ cat /etc/dnsmasq.conf
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.25,255.255.255.0,24h
domain=wlan
address=/gw.wlan/192.168.4.1
pi@raspberrypi:~ $ sudo service dnsmasq status
● dnsmasq.service - dnsmasq - A lightweight DHCP and ca
   Loaded: loaded (/lib/systemd/system/dnsmasq.service;
   Active: active (running) since Mon 2021-05-31 15:17:
```

Figura 9 - Configuração do serviço "Dnsmasq" e estado de funcionamento

```
pi@raspberrypi:~ $ cat /etc/hostapd/hostapd.conf
country_code=PT
interface=wlan0
ssid=SolarNetwork
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
pi@raspberrypi:~ $ sudo service hostapd status
● hostapd.service - Advanced IEEE 802.11 AP and IEEE 802.1X/WPA/WPA2/EAP
   Loaded: loaded (/lib/systemd/system/hostapd.service; enabled; vendor
   Active: active (running) since Mon 2021-05-31 15:17:18 WEST; 1 day 19
```

Figura 10 - Configuração do serviço "Hostapd" e estado de funcionamento

Um cabo Ethernet deve estar conectado ao RaspberryPi 3B para que tenha acesso à internet o próprio dispositivo e os que fazem parte da rede, pelo que também se adicionou uma regra à firewall *Iptables*.

```
pi@raspberrypi:~ $ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 11 - Regras de funcionamento da firewall Iptables

Foi também ativado o serviço SSH para que possa ser acedido de forma remota pelos técnicos e administradores de sistemas caso exista a necessidade de realizar alguma atualização ou manutenção do dispositivo.

```
matiasluna@MacBook-Pro-de-Matias ~ % ssh pi@192.168.4.1
pi@192.168.4.1's password:
Linux raspberrypi 5.10.17-v7+ #1403 SMP Mon Feb 22 11:29:51 GMT 2021 armv7l
```

Figura 12 - Acesso remoto ao RaspberryPi 3B desde um computador pessoal

Foi configurado um servidor *Ngnix* que funciona como *Reverse Proxy* reencaminhando os pedidos para um servidor web interno em *Node.Js*, cumpre a única função de receber os dados dos microcontroladores, criar um objeto com esses dados e inserir a data atual em formato “yyyy-mm-dd h:MM:ss” e enviar os dados para um servidor na Cloud com a plataforma Web.

```
location / {

        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

}
```

Figura 13 - Configuração do servidor Nginx como Reverse Proxy

```
pi@raspberrypi:~/manrenewables $ node index.js
Project of ESS: ManRenewables Gateway Web Server listening on port 8080!
```

Figura 14 - Confirmação de funcionamento do servidor interno em Node.JS

```
app.post('/payload', (req, res) => {
    let body = req.body
    body["full_date"] = dateFormat(new Date(), "yyyy-mm-dd h:MM:ss");
    body.date = {};
    body.date["year"] = dateFormat(new Date(),"yyyy");
    body.date["day"] = dateFormat(new Date(),"dd");
    body.date["month"] = dateFormat(new Date(),"mm");
    body.date["hour"] = dateFormat(new Date(),"hh");
    body.date["minute"] = dateFormat(new Date(),"MM");
    body.date["second"] = dateFormat(new Date(),"ss");
    const postData = JSON.stringify({
        body: body
    })
    //VM na Cloud
    const options = {
        host: 'http://ec2-3-142-69-27.us-east-2.compute.amazonaws.com/#/',
        port: 80,
        path: '/payload',
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
            'Content-Length': Buffer.byteLength(postData)
        }
    };
```

Figura 15 - Excerto de código do tratamento dos dados recebidos e do pedido POST à Plataforma Web na Cloud

O servidor *Ngnix* fica ativo uma vez que o RaspberryPi 3B inicia, para que o servidor interno em *Node.js* também o faça foi adicionado um comando na *crontab* do sistema.

```
@reboot node /home/pi/manrenewables/index.js
```

Figura 16 - Comando de início de funcionamento do servidor interno em Node.Js na crontab do sistema

3.1.5. Cenário real final

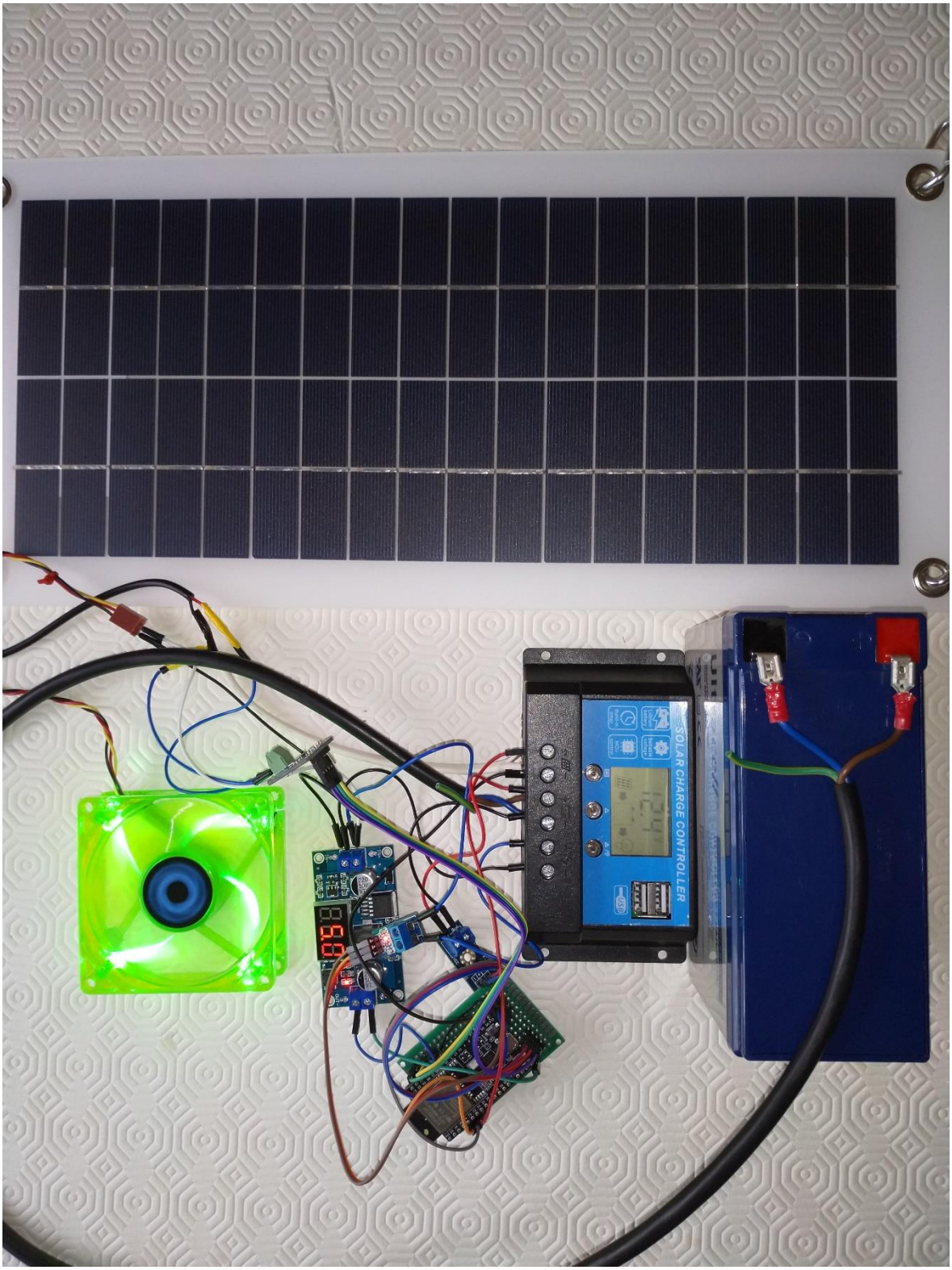


Figura 17 - Fotografia do cenário real final

3.1.6. Plataforma Web (Funcionalidades AnyTime e AnyWhere)

Em todos os projetos é necessário apresentar os dados obtidos ao utilizador final, e tendo em conta esta necessidade foi elaborado um website para seja possível monitorizar em tempo real os dados recolhidos e calculados.

Nesse sentido, resolveu-se elaborar uma plataforma web onde os dados obtidos pelos sensores sejam apresentados sendo possível verificar os consumos num período de tempo passado ou em tempo real. Para tal, divide-se o website em duas partes: um *backend* e um *frontend*.

Para a realização do *backend* recorreu-se à framework *JavaScript Node.Js*, onde esta é responsável por receber os dados que são enviados através do *gateway* (RaspberryPi) e persisti-los numa base de dados não relacional (*MongoDB*), bem como enviar para o *frontend* os dados que se encontram armazenados na mesma.

```
//------------------ LER ULTIMO REGISTO DA BD --------------------------------
//Function that reads the most recent entry in database
function readLast() {…
}
//Endpoint to read the most recent entry in database
app.get('/payload', (req, res) => {…
});

//------------------ LER OS MESES DOS REGISTOS DA BD -------------------------
//Function that reads all the months existing in the data from the database
function readMonths(ano) {…
}
//Endpoint to read all the months existing in the data from the database
app.get('/months/:year', (req, res) => {…
});

//------------------ LER OS DIAS DOS REGISTOS DA BD --------------------------
//Function that reads all the months existing in the data from the database
function readDays(mes, ano) {…
}
//Endpoint to read all the months existing in the data from the database
app.get('/days/:month/:year', (req, res) => {…
});

//------------------ LER OS ANOS DOS REGISTOS DA BD --------------------------
//Function that reads all the years existing in the data from the database
function readYears() {…
}
//Function to read all the months existing in the data from the database
app.get('/years', (req, res) => {…
});
```

Figura 18 - Excerto de código de funções e endpoints existentes no Backend em Node.js da Plataforma Web

```
//------------------- LER ULTIMO REGISTO DA BD  ----------------------------------
//Function that reads the most recent entry in database
function readLast() {
  client.connect();
  client.db("renewable_db").command({ ping: 1 });
  client.db("renewable_db").collection("renewable_db_collection").find({}).toArray(function (err, res) {
    if (err) {
      return err;
    } else {
      response = res[res.length - 1]
      return res;
    }
  })
  client.db.close;
}
```

Figura 19 - Excerto de código da função do Backend da Plataforma Web que recolhe o último registo inserido na base de dados

```
function readDataByDay(dia, mes, ano) {
  client.connect();
  client.db("renewable_db").command({ ping: 1 });
  client.db("renewable_db").collection("renewable_db_collection").find({
    $and: [
      { 'date.year': { $in: [ano] } },
      { 'date.month': { $in: [mes] } },
      { 'date.day': { $in: [dia] } }]
  }).toArray(function (err, res) {
    if (err) {
      return err;
    } else {
      var media_arr = []
      var media = {}
      for (var i = 0; i < 24; i++) {
        media[i] = {
          consumed: 0.0,
          produced: 0.0,
          items: 0.0
        }
        res.forEach(object => {
          if (object.date.hour == i) {
            media[i].consumed += parseFloat(object.consumption_current)
            media[i].produced += parseFloat(object.producing_current)
            media[i].items++;
          }
        })
      }
      for (var i = 0; i < 24; i++) {
        if (media[i].consumed != 0.0 && media[i].produced != 0.0) {
          media[i].consumed = media[i].consumed / media[i].items
          media[i].produced = media[i].produced / media[i].items
        }
        media_arr.push(media[i])
      }
    }
    response = media_arr
    return media_arr;
  })
  client.db.close;
}
```

Figura 20 - Excerto de código da função no Backend da Plataforma Web que recolhe os dados da base de dados de acordo com a data inserida pelo utilizador e calcula a media do produzido e consumido em 24 horas

O *frontend* foi desenvolvido com auxílio da Framework *JavaScript Vue.Js*, comunicando através de pedidos *RESTFull* com o *backend*. Tem como função apresentar ao utilizador dados importantes como a tensão da bateria e do painel solar, a corrente consumida e a corrente produzida.

Figura 21 - Dados apresentados ao utilizador

Existe também, uma parte de estatísticas, onde é possível visualizar a potência consumida e produzida consoante um determinado dia, basta preencher o dia, o mês e o ano, sendo os dados apresentados ao utilizador num gráfico onde estão representadas as 24 horas reais; existe a possibilidade de visualizar apenas as estatísticas por mês, ou seja, num determinado mês escolhido pelo utilizador é apresentada uma média diária da energia consumida e produzida; por fim, existe a possibilidade de visualizar apenas num determinado ano, assim, é apresentado ao utilizador a média da energia produzida e consumida em cada mês.

```
getData() {
    this.loading = true;
    this.dados.produced = [];
    this.dados.consumed = [];
    let url;
    if (this.data.dia) {
        url =
            "http://"+window.location.host.split(":")[0]+":8080/values/hours/" +
            this.data.dia +
            "/" +
            this.data.mes +
            "/" +
            this.data.ano;
    } else if (this.data.mes) {
        url =
            "http://"+window.location.host.split(":")[0]+":8080/values/days/" +
            this.data.mes +
            "/" +
            this.data.ano;
    } else {
        url = "http://"+window.location.host.split(":")[0]+":8080/values/months/" + this.data.ano;
    }
```

Figura 22 - Excerto de código da função no Frontend da Plataforma Web que faz pedidos ao backend de acordo com os valores introduzidos pelo utilizador

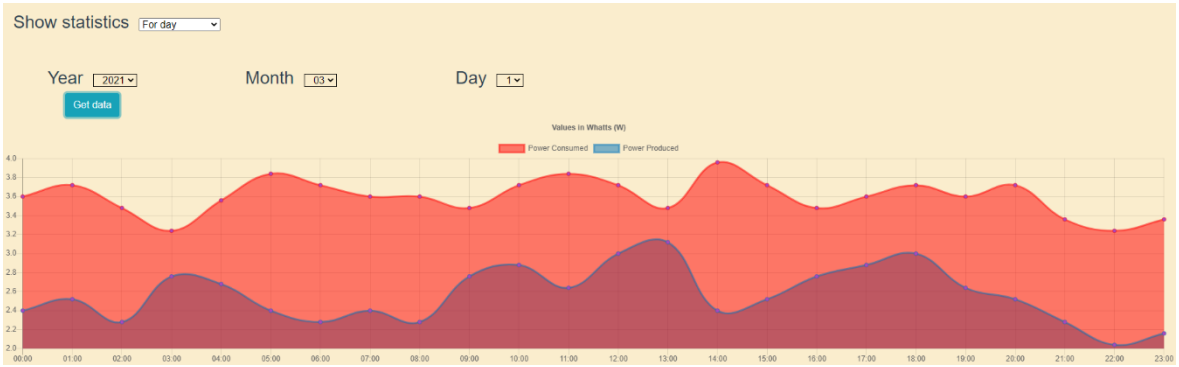


Figura 23 - Gráfico de dados ao logo do dia 1/03/2021

De forma que fosse possível visualizar o histórico dos dados foi decidido implementar uma base de dados ao projeto. Após uma pesquisa, decidiu-se que a melhor opção seria implementar uma base de dados não relacional, uma vez que esta apresenta maior rapidez tendo em conta uma grande quantidade de dados que poderão vir a ser inseridos. Assim, optamos por utilizar o *software MongoDB*.

```
> db.renewable_db_collection.stats()
{
        "ns" : "renewable_db.renewable_db_collection",
        "size" : 2379,
        "count" : 13,
        "avgObjSize" : 183,
        "storageSize" : 36864,
        "freeStorageSize" : 16384,
        "capped" : false,
        "wiredTiger" : {
                "metadata" : {
                        "formatVersion" : 1
                },
```

Figura 24 - Metadados da base de dados MongoDB

Uns dos pressupostos deste projeto era garantir serviços *AnyTime* e *AnyWhere* e para cumprir estes objetivos, recorreu-se à utilização de um servidor na *cloud*, utilizando os serviços disponibilizados pela *Amazon Web Services*, para disponibilizar todo o conjunto base dados, *backend* e *frontend*, existindo assim a possibilidade de o utilizador da aplicação poder visualizar os consumos a qualquer momento e em qualquer lugar. O link de acesso à Plataforma Web é: http://ec2-3-142-69-27.us-east-2.compute.amazonaws.com/#/

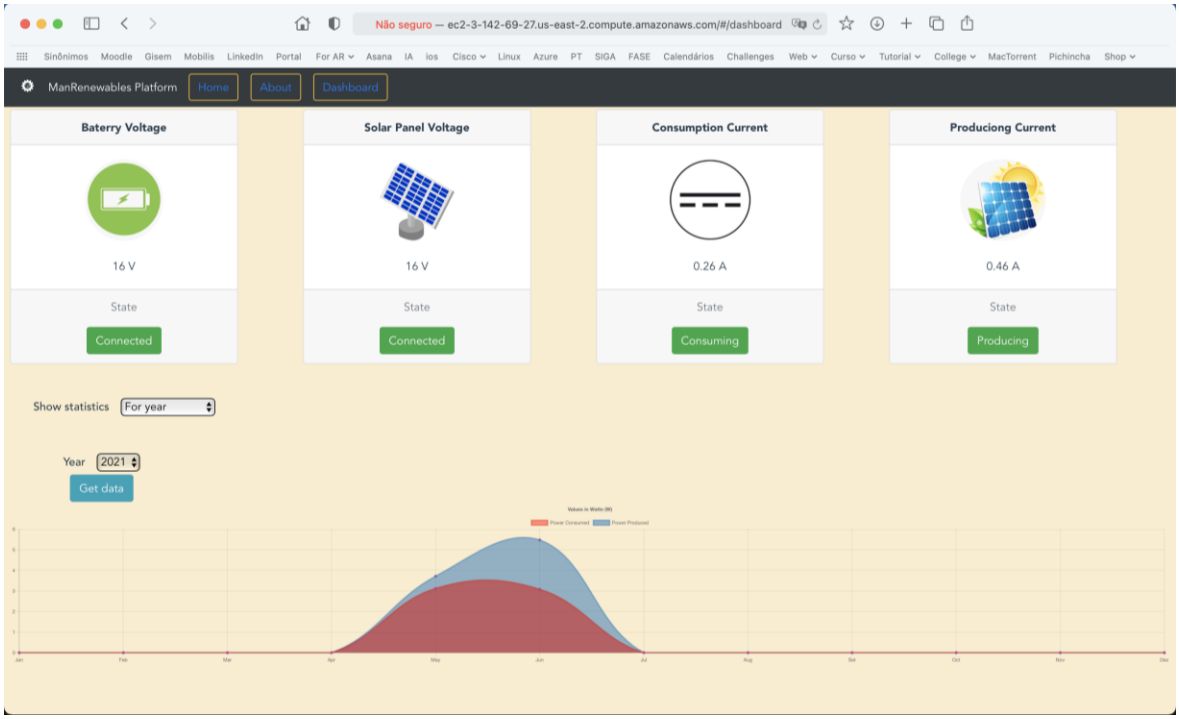


Figura 25 - Dashboard da Plataforma Web

Com vista a garantir a confidencialidade da informação, é realizada autenticação através de uma *password* específica. Quando a aplicação é aberta é pedido ao utilizador para introduzir a *password*. Após isso é realizado um pedido ao *backend* que vai verificar se a *password* está correta. Se estiver, apresenta a *dashboard*. Se não, informa o utilizador do mesmo.

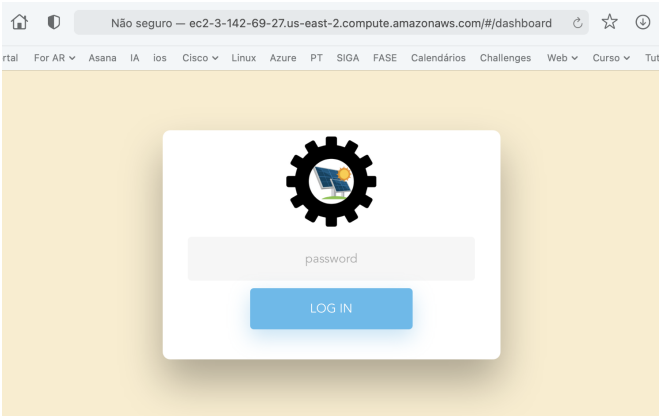


Figura 26 - Página de log in da Plataforma Web

4. Esquemas

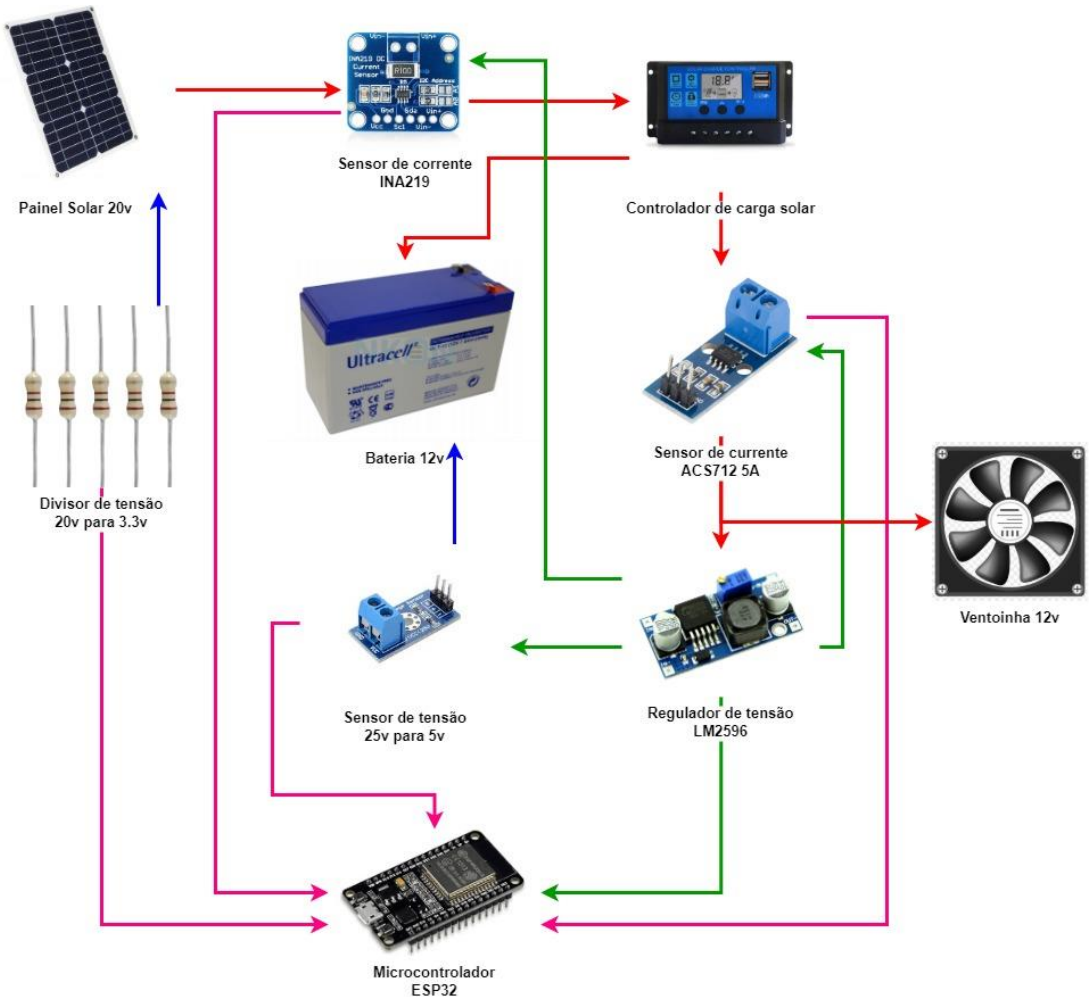


Figura 27 - Cenário de implementação físico

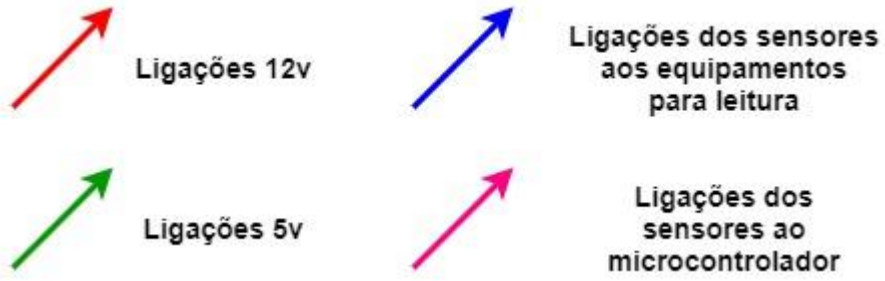


Figura 28 - Legenda do cenário físico

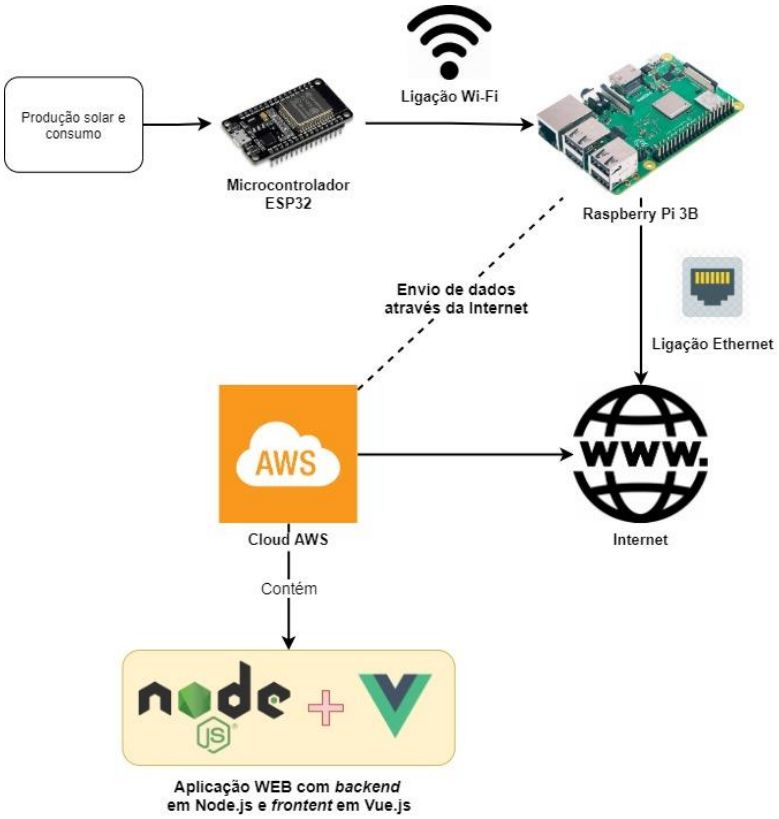


Figura 29 - Cenário das comunicações e do seviço *Anytime & Anywhere*

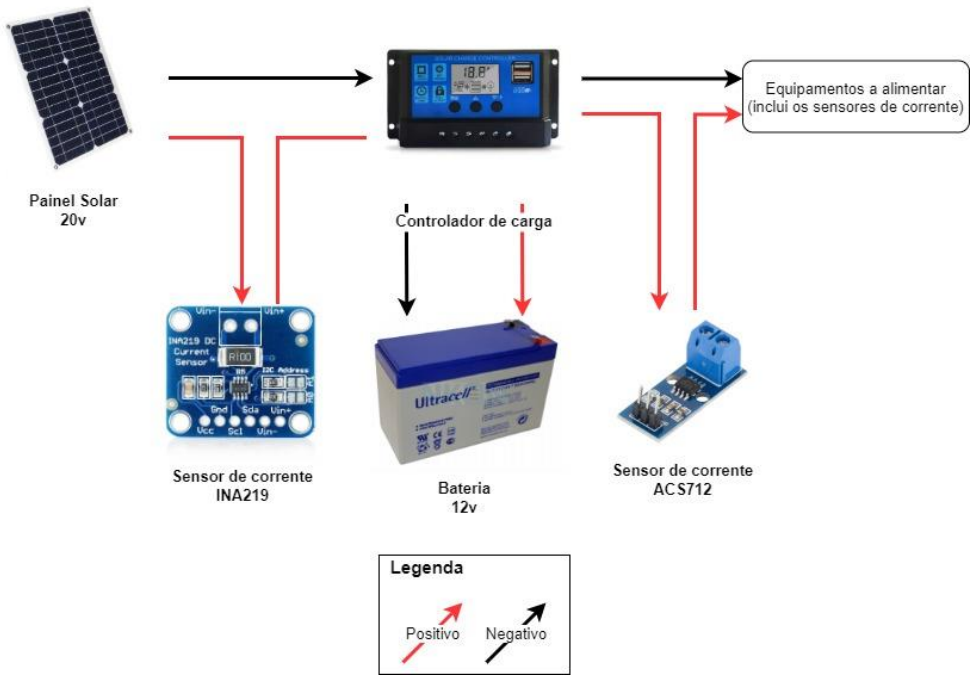


Figura 30 - Ligações dos sensores de corrente

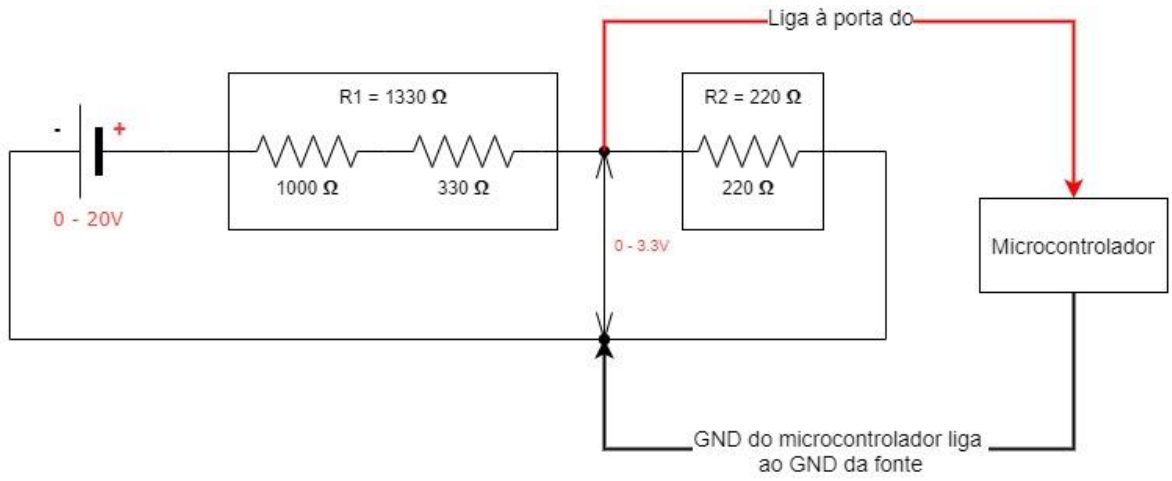


Figura 31 - Esquema do divisor de tensão

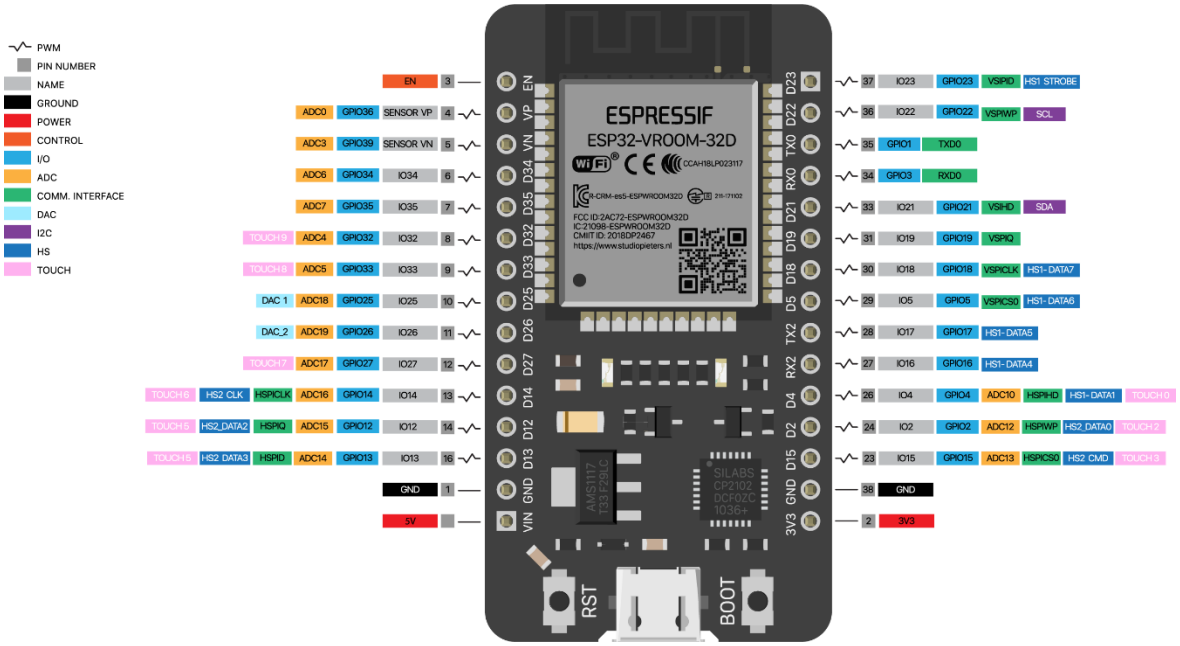


Figura 32 - *Pinout* do ESP32

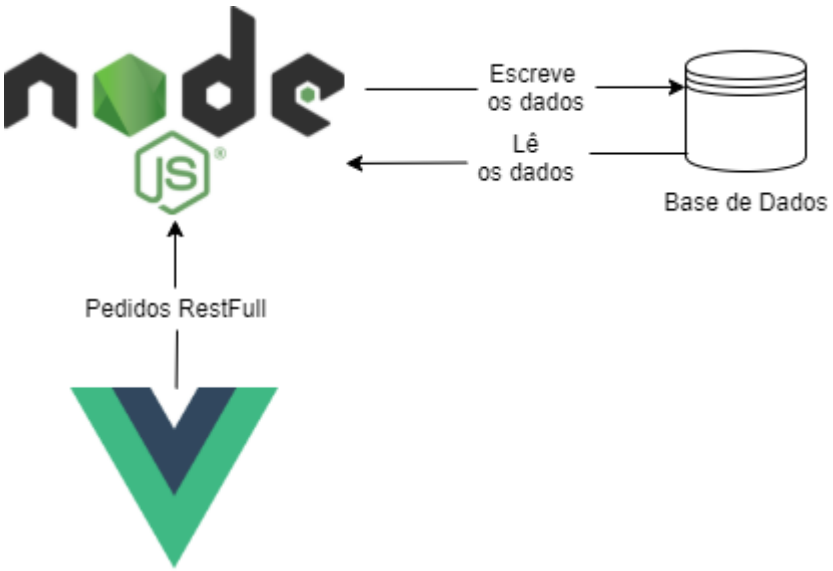


Figura 33 - Comunicações entre o *Backend*/*Frontend*/Base de dados

5. Cenários de Testes

5.1. Testes para a criação do protótipo

5.1.1. Medição da corrente de consumo

Para poder concluir se a bateria e o painel se adequavam ao nosso protótipo, medimos o consumo de todos os equipamentos a funcionar ao mesmo tempo e chegamos à medição de aproximadamente 240 mA (Figura 34), o que é um valor baixo.

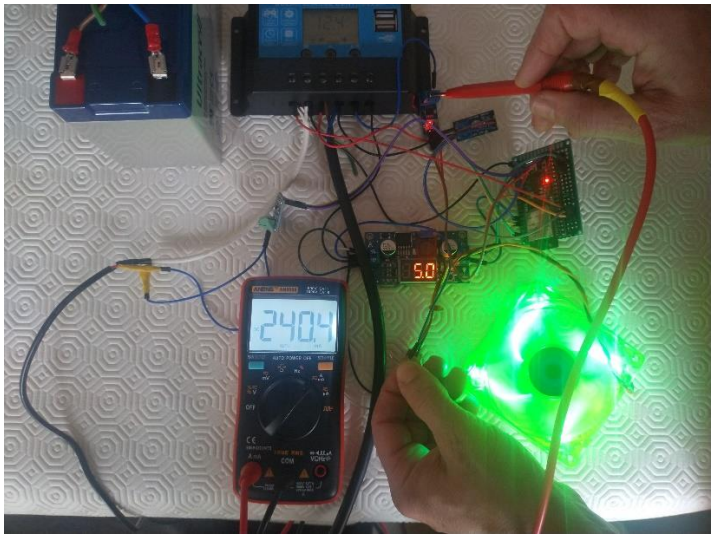


Figura 34 - Medição da corrente de consumo

5.1.2. Medição da corrente de produção

Para começar, medimos a tensão do painel em diversas condições climatéricas e chegamos à conclusão de que mesmo com algumas nuvens, a tensão do painel ainda é aceitável para carregar a bateria, mas não o suficiente para alimentar os componentes. Chegamos a esta conclusão após medir a corrente de produção nas mesmas condições.

Inicialmente, aferimos que a corrente de produção era inferior à de consumo (cerca de 0.15 A), o que foi desmentido após testes mais rigorosos. A corrente máxima de produção foi de aproximadamente 0.41 A (Figura 35), o que é 1.7 vezes maior que a de consumo.

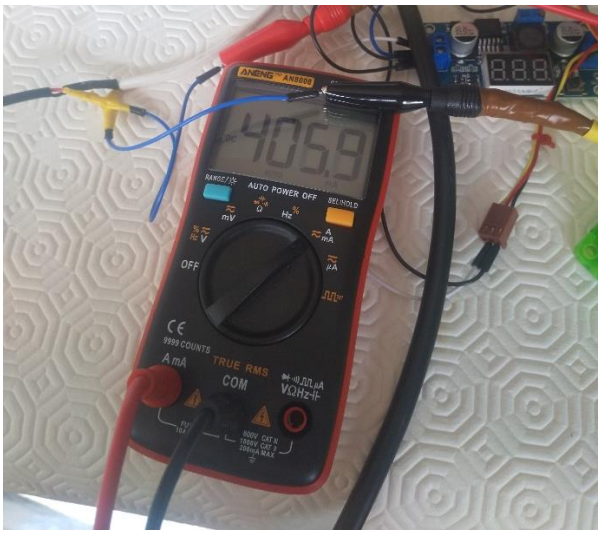


Figura 35 - Medição da corrente de produção

5.1.3. Sensor de tensão

Este teste foi realizado apenas com cálculos com o objetivo de mostrar que não existe perigo em utilizar o divisor de tensão 25v para 5v, uma vez que a tensão que o sensor vai medir nunca vai produzir valores superiores aos suportados pelo microcontrolador (3.3v).

O sensor possui 2 resistências: a primeira de 30kΩ e a segunda de 7.5Ω. Sabendo que a tensão máxima da bateria é de 13.8v (referido nas especificações da mesma), sabendo o valor das resistências e a equação V_{out} = (V_{in}*R2) / (R1+R2), podemos calcular: V_{out}= (13.8 * 7500) / (30000 + 7500) = 2.76v. Com este valor, concluímos que a tensão máxima na porta do microcontrolador é inferior à máxima suportada, e por isso é seguro utilizar o sensor.

5.2. Testes da solução final

5.2.1. Objetivos dos Testes

O objetivo principal dos testes é comprovar o correto funcionamento da leitura dos dados recolhidos pelos sensores, os cálculos realizados pelos diferentes dispositivos assim como o envio, receção e tratamento dos dados para que o utilizador os possa visualizar em tempo real para a melhor toma de decisões. Os objetivos específicos são descritos a seguir:

- Demostrar em tempo real o envio, tratamento e visualização dos dados obtidos e calculados dos sensores e atuadores.

- Verificar o correto funcionamento dos sensores, atuadores e microcontroladores da solução.
- Garantir com sucesso a disponibilidade AnyTime e AnyWhere da Plataforma Web.

5.2.2. Resultados

Para comprovar o correto funcionamento dos diferentes componentes do projeto, são apresentados os resultados recolhidos.

- Dados enviados pelo microcontrolador para o Gateway RaspberryPi 3B, dispositivo que depois os irá enviar em conjunto com a data atual para a Plataforma Web na Cloud que disponibilizará os dados para a sua visualização em tempo real.

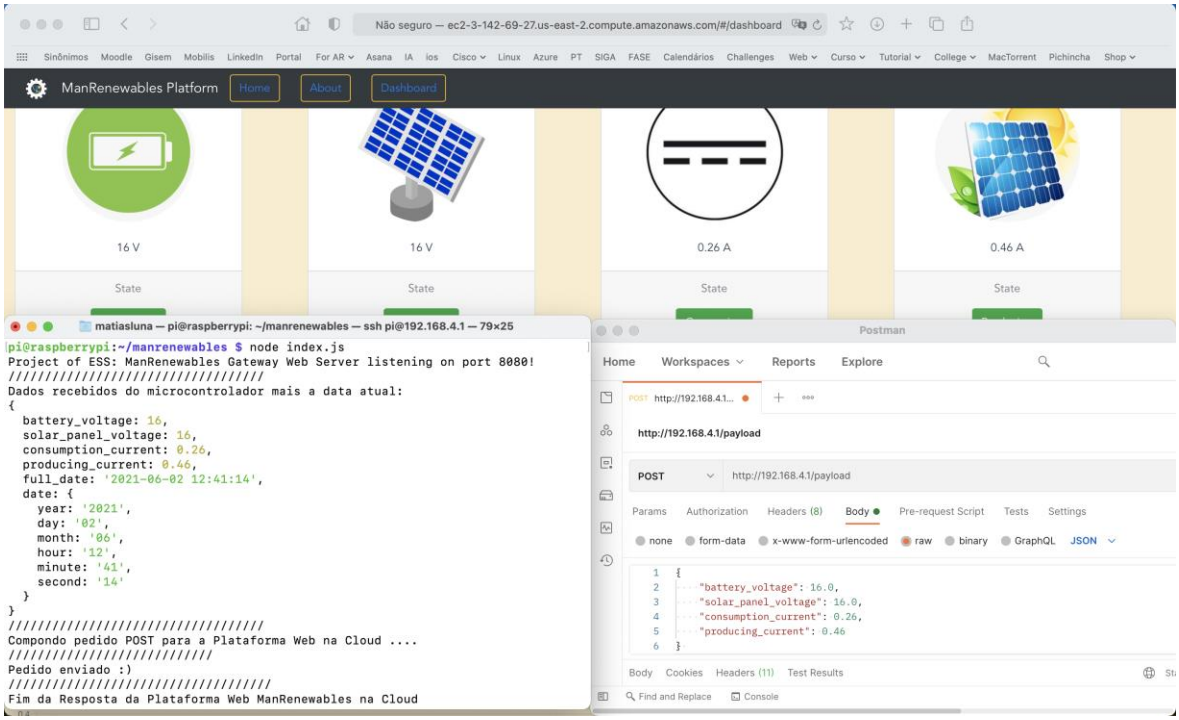


Figura 36 - Envio e visualização de dados em tempo real

- Visualização do gráfico dos dados do dia “02-06-2021” na plataforma Web na Cloud.

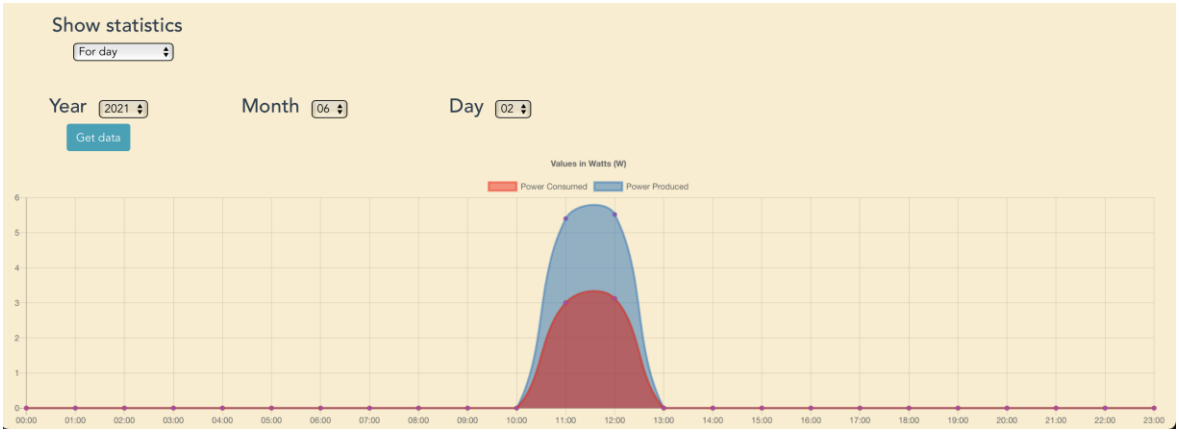
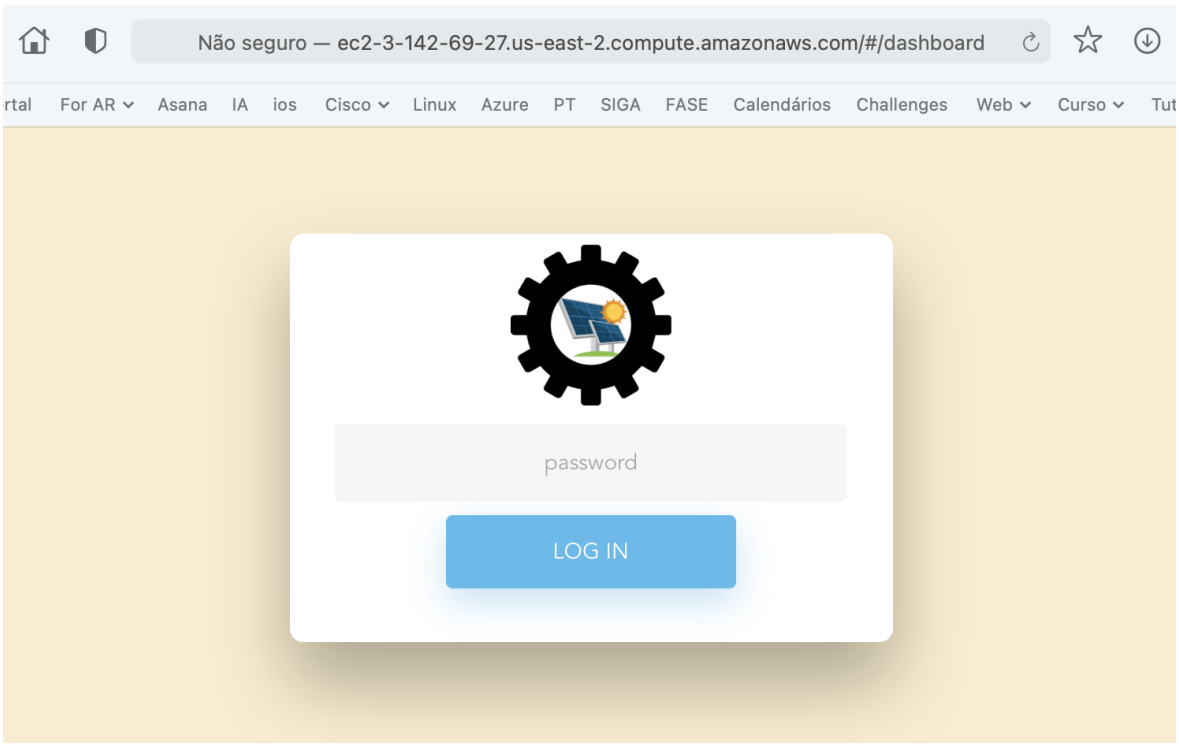
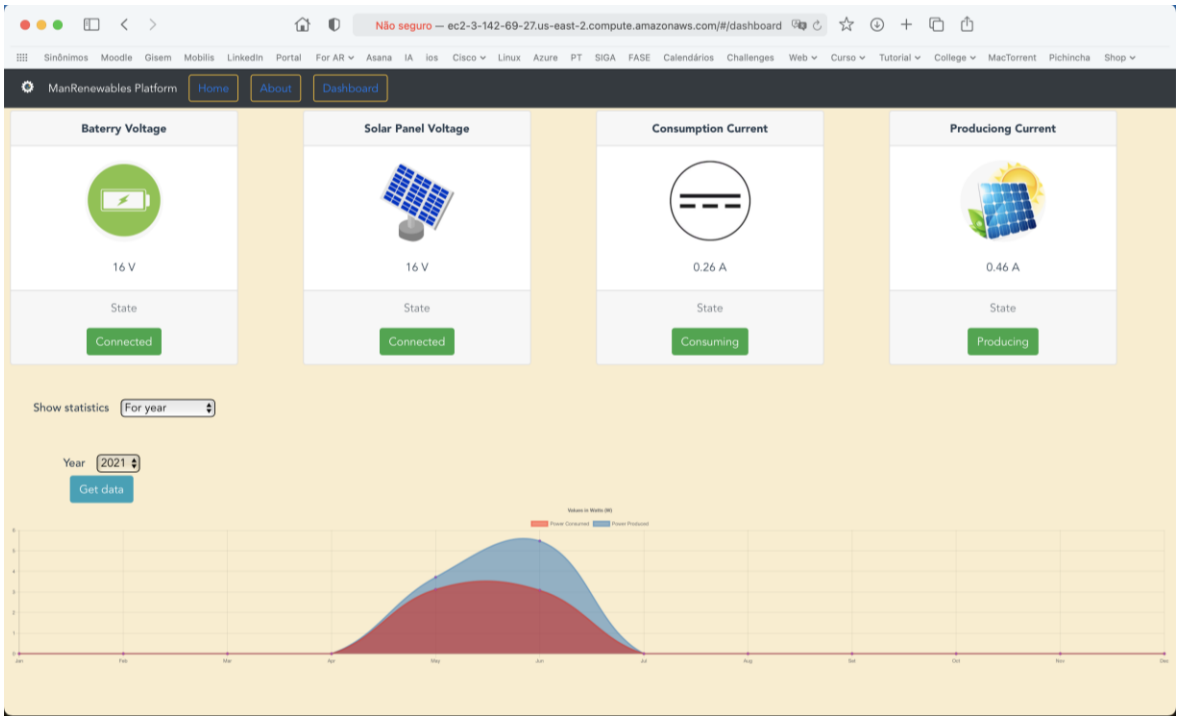


Figura 37 - Gráficos com valores de um dia específico

- Pedido de inserção de password antes de ingressar na Plataforma Web na Cloud



- Visualização do “Dashboard” da plataforma, onde se podem visualizar os dados em tempo real e selecionar uma data especifica para a criação de um gráfico visual.



- Ligação do ESP32 à rede e envio dos dados para o *gateway* (o endereçamento não é o mesmo explicado na secção 3.1.4 uma vez que na altura do teste o RaspberryPi não estava disponível. Então foi utilizado uma máquina virtual para o substituir)

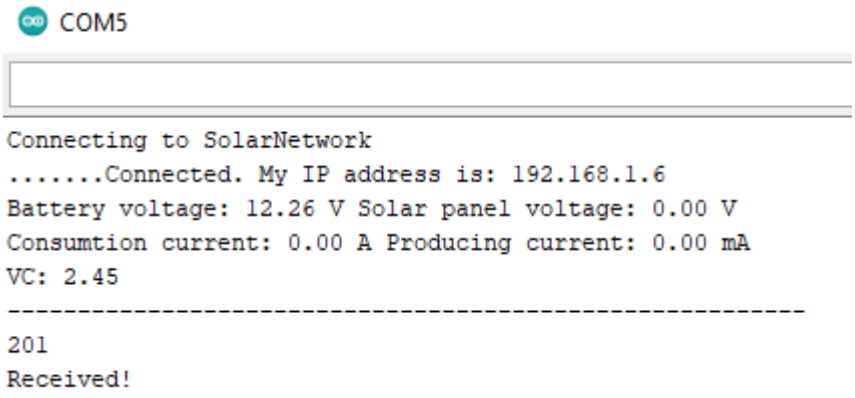


Figura 38 - Ligação do ESP32 à rede e envio dos dados

- Dados recebidos no *gateway* (máquina virtual que simula o RaspberryPi)

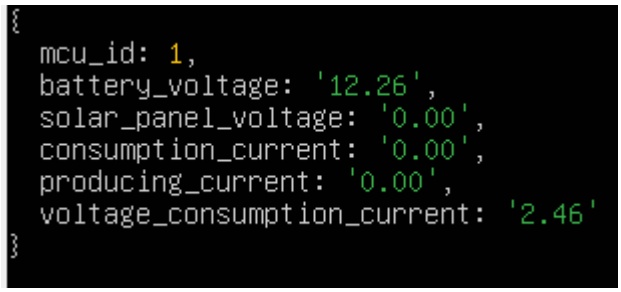


Figura 39 - Dados recebidos no *Gateway Raspberry Pi 3B*

6. Análise crítica e proposta de melhorias

Depois de realizados os testes para a criação do protótipo bem como os testes da solução final e toda a comunicação entre o RaspberryPi 3B e o ESP32, é possível concluir que todos os objetivos a que nos prepusemos se encontram cumpridos e todas as funcionalidades existentes se encontram totalmente funcionais.

Deve-se realçar, que este projeto pode ser facilmente aplicado a qualquer instalação de energia solar sem quaisquer problemas e que cumpre três pressupostos fundamentais: *AnyTime*, *AnyWhere* e *LowCost*.

Num trabalho futuro, existe a possibilidade de melhorar a interface gráfica do *website* bem como melhorar um sistema de autenticação de forma a proteger a aplicação.

7. Conclusão

Cada vez mais, a *Internet of Things* está presente no nosso dia-a-dia e, num mundo altamente tecnológico como aquele em que vivemos, as ferramentas disponibilizadas pelas *IoT* tornam-se cada vez mais poderosas e possibilitam o desenvolvimento de projetos bastante benéficos a um custo reduzido, sendo um exemplo o projeto que desenvolvemos, onde estamos a interligar energias renováveis com a tecnologia.

Com todas as preocupações ambientais existentes, projetos que auxiliem o combate ao aumento do efeito de estufa através do uso de energias renováveis são fundamentais, e o nosso projeto enquadra-se neste âmbito. Sem o uso das tecnologias associadas à *Internet of Things*, este projeto tornar-se-ia um projeto bastante dispendioso assim, e com o auxílio das *IoT* conseguimos transformá-lo num projeto ao alcance financeiro de várias famílias.

Bibliografia

[1] E. Projects, “How to measure current using Arduino and ACS712 Current Sensor,” [Online]. Available: https://www.engineersgarage.com/arduino/acs712-current-sensor-with-arduino/.

[2] M. Ferreira, “INA219 Tutorial for Arduino, ESP8266 and ESP32,” [Online]. Available: https://diyi0t.com/ina219-tutorial-for-arduino-and-esp/.

[3] A. e. Cia, “Calculadora Online – Divisor de tensão com resistores,” [Online]. Available: https://www.arduinoecia.com.br/calculador-divisor-de-tensao-function/.

[4] Espressif, “ESP32 Series Datasheet,” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[5] “Node.js,” [Online]. Available: https://nodejs.org/en/docs/guides/getting-started-guide/. [Acedido em 2 Junho 2021].

[6] “NGINX Docs | NGINX Reverse Proxy,” [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/. [Acedido em 2 Junho 2021].

[7] “Setting up a Raspberry Pi as a routed wireless access point - Raspberry Pi Documentation,” [Online]. Available: https://www.raspberrypi.org/documentation/configuration/wireless/access-point-routed.md. [Acedido em 6 Junho 2021].

Anexos

- Código desenvolvido

Para que o projeto possa ser desenvolvido em conjunto, foi colocado num repositório no Github para que se possa trabalhar em separado. O link de acesso está disponível aqui: https://github.com/matiasarielol/solarNetwork