U.C. de Projeto Integrado de Telecomunicações

Ano Letivo: **2021/2022**

Relatório da Fase B

Grupo 2

- Catarina Neves, a93088
- Eduardo Cardoso, a89627
- José Gomes, a93083
- Luís Oliveira, a89380

27/04/2022

Universidade do Minho
Mestrado Integrado em Engenharia de Telecomunicações e Informática

Índice

Índice de figuras

Lista de siglas e acrónimos

LED	*Light Emitting Diode*
PWM	*Pulse Width Modulation*
TCP	*Transmission Control Protocol*
IP	*Internet Protocol*
HTTP	*Hypertext Transfer Protocol*

1. Introdução

Serve o presente relatório como síntese do trabalho desenvolvido e implementado no decorrer desta segunda fase. Este contém a descrição das estratégias e algoritmos adotados pelo grupo, tal como os respetivos testes realizados. Além dos tópicos acima citados, faz-se referência a trabalhos ou projetos similares, que se enquadram na ótica deste projeto.

O relatório abordará cada etapa desta fase de forma detalhada, ou seja, será apresentada a resposta ou proposta de solução empregue pelo grupo em junção com as ferramentas que foram necessárias para a sua construção.

A figura 1 ilustra o sumário das diferentes tarefas propostas nesta fase.

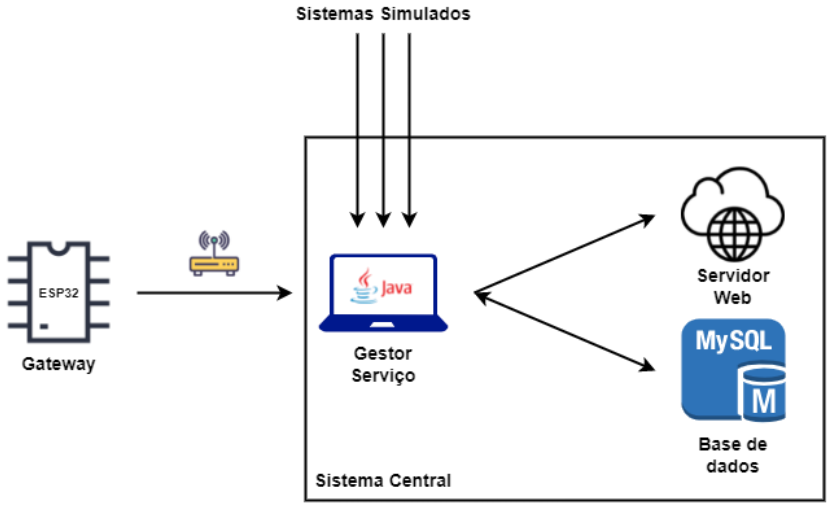

Figura 1 - Tarefas propostas pela fase B.

Nesta fase, o objetivo é a implementação de um sistema central e do protocolo de comunicação entre este sistema e os *gateways* (*Gateway* e Sistemas Simulados), que deve estar assente numa comunicação por mensagens aplicacionais sobre TCP (*Transmission Control Protocol*) /IP (*Internet Protocol*).

O Sistema Central recolhe a informação enviada por todos os *gateways*, que por sua vez, será armazenada na base de dados. Sempre que o serviço *web* queira atualizar os dados apresentados, este faz um pedido ao gestor de serviço que, de forma encadeada, irá buscar os dados à base de dados e enviará os mesmos para o serviço *web*.

2.Trabalho Relacionado

Relativamente aos trabalhos relacionados, enquadram-se os projetos seguintes, que serviram de base à construção deste projeto.

2.1 ESP32 WebSocket Server: Control Outputs

Este projeto, da autoria de Rui Santos [1], consiste numa aplicação que varia o estado de um determinado LED (*Light Emitting Diode*), sendo baseada no modelo cliente-servidor: a placa ESP32 desempenha o papel de servidor e o *browser*, que contém uma página *web* (configurada no código do servidor), atua como cliente, capaz de controlar a placa referida de forma remota.

A página *web* apresenta o estado do LED e possui um botão, que é responsável pela mudança de estado do mesmo (ligar ou desligar). É de salientar, que o LED é conectado à placa referida, através de fios de ligação e com o auxílio de uma *breadboard*.

O cliente estabelece uma ligação via *websocket* com o servidor, e quando esta é estabelecida, o cliente e o servidor podem enviar dados com recurso a *sockets*, de forma *full-duplex*.

A utilização do protocolo baseado em *websockets,* permite que o servidor envie informação para o(s) seu(s) cliente(s) sem necessitar de ser requisitado pelo(s) mesmo(s), logo, sempre que o estado do LED varie, essa informação é enviada para o *browser*, tal como é visível na figura 2.

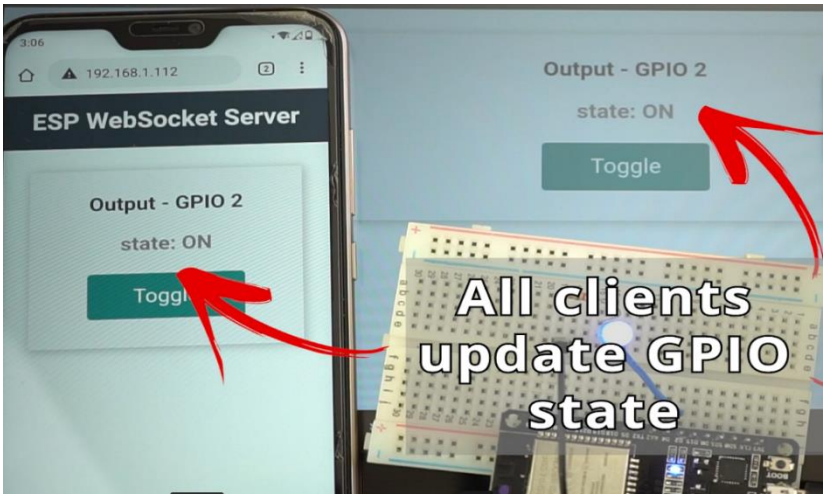

Figura 2 – Aplicação e circuito implementados pelo autor.

A aplicação funciona de acordo com os passos seguintes:

1. Clicar no botão "*Toggle*";
2. O cliente (*browser*) envia a mensagem "*toggle*" via *socket*;
3. O servidor (placa ESP32) recebe essa mensagem e muda o estado do LED;
4. O servidor envia o novo estado do LED para o(s) seu(s) clientes;
5. O(s) cliente(s) recebe(m) a mensagem e atualiza(m) o estado do LED, na *webpage* associada;

2.2 ESP32 WebSocket Server with Multiple Sliders: Control LEDs Brightness

Este projeto, da autoria de Rui Santos [2], consiste numa aplicação que permite variar a luminosidade de vários LEDs, sendo baseada no modelo cliente-servidor: a placa ESP32 desempenha o papel de servidor e o *browser*, que contém uma página *web* (configurada no código do servidor), atua como cliente, capaz de variar a luminosidade dos respetivos LEDs. A comunicação entre cliente(s) e servidor é realizada através de um protocolo assente em *websockets*.

A variação da luminosidade dos LEDs é feita através de um conjunto de *sliders,* que controlam o *duty cycle* dos respetivos sinais PWM (*Pulse Width Modulation*).

A figura seguinte ilustra a página web que contém os *sliders* referidos.

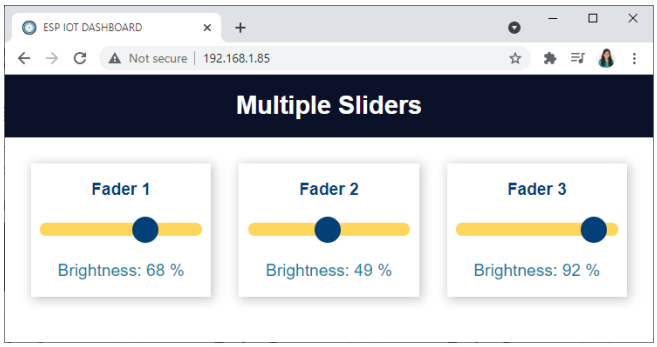

Figura 3 – Página web com 3 sliders.

A página web encontra-se organizada da seguinte forma:

- Um *fader* por LED, num total de 3 LEDs;
- Cada *fader* possui um *slider,* que é responsável pela variação da luminosidade do LED associado;
- A luminosidade do LED (em percentagem) varia entre 0 e 100;
- Sempre que um *slider* varie a luminosidade, esta é atualizada em todos os clientes de forma simultânea;

O servidor coloca na rede local uma página web, que é constituída por 3 *sliders*. Sempre que um *slider* é alterado (luminosidade alterada), o cliente envia o número relativo desse *slider* para o servidor, de acordo com o protocolo referido anteriormente. Por exemplo, se um utilizador colocar a posição do *slider* 3 em 50%, o cliente enviará a mensagem "**3s50**" para o servidor. Após o servidor ter recebido a mensagem mencionada, ajusta o *duty cycle* do sinal PWM, de acordo com os valores incorporados na mensagem e notifica os restantes clientes, com os novos valores.

A figura seguinte ilustra o circuito eletrónico efetuado pelo autor.

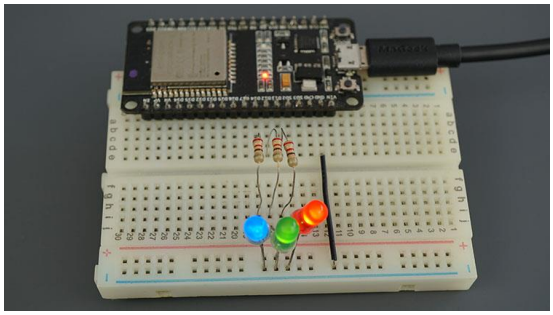

Figura 4 - Circuito eletrónico implementado pelo autor.

3. Etapas do trabalho desenvolvido

Esta secção contém todas as etapas necessárias ao desenvolvimento desta fase, que se encontram descritas pelos mecanismos, ferramentas utilizadas e algoritmos implementados pelo grupo.

3.1. Protocolo de comunicação

Para estabelecer a comunicação entre o Sistema Central e o *Gateway* recorreu-se ao protocolo TCP/IP, de modo a transmitir as tramas de dados recolhidas.

Para tal, o grupo declarou *sockets* do tipo servidor no Sistema Central e *sockets* do tipo cliente no *Gateway*. A partir destas, tirando partido da programação concorrente, foi estabelecida a referida ligação, que permite o bom funcionamento das interações necessárias.

Numa fase mais avançada do projeto, as tramas não irão passar só os valores obtidos pelo Sistema Sensor (e as amostras temporais), como também transportarão informação relativa ao atual funcionamento dos sensores e incluirão ordens de começo, paragem ou retoma da atividade.

A figura 5 apresenta os parâmetros necessários à implementação das *sockets* referidas e o objeto responsável pelo envio da trama e pelo estabelecimento da ligação com o Sistema Central (dentro do código do *Gateway*).

```
const uint16_t port = 5000;
const char * host = "192.168.1.7";

WiFiClient  client;
```

Figura 5 - Declaração da porta, do endereço e do cliente usados para a comunicação com o Sistema Central.

A figura 6 ilustra as mensagens de diagnóstico, relativas à conexão com o Sistema Central, e o envio da amostra para o mesmo.

```
Serial.println("Connected to server successful!");

client.print(temperatureChar);
Serial.println("Disconnecting...");
client.stop();
```

Figura 6 - Envio da trama de dados e finalização da impressão na socket, no Gateway.

A figura 7 ilustra o código do Sistema Central, que é responsável pela inicialização e execução do servidor.

```java
@SpringBootApplication
@RequiredArgsConstructor
public class DemoApplication {
    private static final int PORT = 5000;

    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }

    @Bean
    CommandLineRunner commandLineRunner(WeatherSampleRepo weatherSampleRepo) {
        return args -> {
            try(final var socket = new ServerSocket(PORT)) {
                while(true) {
                    final var attemptSocket :Socket = socket.accept();
                    new Thread(new WeatherSampleJob(attemptSocket, weatherSampleRepo))
                            .start();
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        };
    }
}
```

Figura 7 - Inicialização da socket servidor no sistema.

3.2. Sistema Central

O Sistema Central [3] implementado pelo grupo encontra-se mapeado, estruturado em 3 diretorias distintas:

- ***sample***: incorpora o código relativo ao gestor de serviço, que é responsável pelo processamento das amostras recolhidas e estabelece uma ponte de ligação entre a base de dados relacional e o serviço *web*.
  A figura seguinte ilustra o conjunto de ficheiros (interface e classes) presentes na diretoria descrita.

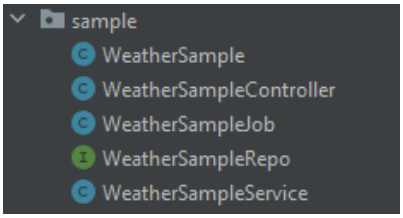

Figura 8 - Ficheiros presentes na diretoria sample.

- ***security*:** engloba o código responsável pela segurança do Sistema Central, ou seja, define um conjunto de funcionalidades de administração e monitorização, que por sua vez, são atribuídas a um conjunto de utilizadores, que possuem um determinado conjunto de funções e permissões.
  A figura seguinte ilustra o conjunto de ficheiros (classes e enumerados) presentes na diretoria descrita.

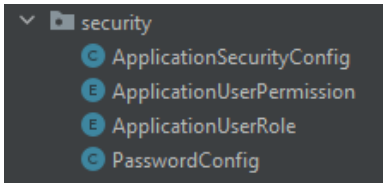

Figura 9 - Ficheiros presentes na diretoria security.

- ***template:*** composta pelo código intrínseco ao serviço *web*, isto é, possui todo o código inerente à interligação entre a componente gráfica do serviço referido e os endereços de ligação associados.
  A figura seguinte ilustra o ficheiro (classe) presente na diretoria descrita.

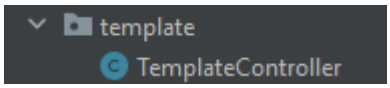

Figura 10 - Ficheiro presente na diretoria template.

- ***resources***: constituída por duas diretorias e por um ficheiro responsável pela comunicação com a base de dados relacional. A diretoria *templates* possui o código inerente à componente gráfica do serviço *web* (menus, imagens, etc.) e o ficheiro *application.properties* contém um conjunto de instruções relativas à aplicação e à base de dados relacional.
  A figura seguinte ilustra a constituição da diretoria referida.

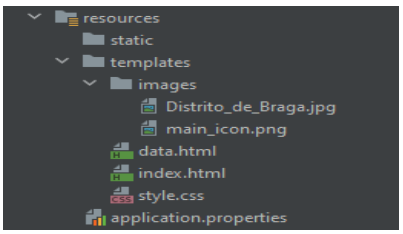

Figura 11 - Constituição da diretoria resources.

3.2.1 Sample

A diretoria *sample* contém um conjunto de classes e uma interface, que são necessários à transformação, tratamento e armazenamento das amostras.

A classe *WeatherSample* define a estrutura, representa o paradigma de qualquer amostra recolhida, isto é, contém todos os atributos e comportamentos inerentes à mesma.

A figura 12 ilustra os estados (variáveis de instância) e os comportamentos (métodos) relativos à classe *WeatherSample*.

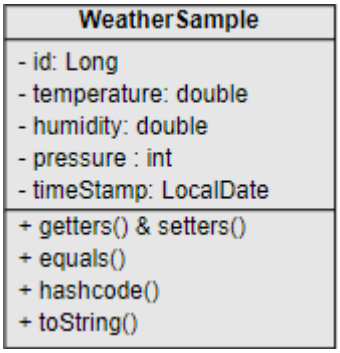

Figura 12 - Estrutura da classe WeatherSample.

É de salientar que dentro da base de dados, cada amostra é identificada pelo seu código identificador (ID), que corresponde à chave primária da tabela *weather*.

A figura seguinte ilustra os atributos da tabela, entidade *weather*.

```
mysql> DESCRIBE weather;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | bigint      | NO   | PRI | NULL    |       |
| humidity    | double      | NO   |     | NULL    |       |
| pressure    | int         | NO   |     | NULL    |       |
| temperature | double      | NO   |     | NULL    |       |
| time_stamp  | datetime(6) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figura 13 - Atributos da tabela weather.

As classes *WeatherSampleController*, *WeatherSampleService* e a interface *WeatherSampleRepo* atuam em diferentes camadas da aplicação.

A interface *WeatherSampleRepo* atua na camada de acesso à base de dados, ou seja, é responsável pela manipulação e gerenciamento da base de dados relacional. Esta interface estende a interface *JpaRepository*¸ que recebe duas classes como argumentos: a classe *WeatherSample*, que define os atributos da tabela *weather* e a classe *Long*, que representa o tipo da chave primária da tabela referida.

É de realçar que não foram definidos quaisquer métodos para a interface *WeatherSampleRepo*, uma vez que os métodos herdados, são suficientes para a complexidade deste projeto.

Relativamente à classe *WeatherSampleService*, esta estabelece a ponte de interligação entre a camada de serviço e a camada de acesso à base de dados, isto é, define um conjunto de métodos que operam sobre os métodos herdados pela interface *WeatherSampleRepo*.

A figura 14 ilustra os estados (variáveis de instância) e os comportamentos (métodos) relativos à classe *WeatherSampleService*.

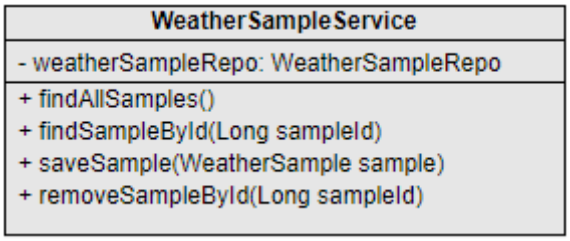

Figura 14 - Estrutura da classe WeatherSampleService.

No caso da classe *WeatherSampleController*, que equivale à camada de controlo, esta efetua um conjunto de *web requests* ao serviço web, do tipo: *GET*, *POST*, *PUT* e *DELETE*. Cada *web request* resulta numa *query* colocada à base de dados.

Em semelhança com o que acontece com a classe *WeatherSampleService*, a camada de controlo também define um conjunto de métodos, anotados com os tipos de *web requests* associados, que operam sobre os métodos definidos pela classe, que é responsável pela camada de serviço.

A figura 15 ilustra os estados (variáveis de instância) e os comportamentos (métodos) relativos à classe *WeatherSampleController*.

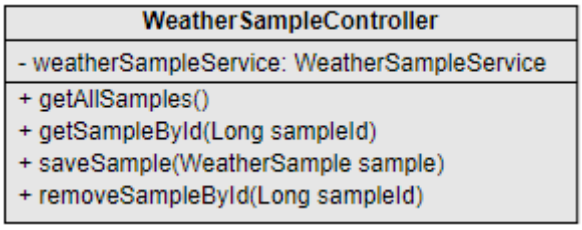

Figura 15 - Estrutura da classe WeatherSampleService.

A interação entre as diversas camadas do gestor de serviço, que foram descritas e analisadas ao longo desta subsecção, pode ser ilustrada de acordo com a figura 16.

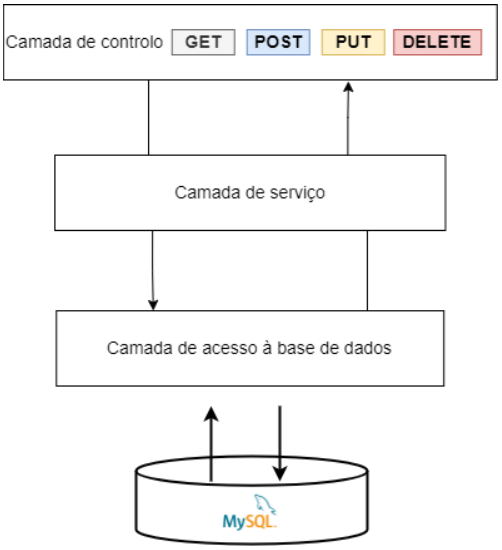

Figura 16 - Interações entre as camadas do gestor de serviço.

3.2.2 Security

A diretoria *security* [4] contém duas classes e dois enumerados, as classes qualificam e determinam a segurança do serviço *web*, enquanto, que os enumerados definem o tipo de utilizadores e respetivas habilidades ou permissões associadas.

A classe *ApplicationSecurityConfig* configura, filtra o protocolo HTTP (*Hypertext Transfer Protocol*), com recurso ao método *configure*, e regista o conjunto de utilizadores, que podem comunicar e interagir com o serviço *web*, através do método *userDetailsService*.

A figura 17 ilustra os estados (variáveis de instância) e os comportamentos (métodos) relativos à classe *ApplicationSecurityConfig*.

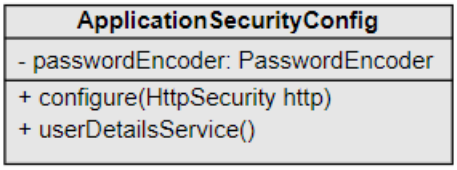

Figura 17 - Estrutura da classe ApplicationSecurityConfig.

Cada utilizador, que se encontra registado na aplicação, é caraterizado pelo seu *username* e por uma *password*, que por sua vez é encriptada, de forma a reforçar a segurança do serviço *web*.

3.2.3 Template

A diretoria *template* contém apenas uma classe que possui dois métodos, que retornam as páginas *web* (***get*** *requests*), que foram previamente definidas na diretoria *resources*.

A figura 18 apresenta os estados (variáveis de instância) e os comportamentos (métodos) relativos à classe *TemplateController*.

Figura 18 - Estrutura da classe TemplateController.

3.2.4 Resources

A diretoria *resources* abrange código relativo à componente gráfica da aplicação, isto é, o serviço *web* e possui o ficheiro de propriedades ou configurações referentes à base de dados relacional, como mostra a figura 19.

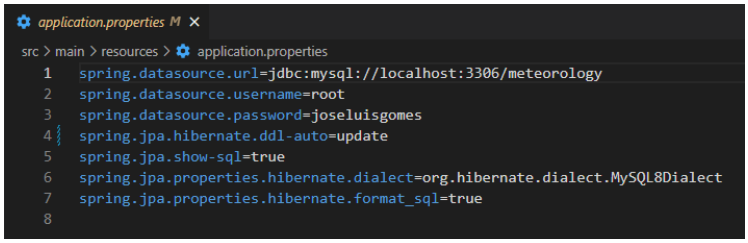

Figura 19 - Ficheiro com as configurações da base de dados.

3.3. Transmissão de assinaturas temporais por BLE

De maneira a obtermos o *timestamp* no servidor recorreu-se às capacidades do protocolo BLE, criando uma característica na qual será escrito o valor em segundos desde 1 de janeiro de 1970 às 00:00:00, como mostra a figura 20.

```
#define timeC
  BLECharacteristic timeCharacteristics("3a3e6d34-6a5d-4205-9b91-94ed7c92f6e1");
  BLEDescriptor timeDescriptor(BLEUUID((uint16_t)0x2902));
```

Figura 20 - Declaração da característica temporal no servidor.

De seguida, este valor é escrito na referida característica, como demonstra a figura 21.

```
char epochString[10];
sprintf(epochString,"%d",epochTime);
byte pSend[10];
for(int i=0;i<10;i++){
  pSend[i]=(byte)epochString[i];

}

 timeCharacteristic ->writeValue(pSend,sizeof(pSend));
```

Figura 21- População do array a ser enviado pela característica.

Assim sendo, o servidor lê os valores nestes impressos, evidenciada na figura 22.

```
timeArr=(char*)timeCharacteristics.getValue().c_str();
Serial.print("\n");
Serial.print(timeConverter());
```

Figura 22 - Leitura do valor temporal.

Tendo obtido o mencionado valor são realizadas um conjunto de operações de modo a converter o tempo em segundos numa *timestamp* com o formato: dia da semana, ano, mês, dia e hh(horas): mm(minutos): ss(segundos), como mostra a figura 23.

```
String timeConverter(){
  float toAdd=atof(timeArr);
  float timePassed = millis();
  int timePassedDiv = (int)roundNo(timePassed/1000);
  double currentTime = (double)timePassedDiv+(double)toAdd;
  time_t currentTimeT = currentTime;

    struct tm  ts;
    char       buf[80];

    // Format time, "ddd yyyy-mm-dd hh:mm:ss zzz"
    ts = *localtime(&currentTimeT);
    strftime(buf, sizeof(buf), "%a %Y-%m-%d %Hhh:%Mmm:%Sss %Z", &ts);

  return buf;
  }

  int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}
```

Figura 23 - Funções de conversão da data para o formato necessário.

É de notar que optamos por calcular o tempo atual adicionando o tempo em segundos desde a inicialização do Arduíno, visto nos termos deparado com faltas de fiabilidade relativamente a obtermos um valor temporal novo, do cliente, a cada segundo (tempos de execução e paragens no momento da abertura de *sockets*, podiam levar ao aumento do tempo de forma não ideal).

3.4. Servidor *Web*

Para esta etapa, o servidor *Web* que desenvolvemos foi inspirado num *template* desenvolvido pela W3Schools [5]. Este template foi depois modificado por nós para obtermos o aspeto e definições que pretendíamos, como mostra a figura 24.

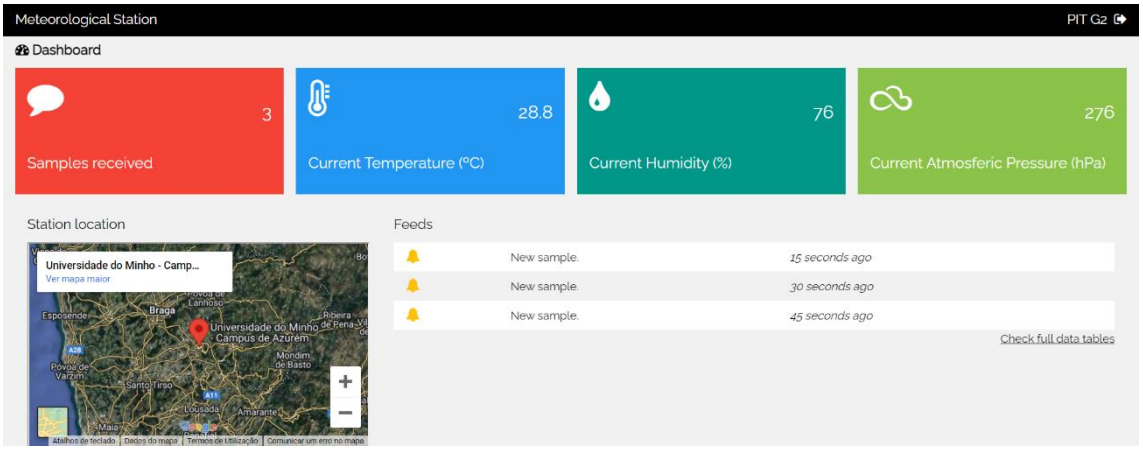

Figura 24 - Página inicial do servidor web.

Nesta figura, podemos ver no topo o número de amostras que o servidor já obteve, tanto como a temperatura, a humidade e a pressão atmosférica da última amostra recebida.

Por sua vez, se se premir o botão "*Check full data tables*", vamos para a página que vemos na figura 25. Nesta página, temos acesso a todas as amostras que foram enviadas para o servidor *web*.

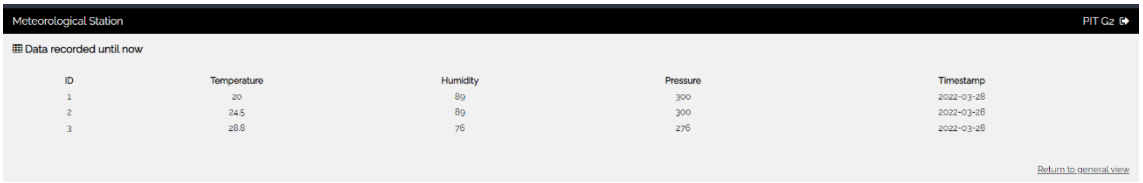

Figura 25 - Página com todas as amostras.

No código, uma parte importante para a apresentação dos dados é o processamento das amostras, para que a tabela dinâmica saiba quantas linhas, que correspondem ao número de amostras, deve criar, como mostra a figura 26.

```
<script type="text/javascript" charset="utf-8">

  fetch('http://localhost:8080/api/group2/samples') //fetch of all samples
          .then(function (response) {
              return response.json();
          }).then(function (getAllSamples) {
              //console.log(getAllSamples);
              renderDataInTheTable(getAllSamples);
          })

      function renderDataInTheTable(todos) { //function to update dynamic table depending on the number of samples
          const mytable = document.getElementById("html-data-table");
          todos.forEach(todo => {
              let newRow = document.createElement("tr");
              Object.values(todo).forEach((value) => {
                  let cell = document.createElement("td");
                  cell.innerText = value;
                  newRow.appendChild(cell);
              })
              mytable.appendChild(newRow);
          });
      }
</script>
</body>
```

Figura 26 - Processamento das amostras.

4. Análise de resultados e testes efetuados

De forma a retificar o funcionamento e o comportamento do Sistema Central e do protocolo de comunicação foram retiradas várias amostras, que evidenciam o funcionamento dos protocolos e mecanismos utilizados.

A figura 27 ilustra um conjunto de 10 amostras enviadas pelo Gateway para o Sistema Central, sendo armazenadas na base de dados pelo mesmo.

```
mysql> SELECT * FROM weather;
+----+----------+----------+-------------+----------------------------+
| id | humidity | pressure | temperature | time_stamp                 |
+----+----------+----------+-------------+----------------------------+
|  1 |       59 |      997 |        18.4 | 2022-04-25 23:10:49.912852 |
|  2 |       58 |      997 |        18.4 | 2022-04-25 23:11:09.874930 |
|  3 |       59 |      997 |        18.4 | 2022-04-25 23:11:29.949153 |
|  4 |       59 |      997 |        18.4 | 2022-04-25 23:11:49.909306 |
|  5 |       59 |      997 |        18.4 | 2022-04-25 23:12:09.877065 |
|  6 |       59 |      997 |        18.4 | 2022-04-25 23:12:29.961544 |
|  7 |       59 |      997 |        18.4 | 2022-04-25 23:12:49.907575 |
|  8 |       59 |      997 |        18.4 | 2022-04-25 23:13:09.983106 |
|  9 |       59 |      997 |        18.3 | 2022-04-25 23:13:29.966519 |
| 10 |       59 |      997 |        18.4 | 2022-04-25 23:13:49.841714 |
+----+----------+----------+-------------+----------------------------+
10 rows in set (0.00 sec)
```

Figura 27 - Amostras armazenadas na base de dados.

A figura 28 ilustra o conjunto de 10 amostras referido, colocadas no serviço *web*.

Meteorological Station PIT G2

Data recorded until now

ID	Temperature	Humidity	Pressure	Timestamp
1	18.4	59	997	2022-04-25T23:10:49.912852
2	18.4	58	997	2022-04-25T23:11:09.87493
3	18.4	59	997	2022-04-25T23:11:29.949153
4	18.4	59	997	2022-04-25T23:11:49.909306
5	18.4	59	997	2022-04-25T23:12:09.877065
6	18.4	59	997	2022-04-25T23:12:29.961544
7	18.4	59	997	2022-04-25T23:12:49.907575
8	18.4	59	997	2022-04-25T23:13:09.983106
9	18.3	59	997	2022-04-25T23:13:29.966519
10	18.4	59	997	2022-04-25T23:13:49.841714

Figura 28 - Amostras impressas no serviço web.

Para se estabelecer um fio de ligação entre a base de dados e o serviço web, o grupo construiu um *script* que se encontra apresentado na figura 29.

```
<script>
  fetch('http://localhost:8080/api/group2/samples') //fetch of all samples
          .then(function (response) {
              return response.json();
          }).then(function (getAllSamples) {
              //console.log(getAllSamples);
              parseData(getAllSamples);
          })

  function parseData(samples){ //function to parse the samples
    samples = JSON.stringify(samples); //samples are converted to JSON

    lastSample = JSON.parse(samples); //JSON samples are then parsed in order to find the last sample
    lastSample = lastSample[lastSample.length - 1];
    displaySample(lastSample);
  }

  function displaySample(sample){ //function to display the last sample on each respective space
    var text;
    var temp = 0;
    var update = [];
    var string = [];

    document.getElementById('num_samples').innerHTML = lastSample.id;
    document.getElementById('cur_temp').innerHTML = lastSample.temperature;
    document.getElementById('cur_hum').innerHTML = lastSample.humidity;
    document.getElementById('cur_pres').innerHTML = lastSample.pressure;
```

Figura 29 - Script que efetua o parsing das amostras.

Relativamente ao domínio dos sistemas simulados, o grupo definiu um conjunto de amostras *hardcoded* e armazenou-as na base de dados, com o intuito de testar as funcionalidades do repositório e a ligação com o serviço *web*.

A figura 30 ilustra os valores *hardcoded* definidos pelo grupo.

```
@Bean
CommandLineRunner commandLineRunner(WeatherSampleRepo weatherSampleRepo) {
        return args -> {
                weatherSampleRepo.deleteAll();

                final var timeStamp = LocalDate.of(2022, Month.MARCH, 28);
                final var sample1 = new WeatherSample(
                                20.0,
                                89.0,
                                300,
                                timeStamp);
                final var sample2 = new WeatherSample(
                                24.5,
                                89.0,
                                300,
                                timeStamp
                );
                weatherSampleRepo.saveAll(List.of(sample1, sample2));
                weatherSampleRepo.findAll().forEach(System.out::println);
                //weatherSampleRepo.saveAll(samples);
        };
}
```

Figura 30 - Definição das amostras hardcoded.

A figura 31 ilustra as amostras armazenadas na base de dados, que por sua vez encontram-se impressas no serviço *web*.

```
mysql> SELECT * FROM weather;
+----+----------+----------+-------------+----------------------------+
| id | humidity | pressure | temperature | time_stamp                 |
+----+----------+----------+-------------+----------------------------+
|  1 |       89 |      300 |          20 | 2022-04-27 23:31:24.795220 |
|  2 |       89 |      300 |        24.5 | 2022-04-27 23:31:24.795220 |
+----+----------+----------+-------------+----------------------------+
2 rows in set (0.00 sec)
```

Meteorological Station PIT G2

Data recorded until now

ID	Temperature	Humidity	Pressure	Timestamp
1	20	89	300	2022-04-27T23:31:24.795220
2	24.5	89	300	2022-04-27T23:31:24.795220

Return to general view

Figura 31 - Armazenamento e impressão das amostras hardcoded.

5. Conclusão

Concluída esta fase, na opinião do grupo, conseguimos atingir todos os objetivos propostos. Esta fase contribuiu para o conhecimento e estudo da implementação de *Sockets*, comunicação sobre TCP/IP e sobre o desenho de Servidores *Web*, tal como comunicação bilateral utilizando o protocolo BLE.

O grupo pretende que, com o trabalho desenvolvido ao longo deste fase intermédia, ter boas fundações para completar a etapa final deste projeto com sucesso.

5.1. Contribuição de cada aluno

Catarina Neves

- Conceção dos algoritmos;
- Tratamento de dados, tabelas e imagens a estes relacionados;
- Transmissão de *timestamps,* sua codificação e descodificação;
- Programação e implementação da aplicação dos Sistema Sensor e *gateway*;
- Desenvolvimento do relatório;

Eduardo Cardoso

- Conceção dos algoritmos;
- Transmissão de *timestamps,* sua codificação e descodificação;
- Programação e implementação da aplicação dos Sistema Sensor e *gateway*;
- Desenvolvimento e implementação das *sockets* e comunicação TCP/IP;
- Desenvolvimento do relatório;

José Gomes

- Conceção dos algoritmos;
- Tradução das classes em diagramas;
- Implementação do Sistema Central;
- Desenvolvimento e implementação das *sockets* e comunicação TCP/IP;
- Desenvolvimento do relatório;

Luís Oliveira

- Conceção dos algoritmos;
- Desenvolvimento do *Frontend* e *Backend* do Servidor *Web*;
- Implementação do Sistema Central;
- Desenho de esquemas e gráficos;
- Desenvolvimento do relatório.

6. Lista de referências

[1] R. Santos, “Random Nerd Tutorials,” 10 2020. [Online]. Available: https://randomnerdtutorials.com/esp32-websocket-server-arduino/.

[2] R. Santos, “ESP32 Web Server (WebSocket) with Multiple Sliders: Control LEDs Brightness (PWM),” 5 2021. [Online]. Available: https://randomnerdtutorials.com/esp32-web-server-websocket-sliders/.

[3] VMware, “Spring Boot,” 2022. [Online]. Available: https://spring.io/projects/spring-boot?msclkid=a8a9e52ac53b11ec97ffa5a63eb35e48.

[4] VMware, “Spring Security,” 2022. [Online]. Available: https://spring.io/projects/spring-security?msclkid=03ed647cc53c11ec824b0986c6465ca1.

[5] W3Schools, “W3Schools W3.CSS,” [Online]. Available: https://www.w3schools.com/w3css/.

[6] NTP Pool Project, “Europe — europe.pool.ntp.org,” [Online]. Available: https://www.pool.ntp.org/zone/europe.