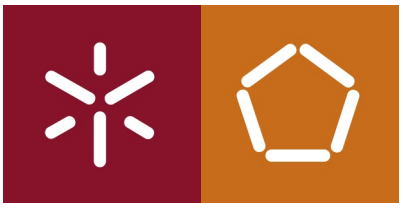
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Inteligência Ambiente: Tecnologias e Aplicações

Monitorização de Parâmetros Ambientais

Guilherme Pereira Martins (A70782)
André Rodrigues Soares (A67654)
Hugo Manuel Gomes Nogueira (A81898)
Joana Esteves Dantas (A84193)

Novembro 2020

Conteúdo

1 Introdução

Neste trabalho prático foi proposto aos alunos o desenvolvimento de um sistema inteligente, capaz de gerar informação útil no contexto escolhido pelo grupo. O foco deste desenvolvimento esteve na sensorização da temperatura recorrendo à API pública disponibilizada pela openweathermap de forma a garantir o acesso aos dados recolhidos foi criada uma aplicação em javascript, usando nodejs, em que através de meios de comunicação entre os sensores, backend e frontend o utilizador tenha à disponibilidade páginas web onde possam ser visualizados os dados através de gráficos, tabelas, etc.

Para além disto, foi também proposto um complemento que se foca em recolher dados de sensores como o teclado e o rato. Desta forma, utilizando código desenvolvido nas aulas, foram captados dados sobre as teclas premidas por um elemento dos grupos durante um pequeno intervalo de tempo e as métricas são também projetadas no frontend desenvolvido para a primeira parte.

2 OpenWeatherMap

2.1 API

Deparados com a necessidade de escolher um ou mais sensores físicos e/ou virtuais, o grupo, decidiu utilizar a API pública disponibilizada pela openweathermap.org. Esta API dá-nos possibilidade de aceder a temperatura ambiente actual em mais de 200.000 cidades em todo o mundo, recolhendo e processando dados de diferentes fontes como satélites, radares, estações de tempo, etc. Os dados podem ser dados em JSON, XML ou HTML e são gratuitos bastando apenas criar uma conta.

```
api.openweathermap.org/data/2.5/
weather?lat={lat}&lon={lon}&units=metric&appid={API key}
```

É desta maneira que recebemos a informação do sensor, o nosso programa lê a latitude e a longitude do utilizador (será explicado mais à frente) e recolhe a informação com unidades métricas.

Exemplo resposta de API:

```
{
  "coord": {
    "lon": -122.08,                \\City geo location, longitude
    "lat": 37.39                   \\City geo location, latitude
  },
  "weather": [
    {
      "id": 800,                   \\Weather condition id
      "main": "Clear",             \\Group of weather parameters
      "description": "clear sky",  \\Condition within the group
      "icon": "01d"                \\Weather icon id
    }
  ],
  "base": "stations",
  "main": {
    "temp": 16.55,                 \\Temperature
    "feels_like": 16.66,           \\Human perception of weather
    "temp_min": 16.37,             \\Minimum temp at the moment
    "temp_max": 16.86,             \\Maximum temp at the moment
    "pressure": 1023,              \\Atmospheric pressure, hpA
    "humidity": 100                \\Humidity, %
  },
  "visibility": 16093,
```

```
"wind": {
  "speed": 1.5,                  \\Wind speed, meter/sec
  "deg": 350                     \\Wind direction, degrees
},
"clouds": {
  "all": 1                       \\Cloudiness, %
},
"dt": 1560350645,                \\Time of data calculation, UNIX
"sys": {
  "type": 1,
  "id": 5122,
  "message": 0.0139,
  "country": "US",                \\Country code
  "sunrise": 1560343627,          \\Sunrise time, UTC
  "sunset": 1560396563            \\Sunset time, UTC
},
"timezone": -25200,               \\Shift in seconds from UTC
"id": 420006353,                  \\City ID
"name": "Mountain View",          \\City name
"cod": 200
}
```

Utilizamos o formato JSON porque tornou mais facil a manipulação de dados na aplicação.

2.2 Funcionamento da aplicação

Para começar, resolvemos criar um servidor que corresse no nosso computador e onde são guardados os dados. Todo o desenvolvimento da aplicação foi feito em JavaScript. Por norma só utilizamos e executamos javascript no browser, com nodejs podemos escrever javascript runtime, ou seja, o código corre no nosso computador. O nosso servidor é uma aplicação que corre e ouve solicitações, "algo"que se queira conectar ao servidor. Assim o objetivo é que se possam fazer requests ao servidor para apanhar dados. Utilizamos um package chamado express, é uma framework simples para fazer webservers que tinha as funcionalidades que queríamos para começar a trabalhar. Inicialmente procuramos opções de como geolocalizar o cliente, ou seja, recolher e mostrar a latitude e longitude do cliente e mandar essa informação para o servidor. A API Navigator (https://developer.mozilla.org/pt-BR/docs/Web/API/Navigator) tem informação sobre a identidade do utilizador como a bateria, linguagem do computador e a localização é uma delas. Quando o utlizador está na página principal é utilizada esta API para determinar a sua latitude e longitude

```
if ('geolocation' in navigator) {
navigator.geolocation.getCurrentPosition( async position => {

    try {
        lat = position.coords.latitude;
        lon = position.coords.longitude;
        const api_url = `weather/${lat},${lon}`;
        const response = await fetch(api_url);
        const json = await response.json();
```

Neste momento, o cliente através da interface Navigator enviou a sua latitude e longitude para o servidor na localização weather/${lat},${lon} No servidor esta informação é recebida com um get:

```
app.get('/weather/:latlon', async (request, response) => {
    const latlon = request.params.latlon.split(',');
    const lat = latlon[0]; const lon = latlon[1];
```

Agora que o servidor tem a latitude e a longitude, podemos utilizar a API do openweathermap com esses valores de localização para sabermos as informações sobre o tempo no local do cliente:

```
app.get('/weather/:latlon', async (request, response) => {
    [...]
    const weather_url = `https://api.openweathermap.org/data/2.5/
        weather?lat=${lat}&lon=${lon}&units=metric&appid={API_KEY}`;
```

```
    const weather_response = await fetch(weather_url);
    const weather_data = await weather_response.json();
    [...]
}
```

Chamamos então a API com os valores de latitude e longitude e agora que temos toda a informação sobre o tempo na constante weather_data, voltamos a enviar esta informação ao cliente para poder informar o cliente da temperatura, vento, humidade, etc:

```
app.get('/weather/:latlon', async (request, response) => {
    [...]
    const data = { weather: weather_data };
    response.json(data);
}
```

Agora que o cliente side tem toda a informação necessária, podemos mostrar ao utilizador os parâmetros desejados na página principal, exemplos desta página são mostrados mais à frente. Resta dizer que aqui os parâmetros são novamente enviados ao servidor desta vez para a localização /api:

```
    const data = { lat, lon, weather};
    const options = {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
    };
    const db_response = await fetch('/api', options);
    const db_json = await db_response.json();
```

Esta informação é mais uma vez recebida no servidor que a vai utilizar para guardar numa base de dados. Uma base de dados é para persistência, a habilidade de guardar dados ao longo do tempo: como o objetivo é ir recolhendo dados ao longo do tempo precisamos de os guardar para os aceder noutra altura mesmo que o servidor tenha sido reiniciado entretanto. Para guardar os dados utilizamos um sistema de base de dados open-source e bastante simples baseada em JavaScript chamada NeDB (https://github.com/louischatriot/nedb)

```
const database = new Datastore('database.db');//criação base de dados
database.loadDatabase();

app.post('/api', (request, response) => {
   const data = request.body;
```

```
    const timestamp = Date.now();
    data.timestamp = timestamp;
    database.insert(data);
    response.json(data);
});
```

Como podemos ver quando recebemos informação do cliente com a latitude, longitude e dados do tempo, guardamos tudo na base de dados database.db. Isto é feito no post do /api, de maneira a que quando for preciso aceder a toda a informação na parte do cliente podemos fazer um get: acedendo ao localhost:3000/api toda a informação presente no database.db vai estar lá em modo JSON:

```
app.get('/api', (request, response) => {
    database.find({}, (err, data) => {
        if (err) {
            response.end();
            return;
        }
        response.json(data);
    });
});
```

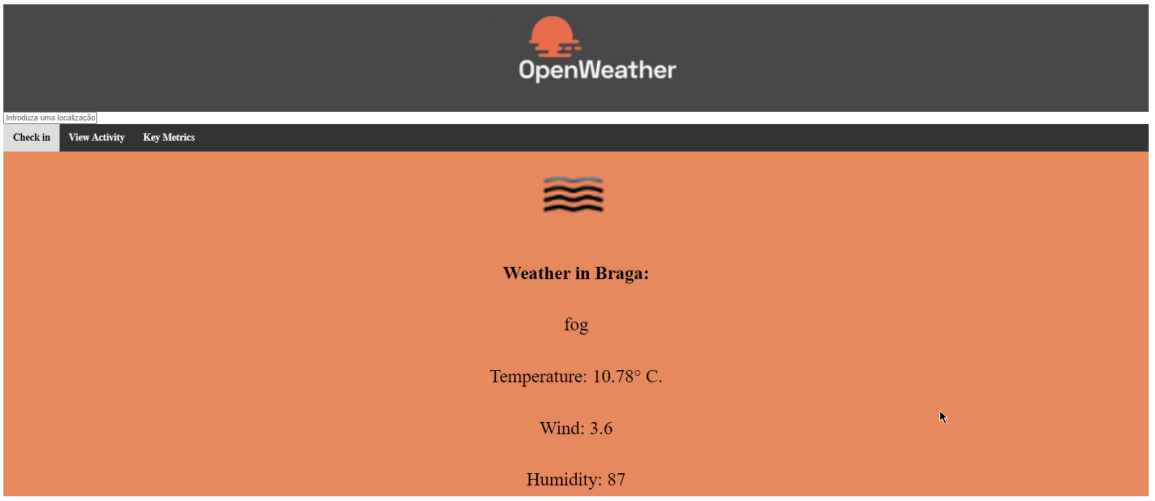
E assim está descrita a maneira principal como os dados são recolhidos e enviados de um lado para o outro na nossa aplicação, uma vez todos os dados poderem ser acedidos em /api podemos ir lá buscar tudo o que é necessário para construir as métricas desejadas. Resumidamente:

- Utilizador recolhe latitude e longitude através da API Navigator
- lat e lon são enviados para o servidor
- servidor recolhe dados da API do openweather associados a essa longitude e latitude
- informação recolhida da API do openweather é enviada de novo para o client side onde é mostrada ao utilizador
- client side volta a enviar info para o servidor com latitude, longitude e informação do tempo
- servidor coloca informação na base de dados e faz um POST da base de dados

2.3 Métricas utilizadas sobre dados recolhidos

2.3.1 Página Principal - Informação atual

Na página principal são nos dadas várias informações do tempo, é nesta página que a latitude e longitude do utilizador é lida e que depois posteriormente é enviada num POST para o servidor que vai colocar tudo na base de dados, resolvemos colocar informação como a temperatura atual, o estado do tempo, a velocidade do vento e a humidade:



Quando esta página é carregada diz-nos automaticamente a informação sobre a nossa localização.

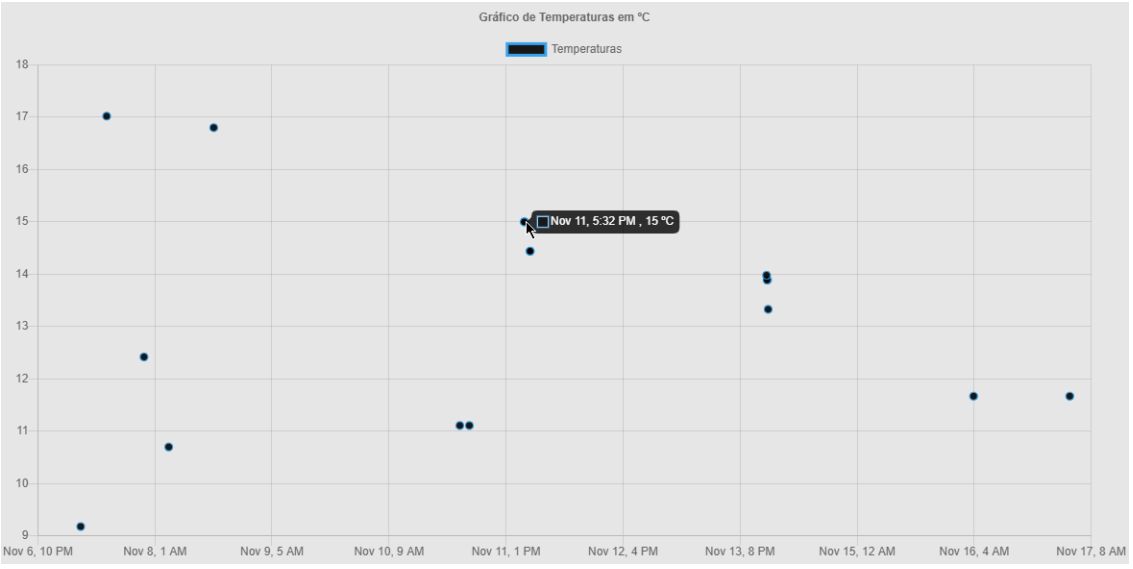
2.3.2 Gráfico de Temperaturas

Uma vez que agora podemos ir buscar toda a informação à localização /api começamos a construir métricas que podiam ser interessantes demonstrar. Através da framework Chart.js construimos um gráfico que mostra todas leituras guardadas na base de dados associadas ao instante temporal que foram lidas.

```
const response = await axios.get('/api'); \\recolha dos dados da BD
for (item of response.data) {
    var date = new Date(item.timestamp).toLocaleString("en-US");
    var temp = item.weather.main.temp;
    [...]
    data_temperature.push({
        x: moment(date),
```

```
        y: temp
    });
[...]
```

Recolhemos os dados do instante temporal, que é dado em UNIX ou seja o número de segundos passados desde 1 de Janeiro de 1970. Este valor é passado para uma data e junto com a temperatura lida naquele instante são colocados num dataset que vai ser utilizado para construir o gráfico de temperaturas:



Neste gráfico cada ponto é uma leitura, e cada leitura tem associada um temperatura e um instante.

2.3.3 Médias e Temperaturas máximas/mínimas

Ao longo da iteração onde recolhemos os dados da base de dados através do get da API, à medida que vamos introduzindo variáveis em datasets que vão ser uteis para o desenho de gráficos vamos também usando variáveis para controlar a temperatura mais alta/baixa lida e a temperatura média:

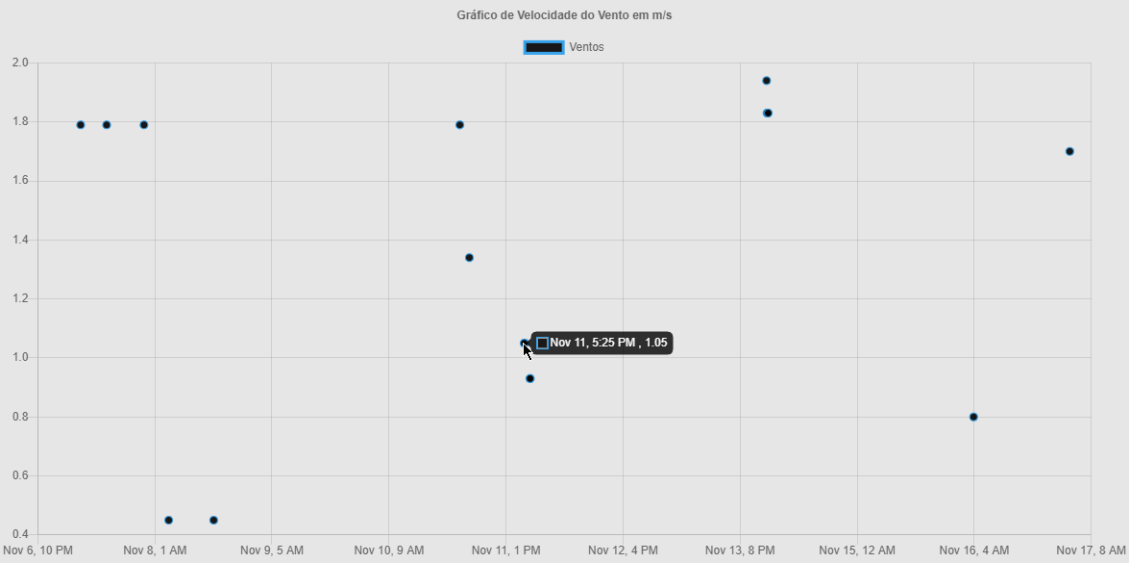
Média de Temperaturas: 13.27ºC

Temperatura mais alta registada: 17.02ºC no dia: 11/7/2020, 2:20:07 PM

Temperatura mais baixa registada: 9.18ºC no dia: 11/7/2020, 8:10:05 AM

2.3.4 Gráfico de velocidade do vento

À semelhança do gráfico de temperatura, também aproveitamos o facto da API do openweathermap nos dizer a velocidade do vento no momento e colocamos a informação num gráfico:



2.3.5 Tabela de leituras

Colocamos ainda uma tabela com todas as leituras lidas visto que nos gráficos podemos ver poucos parâmetros relacionados com cada leitura:

Tabela de leituras:

Click Me to hide table

Scanning Time	City	Temperature, °C	Description	Vento, m/s	Atmos Pressure, hPa	Humidity, %	Clouds, %
11/16/2020, 4:31:19 AM	Braga	11.67	overcast clouds	0.8	1023	90	100
11/11/2020, 4:25:10 AM	Braga	11.11	broken clouds	1.34	1023	82	77
11/11/2020, 5:25:52 PM	Braga	15	overcast clouds	1.05	1021	79	100
11/14/2020, 2:57:15 AM	Nogueiró	13.98	scattered clouds	1.94	1017	77	33
11/14/2020, 3:20:01 AM	Braga	13.33	scattered clouds	1.83	1017	78	33
11/17/2020, 3:50:48 AM	Braga	10.78	fog	3.6	1024	87	0

2.4 Adafruit + IFTT

Tal como desafiado pela equipa docente decidimos implementar a utilização do Adafruit e da aplicação IFTT para telemóveis de modo a criar um feed e lançar notificações para o nosso telemóvel dependendo de um determinado acontecimento que nós definimos. Quando é lida a temperatura na página principal é lançado esse valor para um temperature_feed criado no adafruit:

```
//create a client instance
client = new Paho.MQTT.Client("io.adafruit.com",Number(443),"JS_Client");
//set callback handlers
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
//connect the client
    client.connect({onSuccess:onConnect, userName:"{NAME}",
    password:"{AIO_KEY}", useSSL:true, mqttVersion:4});
```

Aqui podemos ver a criação da ligação ao ADAFRUIT. Quando a criação é bem sucedida é chamada a função onConnect() que envia a temperatura (guardada na variável temps) para o feed:

```
function onConnect() {
    console.log("onConnect");
    //subscribe
    client.subscribe("iata_2020/feeds/temperature_feed");
    var temps = temperat.toString();
    message = new Paho.MQTT.Message(temps);
    message.destinationName = "iata_2020/feeds/temperature_feed";
    client.send(message);
}
```

Depois da conexão da conta do adafruit ao IFTTT e da criação de uma condição que neste caso foi a temperatura lidar ser menor que 20 por questões de teste recebemos a notificação:

3 Adenda

3.1 Descrição dos sensores utilizados

A realização desta adenda tinha como propósito recolher dados de sensores como teclado e/ou rato. Neste caso o grupo optou por apenas fazer sensorização do teclado. Assim sendo criou-se um programa em *python*, que faz o *"log"* das teclas pressionadas enquanto o programa está a executar. Este programa usa o *package keyboard* que pertence à *library pynput*. Este *package* é usado para controlar e monitorizar o teclado, sensor do qual pretendemos recolher dados. Esses dados são escritos em um ficheiro JSON para posteriormente serem usados.

3.2 Descrição dos dados recolhidos e do tratamento efetuado

Quanto aos dados recolhidos, apenas são guardados o *timestamp* do momento em que uma tecla foi premida e o valor da própria tecla. Posto isto, é criado um ficheiro JSON que contém nele guardado um *array* de objetos, sendo que cada objeto é composto pelo *timestamp* e pelo valor da tecla. Ficamos então com um ficheiro JSON neste formato:


```
[{"Timestamp":1605552727,"Key":"Key.shift"},
{"Timestamp":1605552727,"Key": "I"},
{"Timestamp":1605552728,"Key": "n"},
{"Timestamp":1605552728,"Key": "t"},
{"Timestamp":1605552728,"Key": "e"},
{"Timestamp":1605552728,"Key": "l"},
{"Timestamp":1605552728,"Key": "i"},
{"Timestamp":1605552729,"Key": "g"},
{"Timestamp":1605552729,"Key":"Key.shift"},
{"Timestamp":1605552729,"Key": "^"},
{"Timestamp":1605552730,"Key": "e"},
{"Timestamp":1605552730,"Key": "n"},
{"Timestamp":1605552730,"Key": "c"},
{"Timestamp":1605552730,"Key": "i"},
{"Timestamp":1605552730,"Key": "a"},
{"Timestamp":1605552730,"Key":"Key.space"},
```


Tanto a monitorização do teclado, como a respectiva criação do ficheiro JSON são feitas através da função *"on_press"* que irá ser executada sempre que ocorra o evento de uma tecla ser premida. Esta função irá abrir o ficheiro indicado, e escrever nele um objecto JSON com os valores referentes à tecla premida, fazendo isto para todas as teclas premidas para desta forma ir construindo o nosso array de objetos.

```
def on_press(key):
    f = open("logger.JSON", "a")
    try:
        print('alphanumeric key {0} pressed'.format(key.char))
        value = '{0}'.format(key.char)
        timestamp = calendar.timegm(time.gmtime())
        f.write("{\"Timestamp\":" + str(timestamp) + "," + "\"Key\": \"" + value + '\"},\n')
    except:
        print('special key {0} pressed'.format(key))
        value = '{0}'.format(key)
        timestamp = calendar.timegm(time.gmtime())

        if key == keyboard.Key.esc:
            f.write("{\"Timestamp\":" + str(timestamp) + "," + "\"Key\":\"" + value + '\"}\n')
        else:
            f.write("{\"Timestamp\":" + str(timestamp) + "," + "\"Key\":\"" + value + '\"},\n')
    f.close()


# detect key releases
def on_release(key):
    print('{0} released'.format(key))

    if key == keyboard.Key.esc:
        # Stop Listener
        print('Stop')
        return False


def js(s):
    f = open("logger.JSON", "a")
    f.write(s)
    f.close()


# Collect events
js("[")
with keyboard.Listener(
        on_press=on_press,
        on_release=on_release) as listener:
    listener.join()
js("]")
```

Depois de criado o ficheiro JSON, é possível no *frontend* criado (página HTML), escolher o ficheiro sobre o qual serão calculadas as métricas definidas. Com o ficheiro seleccionado, e através do uso de *javascript*, o mesmo é lido por meio de um *filereader*. Obtendo o resultado dessa leitura, é feito o *JSON.parse* do mesmo, ou seja, é guardado numa variável o conjunto de objetos JSON contido no ficheiro, podendo agora esses mesmos objetos serem manipulados usando essa variável.

```
document.getElementById('inputfile')
            .addEventListener('change', function () {
                var fr = new FileReader();
                fr.onload = function () {
                    /**Parse do ficheiro JSON**/
                    res = JSON.parse(fr.result);
```

Para poder ser possível efetuar o calculo das métricas, foram criados vários *arrays/objetos*.

Os arrays *arr* e *t* contêm, respetivamente, todas as *keys* registadas no ficheiro e os seus *timestamps*.

```
const v = Object.entries(res);

                    /**Pôr o timestamp e a key**/
```

```
for (let i = 0; i < v.length; i++) {
    arr[i] = v[i][1].Key;
    t[i] = v[i][1].Timestamp
    //arr.forEach
}
```

O objeto *result* foi criado para, em vez de conter todos os valores das *keys* premidas, para passar a conter todos os valores de *keys* distintos, associados com o número que representa as vezes que essa *key* foi premida. (ex: {"a": 4, "b": 1,(...)})

```
/**Criar objeto com as keys e o nº de vezes primidas**/
let j = 0;
for (var i = 0; i < arr.length; ++i) {
    if (!result[arr[i]]) {
        result[arr[i]] = 0;
        k[j] = arr[i];
        j++;
    }
    ++result[arr[i]];
}
```

3.3 Descrição das métricas utilizadas sobre os dados recolhidos

São apresentadas de seguida, as métricas implementadas no trabalho.

3.3.1 Total de teclas pressionadas

Nesta métrica, é calculada o número total de teclas registadas no ficheiro. Para efectuar este calculo, foram percorridos e somados todos os *values* do objeto *result*, ou seja, foram somados os valores totais de todas as ocorrências das *keys* distintas.

```
for (var i = 0; i < Object.keys(result).length; ++i) {
    tot += parseInt(Object.values(result)[i]);
}
```

3.3.2 Total de caracteres apagados

Nesta métrica, é calculada o número total de caracteres apagados, ou seja, o número de ocorrências da tecla *Key.backspace*. Desta forma, foram percorridos todos as *Keys* do objeto *result*, de forma a ver se seu valor incluía *backspace*. Caso inclui-se, o *value* associado a essa *key* era adicionado a variável definida para o calculo em questão.

```
for (var i = 0; i < Object.keys(result).length; ++i) {
    if (Object.keys(result)[i].includes('backspace')) {
        back += parseInt(Object.values(result)[i]);
    }
}
```

3.3.3 Velocidade de digitação

Nesta métrica, é calculada a velocidade de digitação, ou seja, o número de teclas premidas por segundo. Para isto é feita a divisão do número total de teclas pressionadas pela diferença entre o *timestamp* da última tecla premida e o *timestamp* da primeira, em segundos.

```
var fim = new Date(t[t.length - 1]).getTime();
var ini = new Date(t[0]).getTime();

vel = tot / Math.abs(fim - ini);
```

3.3.4 Número de vezes que cada tecla foi premida

Nesta métrica é criada uma tabela que contém todas as teclas distintas contidas no ficheiro e o número de vezes que essas foram pressionadas. A tabela encontra-se ordenada por ordem decrescente em relação a esse número. Para conseguir criar esta tabela foi necessário dividir o objeto *result* em dois arrays de modo a ser possível ordena-los por ordem decrescente e inseri-los em diferentes colunas da tabela. Desta forma, criou-se o *array Sorted* que contem todos os valores das teclas ordenados (ex: [“k”, “o”, ...]) e o *array Sorted_val* que contém ordenadamente os números de vezes pressionada. Ou seja, se na posição 0 do *array Sorted* estiver o valor “r”e na posição 0 do *array Sorted_val* estiver 37, significa que a tecla “r”foi premida 37 vezes.

```
Sorted = Object.keys(result).sort(function (a, b) {
        return result[a] - result[b]
        })
Sorted_val = Object.values(result).sort(function (a, b) {
        return a - b
        })

/**Criar Tabela e Inserir cabeçalho**/
    const table = document.getElementById("Top10_table");
    var row_header = table.insertRow(0);
    row_header.insertCell(0).outerHTML = "<th>Keys</th>
              <th>Nr. de vezes pressionada</th>";
    var tablerow = 1;
```

```
/**Inserir valores na tabela por ordem**/
for (var i = 0; i < Object.keys(result).length; ++i) {
    var row = table.insertRow(tablerow);

    var x = row.insertCell(0);
    var y = row.insertCell(1);

    x.innerHTML = Sorted[i];
    y.innerHTML = Sorted_val[i];
}
```

3.3.5 Percentagem de grupo de teclas

Nesta métrica são calculadas as diferentes percentagens do número de teclas premidas de cada grupo, em relação ao número total de teclas. Os grupos são *left hand* (LH), *right hand* (RH) e *space and others* (S/Others). No fim os resultados são apresentados no *frontend* em forma de *pie chart*. Inicialmente definiu-se os representantes dos diferentes grupos:

```
lh.push("q", "w", "e", "r", "t", "a", "s", "d", "f", "g", "z",
"x", "c", "v", "b");
lh.push("Q", "W", "E", "R", "T", "A", "S", "D", "F", "G", "Z",
"X", "C", "V", "B");
lh.push("Key.tab", "Key.caps_lock", "Key.shift", "Key.ctrl_l",
"Key.cmd", "Key.alt_l");

rh.push("y", "u", "i", "o", "p", "h", "j", "k", "l", "ç", "n",
"m", ",", ".", "-");
rh.push("Y", "U", "I", "O", "P", "H", "J", "K", "L", "Ç", "N",
"M");
rh.push("Key.ctrl_l", "Key.alt_gr", "Key.menu", "Key.ctrl_r",
"Key.shift_r", "Key.enter", "º", "ª", "~", "^", "(", ")", "?");
```

De seguida percorreu-se o *array* contendo todas as teclas registadas e verificou-se a que grupo pertenciam, incrementado o valor da variável referente ao respetivo grupo.

```
var l_keys = 0;
var r_keys = 0;
var other_keys = 0;

    /**Calcular Nr de Keys pertencentes a cada grupo**/
```

```
        for (let i = 0; i < arr.length; i++) {
            if (lh.includes(arr[i])) {
                l_keys++;
            }
            else {
                if (rh.includes(arr[i])) {
                    r_keys++;
                }
                else {
                    other_keys++;
                }
            }
        }
```

Depois de calculados os valores totais de cada grupo, passamos para a configuração do gráfico em si. Em *type* é definido o tipo de gráfico pretendido, em *data.datasets* são indicadas as percentagens e as cores que cada fatia irá apresentar e em *data.labels* são definidas os nomes que irão corresponder a cada fatia. Ficamos assim então com:

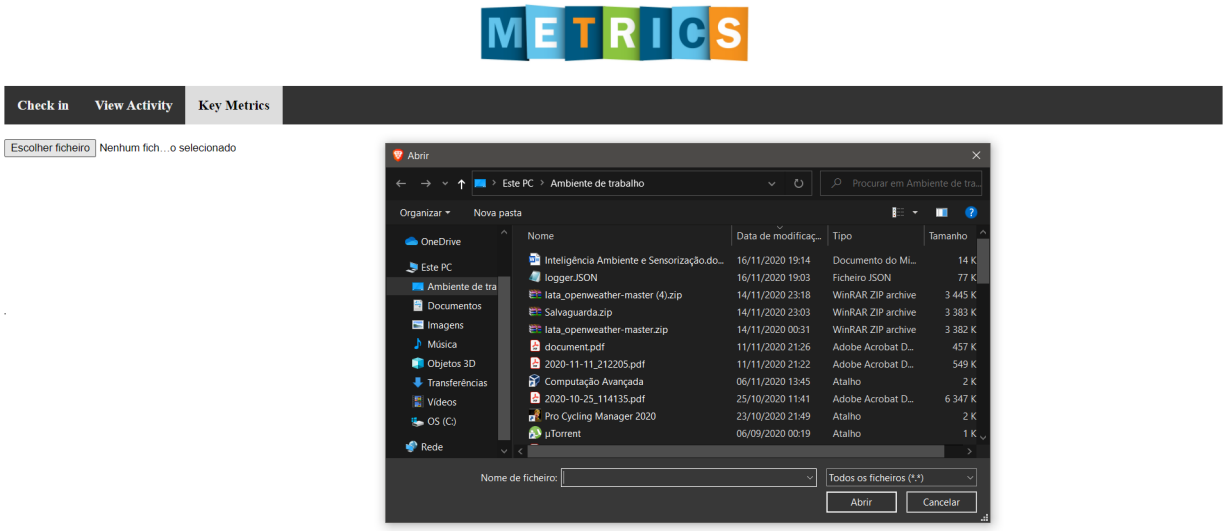
```
/**Configurar pie chart**/
    var config = {
        type: 'pie',
        data: {
            datasets: [{
                data: [
                    (l_keys / tot) * 100,
                    (r_keys / tot) * 100,
                    (other_keys / tot) * 100,

                ],
                backgroundColor: [
                    color(window.chartColors.red).alpha(0.9).rgbString(),
                    color(window.chartColors.orange).alpha(0.9).rgbString(),
                    color(window.chartColors.blue).alpha(0.9).rgbString(),
                ],
                label: 'Dataset 1'
            }],
            labels: [
                'LH',
```

```
                'RH',
                'S/Others'
            ]
        },
        options: {
            responsive: true,
            title: {
                display: true,
                text: 'Key Group Pressed Percentage'
            },
        }
    };

    var ctx = document.getElementById('pie_chart').getContext('2d');
    window.pie_chart = new Chart(ctx, config);
```

3.4 Sumário dos resultados obtidos



3.4 Sumário dos resultados obtidos

Escolher ficheiro logger.JSON

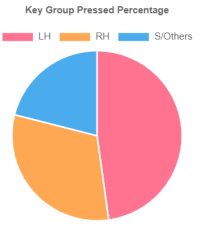
Total de teclas pressionadas: 1979

Total de caracteres apagados: 37

Velocidade (Nr. de Teclas/segundo): 3.0540123456790123

Top Teclas pressionadas:

Keys	Nr. de vezes pressionada
Key.space	272
e	194
o	173
a	149
s	139
i	104
r	101
t	89
n	85
d	79
m	55
Key.shift	50
u	47
c	45
l	38
Key.backspace	37
Key.left	33



4 Conclusão

Numa primeira fase, foram estudados tanto os enunciados do trabalho prático como também os *slides* disponibilizados na *blackboard* referentes ao trabalho efectuado nas aulas de modo a compreender melhor o que nos era proposto a fazer. De seguida, e depois de escolhido o enunciado, o grupo foi trocando ideias de modo a encontrar qual a melhor abordagem para a concepção da resolução do trabalho prático. Depois definida a abordagem, passou-se para a sua produção. O trabalho foi realizado em *javascript*, linguagem de programação com a qual o grupo ainda não tinha tido grande contacto, o que fez com que houvessem algumas dificuldades iniciais que acabaram por ser ultrapassadas à medida que se progredindo. Em suma, o grupo pensa que abordou tudo o que era pretendido no enunciado do trabalho prático e que o trabalho satisfaz os requisitos pretendidos, pretendendo mesmo assim no futuro acrescentar mais funcionalidades ao mesmo.