Sistema de gestão/monitorização de consumo de energia para o setor doméstico

Filipe de Queirós Campos
Mestrado Integrado em Engenharia Física
Departamento de Física e Astronomia
2014

Orientador
Paulo Marques, Professor Auxiliar, FCUP

Coorientador
Carlos Santos, Engenheiro, EWEN

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Resumo

O objetivo deste projeto era o desenvolvimento um sistema de gestão e monitorização de consumo energético para o setor doméstico. Pretendia-se que o utilizador conseguisse aceder aos dados autonomamente, e caso deseja-se, que estes fossem tratados nos servidores da empresa EWEN, empresa na qual a totalidade deste trabalho foi realizado. Este projeto visou também o desenvolvimento de um sistema baixo custo e por isso teve como principal equipamento um microcontrolador do tipo Arduino.

Neste relatório são descritos os objetivos globais do projeto, bem como todo o trabalho que foi desenvolvido e algumas ideias para um futuro melhoramento do protótipo.

Ao longo de 10 meses foi desenvolvido um protótipo capaz de suportar 6 sensores, fazer a recolha, organização e envio de dados de cada um deles para o servidor para tratamento destes. O protótipo pode também guardar os dados na sua memória interna de forma a estes serem apresentados numa plataforma Web.

Nesta plataforma Web poderemos configurar o modo de funcionamento do protótipo, escolhendo se queremos que envie os dados para o servidor ou se queremos guardá-los na memória, havendo também a possibilidade de estes modos funcionarem em conjunto. É possível também configurar parâmetros temporais, como a frequência de leitura dos sensores ou o período de tempo em que os dados são agregados para serem apresentados no gráfico.

O protótipo foi construído pensando na possibilidade de escolher diversos tipos de sensores, tendo disponíveis as opções mais encontradas no mercado, ou seja, sensores com sinal de saída de 0-5V, 0-10V, 0-20mA e 4-20mA. Devido a esta possibilidade, para além da monitorização de consumo energético, este protótipo conseguirá ler outras grandezas físicas, como temperatura e humidade por exemplo.

Summary

The goal of this project was to develop a management and monitoring system of energy consumption for the domestic sector. It was intended that the user could access the data autonomously and, if he wished, data could be treated in EWEN's servers, the company where all project was developed. This project also aimed to develop a low cost system and for that reason, the main equipment was an Arduino microcontroller.

This report describes the project main goals, all the work developed and also some ideas for future improvements to the final prototype.

Over 10 months it was developed a prototype able of supporting 6 sensors, collect, organize and send the data of each sensor to a server to process it. The prototype can also store the data in its internal memory in order to display it in a Web platform.

In this Web platform we can configure the operating mode of the prototype, choosing if we want to send data to a server, or to store it in its memory; there is also the possibility of these modes run together. It is also possible to configure temporal parameters such as the reading frequency of the sensors or the time period in which data is aggregated to be displayed on the graph.

The prototype was built with a possibility of choosing several types of sensors, having the most available options found in the market, i.e., sensors with output signal ranging from 0-5V, 0-10V, 0-20mA and 4-20mA. Because of this possibility, in addition to monitoring of energy consumption, this prototype is able to read other physical quantities such as temperature and humidity for example.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

AJAX – Asynchronous JavaScript and XML
BCD – Binary Coded Decimal
CPU – Central Processing Unit
DFA – Departamento de Física e Astronomia
EEPROM – Electrically Erasable Programmable Read-Only Memory
GND - Ground
HTML – HyperText Markup Language
IDE – Integrated Development Environment
I^2C – Inter-Integrated Circuit
I/O – Input/Output
IP – Internet Protocol
JSON – JavaScript Object Notation
NTP – Network Time Protocol
PCB – Printed Circuit Board
PoE – Power over Ethernet
PWM – Pulse-Width Modulation
RTC – Real Time Clock
SD – Secure Digital
SPI – Serial Peripheral Interface
SRAM – Static Random-Access Memory
TCP – Transmission Control Protocol
UART – Universal Asynchronous Receiver/Transmitter
UDP – User Datagram Protocol
USB – Universal Serial Bus
XML – Extensible Markup Language

1- Introdução

1.1- Objetivo geral

O objetivo deste projeto era a conceção, construção, montagem e o ensaio de um sistema de gestão e monitorização de consumos de energia para o sector doméstico, utilizando um *hardware* de baixo custo (um microcontrolador Arduino, Raspberry Pi ou equivalente). O sistema deveria enviar os dados dos consumos para o servidor da empresa EWEN, para serem tratados e disponibilizados para o cliente via Internet; alternativamente, o consumidor final poderia visualizar um gráfico do consumo, acedendo diretamente ao Arduino.

O projeto foi implementado com um Arduino porque a velocidade de arranque em comparação a um Raspberry é superior, devido ao Raspberry Pi possuir um sistema operativo. Isto era importante, pois caso houvesse uma falha de energia, pretendia-se que o sistema estivesse novamente operacional o mais rápido possível para recolha de dados. Desejava-se ainda que o utilizador pudesse aceder diretamente ao Arduino sem que estes dados fossem enviados para os servidores da empresa EWEN, estando este ligado por Ethernet e através da rede local fosse possível a visualização dos dados tratados (não seria necessário estar ligado ao exterior pois o dispositivo e o computador estariam ligados pela mesma rede local). Caso o cliente quisesse os dados tratados de forma mais detalhada, estes seriam enviados para os servidores da empresa e poderia visualizá-los através do computador (neste caso precisaria de acesso à internet).

Um dos objetivos era também criar algum mecanismo que permitisse ao cliente configurar o protótipo, incluindo por exemplo informações sobre os sensores, para que os valores enviados para os servidores da empresa EWEN fossem corretos.

Desejava-se que o protótipo final estivesse pronto a receber vários de sensores, para que o utilizador conseguisse ter uma alguma liberdade de escolha na compra destes. Para isso foi necessário a construção de *hardware* que permitisse a introdução de sensores com diferentes sinais de saída.

Na página seguinte estão listados os objetivos propostos para este projeto.

1.2- Objetivos propostos

Lista de objetivos:

- Definição das funcionalidades pretendidas para o protótipo
- Definição de entradas e saídas a monitorizar ou de controlo
- Definição do tipo de tratamento de dados
- Definição do diagrama de blocos
- Análise de *software/hardware* disponíveis no mercado
- Experimentação dos módulos necessários e transdutores
- Aprendizagem da linguagem de programação do *hardware*
- Definição da página/modo de interação para o cliente final
- Definição de uma caixa para o *hardware* em material atraente
- Estudo de custos de produção em massa
- Montagem do sistema completo
- Elaboração de toda a programação do sistema
- Ensaios e validação de valores lidos e processados
- Montagem e utilização do acesso via Web
- *Debugging*

1.3- Motivação

Nos últimos anos devido a acontecimentos como o aquecimento global, o pensamento ecológico tem começado a aumentar na população. As pessoas têm vindo a optar por eletrodomésticos mais eficientes de modo a poupar energia e assim reduzir o impacto negativo que estes têm para o ambiente.

A crise económica que o país atravessa também influencia as pessoas a tentar reduzir os seus gastos, muitas vezes não sabendo onde está o foco das despesas e com este projeto podemos aliar a necessidade ecológica e financeira das pessoas, ajudando estas a monitorizar os consumos energéticos das suas habitações.

Num tempo de crise, não se está disposto a abdicar de grandes quantias de dinheiro mesmo que o intuito seja para conseguir poupar, portanto é necessário criar um dispositivo de baixo custo.

Dum ponto de vista empresarial, tem que ser produzido um dispositivo o mais barato possível e ser feito uma análise de mercado para poder vender o produto de forma a tirar o maior lucro possível.

Fazendo uma breve pesquisa sobre o estado da arte no que toca a monitores de energia verifica-se que existe uma grande diversidade sendo alguns até muito baratos (encontra-se vários monitores entre 50€ a 70€), mas neste projeto o importante é tentar reproduzir as funções de monitorização num microcontrolador como o Arduino, para estudar se é possível moldar um monitor de consumo energético com as funcionalidades desejadas pela empresa.

1.4- Conceito geral do sistema de monitorização

Como este projeto está direcionado para o setor doméstico, por norma, teremos o nosso sistema de monitorização junto ao quadro de distribuição elétrica, aplicando um sensor de corrente ao cabo elétrico que liga os disjuntores do quadro aos vários circuitos da casa, e assim monitorizar o consumo energético referente ao disjuntor onde o sensor está aplicado (figura 1a).

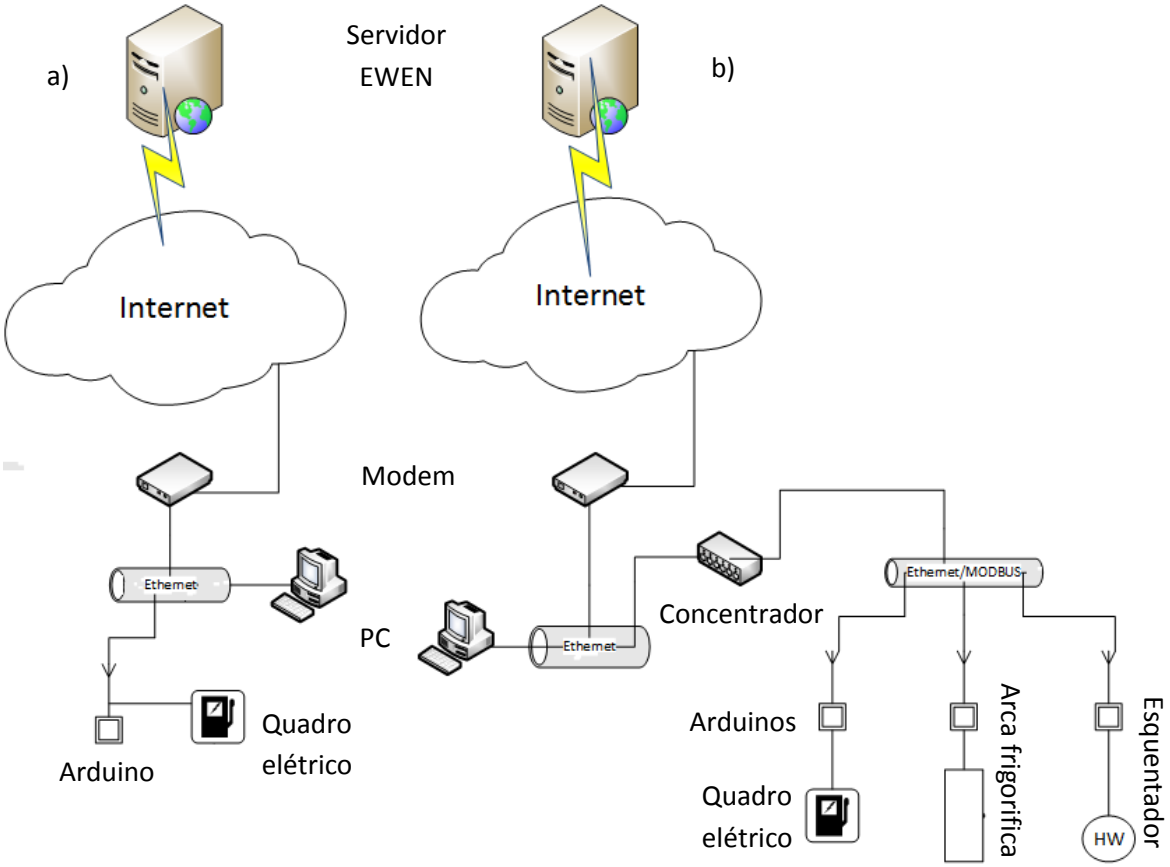

Fig. 1 – Esquema das opções pensadas inicialmente para o funcionamento do sistema monitorização energética.

Inicialmente ponderou-se também, caso o cliente quisesse, usar outros Arduinos para monitorizar aparelhos de maior consumo energético, como por exemplo arcas frigoríficas ou ar-condicionado de forma a ter dados mais concretos sobre o consumo destes aparelhos específicos. Neste caso um Arduino serviria de concentrador, isto é, receberia dados de outros Arduinos, por Ethernet ou usando um protocolo de

comunicação MODBUS (protocolo muito usado em automação industrial). A figura 1b mostra uma representação esquemática de um circuito de monitorização que utiliza vários Arduinos.

Devido ao número de sensores que o protótipo consegue ler, a configuração final do projeto foi desenhada para monitorizar o quadro elétrico e também outros aparelhos, utilizando apenas um Arduino. A figura 2 mostra uma representação esquemática do circuito de monitorização utilizado.

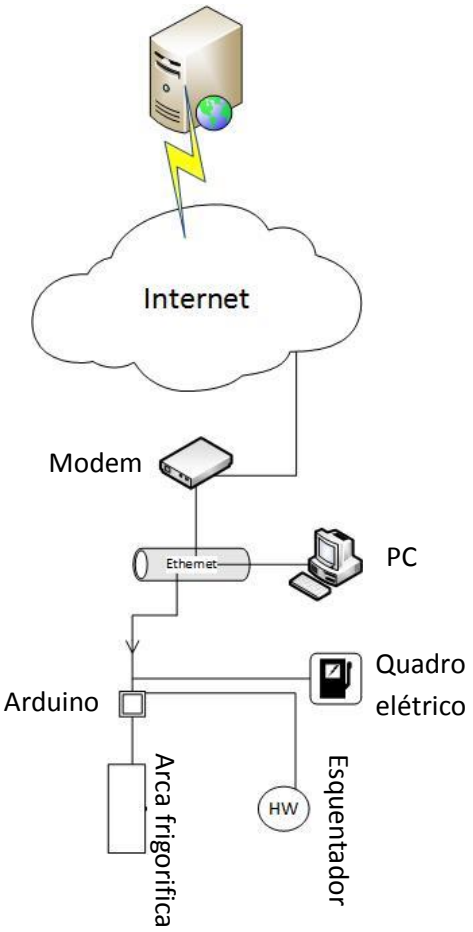

Fig. 2 – Esquema final para o funcionamento do sistema monitorização energética.

A opção de múltiplos Arduinos a enviar para um concentrador foi descartada, devido à reduzida memória *flash* do Arduino Uno, sendo que a adição de uma rotina de receção de dados iria ultrapassar a memória deste, mesmo retirando a rotina de leitura de sensores no código do concentrador. Como cada Arduino necessita de um endereço IP associado, se quisermos ter vários dispositivos em casa poderemos observar os dados de cada um separadamente e os custos para o cliente serão os mesmos.

O nosso sistema de monitorização atualmente dispõe de um Arduino Uno, um Ethernet Shield, um RTC Shield e um Sensor Shield emparelhados, sendo o último desenhado e fabricado no laboratório de eletrónica do Departamento de Física e Astronomia e os outros adquiridos. Foi também construída uma caixa para conter o protótipo com terminais de parafusos e um botão para reiniciar o dispositivo. Podemos ver a montagem final na figura 3.

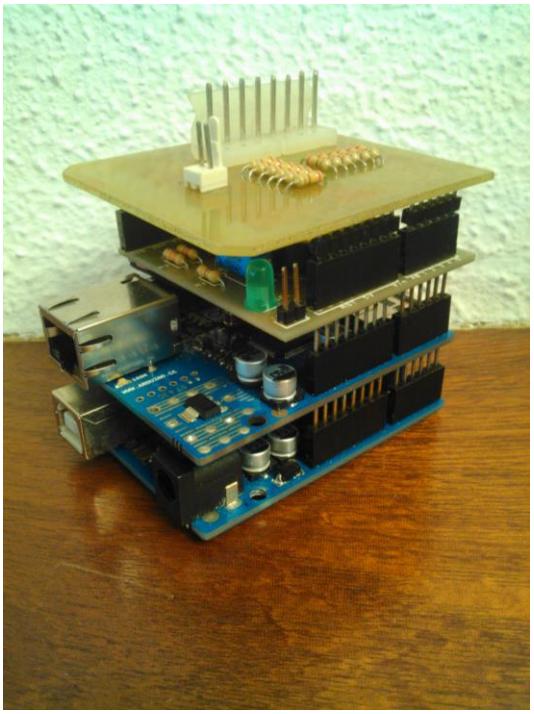

Fig. 3 – Montagem do sistema de monitorização.

1.5- Arduino

O Arduino é uma plataforma física de computação constituída por uma placa com um microcontrolador, e uma plataforma de escrita de *software* para a nossa placa.

O Arduino pode ser utilizado para desenvolver objetos interativos, suportando como entrada uma vasta gama de sensores ou interruptores, e outras saídas físicas para controlo de uma variedade de LEDs, motores, etc. Os projetos realizados em Arduino podem ter funcionamento autónomo, ou podem comunicar com outros tipos de *software* a correr no computador (como por exemplo, Flash, Processing, MaxMSP). As placas podem ser montadas manualmente pelo utilizador ou compradas pré-montadas, e o download do IDE (ambiente de programação) para o desenvolvimento de código é gratuito.

A linguagem de programação Arduino é baseada na linguagem usada pela Wiring nos seus microcontroladores (esta empresa nasceu primeiro que a Arduino criando os seus controladores). Já o IDE usado para programar o nosso Arduino é baseado no IDE Processing.

Existe muitos tipos de microcontroladores e plataformas disponíveis para este tipo de computação como o Basic Stamp Parallax, Netmedia do BX -24, Phidgets, Handyboard do MIT, e muitos outros que oferecem funcionalidades semelhantes. O Arduino possui um processo mais simples de trabalhar com microcontroladores, comparando com os referidos anteriormente, e oferece algumas vantagens como o preço (sendo relativamente mais baratos em comparação aos outros). Outra vantagem reside no facto de o *software* Arduino correr tanto em Windows, como no Macintosh OSX e nos sistemas Linux, enquanto a maioria dos sistemas dos outros microcontroladores só correm em Windows. O IDE do Arduino é muito fácil de usar para pessoas que estão a começar a programar, sendo também flexível para programadores avançados.

O Arduino é baseado nos microcontroladores ATmega8 e ATmega168 da Atmel, e os códigos de Arduino são disponibilizados em *open source*. Estes códigos podem ser melhorados através de bibliotecas de C++, ou para utilizadores que pretendam ir mais ao detalhe, podendo programar os códigos AVR-C diretamente no Arduino, que é uma linguagem de programação de baixo nível (código de máquina) que os microprocessadores entendem nativamente. Nesta linguagem trabalhamos diretamente

com código binário e é necessário compreender as características da arquitetura do microprocessador.

Existem placas que encaixam no Arduino que podem permitir que este tenha funções como conectar-se à internet via cabo Ethernet ou por Wi-Fi, se ligue a um relógio de tempo real ou até a um motor. Vamos falar de seguida de algumas destas placas (geralmente designadas por *shields*) que são usadas neste projeto.

1.6- Ethernet Shield para Arduino

O Ethernet Shield para o Arduino é uma placa que encaixa no nosso Arduino permitindo que este se consiga conectar à internet, utilizando para isso um chip *ethernet* Wiznet W5100 que é capaz de comunicar por TCP e por UDP (protocolos de comunicação). Este suporta até quatro *sockets* de conexão ao mesmo tempo. A placa Ethernet conecta-se com o Arduino mantendo o mesmo esquema de *pins*, permitindo empilhar outras placas. A conexão da placa é RJ-45 *standard* e na placa existe um *slot* para um cartão micro-SD, que pode ser utilizado para armazenar ficheiros.

Ao Ethernet Shield pode ser adicionado um componente para que este tenha PoE (Power over Ethernet) de forma a obter a energia necessária ao funcionamento a partir do cabo Ethernet. Este componente não é necessário para o decorrer do projeto.

O Arduino comunica com o chip W5100 e com o cartão SD, mas nunca simultaneamente, pelo que tem que se ter atenção a este facto na construção do código. Caso não se use um dos componentes o mesmo deverá ser desativado no código.

1.7- Real Time Clock Shield para Arduino

Esta placa para Arduino permite-nos ter um relógio preciso, o que evita a programação de um relógio, libertando memória do Arduino que seria gasta com a implementação deste, e assim aliviar o processador do Arduino. Esta placa é baseada no Maxim - Dallas DS1307, que é um contador integrado BCD (Binary Coded Decimal) de baixo consumo energético, e possui 56 bytes de SRAM não volátil. O nosso relógio de tempo real (RTC)

comunica com o Arduino através do barramento I^2C (barramento criado para conectar periféricos de baixa velocidade à placa mãe).

Um relógio de tempo real é basicamente um relógio que funciona com uma bateria e consegue manter o tempo, mesmo quando há uma falha de energia, o que é bastante importante para este projeto. Podemos reprogramar o nosso Arduino, desconectá-lo do computador através do USB ou até mesmo do cabo de alimentação, que o RTC mantém a sua contagem e assim podemos obter sempre uma hora precisa. O chip RTC é um chip especializado para manter o controlo do tempo.

1.8- Sensor Shield para Arduino

Com a necessidade de criar uma interface entre os sensores e o Arduino, foi desenhada uma placa que pudesse receber sensores com sinal de saída de corrente ou de tensão simultaneamente. Os conversores analógico-digitais do Arduino lêm valores de tensão entre 0-5V, portanto quando usamos sensores com sinal de saída de 4-20mA ou 0-20mA, teremos que ler a queda de potencial entre uma resistência de 250Ω. Como existem no mercado muitos sensores que o sinal de saída é de 0-10V, é necessário um divisor de tensão com resistências iguais para a saída se converter num sinal de 0-5V. Tendo um divisor de tensão com duas resistências de 250Ω satisfaz-se os 3 tipos de sensores referidos anteriormente (figura 4). Ao usar sensores com sinal de saída de 0-5V estes são convertidos num sinal de 0-2.5V, que reduz para metade a resolução.

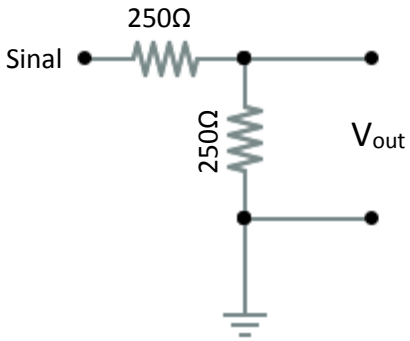

Fig. 4 – Divisor de tensão necessário para leitura dos sensores disponíveis.

Foi decidido que os sensores não seriam inseridos diretamente nesta placa, de forma a não forçar toda a estrutura (Arduino e respetivos *shields*) e por isso criou-se uma pequena placa que estaria presa na própria caixa, fazendo a ligação ao Sensor Shield através de terminais *molex*. O uso destes terminais facilita a montagem e desmontagem do protótipo pois são fáceis de colocar e remover.

Desejava-se também que o protótipo tivesse um botão *reset* e por isso nesta placa teria também que ter uma ligação a um botão que está na caixa utilizando também terminais *molex* para a ligação.

Depois de projetada a placa, foi utilizado o programa Eagle para desenhá-la, podendo observar na figura 5 o desenho final desta. Apresentamos também a placa que estará presa na caixa na figura 6.

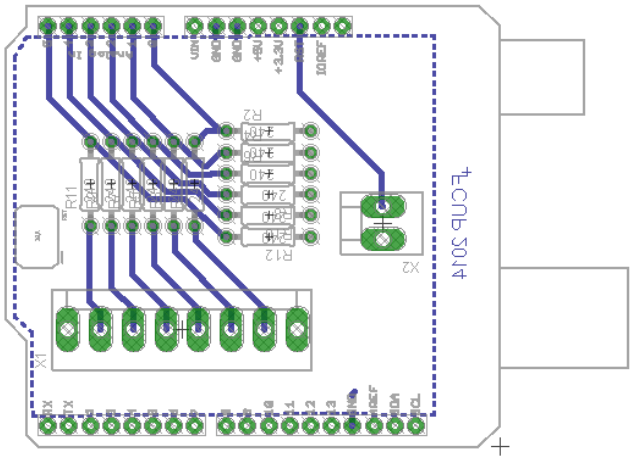

Fig. 5 – Desenho do Sensor Shield.

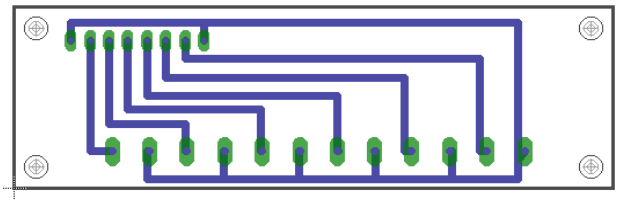

Fig. 6 – Desenho da placa de interface entre os sensores e Sensor Shield.

Foram também projetadas e desenhadas as placas para o projeto do aluno Tiago Costa desenvolvido na empresa EWEN mas direcionado para o sector industrial. As imagens destas estão apresentas no anexo 1.

1.9- Produção de PCB

Para a produção dos PCBs foi usado um processo de fotolitografia. Como as placas usadas já vêm com o material fotossensível, este processo tem como passos a exposição a radiação, remoção do material fotossensível exposto, seguida da remoção do cobre da zona exposta a radiação (*wet etching*) e no final a remoção do material fotossensível restante.

Para a exposição à radiação UV é preciso ter uma caixa de exposição como a apresentada na figura 7. Usando uma máscara podemos aplicar radiação UV para atacar o material fotossensível que queremos remover. Esta exposição a radiação causa mudanças químicas no material, tornando-o solúvel ao reagente que é usado na remoção deste.

Fig. 7 – Caixa de exposição de raios UV.

No processo de remoção do material fotossensível na produção de PCBs, depois da exposição UV foi usado um reagente para remover o material exposto. O reagente utilizado foi a soda cáustica (também conhecido por hidróxido de sódio) e o processo consistiu em mergulhar a placa num recipiente com o reagente e agitando esta, observamos o material exposto a deixar a placa.

No processo de *wet etching* foi usado como reagente químico o percloreto de ferro para remover o cobre das zonas expostas à radiação UV. Pode-se observar a montagem na figura 8, em que foi utilizada uma resistência de aquário para aquecer (~33ºC) o reagente e assim acelerar o processo. Existe uma entrada de ar para regenerar a solução, não ficando esta estática.

Fig. 8 – Montagem para processo de *wet etching*.

No final, depois de já se ter removido o cobre nas zonas expostas à radiação UV, usa-se palha-de-aço para remover o material fotossensível não-exposto, obtendo assim as pistas eletrónicas em cobre.

2- Desenvolvimento do Projeto

2.1- Comparação dos principais Arduinos

Um dos objetivos era a comparação dos principais Arduinos, de modo a ver qual o modelo mais adequado à implementação do projeto. Podemos observar na tabela 1 as características principais dos Arduinos mais comuns.

Podemos ver que para o caso de precisarmos de ler vários sensores o ideal será o Mega 2560, mas o seu preço ronda o dobro do Uno e Leonardo, que são os dois Arduinos com o preço de custo mais baixo, sendo que o Leonardo para além de ser ligeiramente mais barato tem a possibilidade de ler mais sensores (para além das 6 portas analógicas comuns ao Uno, o Leonardo consegue usar algumas das portas definidas como digitais como portas analógicas duplicando o número de portas do Arduino Uno).

O projeto começou a ser trabalhado com um Arduino Leonardo mas este foi abandonado devido a problemas com a comunicação com a porta série, que depois de uma pesquisa em fóruns verificou-se ser um problema comum neste Arduino, principalmente quando se tentava aceder ao *slot* SD.

O Arduino Leonardo é relativamente recente, pelo que se torna mais difícil encontrar soluções para os problemas encontrados, e os erros ainda não estão todos corrigidos até ao momento, ao contrário do Arduino Uno que está mais isento destes problemas visto que já vai na terceira versão.

Depois de ver que estes problemas eram sistemáticos e sendo incapaz de os resolver, foi testado um Arduino Uno e verificou-se que os problemas anteriores foram eliminados.

Após análise chegou-se à conclusão que para um projeto desta natureza, o Arduino Uno é o que tem uma melhor relação necessidade/preço/funcionalidade tendo em conta que este projeto é direcionado para o setor doméstico e que não haverá necessidade de se fazer a leitura de um grande número de sensores.

		Arduinos			
		Uno	Due	Leonardo	Mega 2560
Características	Processador	ATmega328	AT91SAM3X8E	ATmega32u4	ATmega2560
	Tensão de funcionamento/ Tensão de Entrada	5 V/ 7-12 V	3.3 V/ 7-12 V	5 V/ 7-12 V	5 V/ 7-12 V
	Velocidade CPU	16 MHz	84 MHz	16 MHz	16 MHz
	I/O Analógicas	6/0	12/2	12/0	16/0
	IO/PWM Digitais	14/6	54/12	20/7	54/15
	EEPROM [Kb]	1	-	1	4
	SRAM [Kb]	2	96	2.5	8
	Flash [Kb]	32	512	32	256
	USB	Regular	2 Micro	Micro	Regular
	UART	1	4	1	4
	Preço (Loja Arduino)	€20.00	€39.00	€18.00	€39.00

Tabela 1 – Comparação dos principais Arduinos vendidos atualmente.

2.2- Produto final da programação

O código desenvolvido é capaz de fazer a leitura de 6 portas analógicas do Arduino, todas elas testadas com sensores reais. As várias leituras são guardadas em variáveis para posteriormente serem gravadas as médias destas leituras, juntamente com o registo do instante temporal em que a escrita no ficheiro é efectuada. Os dados são guardados em ficheiros que vêm no formato “S/YYMMDD.CSV”, sendo “S/” uma pasta em que “S” é o sensor, “YY” são os dois últimos dígitos do ano, “MM” corresponde ao mês e “DD” corresponde ao dia (limitado a um ficheiro por dia). No fim temos “.CSV” que corresponde à extensão relativa ao formato do ficheiro que vai ser gravado na memória SD.

Enquanto o nosso microcontrolador não está a efetuar medidas, este funcionará como um *Webserver*, permitindo aceder pela internet através do IP atribuído ao Arduino, aos ficheiros alojados no cartão SD. Os ficheiros são listados numa página Web, sendo estes agregados por sensor e quando um dos ficheiros é selecionado podemos observar o gráfico respetivo.

Através do servidor Web pode-se alterar o período de tempo em que os dados são agregados num ficheiro, sendo que a data correspondente ao nome do ficheiro é a do primeiro registo. Existe a possibilidade de serem agregados diariamente, semanalmente ou mensalmente. Outro parâmetro temporal que se pode configurar é a frequência de leitura de dados, sendo que o seu máximo esta limitado a uma leitura por segundo.

Outras alterações que se podem fazer é a nível da escolha de sensores, podendo escolher o tipo de sinal de saída do sensor, tendo como opção os mais comuns do mercado. Cada tipo de sensor tem uma função de calibração associada, tendo como variáveis o máximo e mínimo da grandeza lida, para uma conversão correta dos valores quando o gráfico é apresentado.

Tipo de sinal de saída	**Fórmula de calibração**
0-5 V	$y = \frac{2(max - min) * x}{1023} + min$
0-10 V	$y = \frac{(max - min) * x}{1023} + min$
0-20 mA	$y = \frac{(max - min) * x}{1023} + min$
4-20 mA	$y = \frac{5}{4}(max - min)\left(\frac{x}{1023} - 1\right) + max$

Tabela 2 – Fórmulas de calibração.

O protótipo está preparado também para enviar os dados para um servidor na empresa EWEN, portanto existe a capacidade de configurar o modo de operação do dispositivo, tendo como opções, poder apenas enviar dados para o servidor da empresa para estes serem tratados, ou por outro lado ser utilizado como servidor local, guardando os dados para que o utilizador possa visualizar gráficos dos seus consumos. No final do projeto foi adicionada também a capacidade de funcionar de ambas as formas.

Por fim é possível também configurar os parâmetros de *network*, como o endereço MAC e IP como também o Gateway e DNS para garantirmos o melhor funcionamento do nosso servidor local, como também um melhor o envio de dados para o servidor EWEN.

2.3- Desenvolvimento do código

Nesta secção é exposto o desenvolvimento do código ao longo de todo o projeto, numa ordem cronológica.

Os primeiros passos deste projeto tiveram como base um projeto chamado Super Graphing Data Logger [6], no qual era adquirido o sinal de um sensor de luz, os dados eram guardados no cartão SD e observados na internet. Para seguir a ideia deste projeto, este código teria que ser bastante otimizado, visto que o programa ocupava completamente a memória impedindo a leitura de múltiplos sensores.

Inicialmente foi reformulada a rotina de criação de ficheiros de forma a conseguir poupar memória para conseguir aumentar o número de sensores. O passo seguinte foi alterar o modo de listagem dos ficheiros no *Website*, fazendo todas as linhas de escrita de dados no formato *string* (cadeia de caracteres), guardadas agora na memória *flash* de modo a impedir o transbordamento de dados (também conhecido como *overflow*) da memória SRAM, através da função “F()” [7]. De seguida, houve uma mudança em relação ao código que serviu de base para este projeto, que consistiu na obtenção do tempo através do nosso *shield* RTC em vez da utilização de um servidor NTP.

Posteriormente, o foco do trabalho foi criar uma página Web, em que os gráficos fossem criados a partir dos dados no nosso cartão SD. Este não poderia ser baseado no projeto encontrado, e referido anteriormente, pois a plataforma usada por este, highcharts.js, não podia ser usada para fins comerciais. A plataforma usada foi a Google Charts [11] que é uma plataforma gratuita.

De modo a aprender a trabalhar com linguagens de programação Web foram efetuados alguns tutoriais em HTML e JavaScript [8], permitindo criar várias páginas de forma a termos uma plataforma Web organizada, sendo que as primeiras páginas foram as de configuração dos sensores. Para isto foram construídas rotinas de envio de dados para o *browser* usando XML, para atualizar a página através da técnica AJAX, permitindo que tenhamos os dados atualizado no momento prévio à configuração.

Nesta altura do projeto, o transbordamento de dados na SRAM era muito comum e efetuou-se uma pesquisa para conseguir reduzir esta, e assim, houve uma passagem do uso da biblioteca “SdFat.h” em detrimento da “SD.h”, esta última mais recente e fornecida pela empresa Arduino, sendo esta baseada na anterior [9]. A mudança de biblioteca trouxe alguns problemas iniciais pois sendo um pouco antiga, os códigos

exemplos não funcionavam na nova IDE e desta forma alguns dias foram passados a desvendar como colocar esta nova biblioteca a funcionar com a anterior. No final, para além de se resolver os problemas da SRAM, também houve uma grande redução da memória *flash* (cerca de 5Kb) o que levou à possibilidade de usar as 6 portas analógicas pela primeira vez.

Foi desenvolvida de seguida uma rotina "Setinputs" que, dependendo dos dados que forem configurados na plataforma Web, permitia alterar os parâmetros correspondentes. Esta rotina foi aumentando conforme as várias ideias de configurações foram aparecendo ao longo do desenvolvimento do projeto, estando este aumento sempre condicionado pela memória *flash*.

Depois de ter um servidor que já lia sensores, gravava os dados e os apresentava na plataforma Web, chegou a altura de enviar os dados para o servidor da empresa EWEN e para isso foram aprendidas algumas noções de node.js e SQL de forma a saber como estruturar os dados e como estes seriam recebidos.

Com o envio de dados para o servidor, iniciou-se a tentativa de conseguir colocar os dois modos de operação em conjunto. Para isso foi preciso uma grande otimização do código pois este tinha a memória *flash* totalmente ocupada. Para isso foram seguidas várias instruções dum documento da Atmel [10] e depois de otimizar tudo o que era necessário foi possível o funcionamento em ambos os modos.

A última fase foi a edição da plataforma Web, e para isso foram estudados tutoriais em CSS de forma a conseguir dar o aspeto pretendido plataforma. Foram usadas animações do Google Charts para evitar monotonia da observação, assim como todas as configurações foram trabalhadas de forma a serem interativas. Os dados de cada sensor são guardados em pastas diferentes o que leva a uma apresentação mais organizada no *browser*.

As últimas alterações do código foram efetuadas depois de ser construído o *shield* para os sensores. Isto aconteceu devido a uma má interpretação da *datasheet* do Arduino que dava a entender que podíamos escolher entre os conjuntos de portas SCL/SDA ou A4/A5, para a utilização do relógio e afinal estes conjuntos de portas estavam ligados internamente, impedindo a leitura de duas portas analógicas, reduzindo o projeto a 4 sensores. Estas não davam para ser usadas alternadamente pois o nosso Sensor Shield ligava estas portas ao GND e sempre que o Arduino tentava conectar-se ao nosso RTC todo o dispositivo bloqueava.

Apesar de se indicar para deixar o projeto com a possibilidade de leitura de apenas 4 sensores, tentou-se reprogramar a comunicação I^2C de forma a comunicar com o nosso relógio através de outras portas. Com a ajuda da nova biblioteca "DigitalIO.h", criaram-se rotinas de comunicação com o relógio através de duas portas digitais, conseguindo assim voltar a ter a possibilidade de leitura de todas as portas analógicas disponíveis.

Assim foi terminada toda a programação do protótipo, sendo a memória *flash* ocupada com 31104 bytes num total de 32256 bytes disponíveis.

O RTC foi configurado através de um programa à parte na biblioteca "Time". Foi também criado um código separado que configura a EEPROM inicialmente. Foi decidido criá-lo separadamente pois seria desnecessário ocupar memória de código em algo que apenas é utilizado uma vez.

Este é a única parte do projeto que ainda é semelhante ao SGDL, o projeto que ajudou a dar os primeiros passos, sendo que todo o código que faz correr o protótipo foi totalmente reformulado, não apresentando qualquer semelhança.

2.4- Fabricação de placas

Para a interface entre o Arduino e os sensores foi necessário construir uma placa que estivesse preparada para receber os vários sensores que o protótipo consegue receber. Depois de desenhada, como foi mostrado na introdução, passou-se para o processo de fabricação.

O primeiro passo foi imprimir o circuito num acetato para criar uma máscara para realizar fotolitografia. Depois colocou-se a máscara sobre a placa que tinha um material fotossensível e aplicou-se radiação UV durante 110 segundos.

Após a exposição, limpou-se o material fotossensível exposto com soda cáustica para de seguida fazer a remoção química do cobre nas zonas que foram expostas a radiação (*etching*). Para essa remoção foi usado percloreto de ferro durante aproximadamente 7 minutos (é necessário verificar a placa durante o processo até que todo o material desejado seja removido). Depois do *etching* removeu-se o material fotossensível não exposto com a ajuda de palha-de-aço.

Seguiu-se o corte da placa com a ajuda de uma guilhotina e usando uma lima moldou-se o PCB para este ficar com a forma do Arduino. De seguida efetuaram-se os furos na placa para podermos soldar os componentes.

Os componentes usados no *shield* foram 12 resistências de 240Ω, 1 terminal *molex* macho com 2 *pins* e 2 terminais *molex* machos com 4 *pins*, soldaram-se também os *pins* para encaixe no Arduino, 2 conjuntos de 6 *pins* e 2 conjuntos de 8. Podemos observar o produto final na figura 9.

Quando se usa resistências de 240Ω para converter um sensor com sinal de saída de corrente (por exemplo 0-20mA) este sinal transforma-se num sinal de 0-4.8V em vez de do desejado sinal de 0-5V, por isso a conversão não é tão boa. No final do relatório serão dadas ideias para melhorar a conversão deste sinal.

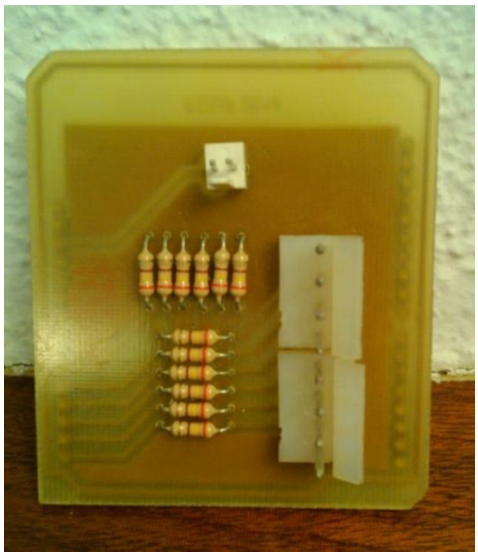

Fig. 9 – Sensor Shield.

A fabricação da placa inserida na caixa passou pelos mesmo processos descritos anteriormente tendo como componentes um *molex* macho de 8 *pins* e 12 terminais de parafusos para receber os sensores. Podemos observar esta placa na figura 10. Foram também produzidas as placas representadas no anexo 1 para o projeto do aluno Tiago Costa.

Fig. 10 – Placa exterior com terminais de parafuso.

Finalmente descrevemos as alterações efetuadas ao RTC Shield para que este possa ser usado e conseguirmos tirar proveito dos seis sensores. Foi ligado o pin SCL à porta digital 3 e o pin SDA à porta digital 2. Os *pins* SCL e SDA que ligariam ao Ethernet Shield foram dobrados e isolados com uma manga termoretrátil (figura 11).

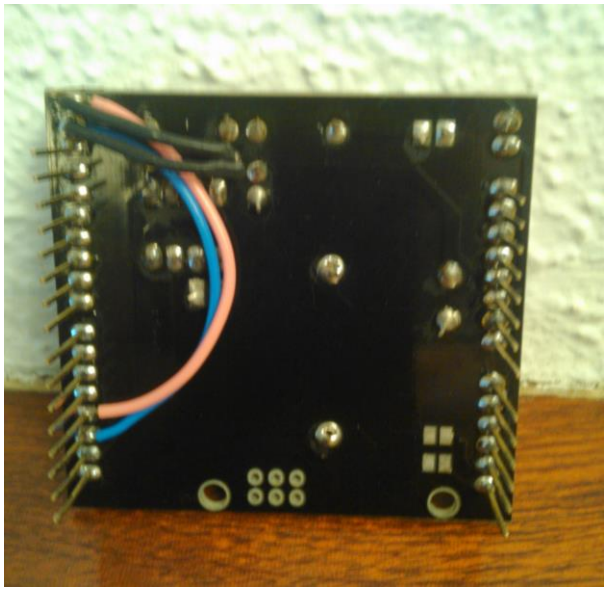

Fig. 11 – Alteração ao RTC Shield.

2.6- Encapsulamento

Para o encapsulamento do protótipo usou-se uma caixa disponível no DFA devido ao custo excessivo que seria caso fosse moldada uma caixa à medida. Foram criadas aberturas para podermos ter acesso à entrada USB, porta Ethernet e para a fonte de alimentação. Essas aberturas efetuadas com ajuda de brocas e moldadas com limas.

Também foram realizados furos para colocar parafusos para o suporte do dispositivo dentro da caixa, outros para prender a placa exterior com os terminais de parafusos e por último o furo para o botão de *reset*. Podemos observar a caixa na figura 12.

Fig. 12 – Caixa final.

2.7- Apresentação dos dados

Nesta secção vamos comentar como são representados os dados tanto na plataforma Web como os recebidos pelo servidor.

Na plataforma Web o utilizador pode observar um gráfico dos dados guardados no cartão SD. No eixo dos *yy* não temos indicação da grandeza medida nem das respetivas unidades devido ao grande leque de grandezas que o dispositivo pode ler; assume-se que o utilizador conheça a grandeza que está a medir, assim como as suas unidades, pois foi o utilizador que comprou e instalou o sensor. Para o eixo dos *xx*, as unidades não estão representadas porque estas variam com a quantidade de dados, por exemplo na figura 13 que corresponde à leitura durante uma semana aparecem datas, na figura 37 já aparecem horas. O número do sensor apresentado no título do gráfico corresponde ao número representado na caixa como podemos ver na figura 10 apresentada anteriormente.

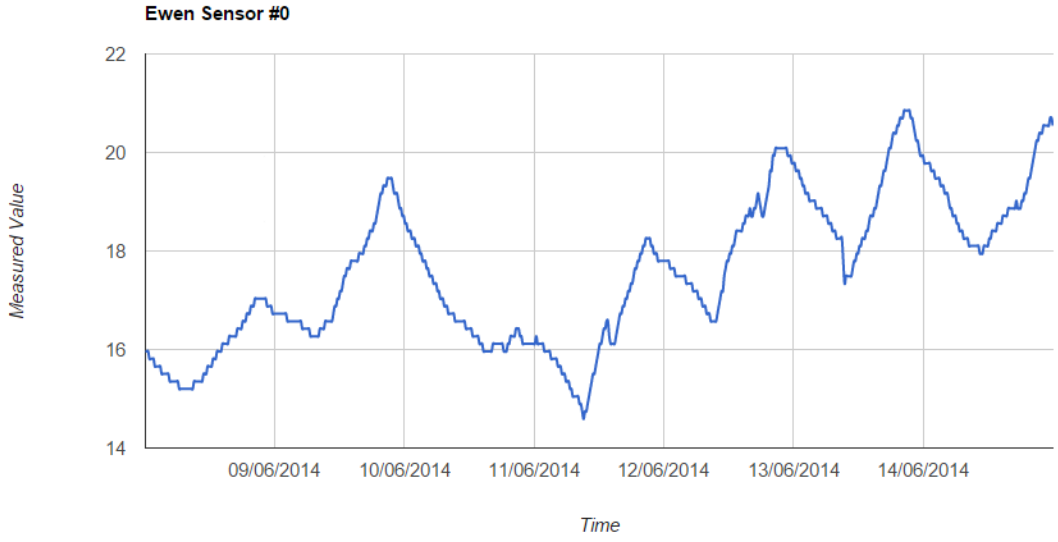

Fig. 13 – Leitura de um sensor de temperatura durante uma semana.

Quando enviamos os dados para o servidor, este tem um programa produzido pelo aluno Tiago Costa em node.js que descodifica a mensagem enviada e introduz os dados numa base de dados MySQL [13]. Observamos na figura 14, alguns dados de um sensor de temperatura.

id	origin	uniq_ref	channel	utc_dt	value
156	192.168.0.10	Arduino55330333630351F0200	1	1408114800	21.12
157	192.168.0.10	Arduino55330333630351F0200	1	1408115700	21.01
158	192.168.0.10	Arduino55330333630351F0200	1	1408116600	21.06
159	192.168.0.10	Arduino55330333630351F0200	1	1408117500	21.05
160	192.168.0.10	Arduino55330333630351F0200	1	1408118400	21.08

Fig. 14 – Dados recebidos num servidor MySQL.

2.8- Análise do código principal

O código principal faz uso da chamada regular de módulos ou sub-rotinas que serão analisadas nas próximas páginas. Este código poderá ser observado na integral no anexo 2.

2.8.1- Módulos

Para ativar algumas funcionalidades é necessário importar bibliotecas que têm utilidade ao longo do código. Podemos observar essa importação na figura 15.

Para trabalhar com o nosso cartão SD, utiliza-se o módulo “SdFat.h”, embora durante grande parte do projeto se tenha usado a “SD.h”. Quando o programa começou a ter problemas de *overflow* da memória SRAM este módulo foi substituído em prol da otimização do código do protótipo. Quando a mudança foi feita estes problemas desapareceram e também conseguimos poupar mais de 5kB de memória *flash* nestas alterações.

Depois chamam-se as bibliotecas “Ethernet.h” e “SPI.h”. A primeira serve para toda a comunicação via Ethernet por TCP, sendo que toda a parte do código do servidor Web

tem como suporte este módulo. A segunda biblioteca é necessária para a comunicação com o Ethernet Shield via SPI permitindo usar o cartão SD e o controlador Ethernet.

```
#include <SdFat.h>
#include <Ethernet.h>
#include <SPI.h>
#include <EEPROM.h>
#include <EEPROMAnything.h>
#include <string.h>
#include <Time.h>
#include <DigitalIO.h>
```

Fig. 15 – Módulos importados do código principal.

Os módulos seguintes são “EEPROM.h” que permite a escrita e leitura de bytes na EEPROM enquanto o módulo “EEPROMAnything.h” complementa o anterior para que seja possível gravar qualquer tipo de informação, o que é bastante útil pois queremos gravar diferentes tipos de dados. A razão para usar esta memória é para caso haja uma falha de energia, o dispositivo reinicie com os parâmetros atualizados e não os predefinidos. Os dados que são guardados serão indicados mais à frente na secção 2.8.2.

O módulo “string.h” contém funções que ajudam a construir *strings*, assim como a conversão de outro tipo de dados em *string* e vice-versa. O módulo “Time.h” é utilizado para todas as funções relacionadas com o tempo, sendo que a principal função é a obtenção da data atual no formato *unix timestamp*.

O último módulo, “DigitalIO.h” é utilizado para reformular a comunicação I^2C para o RTC. Antes era usado o módulo “Wire.h” que é distribuído pela empresa Arduino, mas como o *hardware* usa as portas analógicas 4 e 5 para esta comunicação, perder-se-iam duas portas para leitura de sensores usando o nosso relógio. Com este módulo conseguimos reprogramar o Arduino de forma a fazer a comunicação I^2C através de duas portas digitais.

2.8.2- Inicialização de parâmetros

Uma parte importante é a inicialização de variáveis usadas no código (figura 16), começando pelo conjunto 1 que define as portas usadas pela comunicação I^2C e inicializa o endereço de comunicação. No conjunto 2 temos as variáveis para o uso do cartão SD, inicializando o cartão, as partições do cartão (volume) e inicializando variáveis que são do tipo pasta ou ficheiro. No pequeno conjunto 3 definimos o IP do servidor da empresa EWEN e a linha seguinte serve para indicar a porta em que vamos receber dados quando o nosso *hardware* funcionar como um servidor.

```
#define DS1307ADDR 0XD0
const uint8_t SDA_PIN = 2;                    1
const uint8_t SCL_PIN = 3;
FastI2cMaster<SCL_PIN, SDA_PIN> rtc;

Sd2Card card;
SdVolume volume;
SdFile root;                                  2
SdFile file;
SdFile workingdir;

IPAddress serverIP(37,187,134,39);
EthernetServer server(80);                    3

unsigned long lastIntervalTime;
unsigned long lastIntervalMes;
unsigned long timetosend;
unsigned long sum[6]={0L,0L,0L,0L,0L,0L};
unsigned long sumenv[6]={0L,0L,0L,0L,0L,0L};
unsigned int count;
unsigned long countenv;
uint8_t ns;
char* networks[6];                                    4
```

Fig. 16 – Parâmetros inicializados no código principal.

No conjunto 4 inicializamos várias variáveis, começando pela data em que ocorreu a última escrita em ficheiros, seguida da data em que foi feita a última leitura dos sensores e depois a data em que enviamos pela última vez dados para o servidor. Depois guardamos dois conjuntos de 6 valores que correspondem aos 6 sensores, em que se efetuam somas, para posteriormente serem usadas para fazer médias quando se

escrever os dados no ficheiro ou quando se enviar os dados para o servidor. São necessários dois conjuntos pois poderemos querer uma frequência de escrita nos ficheiros diferente da frequência de envio de dados. Temos também dois contadores, que serão incrementados sempre que os sensores forem lidos, e depois estes valores serão utilizados para fazer as médias.

A próxima variável é para ser usada quando estamos na plataforma Web, e o valor desta variável muda dependendo do sensor que estamos a consultar ou configurar. Por último temos um conjunto de variáveis que servem de apoio para quando estivermos a configurar parâmetros relacionados com a rede.

O último passo é criar uma estrutura para ler e escrever na memória EEPROM (figura 17). Nesta estrutura temos variáveis como a data de criação do último ficheiro e os caminhos do ficheiros que estão gravados no cartão SD para facilitar a escrita e leitura. Temos 4 conjuntos de 6 valores, para definirmos para cada sensor, o seu tipo de sinal de saída, máximo e mínimo para o cálculo da fórmula de calibração e se estes estão ativados ou não. Depois temos variáveis como o modo de operação que tem 3 opções, a primeira é apenas gravar os dados e apresentá-los em gráficos, a segunda apenas enviar os dados para o servidor da empresa, sendo a terceira o funcionamento de ambos os modos anteriores. Temos ainda o período do ficheiro que poderá ser diário, semanal ou mensal, e também a frequência de escrita no ficheiro.

```
typedef struct{
    unsigned long newFileTime;
    char workingFilename[6][14];
    uint8_t tipo[6];
    double maxi[6];
    double mini[6];
    uint8_t check[6];
    uint8_t modo;
    uint8_t per;
    int freq;
    boolean first;
    byte mac[6];
    byte ip[4];
    byte dnsgateway[4];
} configuration;
```

Fig. 17 – Estrutura de dados guardados na memória EEPROM.

Existe uma variável booleana chamada *first*, que é verdadeira quando usada pela primeira vez (permitindo a criação imediata de ficheiros), tornando-se falsa posteriormente. Esta variável também pode ser alterada em alguns casos de forma a criar ficheiros em certas condições que veremos mais à frente. Por último surgem os conjuntos de variáveis que definem o endereço MAC, IP do nosso Arduino e o DNS ou Gateway usado.

2.8.3- Sub-rotinas

Outro bloco importante diz respeito às funções utilizadas para o funcionamento do código começando pela "readDS1307". Esta função permite a transferência de dados entre o chip DS1307 do RTC e o Arduino (esta função foi baseada num exemplo da biblioteca "DigitalIO.h" com pequenas modificações). A função "bcd2dec" permite transformar dados do tipo BCD para decimal, o que vai ser necessário para definir o tempo usando a função "setTime" da biblioteca "Time.h" quando é chamada a função "readtime" (figura 18).

```
uint8_t readDS1307(uint8_t address, uint8_t *buf, uint8_t count){
  if (!rtc.transfer(DS1307ADDR | I2C_WRITE, &address, 1)){
      return false;
  }
  return rtc.transfer(DS1307ADDR | I2C_READ, buf, count);
}

uint8_t bcd2dec(uint8_t num){
  return ((num/16 * 10) + (num % 16));
}

time_t readtime(void){
  uint8_t r[8];
  readDS1307(0, r, 8);
  setTime(bcd2dec(r[2]),bcd2dec(r[1]),bcd2dec(r[0]),bcd2dec(r[4]), bcd2dec(r[5]), bcd2dec(r[6]));
  return now();
}
```

Fig. 18 – Funções readDS1207, bcd2dec e readtime.

A função "XML_response" (figura 19) estabelece uma comunicação XML com o *browser*, para quando o utilizador for configurar, ser informado dos estados atuais do sistema de monitorização e dos sensores. Através do *browser* são comunicadas informações de rede como, o IP, endereço MAC, DNS e Gateway; são também comunicados

parâmetros temporais como a frequência de escrita e o período do ficheiro, como também o modo atual de funcionamento e o número do sensor que estamos a configurar.

Indica-se também o tipo de sensor que se está a configurar, como também o seu máximo e o mínimo para cálculo da fórmula de calibração do gráfico. Por fim é comunicado o estado de todos os sensores para se saber quais os que estão ativos ou não.

```
void XML_response(EthernetClient cl){
    cl.print(F("<?xml version = \"1.0\" ?><inputs>"));
    for (uint8_t i=0; i<4; i++){
      cl.print(F("<ip>"));
      cl.print(config.ip[i]);
      cl.print(F("</ip>"));
    }
    for (uint8_t i=0; i<6; i++){
      cl.print(F("<mac>"));
      if (config.mac[i]<16){
        cl.print(F("0"));
      }
      cl.print(config.mac[i], HEX);
      cl.print(F("</mac>"));
    }
    for (uint8_t i=0; i<4; i++){
      cl.print(F("<dnsg>"));
      cl.print(config.dnsgateway[i]);
      cl.print(F("</dnsg>"));
    }
    cl.print(F("<mode>"));
    cl.print(config.modo);
    cl.print(F("</mode><per>"));
    cl.print(config.per);
    cl.print(F("</per><freq>"));
    cl.print(config.freq);
    cl.print(F("</freq><ns>"));
    cl.print(ns);
    cl.print(F("</ns><tipo>"));
    cl.print(config.tipo[ns]);
    cl.print(F("</tipo><max>"));
    cl.print(config.maxi[ns]);
    cl.print(F("</max><min>"));
    cl.print(config.mini[ns]);
    cl.print(F("</min>"));
    for (uint8_t i = 0; i < 6; i++) {
      cl.print(F("<check>"));
      cl.print(config.check[i]);
      cl.print(F("</check>"));
    }
    cl.print(F("</inputs>"));
}
```

Fig. 19 – Função XML_response.

A função “Setinputs” (figura 20) recebe como argumentos uma *string* com informação para ser decifrada e um algarismo que corresponde a uma tarefa que esta função vai desempenhar. Esta *string* com parâmetros estrategicamente construída é enviada pelo *browser*, para posteriormente se poder decompor, guardando todos os valores que estejam entre os símbolos “=” e “&”.

Quando se recebe o comando 0, nessa *string* estão as informações do tipo de sensor, máximo, mínimo e o estado de ativação de um sensor “ns”, que já está definido. Caso o comando seja o número 1, então este vai configurar o modo de operação do sistema de monitorização.

No comando 2, é feita a configuração temporal; quando há a mudança no período de agregação de dados num ficheiro, deve ser criado um novo, alterando a variável *first* guardada na memória EEPROM.

O comando 3 serve para apagar do cartão SD o ficheiro que corresponde ao nome enviado pela *string*, enquanto o comando 4 serve para definir o sensor para o qual vamos ver os gráficos ou configurar.

Por último temos o comando 5 que serve para configurar os comandos de rede. Este comando tem ajuda de uma outra função, chamada “network”, pois o IP, o endereço MAC e o DNS/Gateway são definidos por conjuntos de números, separados por “.”, servindo esta nova função para separar os dados e fazer as devidas alterações corretamente.

```
void Setinputs(char* clientlinel, uint8_t n)
{
  char* pch;
  char* z[4];
  uint8_t i=0;
  pch = strtok (clientlinel,"=&");
  while (pch != NULL)
  {
    if (i%2!=0){
      z[i/2]=pch;
    }
    pch = strtok (NULL,"=&");
    i++;
  }
  if (n==0){
    config.tipo[ns]=atoi(z[0]);
    config.maxi[ns]=atof(z[1]);
    config.mini[ns]=atof(z[2]);
    config.check[ns]=atoi(z[3]);
    EEPROM_writeAnything(0, config);
  }
  else if (n==1){
    config.modo=atoi(z[0]);
    EEPROM_writeAnything(0, config);
  }
  else if (n==2) {
    config.freq=atoi(z[1]);
    if (config.per!=atoi(z[0])){
      config.per=atoi(z[0]);
      config.first=true;
    }
    EEPROM_writeAnything(0, config);
  }
  else if (n==3){
    file.open(&root,z[0], O_RDWR | O_AT_END);
    file.remove();
  }
  else if (n==4) {
    ns=atoi(z[0]);
  }
  else {
    network(z[0]);
    for (uint8_t i=0; i<4; i++){
      config.ip[i]=atoi(networks[i]);
    }
    network(z[1]);
    for (uint8_t i=0; i<4; i++){
      config.dnsgateway[i]=atoi(networks[i]);
    }
    network(z[2]);
    for (uint8_t i=0; i<6; i++){
      config.mac[i]=atoi(networks[i]);
    }
    EEPROM_writeAnything(0, config);
  }

void network(char* clientlinel){
  char* pch;
  uint8_t i=0;
  pch = strtok (clientlinel,".");
  while (pch != NULL)
  {
    networks[i]=pch;
    pch = strtok (NULL,".");
    i++;
  }
}
```

Fig. 20 – Funções Setinputs e network.

Na próxima função “sendtoewen” (figura 21), os dados são mandados através de HTTP POST, já estruturados para que o servidor consiga lê-los. A estrutura de dados é no formato JSON, com a seguinte ordem: primeiro envia-se o “id” que é a palavra Arduino seguido do número de série (posteriormente será descrito como se consegue obter), depois “utc” que é o tempo em que os dados foram enviados no formato *unix timestamp*, em terceiro escreve-se uma máscara hexadecimal que permite ao servidor saber quais os sensores que estão ligados, e por fim os valores dos sensores. Os valores são colocados por ordem do sensor 0 para o 5 e através da máscara o servidor consegue identificar a correspondência valor-sensor. Ao enviar os dados é executada a fórmula de calibração que depende do tipo de sinal de saída do sensor e este valor é uma média de várias leituras realizadas à taxa de 1 Hz durante 15 minutos.

```
void sendtoewen(EthernetClient cliente, unsigned long rawTime){
  byte mask=0;
  for (uint8_t i=0; i < 6; i++){
    if (config.check[i]==1){
      mask+=(1<<i);
    }
  }
  cliente.println(F("POST /arduinotest.php HTTP/1.1"));
  cliente.print(F("Host: "));
  cliente.println(serverIP);
  cliente.println(F("Connection: close\r\n Content-Type: application/x-www-form-urlencoded"));
  cliente.println(F("Content-Length:150 \r\n"));
  cliente.print(F("id=Arduino55330333630351F0200&utc="));
  cliente.print(rawTime);
  cliente.print(F("&mask="));
  cliente.print(mask, HEX);
  cliente.print(F("&value="));
  for (uint8_t i=0; i < 6; i++){
    if (config.check[i]==1){
      if (config.tipo[i]==0){
        cliente.print((config.maxi[i]-config.mini[i])*(sumenv[i]/countenv)/512.0+config.mini[i]);
      }
      else if (config.tipo[i]==3){
        cliente.print((5/4.0)*(config.maxi[i]-config.mini[i])*((sumenv[i]/countenv)/1023.0-1)+config.maxi[i]);
      }
      else {
        cliente.print((config.maxi[i]-config.mini[i])*(sumenv[i]/countenv)/1023.0+config.mini[i]);
      }
      cliente.print(sumenv[i]/countenv);
      cliente.print(F(","));
    }
    sumenv[i]=0;
  }
  cliente.print(F("                                                    "));
  cliente.print(F("                                                    "));
  countenv=0;
}
```

Fig. 21 – Funções sendtoewen.

No fim são mandados espaços em branco pois em HTTP POST é necessário informar o número de caracteres que são enviados. Como este número é variável, impôs-se um limite máximo de 150 caracteres, que corresponde a ter os seis sensores a enviar valores de grande dimensão. Estes espaços em branco servem de compensação para casos em que os valores enviados não possuem muitos caracteres ou para leituras de poucos sensores.

A função “ListFiles” (figura 22) é criada para listar os ficheiros guardados no cartão SD. Esta é a única parte do projeto em que a escrita HTML foi feita a partir do Arduino sendo que o resto das páginas ficam guardadas no cartão. Isto acontece porque o cliente não pode aceder a ficheiros alojados no servidor e portanto tem que ser o servidor a escrever os caminhos do ficheiro no *browser*. No início do código é dada a informação do espaço livre e total do cartão SD e só depois é feita a listagem dos ficheiros do sensor que se estiver a consultar no momento. Também é colocada a possibilidade de fazer o *download* do ficheiro com os respetivos dados, assim como a possibilidade de remover o ficheiro se assim se desejar.

A função “createfile” (figura 23) tem o propósito de criar o caminho dos próximos ficheiros e no fim guardar esses caminhos na memória EEPROM. Os caminhos dos ficheiros vêm no formato “S/YYMMDD.CSV”, sendo “S/” uma pasta em que “S” é o sensor, “YY” são os dois últimos dígitos do ano, “MM” corresponde ao mês e “DD” corresponde ao dia. No fim temos “.CSV” que corresponde ao formato do ficheiro em que vai ser gravado na memória SD.

2.8.4- Inicialização

Em programação na linguagem Arduino, o Setup (figura 24) corresponde à secção do código que apenas será corrida uma vez. Nesta parte do código deve-se colocar todas as funcionalidades do Arduino que precisam de ser inicializadas.

Começamos por colocar a porta digital 10 como saída e atribui-se um valor de HIGH para assim desligar o chip W5100, pois tanto o chip como o cartão SD comunicam através de comunicação SPI pelo que não se devem iniciar simultaneamente.

```
void ListFiles(EthernetClient client) {
  char f[2];
  itoa(ns,f,10);
  client.print(F("<p align=center><span class=\"sub\">Free Memory:</span> "));
  client.print(volume.freeClusterCount()*volume.blocksPerCluster()/2097152.0);
  client.print(F(" GB of "));
  client.print(card.cardSize()/2097152.0);
  client.print(F(" GB</p><p><span class=\"sub\">Sensor #"));
  client.print(f);
  client.print(F("</span> (YYMMDD format)</p>"));
  dir_t p;
  workingdir.open(&root, f , O_READ);
  workingdir.rewind();
  client.println(F("<ul>"));
  while (workingdir.readDir(&p) > 0) {
    if (p.name[0] == DIR_NAME_FREE) break;
    if (p.name[0] == DIR_NAME_DELETED || p.name[0] == '.') continue;
    if (!DIR_IS_FILE_OR_SUBDIR(&p)) continue;
    client.print(F("<li><a href=\"/HC.htm?file="));
    client.println(f);
    client.print(F("/"));
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print((char)p.name[i]);
    }
    client.print(F("\">"));
    for (uint8_t i = 0; i < 6; i++) {
      client.print((char)p.name[i]);
    }
    client.print(F("</a><span class=\"tab\"></span><a href=\""));
    client.print(f);
    client.print(F("/"));
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print((char)p.name[i]);
    }
    client.println(F("\" download>download</a><span class=\"tab\"></span>"));
    client.print(F("<a href=javascript:removefile('"));
    client.print(f);
    client.print(F("/"));
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print((char)p.name[i]);
    }
    client.print(F("');>Remove File</a></li>"));
  }
  client.println(F("</ul>"));
  workingdir.close();
}
```

Fig. 22 – Função ListFiles.

```
void createfile(unsigned long rawTime){
  int dayInt = day(rawTime);
  int monthInt = month(rawTime);
  int yearInt = year(rawTime);
  char dayStr[3];
  char monthStr[3];
  char yearStr[5];
  char subYear[3];
  char f[2];
  itoa(yearInt,yearStr,10);
  itoa(monthInt,monthStr,10);
  itoa(dayInt,dayStr,10);
  memcpy( subYear, &yearStr[2], 3 );
  for (uint8_t i = 0; i < 6; i++) {
    char newFilename[13] = {""};
    itoa(i,f,10);
    strcat(newFilename,f);
    strcat(newFilename,"/");
    strcat(newFilename,subYear);
    if (monthInt < 10){
      strcat(newFilename,"0");
    }
    strcat(newFilename,monthStr);
    if (dayInt < 10){
      strcat(newFilename,"0");
    }
    strcat(newFilename,dayStr);
    strcat(newFilename,".CSV");
    strcpy(config.workingFilename[i],newFilename);
  }
  config.newFileTime = rawTime;
  EEPROM_writeAnything(0, config);
}
```

Fig. 23 – Funções createfile.

As três linhas seguintes servem para inicializar o cartão SD e todas as suas propriedades e partições. Segue-se a sincronização do relógio e a leitura dos dados da memória EEPROM, como também a inicialização do chip W5100, atribuindo um endereço MAC, IP, Gateway e DNS e depois inicializa-se o servidor.

Por fim atribui-se a data de quando se liga o aparelho às variáveis já inicializadas anteriormente na secção 2.8.2, que correspondem à data em que ocorreu a última escrita em ficheiros, seguida da data em que foi feita a última leitura dos sensores e depois a data em que se enviou pela última vez dados para o servidor.

```
void setup() {
  pinMode(10, OUTPUT);
  digitalWrite(10, HIGH);
  card.init(SPI_HALF_SPEED, 4);
  volume.init(&card);
  root.openRoot(&volume);
  setSyncProvider(readtime);
  EEPROM_readAnything(0,config);
  Ethernet.begin(config.mac, config.ip, config.dnsgateway, config.dnsgateway);
  server.begin();
  lastIntervalTime= now();
  lastIntervalMes= lastIntervalTime;
  timetosend= lastIntervalTime;
}
```

Fig. 24 – Setup.

2.8.5- *Loop*

A secção *loop* é onde se define como o código se vai desenrolar e sempre que o código seja todo executado, volta ao princípio mais uma vez. Esta secção foi estruturada em duas partes, a primeira relativa aos sensores, que engloba a leitura de sensores, criação e escrita de ficheiros e na segunda parte o acesso ao *Webserver* e envio de dados para o servidor.

Na parte de leitura (figura 25) a condição para entrar no ciclo é ter passado um segundo desde a última leitura, leitura essa feita para os sensores ativos e o seu valor é adicionado a uma variável de soma, onde também é incrementado um contador para uma posterior média.

Caso seja o momento de escrever nos ficheiros (isto depende da frequência de escrita), entramos noutro ciclo. Antes de escrevermos no ficheiro, neste novo ciclo, é necessário verificar se é preciso criar um ficheiro antes e para isso temos uma série de condições (figura 26). A primeira é se a variável *first* é verdadeira e de seguida o período a que os ficheiros dizem respeito. Por exemplo, num caso de um ficheiro diário, caso o dia registado pelo relógio seja diferente do dia correspondente à data de criação do último ficheiro então é criado um novo.

```
#define BUFSIZ 100
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    for (uint8_t i = 0; i < 6; i++) {
      if (config.check[i]==1) {
        unsigned int k=analogRead(i);
        if (config.modo!=1){
          sum[i]+=k;
        }
        if (config.modo!=0){
          sumenv[i]+=k;
        }
      }
    }
    if (config.modo!=1){
      count++;
    }
    if (config.modo!=0){
      countenv++;
    }
    lastIntervalMes = rawTime;
    ...
  }
}
```

Fig. 25 – Leitura de sensores e valores guardados para futura média.

A próxima condição pertence ao caso de desejarmos ficheiros semanais, sendo este o caso mais complexo. Esta condição foi pensada para que seja criado um ficheiro no início de cada domingo e a função “weekday” é uma função que atribui um número ao dia da semana, sendo que 1 é para Domingo e 7 para Sábado. A complexidade está no facto de o aparelho poder estar vários dias sem estar ligado e não se pode apenas ler o valor dado pela função “weekday” e dizer para criar no valor 1, pois caso se ligue o dispositivo passado umas semanas este vai continuar a escrever no mesmo ficheiro.

Portanto temos algumas subcondições, como seja para o caso de que o valor “weekday” do nosso momento da leitura for menor que o do valor da última escrita o que significará que estaremos noutra semana. A segunda subcondição é para o caso em que o valor “weekday” é o mesmo. Caso a diferença entre o momento da atual escrita e a última for mais de um dia então estaremos numa nova semana e deveremos criar um novo ficheiro. Esta condição permite também que não se esteja sempre a tentar criar um ficheiro no mesmo dia. A última subcondição será para os casos do aparelho estar desligado há muito tempo e não corresponder às subcondições anteriores. Neste caso,

se a diferença entre o momento atual da escrita e da última escrita for mais de uma semana cria-se um novo ficheiro.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
    if ((rawTime - lastIntervalTime) >= config.freq && config.modo!=1){
      if (config.first==true){
        createfile(rawTime);
        config.first=false;
        EEPROM_writeAnything(0, config);
      }
      else if (config.per==0 & day(rawTime)!=day(config.newFileTime)){
        createfile(rawTime);
      }
      else if (config.per==1){
        if (weekday(rawTime)<weekday(config.newFileTime)){
          createfile(rawTime);
        }
        else if(weekday(rawTime)==weekday(config.newFileTime) && rawTime-config.newFileTime>=86400){
          createfile(rawTime);
        }
        else if((rawTime-config.newFileTime)>604800){
          createfile(rawTime);
        }
      }
      else if (config.per==2 & month(rawTime)!=month(config.newFileTime)){
        createfile(rawTime);
      }
      ...
    }
  }
}
```

Fig. 26 – Condições de criação de ficheiro.

Como última subcondição, temos o caso de ficheiros mensais. Se o valor do mês na altura da escrita for diferente do valor do mês da última, então criaremos um ficheiro.

Depois de todas estas subcondições, é altura de escrever nos ficheiros e para isso é necessário a construção duma *string* (figura 27). Esta *string* é construída colocando a data da escrita no formato *unix timestamp*, seguido duma “,” e por final o valor do sensor que será a média de várias leituras com valores entre 0 a 1023 que corresponde a 10 bits.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
    if ((rawTime - lastIntervalTime) >= config.freq && config.modo!=1){
     ...
      char timeStr[12];
      ultoa(rawTime,timeStr,10);
      char sensorStr[6];
      for (uint8_t i = 0; i < 6; i++) {
        if (config.check[i]==1){
          char dataString[20] = {""};
          unsigned int sensor = sum[i]/count;
          itoa(sensor,sensorStr,10);
          strcat(dataString,timeStr);
          strcat(dataString,",");
          strcat(dataString,sensorStr);
          file.open(&root,config.workingFilename[i], O_RDWR | O_CREAT | O_AT_END);
          file.println(dataString);
          file.close();
        }
        sum[i]=0;
      }
      lastIntervalTime =rawTime;
      count=0;
    }
  }
}
```

Fig. 27 – Escrita no ficheiro.

A parte do código relacionado com a Web está sempre ativa quando não estão a ocorrer leituras dos sensores. Antes de ativarmos o servidor temos que verificar se é necessário enviar os dados ao servidor e efetuar o envio caso seja o caso (figura 28).

Depois desta primeira ação podemos começar a ligação ao servidor, verificando se este está ativo e começando a receber a linha de endereço (figura 29).

Se nessa linha estiver a palavra “set” então começa a comunicação XML e enviaremos os dados para o nosso *Webserver* através da função “XML_response” (figura 30). Em conjunto com a palavra “set” existe uma leque de outras palavras que são detetadas e dependendo da palavra um comando na função “Setinputs” é ativado configurando assim o dispositivo.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
  }
  else{
    if ((rawTime-timetosend >= 900) && config.modo!=0){
      EthernetClient cliente;
      if (cliente.connect(serverIP, 8888)){
        sendtoewen(cliente,rawTime);
        timetosend=rawTime;
        delay(1);
        cliente.stop();
      }
    }
    ...
  }
}
```

Fig. 28 – Envio de dados para servidor EWEN.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
  }
  else{
    ...
    EthernetClient client = server.available();
    if (client) {
      char clientline[BUFSIZ];
      uint8_t index = 0;
      while (client.connected()) {
        if (client.available()) {
          char c = client.read();
          if (c != '\n' && c != '\r') {
            clientline[index] = c;
            index++;
            if (index >= BUFSIZ){
              index = BUFSIZ -1;
            }
            continue;
          }
          clientline[index] = 0;
          client.println(F("HTTP/1.1 200 OK"));
          ...
        }
      }
    }
  }
}
```

Fig. 29 – Ativação do servidor Web.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
  }
  else{
    ...
    if (client) {
      ...
      while (client.connected()) {
        if (client.available()) {
          ...
          if (strstr(clientline, "set")){
              client.println(F("Content-Type: text/xml"));
              client.println(F("Connection: keep-alive"));
              client.println();
              if (strstr(clientline, "t=")){
                Setinputs(clientline,0);
              }
              else if (strstr(clientline, "mode=")){
                Setinputs(clientline,1);
              }
              else if (strstr(clientline, "per=")){
                Setinputs(clientline,2);
              }
              else if (strstr(clientline, "filename=")){
                Setinputs(clientline,3);
              }
              else if (strstr(clientline, "ns=")){
                Setinputs(clientline,4);
              }
              else if (strstr(clientline, "ip=")){
                Setinputs(clientline,5);
              }
              XML_response(client);
          }
          ...
        }
      }
    }
  }
}
```

Fig. 30 – Comunicação XML e casos para ativação da função Setinputs.

No caso de não se receber a palavra “set” haverá a escrita em HTML, com três condições (figura 31). Se para além do IP a linha de endereços não contiver nada, abre-se a página “index.htm” que está guardada no cartão SD. Depois temos o caso de conter na linha “/list.htm”; nesse caso será aberta a página com o mesmo nome que é completa com a função “ListFiles”.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
  }
  else{
    ...
    if (client) {
      ...
      while (client.connected()) {
        if (client.available()) {
          ...
          if (strstr(clientline, "set")){
              ...
          }
          ...
          else {
              client.println(F("Content-Type: text/html"));
              client.println(F("Connection: keep-alive"));
              client.println();
          }
          if (strstr(clientline, "GET / ")) {
              file.open(&root,"index.htm",O_READ);
              while ((c = file.read()) > 0) {
                client.print(c);
              }
              file.close();
          }
          else if (strstr(clientline, "GET /list.htm")) {
              file.open(&root,"list.htm", O_READ);
              while ((c = file.read()) > 0) {
                client.print(c);
              }
              ListFiles(client);
              client.println(F("</body></html>"));
              file.close();
          }
          ...
        }
      }
    }
  }
}
```

Fig. 31 – Escrita HTML e casos específicos das páginas index.htm e list.htm.

Por último temos uma condição que interpreta a linha recebida de forma a abrir a página correspondente (figura 32). No caso da comunicação via Web costuma-se dar um compasso de espera de um segundo de forma a garantirmos que toda a informação seja transferida.

```
void loop(){
  unsigned long rawTime = now();
  if ((rawTime - lastIntervalMes) >= 1){
    ...
  }
  else{
    ...
    if (client) {
      ...
      while (client.connected()) {
        if (client.available()) {
          ...
          if (strstr(clientline, "GET / ")) {
              ...
          }
          else if (strstr(clientline, "GET /list.htm")) {
              ...
          }
          else if (strstr(clientline, "GET /") != 0) {
            char* filename = strtok(clientline + 5, "?");
            strstr(clientline, " HTTP")[0] = 0;
            file.open(&root,filename, O_READ);
            while ((c = file.read()) > 0) {
                client.print(c);
              }
            file.close();
          }
          break;
        }
      }
      delay(1);
      client.stop();
    }
  }
}
```

Fig. 32 – Código para interpretação da linha recebida.

Para que se possa entender melhor o funcionamento do código são apresentados nas figuras 33 e 34, dois fluxogramas que mostram o modo de leitura de sensores e o modo de rede duma forma simplificada.

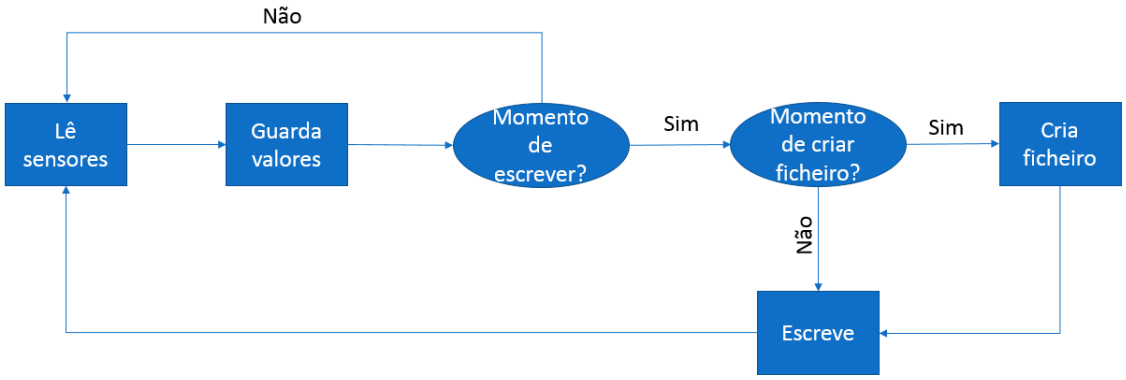

Fig. 33 – Fluxograma do modo de leitura de sensores.

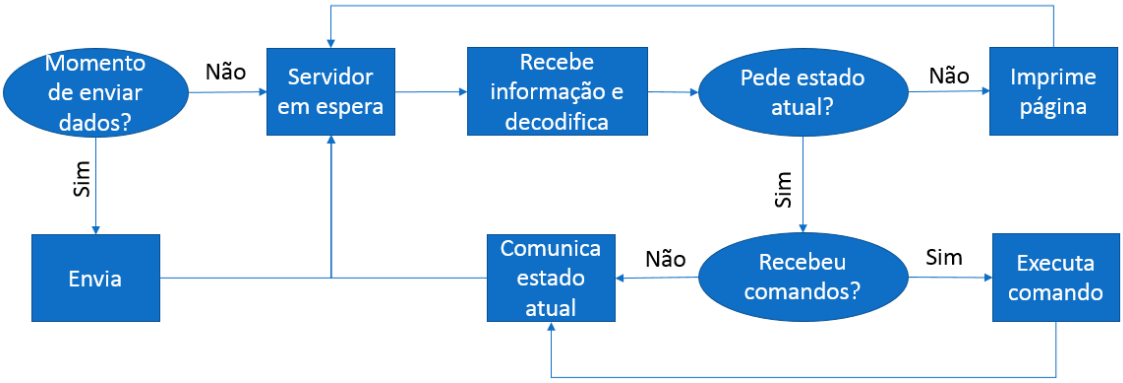

Fig. 34 – Fluxograma do modo de rede.

2.8.6- Configuração EEPROM

Usando um programa autónomo configura-se a EEPROM, definindo uma estrutura igual à apresentada na figura 17, em que se atribui um valor predefinido a cada parâmetro. Estre procedimento é necessário porque nas páginas de configurações, o valor atual é sempre apresentado e por isso deve ser colocado um valor predefinido para não ocorrer um erro no nosso *Webserver*. Também temos que ter um endereço MAC, IP, DNS e Gateway generalizados para que funcione na primeira utilização.

Poderemos observar o código deste programa no anexo 3.

2.9- Análise da plataforma Web

A plataforma Web consiste num conjunto de 10 páginas Web e 6 pastas onde os ficheiros com os dados serão guardados na pasta correspondente ao sensor. Todos estes ficheiros estão guardados no cartão SD e nesta secção vamos fazer uma breve descrição de como cada página funciona.

Serão descritas todas as 10 páginas da plataforma nesta secção, podendo observar os códigos destas entre os anexos 4 e 13.

2.9.1- index.htm

Ao entrar na plataforma Web a primeira página em que se encontra é a "index.htm" (figura 35). Esta página apresenta duas secções, que correspondem a duas outras páginas. Do lado esquerdo temos uma secção fixa que é a página "menu.htm", enquanto a secção da direita é variável ("select.htm") e vai depender da opção do menu que selecionarmos.

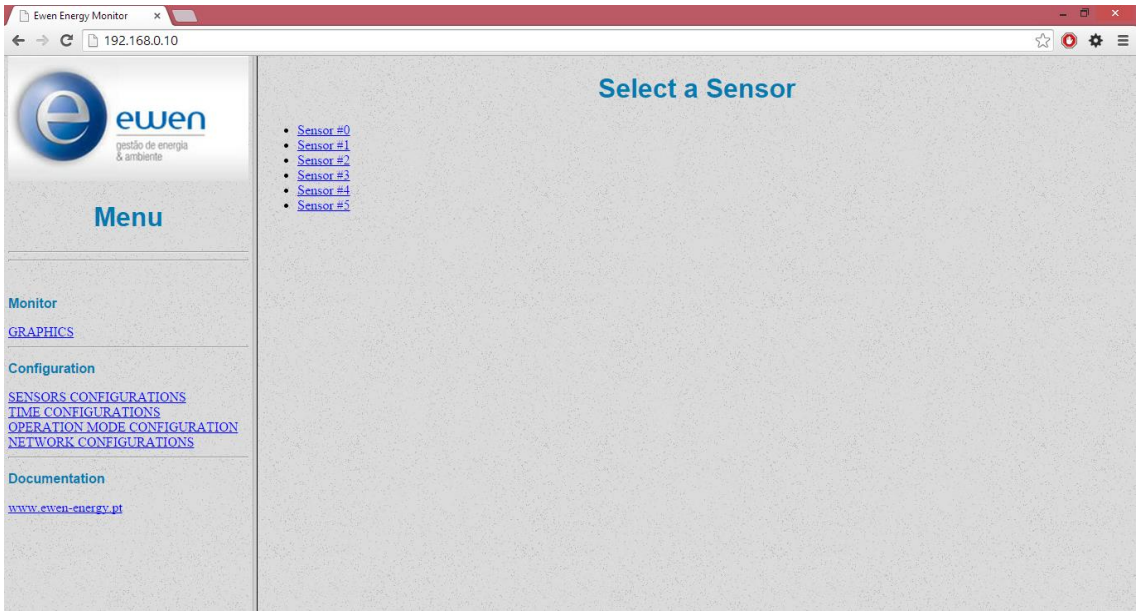

Fig. 35 – Vista geral da página index.htm no primeiro acesso.

2.9.2- menu.htm

Nesta página temos os *links* para todas as páginas da plataforma, pode-se optar entre monitorar gráficos ou configurar todo o sistema (figura 36).

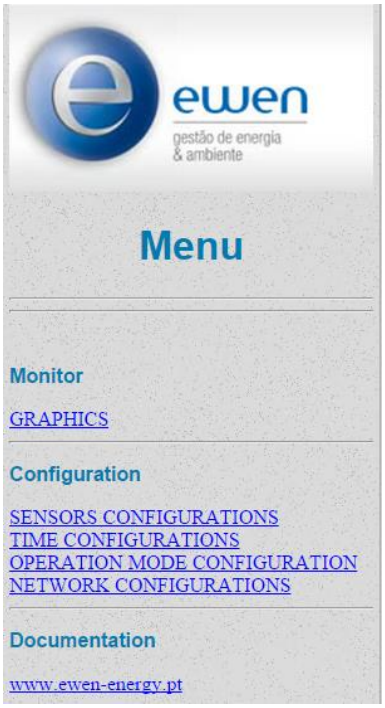

Fig. 36– Frame esquerdo contendo menu.htm.

2.9.3- select.htm

Esta página serve como intermediária à página “list.htm”, tendo os 6 sensores como opção, e quando selecionamos um, este informa o dispositivo sobre qual o sensor que se vai configurar e redireciona para a página adequada (figura 37).

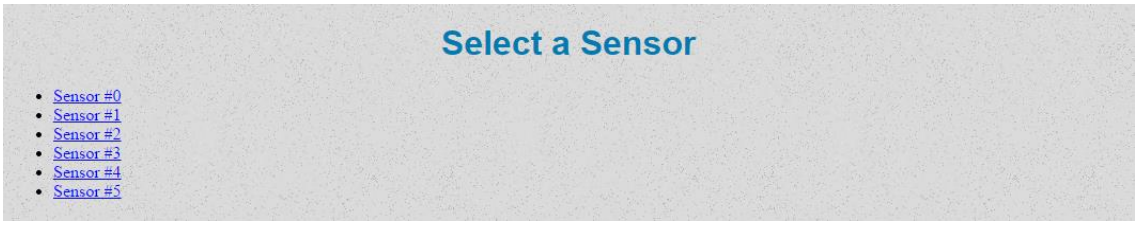

Fig. 37 – Frame direito contendo select.htm.

2.9.4- list.htm

A nível de código esta página é diferente das restantes estando apenas guardado no cartão SD parte desta, sendo necessário a função “ListFiles” (figura 22) para completar a página.

Depois de a página carregar, para além de se poder observar o estado da memória atual do cartão SD, pode-se selecionar o ficheiro que redirecionará o utilizador para a página “HC.htm” que apresentará o gráfico correspondente ao ficheiro; pode-se também a efetuar o *download* do ficheiro no formato CSV ou remover o ficheiro para libertar memória (figura 38).

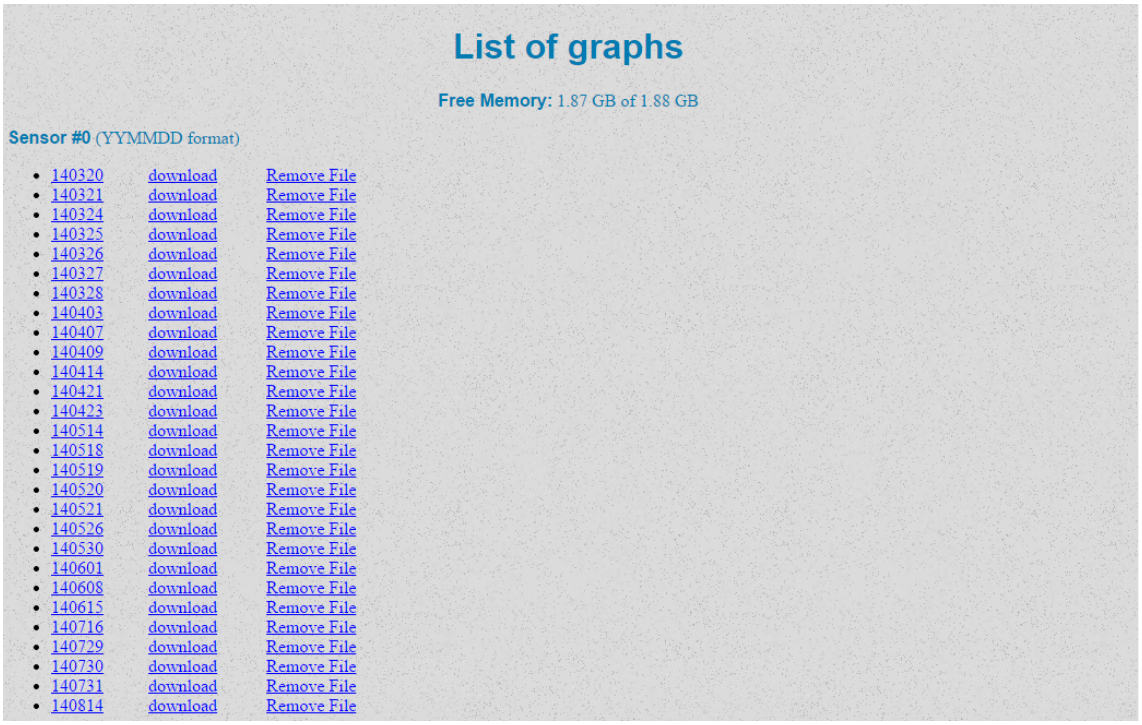

Fig. 38 – Frame direito contendo list.htm.

2.9.5- HC.htm

A função desta página é apresentar gráficos através da plataforma Google Charts, usando a informação recebida por comunicação XML com o dispositivo. Esta página tem várias fórmulas de calibração, correspondentes ao tipo de sensor que está a fazer

a leitura, o que permite desenhar o gráfico na escala desejada. Os valores necessários para desenhar o gráfico são o tipo de sinal de saída do sensor (devido à fórmula de calibração associada) e o máximo e mínimo da unidade que estamos a ler (figura 39).

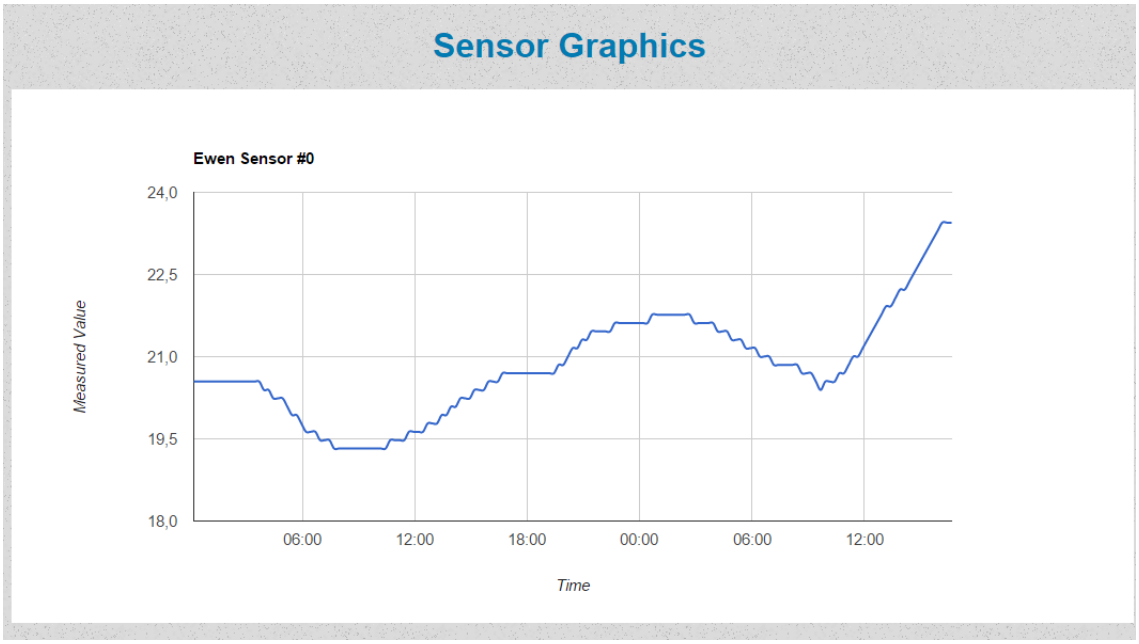

Fig. 39 – Frame direito contendo HC.htm.

2.9.6- sconfig.htm

O campo das configurações é muito semelhante à página “select.htm”, pois serve de intermediária à página “config.htm”, sendo que a grande diferença é esta indicar se cada um dos sensores está ativo ou não (figura 40).

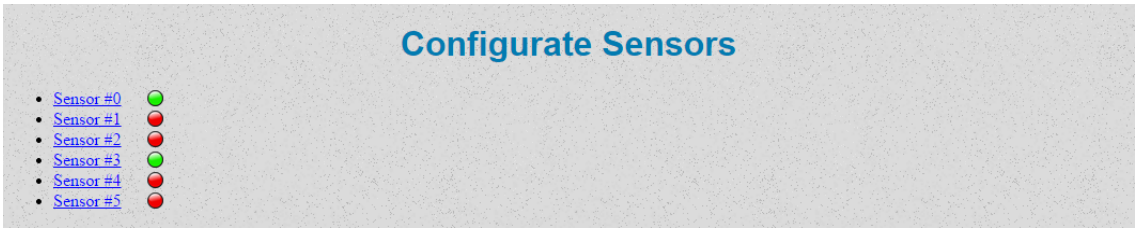

Fig. 40 – Frame direito contendo sconfig.htm.

2.9.7- config.htm

Esta página serve para configurar o sensor que selecionamos na página “sconfig.htm”; são comunicados os valores atuais do sensor através de XML, e assim pode-se alterar apenas o que se desejar. Para além de ativar ou desativar o sensor pode-se alterar o tipo de sinal de saída do sensor bem como o máximo e mínimo da unidade lida (figura 41).

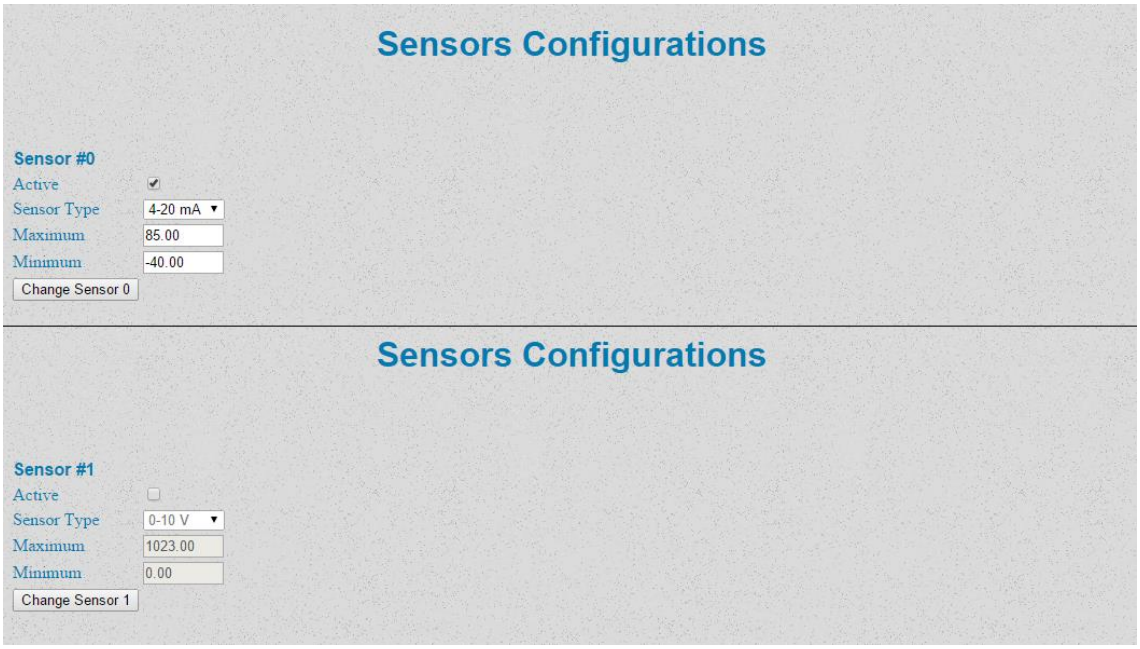

Fig. 41 – Frames direitos contendo config.htm, para quando o sensor está ativo em cima e desativo em baixo.

2.9.8- tconfig.htm

Em “tconfig.htm” configura-se a frequência de escrita no ficheiro e o período de tempo em que os dados são agregados num ficheiro. Tal como a página anterior, os dados atuais são comunicados através de XML (figura 42).

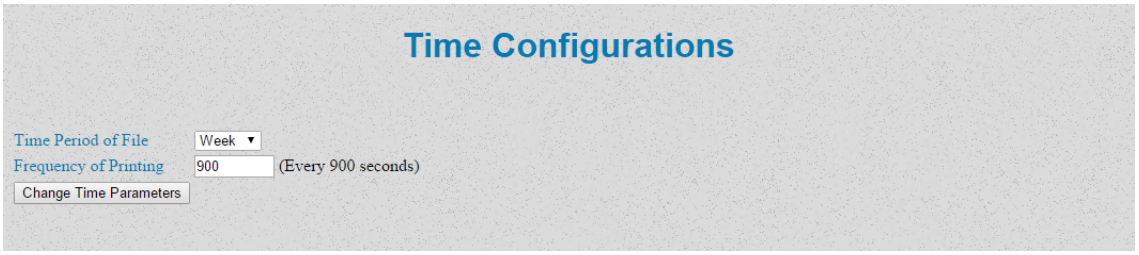

Fig. 42 – Frame direito contendo tconfig.htm.

2.9.9- mconfig.htm

Nesta página a única opção é configurar o modo de funcionamento do dispositivo, se este apenas escreve os dados no ficheiro, se envia os dados para um servidor da EWEN ou se faz ambos em simultâneo (figura 43).

Fig. 43 – Frame direito contendo mconfig.htm.

2.9.10- network.htm

Por fim, temos a página “network.htm” que nos permitirá configurar dados como o endereço MAC, IP e DNS/Gateway (figura 44).

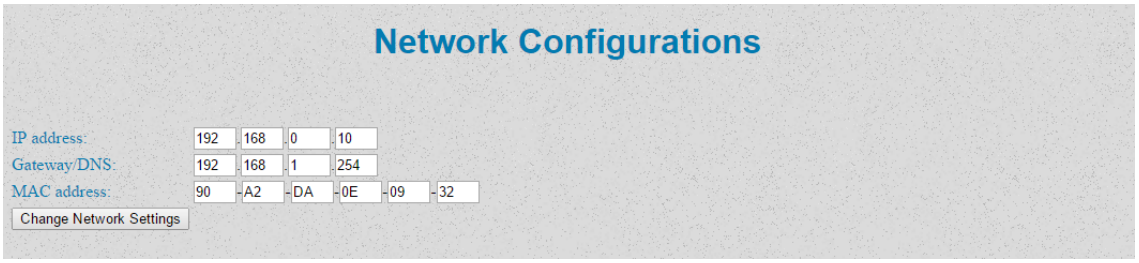

Fig. 44 – Frame direito contendo network.htm.

3- Propostas para futuro desenvolvimento

Como propostas para a continuação do desenvolvimento do protótipo, começamos por sugerir a obtenção do número de série automaticamente visto que a única maneira de o obter é ligando o dispositivo por USB a um computador e depois ir às suas propriedades para obter esse número. A ideia seria criar um programa que consiga ir encontrar esse valor e que o insira no nosso código de configuração da memória EEPROM por exemplo.

Seria ideal criar um manual para que fosse possível que o cliente configurasse o dispositivo sem qualquer dificuldade. No caso da configuração do IP pode acontecer problemas se colocarmos um IP já atribuído. Uma ideia seria utilizar uma função do Arduino que cria um IP automático quando o dispositivo fosse ligado via *Ethernet*, e o dispositivo criasse um ficheiro no cartão SD com essa informação para facilitar o primeiro acesso. O problema prende-se no facto de ao permitir o acesso exterior ao cartão, o utilizador pode sem querer apagar os ficheiros Web, comprometendo o funcionamento do dispositivo.

A nível da placa de sensores poderiam ser usadas resistências de 250Ω em vez de 240Ω, que levariam a uma melhor conversão do sinal analógico quando usamos sensores com sinal de saída 4-20mA ou 0-20mA, antes de ser lido pelo Arduino. Outra solução seria o uso de potenciómetros pois seria possível ajustar o valor da resistência.

É necessário ainda fazer um bom estudo de produção em massa, contactando várias empresas de modo a criar preços das placas e componentes mais baratos do que se vê no mercado para poder tornar o produto economicamente viável. Seria vantajoso encriptar o código para que este fosse compilado onde o dispositivo é produzido em massa, de forma a reduzir trabalho por parte da empresa sem que o código pudesse ser copiado por outras.

Poderia haver a possibilidade de termos um concentrador a receber dados de vários Arduinos, mas este teria que ter uma memória maior como por exemplo o Arduino Mega 2560, ou outro tipo de microcontrolador, visto que ultimamente têm saído outros *hardwares* de baixo-custo, com mais especificações que poderiam ser mais rentáveis, tanto para criar um concentrador, como para substituir o Arduino. O que mais me chamou a atenção foi o Intel Galileo [12], que é um produto compatível com a programação Arduino.

4- Conclusões

Depois da conclusão deste projeto, podemos concluir que este superou a espectativas, pois, para além de realizar o que foi proposto nos objetivos, conseguiu-se que o protótipo final fosse mais que um monitor de consumo energético, tendo a versatilidade de ler as unidades físicas que quisermos caso se compre um sensor com um dos outputs permitidos.

Quase todos objetivos foram cumpridos, menos o que diz respeito ao estudo de produção em massa devido a dificuldades em encontrar preços para grandes quantidades e para todos os componentes, assim com na procura de preços para o encapsulamento personalizado. Como esta foi a tarefa que foi deixada para último lugar, por falta de tempo esta não foi realizada com sucesso e por isso não apresentada no trabalho.

Apesar do projeto ter a finalidade de ler um quadro elétrico, esse teste não foi feito, pois os sensores de corrente não foram adquiridos. Isto não implica nada no projeto pois foram efetuados testes com outros tipos de sensores e o relevante é que o output dos sensores seja compatível com os que o dispositivo possa suportar, para posteriormente fazer uma conversão do sinal analógico para digital.

O desenvolvimento deste projeto foi uma mais-valia pois, para além de ter aprofundado conhecimentos em programar um Arduino de forma a criar um dispositivo com diversas funcionalidades, tive a oportunidade de aprender várias linguagens de programação Web, tanto para construção de *sites* e todo um *Webserver* como também para a comunicação com servidores externos. Foi bastante motivante estar a realizar um projeto vendo que este tem potencial comercial e por isso penso que a experiência deste estágio foi bastante positiva, pois existe um contacto com o mundo empresarial, e o tipo de pressão para resultados é diferente do que nos acostumamos a um nível universitário, o que me fez crescer enquanto estudante.

5- Referências bibliográficas

[1] http://www.arduino.cc/, Arduino

[2] http://arduino.cc/en/Main/ArduinoEthernetShield, Arduino Ethernet Shield

[3] http://store.open-electronics.org/RTC%20shield, RTC Shield para Arduino

[4] http://learn.adafruit.com/ds1307-real-time-clock-breakout-board-kit/what-is-an-rtc, DS1307 RTC

[5] http://arduino.cc/en/Products.Compare, Comparação de Arduinos

[6] http://everettsprojects.com/2012/12/31/arduino-super-graphing-data-logger/, Super Graphing Data Logger

[7] http://playground.arduino.cc/Learning/Memory, Função F()

[8] http://www.w3schools.com/, Tutoriais para programação Web

[9] http://purposefulscience.blogspot.pt/2013/06/saving-ram-in-arduino-to-fit-sd-library.html, Dicas para poupar RAM

[10] http://www.atmel.com/images/doc8453.pdf, Dicas para optimização do código

[11] https://developers.google.com/chart/, Google Charts

[12] http://arduino.cc/en/ArduinoCertified/IntelGalileo, Inter Galileo

[13] http://www.mysql.com/, MySQL

6- Anexos

Anexo 1 – Esquemas das placas produzidas para o projeto do aluno Tiago Costa

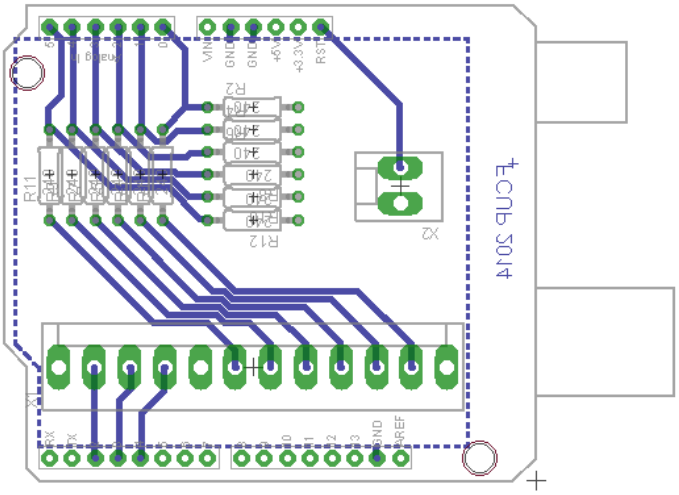

Fig. 45 – Desenho de um *shield* para um sistema de deslastre.

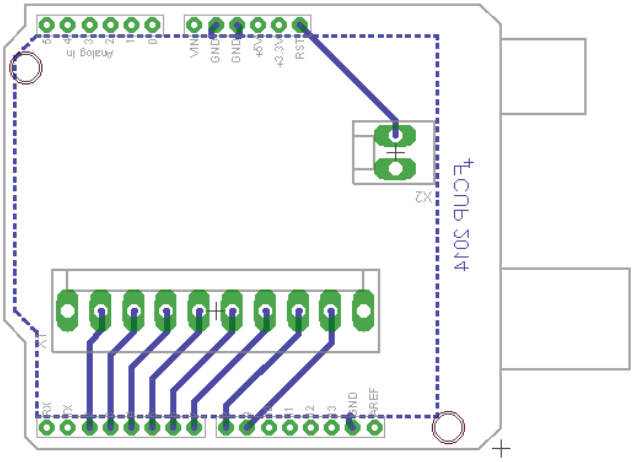

Fig. 46 – Desenho de um *shield* para um contador de impulsos.

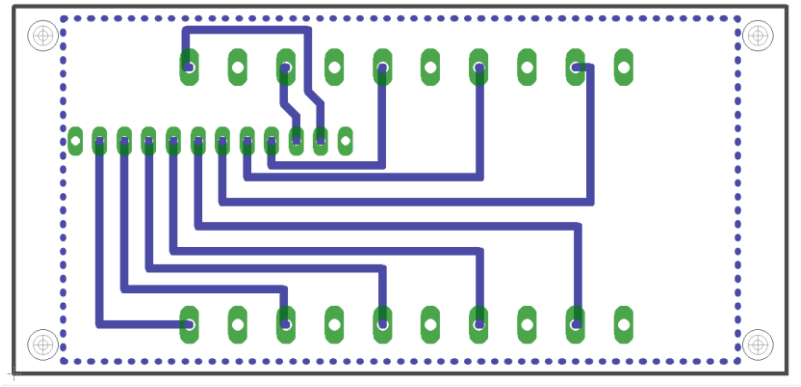

Fig. 47 – Desenho da placa de interface para o sistema de deslastre e contador de impulsos.

Anexo 2 – EEM.ino

```
#include <SdFat.h>
#include <Ethernet.h>
#include <SPI.h>
#include <EEPROM.h>
#include <EEPROMAnything.h>
#include <string.h>
#include <Time.h>
#include <DigitalIO.h>

#define DS1307ADDR 0XD0
const uint8_t SDA_PIN = 2;
const uint8_t SCL_PIN = 3;
FastI2cMaster<SCL_PIN, SDA_PIN> rtc;

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
SdFile workingdir;

IPAddress serverIP(37,187,134,39);
EthernetServer server(80);

unsigned long lastIntervalTime;
unsigned long lastIntervalMes;
unsigned long timetosend;
unsigned long sum[6]={
  0L,0L,0L,0L,0L,0L};
unsigned long sumenv[6]={
  0L,0L,0L,0L,0L,0L};
unsigned int count;
unsigned long countenv;
uint8_t ns;
char* networks[6];

typedef struct{
  unsigned long newFileTime;
  char workingFilename[6][14];
  uint8_t tipo[6];
  double maxi[6];
  double mini[6];
  uint8_t check[6];
  uint8_t modo;
  uint8_t per;
  int freq;
  boolean first;
  byte mac[6];
  byte ip[4];
  byte dnsgateway[4];
}
configuration;

configuration config;

uint8_t readDS1307(uint8_t address, uint8_t *buf, uint8_t count){
  if (!rtc.transfer(DS1307ADDR | I2C_WRITE, &address, 1)){
    return false;
  }
  return rtc.transfer(DS1307ADDR | I2C_READ, buf, count);
}
```

```
61.
62. uint8_t bcd2dec(uint8_t num){
63.   return ((num/16 * 10) + (num % 16));
64. }
65.
66. time_t readtime(void){
67.   uint8_t r[8];
68.   readDS1307(0, r, 8);
69.   setTime(bcd2dec(r[2]),bcd2dec(r[1]),bcd2dec(r[0]),bcd2dec(r[4]), bcd2dec(r[5]
    ), bcd2dec(r[6]));
70.   return now();
71. }
72.
73. void XML_response(EthernetClient cl){
74.   cl.print(F("<?xml version = \"1.0\" ?><inputs>"));
75.   for (uint8_t i=0; i<4; i++){
76.     cl.print(F("<ip>"));
77.     cl.print(config.ip[i]);
78.     cl.print(F("</ip>"));
79.   }
80.   for (uint8_t i=0; i<6; i++){
81.     cl.print(F("<mac>"));
82.     if (config.mac[i]<16){
83.       cl.print(F("0"));
84.     }
85.     cl.print(config.mac[i], HEX);
86.     cl.print(F("</mac>"));
87.   }
88.   for (uint8_t i=0; i<4; i++){
89.     cl.print(F("<dnsg>"));
90.     cl.print(config.dnsgateway[i]);
91.     cl.print(F("</dnsg>"));
92.   }
93.   cl.print(F("<mode>"));
94.   cl.print(config.modo);
95.   cl.print(F("</mode><per>"));
96.   cl.print(config.per);
97.   cl.print(F("</per><freq>"));
98.   cl.print(config.freq);
99.   cl.print(F("</freq><ns>"));
100.  cl.print(ns);
101.  cl.print(F("</ns><tipo>"));
102.  cl.print(config.tipo[ns]);
103.  cl.print(F("</tipo><max>"));
104.  cl.print(config.maxi[ns]);
105.  cl.print(F("</max><min>"));
106.  cl.print(config.mini[ns]);
107.  cl.print(F("</min>"));
108.  for (uint8_t i = 0; i < 6; i++) {
109.    cl.print(F("<check>"));
110.    cl.print(config.check[i]);
111.    cl.print(F("</check>"));
112.  }
113.  cl.print(F("</inputs>"));
114.}
115.
116.void Setinputs(char* clientline1, uint8_t n)
117.{
118.  char* pch;
119.  char* z[4];
120.  uint8_t i=0;
121.  pch = strtok (clientline1,"=&");
122.  while (pch != NULL)
123.  {
124.    if (i%2!=0){
```

```
125.      z[i/2]=pch;
126.    }
127.    pch = strtok (NULL,"=&");
128.    i++;
129.  }
130.  if (n==0){
131.    config.tipo[ns]=atoi(z[0]);
132.    config.maxi[ns]=atof(z[1]);
133.    config.mini[ns]=atof(z[2]);
134.    config.check[ns]=atoi(z[3]);
135.    EEPROM_writeAnything(0, config);
136.  }
137.  else if (n==1){
138.    config.modo=atoi(z[0]);
139.    EEPROM_writeAnything(0, config);
140.  }
141.  else if (n==2) {
142.    config.freq=atoi(z[1]);
143.    if (config.per!=atoi(z[0])){
144.      config.per=atoi(z[0]);
145.      config.first=true;
146.    }
147.    EEPROM_writeAnything(0, config);
148.  }
149.  else if (n==3){
150.    file.open(&root,z[0], O_RDWR | O_AT_END);
151.    file.remove();
152.  }
153.  else if (n==4) {
154.    ns=atoi(z[0]);
155.  }
156.  else {
157.    network(z[0]);
158.    for (uint8_t i=0; i<4; i++){
159.      config.ip[i]=atoi(networks[i]);
160.    }
161.    network(z[1]);
162.    for (uint8_t i=0; i<4; i++){
163.      config.dnsgateway[i]=atoi(networks[i]);
164.    }
165.    network(z[2]);
166.    for (uint8_t i=0; i<6; i++){
167.      config.mac[i]=atoi(networks[i]);
168.    }
169.    EEPROM_writeAnything(0, config);
170.  }
171.}
172.
173.void network(char* clientline1){
174.  char* pch;
175.  uint8_t i=0;
176.  pch = strtok (clientline1,".");
177.  while (pch != NULL)
178.  {
179.    networks[i]=pch;
180.    pch = strtok (NULL,".");
181.    i++;
182.  }
183.}
184.
185.void sendtoewen(EthernetClient cliente, unsigned long rawTime){
186.  byte mask=0;
187.  for (uint8_t i=0; i < 6; i++){
188.    if (config.check[i]==1){
189.      mask+=(1<<i);
```

```
    }
  }
  cliente.println(F("POST /arduinotest.php HTTP/1.1"));
  cliente.print(F("Host: "));
  cliente.println(serverIP);
  cliente.println(F("Connection: close\r\n Content-Type: application/x-www-form-urlencoded"));
  cliente.println(F("Content-Length:150 \r\n"));
  cliente.print(F("id=Arduino55330333630351F0200&utc="));
  cliente.print(rawTime);
  cliente.print(F("&mask="));
  cliente.print(mask, HEX);
  cliente.print(F("&value="));
  for (uint8_t i=0; i < 6; i++){
    if (config.check[i]==1){
      if (config.tipo[i]==0){
        cliente.print((config.maxi[i]-config.mini[i])*(sumenv[i]/countenv)/512.0+config.mini[i]);
      }
      else if (config.tipo[i]==3){
        cliente.print((5/4.0)*(config.maxi[i]-config.mini[i])*((sumenv[i]/countenv)/1023.0-1)+config.maxi[i]);
      }
      else {
        cliente.print((config.maxi[i]-config.mini[i])*(sumenv[i]/countenv)/1023.0+config.mini[i]);
      }
      cliente.print(sumenv[i]/countenv);
      cliente.print(F(","));
    }
    sumenv[i]=0;
  }
  cliente.print(F("                                                    "));
  cliente.print(F("                                                    "));
  countenv=0;
}

void ListFiles(EthernetClient client) {
  char f[2];
  itoa(ns,f,10);
  client.print(F("<p align=center><span class=\"sub\">Free Memory:</span> "));

  client.print(volume.freeClusterCount()*volume.blocksPerCluster()/2097152.0);

  client.print(F(" GB of "));
  client.print(card.cardSize()/2097152.0);
  client.print(F(" GB</p><p><span class=\"sub\">Sensor #"));
  client.print(f);
  client.print(F("</span> (YYMMDD format)</p>"));
  dir_t p;
  workingdir.open(&root, f , O_READ);
  workingdir.rewind();
  client.println(F("<ul>"));
  while (workingdir.readDir(&p) > 0) {
    if (p.name[0] == DIR_NAME_FREE) break;
    if (p.name[0] == DIR_NAME_DELETED || p.name[0] == '.') continue;
    if (!DIR_IS_FILE_OR_SUBDIR(&p)) continue;
    client.print(F("<li><a href=\"/HC.htm?file="));
    client.println(f);
    client.print(F("/"));
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
```

```
249.      client.print((char)p.name[i]);
250.    }
251.    client.print(F("\">"));
252.    for (uint8_t i = 0; i < 6; i++) {
253.      client.print((char)p.name[i]);
254.    }
255.    client.print(F("</a><span class=\"tab\"></span><a href=\""));
256.    client.print(f);
257.    client.print(F("/"));
258.    for (uint8_t i = 0; i < 11; i++) {
259.      if (p.name[i] == ' ') continue;
260.      if (i == 8) {
261.        client.print('.');
262.      }
263.      client.print((char)p.name[i]);
264.    }
265.    client.println(F("\" download>download</a><span class=\"tab\"></span>"));

266.    client.print(F("<a href=javascript:removefile('"));
267.    client.print(f);
268.    client.print(F("/"));
269.    for (uint8_t i = 0; i < 11; i++) {
270.      if (p.name[i] == ' ') continue;
271.      if (i == 8) {
272.        client.print('.');
273.      }
274.      client.print((char)p.name[i]);
275.    }
276.    client.print(F("');>Remove File</a></li>"));
277.  }
278.  client.println(F("</ul>"));
279.  workingdir.close();
280.}
281.
282.void createfile(unsigned long rawTime){
283.  int dayInt = day(rawTime);
284.  int monthInt = month(rawTime);
285.  int yearInt = year(rawTime);
286.  char dayStr[3];
287.  char monthStr[3];
288.  char yearStr[5];
289.  char subYear[3];
290.  char f[2];
291.  itoa(yearInt,yearStr,10);
292.  itoa(monthInt,monthStr,10);
293.  itoa(dayInt,dayStr,10);
294.  memcpy( subYear, &yearStr[2], 3 );
295.  for (uint8_t i = 0; i < 6; i++) {
296.    char newFilename[13] = {
297.      ""    };
298.    itoa(i,f,10);
299.    strcat(newFilename,f);
300.    strcat(newFilename,"/");
301.    strcat(newFilename,subYear);
302.    if (monthInt < 10){
303.      strcat(newFilename,"0");
304.    }
305.    strcat(newFilename,monthStr);
306.    if (dayInt < 10){
307.      strcat(newFilename,"0");
308.    }
309.    strcat(newFilename,dayStr);
310.    strcat(newFilename,".CSV");
311.    strcpy(config.workingFilename[i],newFilename);
312.  }
```

```
313.   config.newFileTime = rawTime;
314.   EEPROM_writeAnything(0, config);
315. }
316. 
317. void setup() {
318.   pinMode(10, OUTPUT);
319.   digitalWrite(10, HIGH);
320.   card.init(SPI_HALF_SPEED, 4);
321.   volume.init(&card);
322.   root.openRoot(&volume);
323.   setSyncProvider(readtime);
324.   EEPROM_readAnything(0,config);
325.   Ethernet.begin(config.mac, config.ip, config.dnsgateway, config.dnsgateway);

326.   server.begin();
327.   lastIntervalTime= now();
328.   lastIntervalMes= lastIntervalTime;
329.   timetosend= lastIntervalTime;
330. }
331. 
332. #define BUFSIZ 100
333. void loop(){
334.   unsigned long rawTime = now();
335.   if ((rawTime - lastIntervalMes) >= 1){
336.     for (uint8_t i = 0; i < 6; i++) {
337.       if (config.check[i]==1) {
338.         unsigned int k=analogRead(i);
339.         if (config.modo!=1){
340.           sum[i]+=k;
341.         }
342.         if (config.modo!=0){
343.           sumenv[i]+=k;
344.         }
345.       }
346.     }
347.     if (config.modo!=1){
348.       count++;
349.     }
350.     if (config.modo!=0){
351.       countenv++;
352.     }
353.     lastIntervalMes = rawTime;
354.     if ((rawTime - lastIntervalTime) >= config.freq && config.modo!=1){
355.       if (config.first==true){
356.         createfile(rawTime);
357.         config.first=false;
358.         EEPROM_writeAnything(0, config);
359.       }
360.       else if (config.per==0 & day(rawTime)!=day(config.newFileTime)){
361.         createfile(rawTime);
362.       }
363.       else if (config.per==1){
364.         if (weekday(rawTime)<weekday(config.newFileTime)){
365.           createfile(rawTime);
366.         }
367.         else if(weekday(rawTime)==weekday(config.newFileTime) && rawTime-
    config.newFileTime>=86400){
368.           createfile(rawTime);
369.         }
370.         else if((rawTime-config.newFileTime)>604800){
371.           createfile(rawTime);
372.         }
373.       }
374.       else if (config.per==2 & month(rawTime)!=month(config.newFileTime)){
375.         createfile(rawTime);
```

```
376.       }
377.       char timeStr[12];
378.       ultoa(rawTime,timeStr,10);
379.       char sensorStr[6];
380.       for (uint8_t i = 0; i < 6; i++) {
381.         if (config.check[i]==1){
382.           char dataString[20] = {
383.             ""           };
384.           unsigned int sensor = sum[i]/count;
385.           itoa(sensor,sensorStr,10);
386.           strcat(dataString,timeStr);
387.           strcat(dataString,",");
388.           strcat(dataString,sensorStr);
389.           file.open(&root,config.workingFilename[i], O_RDWR | O_CREAT | O_AT_E
    ND);
390.           file.println(dataString);
391.           file.close();
392.         }
393.         sum[i]=0;
394.       }
395.       lastIntervalTime =rawTime;
396.       count=0;
397.     }
398.   }
399.   else{
400.     if ((rawTime-timetosend >= 900) && config.modo!=0){
401.       EthernetClient cliente;
402.       if (cliente.connect(serverIP, 8888)){
403.         sendtoewen(cliente,rawTime);
404.         timetosend=rawTime;
405.         delay(1);
406.         cliente.stop();
407.       }
408.     }
409.     EthernetClient client = server.available();
410.     if (client) {
411.       char clientline[BUFSIZ];
412.       uint8_t index = 0;
413.       while (client.connected()) {
414.         if (client.available()) {
415.           char c = client.read();
416.           if (c != '\n' && c != '\r') {
417.             clientline[index] = c;
418.             index++;
419.             if (index >= BUFSIZ){
420.               index = BUFSIZ -1;
421.             }
422.             continue;
423.           }
424.           clientline[index] = 0;
425.           client.println(F("HTTP/1.1 200 OK"));
426.           if (strstr(clientline, "set")){
427.             client.println(F("Content-Type: text/xml"));
428.             client.println(F("Connection: keep-alive"));
429.             client.println();
430.             if (strstr(clientline, "t=")){
431.               Setinputs(clientline,0);
432.             }
433.             else if (strstr(clientline, "mode=")){
434.               Setinputs(clientline,1);
435.             }
436.             else if (strstr(clientline, "per=")){
437.               Setinputs(clientline,2);
438.             }
439.             else if (strstr(clientline, "filename=")){
```

```
440.               Setinputs(clientline,3);
441.             }
442.             else if (strstr(clientline, "ns=")){
443.               Setinputs(clientline,4);
444.             }
445.             else if (strstr(clientline, "ip=")){
446.               Setinputs(clientline,5);
447.             }
448.             XML_response(client);
449.           }
450.           else {
451.             client.println(F("Content-Type: text/html"));
452.             client.println(F("Connection: keep-alive"));
453.             client.println();
454.           }
455.           if (strstr(clientline, "GET / ")) {
456.             file.open(&root,"index.htm",O_READ);
457.             while ((c = file.read()) > 0) {
458.               client.print(c);
459.             }
460.             file.close();
461.           }
462.           else if (strstr(clientline, "GET /list.htm")) {
463.             file.open(&root,"list.htm", O_READ);
464.             while ((c = file.read()) > 0) {
465.               client.print(c);
466.             }
467.             ListFiles(client);
468.             client.println(F("</body></html>"));
469.             file.close();
470.           }
471.           else if (strstr(clientline, "GET /") != 0) {
472.             char* filename = strtok(clientline + 5, "?");
473.             strstr(clientline, " HTTP")[0] = 0;
474.             file.open(&root,filename, O_READ);
475.             while ((c = file.read()) > 0) {
476.               client.print(c);
477.             }
478.             file.close();
479.           }
480.           break;
481.         }
482.       }
483.       delay(1);
484.       client.stop();
485.     }
486.   }
487. }
```

Anexo 3 – EEPROM_config.ino

```
#include <string.h>
#include <EEPROM.h>
#include <EEPROMAnything.h>

typedef struct{
    unsigned long newFileTime;
    char workingFilename[6][14];
    uint8_t tipo[6];
    double maxi[6];
    double mini[6];
    uint8_t check[6];
    uint8_t modo;
    uint8_t per;
    int freq;
    boolean first;
    byte mac[6];
    byte ip[4];
    byte dnsgateway[4];
} configuration;

 configuration config;

void setup(){
  Serial.begin(9600);
  config.newFileTime=1414800000L;
  strcpy(config.workingFilename[0],"0/141101.CSV");
  strcpy(config.workingFilename[1],"1/141101.CSV");
  strcpy(config.workingFilename[2],"2/141101.CSV");
  strcpy(config.workingFilename[3],"3/141101.CSV");
  strcpy(config.workingFilename[4],"4/141101.CSV");
  strcpy(config.workingFilename[5],"5/141101.CSV");
  config.tipo[0]=0;
  config.tipo[1]=0;
  config.tipo[2]=0;
  config.tipo[3]=0;
  config.tipo[4]=0;
  config.tipo[5]=0;
  config.maxi[0]=81023;
  config.maxi[1]=1023;
  config.maxi[2]=1023;
  config.maxi[3]=1023;
  config.maxi[4]=1023;
  config.maxi[5]=1023;
  config.mini[0]=0;
  config.mini[1]=0;
  config.mini[2]=0;
  config.mini[3]=0;
  config.mini[4]=0;
  config.mini[5]=0;
  config.check[0]=0;
  config.check[1]=0;
  config.check[2]=0;
  config.check[3]=0;
  config.check[4]=0;
  config.check[5]=0;
  config.modo=0;
  config.per=0;
  config.freq=1;
  config.first=true;
  config.mac[0] = 0x90;
```

```
61.   config.mac[1] = 0xA2;
62.   config.mac[2] = 0xDA;
63.   config.mac[3] = 0x0E;
64.   config.mac[4] = 0x09;
65.   config.mac[5] = 0x32;
66.   config.ip[0] = 192;
67.   config.ip[1] = 168;
68.   config.ip[2] = 0;
69.   config.ip[3] = 1;
70.   config.dnsgateway[0] = 192;
71.   config.dnsgateway[1] = 168;
72.   config.dnsgateway[2] = 1;
73.   config.dnsgateway[3] = 254;
74.   EEPROM_writeAnything(0, config);
75. }
76.
77. void loop(){
78. }
```

Anexo 4 – index.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Ewen Energy Monitor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<frameset cols="300,*" noresize>
<frame name="leftFrame" src="menu.htm" noresize>
<frame name="rightFrame" src="select.htm">
</frameset>
<noframes>
<body>
</body>
</noframes>
</html>
```

Anexo 5 – menu.htm

```
<!DOCTYPE html>
<html>
<head>
<title>EWEN Menu</title>
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
.sub {color: #017bb1; font-family: arial; font-size: : 12px; font-weight: bold;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
</head>
<body leftmargin="5" topmargin="0">
<img  width="290" src="http://www.ewen-energy.com/img/logo.jpg">
<br>
<h1 align="center" class="title">Menu</h1><hr><hr>
<form>
<br>
  <p align="left" class="sub">Monitor</p>
  <a href=select.htm target=rightFrame>GRAPHICS</a><br>
    <hr>
    <p align="left" class="sub">Configuration</p>
  <a href=sconfig.htm target=rightFrame>SENSORS CONFIGURATIONS</a><br>
  <a href=tconfig.htm target=rightFrame>TIME CONFIGURATIONS</a><br>
  <a href=mconfig.htm target=rightFrame>OPERATION MODE CONFIGURATION</a><br>
  <a href=network.htm target=rightFrame>NETWORK CONFIGURATIONS</a><br>
    <hr>
    <p align="left" class="sub">Documentation</p>
  <p><a href="http://www.ewen-energy.pt" target="_blank">www.ewen-energy.pt</a></p>
</form>
</body>
</html>
```

Anexo 6 – select.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Select Sensor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-
weight: bold;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<script>
    function listfile(ns)
    {
        strInputs = "set";
        strInputs += "ns="+ns+"&";
        var request = new XMLHttpRequest();
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
        location.href="list.htm";
    }
</script>
</head>
<body>
<h1 align="center" class="title">Select a Sensor</h1>
<ul>
    <li><a href="javascript:listfile(0);">Sensor #0</a></li>
    <li><a href="javascript:listfile(1);">Sensor #1</a></li>
    <li><a href="javascript:listfile(2);">Sensor #2</a></li>
    <li><a href="javascript:listfile(3);">Sensor #3</a></li>
    <li><a href="javascript:listfile(4);">Sensor #4</a></li>
    <li><a href="javascript:listfile(5);">Sensor #5</a></li>
</ul>
</body>
</html>
```

Anexo 7 – list.htm

```
<!DOCTYPE html>
<html>
<head>
<title>List of data</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-
weight: bold;}
p {color: #017bb1; font-size: : 12px;}
.sub {color: #017bb1; font-family: arial; font-size: : 12px; font-
weight: bold;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<style type="text/css">
<!--
span.tab {padding-left: 2.6em;}
-->
</style>
<script>
    function removefile(filename)
    {
        strInputs = "set";
        strInputs += "filename="+filename+"&";
        var request = new XMLHttpRequest();
        request.open("GET", strInputs, false);
        request.send(null);
        location.reload();
        strInputs = "set";
    }
</script>
</head>
<body>
<h1 align="center" class="title">List of graphs</h1>
```

Anexo 8 – HC.htm

```
<!DOCTYPE html>
<html>
<head>
   <title>Google Chart Example</title>
    <style>
    a:link    {color:blue;}
    a:visited {color:blue;}
    a:hover   {color:green;}
    .title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
    .sub {color: #017bb1; font-family: arial; font-size: : 12px; font-weight: bold;}
    body
    {
    background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
    }
    </style>
   <script src="https://www.google.com/jsapi"></script>
   <script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
   <script src="http://cdnjs.cloudflare.com/ajax/libs/jquery-csv/0.71/jquery.csv-0.71.min.js"></script>
   <script>

    function Getvalues()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        ns = this.responseXML.getElementsByTagName('ns')[0].childNodes[0].nodeValue;
                        tipo=parseInt(this.responseXML.getElementsByTagName('tipo')[0].childNodes[0].nodeValue);
                        max=parseFloat(this.responseXML.getElementsByTagName('max')[0].childNodes[0].nodeValue);
                        min=parseFloat(this.responseXML.getElementsByTagName('min')[0].childNodes[0].nodeValue);
                    }
                }
            }
        }
        request.open("GET", "set", false);
        request.send(null);
    }
      google.load("visualization", "1", {packages:["corechart"]});
      google.setOnLoadCallback(drawChart);

      function getDataFilename(str){
        point = str.lastIndexOf("file=")+4;
         tempString = str.substring(point+1,str.length)
        if (tempString.indexOf("&") == -1){
            return(tempString);
        }
        else{
            return tempString.substring(0,tempString.indexOf("&"));
        }
      }
      query  = window.location.search;
```

```
54.       var dataFilePath = getDataFilename(query);
55.       function drawChart() {
56.         $.get(dataFilePath, function(csvString) {
57.             Getvalues();
58.             var arrayData = $.csv.toArrays(csvString, {onParseValue: $.csv.hook
    s.castToScalar});
59.             arrayData.unshift(["Time","Measured Value"]);
60.             for (var i=1; i<arrayData.length; i++) {
61.                 arrayData[i][0]=new Date(arrayData[i][0]*1000);
62.                 if (tipo==0){
63.                     arrayData[i][1]=(max-min)*arrayData[i][1]/512.0+min;
64.                 }
65.                 else if (tipo==3) {
66.                     arrayData[i][1]=(5/4.0)*(max-min)*(arrayData[i][1]/1023.0-
    1)+max;
67.                 }
68.                 else {
69.                     arrayData[i][1]=(max-min)*arrayData[i][1]/1023.0+min;
70.                 }
71.
72.             }
73.             var data = new google.visualization.DataTable();
74.             data.addColumn('datetime', arrayData[0][0]);
75.             data.addColumn('number', arrayData[0][1]);
76.
77.             var options = {
78.                title: "Ewen Sensor #"+ns,
79.                hAxis: {title: data.getColumnLabel(0), minValue: data.getColumnR
    ange(0).min, maxValue: data.getColumnRange(0).max},
80.                vAxis: {title: data.getColumnLabel(1), minValue: data.getColumnR
    ange(1).min, maxValue: data.getColumnRange(1).max},
81.                curveType: 'function',
82.                legend: 'none',
83.                width: 1050,
84.                height: 500,
85.                animation: {duration: 1}
86.             };
87.
88.             var chart = new google.visualization.LineChart(document.getElementB
    yId('chart'));
89.             var index = 1;
90.             var drawChart1 = function() {
91.                 if (index <arrayData.length ) {
92.                     data.addRow(arrayData[index]);
93.                     index++;
94.                     chart.draw(data, options);
95.                 }
96.             }
97.
98.             google.visualization.events.addListener(chart, 'animationfinish', d
    rawChart1);
99.             chart.draw(data, options);
100.             drawChart1();
101.          });
102.       };
103.   </script>
104.</head>
105.<body>
106.    <h1 align="center" class="title">Sensor Graphics</h1>
107.   <div id="chart">
108.   </div>
109.</body>
110.</html>
```

Anexo 9 – sconfig.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Configurate Sensor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
img
{
position:relative;
top:2px;
}
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<style type="text/css">
<!--
span.tab {padding-left: 1.5em;}
-->
</style>
<script>
    strInputs = "set";
    var check=["1","1","1","1","1","1"];
    function Getinputs()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        var count;
                        for (count = 0; count < 6; count++) {
                            check[count] = this.responseXML.getElementsByTagName('check')[count].childNodes[0].nodeValue;
                        }
                    }
                }
            }
        }
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
    }

    function Setsens(ns)
    {
        strInputs += "ns="+ns+"&";
        Getinputs();
        location.href="config.htm";
    }
</script>
</head>
<body>
```

```
59. <h1 align="center" class="title">Configurate Sensors</h1>
60. <ul>
61.     <script>
62.         Getinputs();
63.         var i;
64.         for (i=0; i<6; i++)
65.         {
66.             document.write("<li><a href=\"javascript:Setsens("+i+");\">Sensor #
    "+i+"</a><span class=\"tab\"></span>");
67.             if (check[i]==1)
68.             {
69.                 document.write("<img width=\"15\" height=\"15\" src=\"http://up
    load.wikimedia.org/wikipedia/commons/thumb/4/4b/Green_Light_Icon.svg/120px-
    Green_Light_Icon.svg.png\"></li>");
70.             }
71.             else
72.             {
73.                 document.write("<img width=\"15\" height=\"15\" src=\"http://up
    load.wikimedia.org/wikipedia/commons/thumb/1/1f/Red_Light_Icon.svg/120px-
    Red_Light_Icon.svg.png\"></li>");
74.             }
75.         }
76.     </script>
77. </ul>
78. </body>
79. </html>
```

Anexo 10 – config.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Ewen Configuration</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
.para {color: #017bb1; font-size: : 12px;}
.sub {color: #017bb1; font-family: arial; font-size: : 12px; font-weight: bold;}
body
{background-image:url("http://subtlepatterns.com/patterns/old_moon.png");}
</style>
<script>
    strInputs = "set";
    function Getinputs()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        ns = this.responseXML.getElementsByTagName('ns')[0].childNodes[0].nodeValue;
                        wtipo = this.responseXML.getElementsByTagName('tipo')[0].childNodes[0].nodeValue;
                        wmax = this.responseXML.getElementsByTagName('max')[0].childNodes[0].nodeValue;
                        wmin = this.responseXML.getElementsByTagName('min')[0].childNodes[0].nodeValue;
                        check = this.responseXML.getElementsByTagName('check')[parseInt(ns)].childNodes[0].nodeValue;
                    }
                }
            }
        }
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
    }

    function GetCheck(Oform)
    {
        if (Oform.elements["check"].checked)
        {
            Oform.elements["check"].value=1
            Oform.elements["type"].disabled=false;
            Oform.elements["max"].disabled=false;
            Oform.elements["min"].disabled=false;
        }
        else
        {
            Oform.elements["check"].value=0
            Oform.elements["type"].disabled=true;
            Oform.elements["max"].disabled=true;
```

```
54.             Oform.elements["min"].disabled=true;
55.         }
56.
57.     }
58.
59.     function Setinputs(Oform)
60.     {
61.         strInputs += "t="+Oform.elements["type"].value;
62.         strInputs += "&maxmax="+Oform.elements["max"].value;
63.         strInputs += "&minmin="+Oform.elements["min"].value;
64.         strInputs += "&c="+Oform.elements["check"].value;
65.         strInputs +="&"
66.         Getinputs();
67.         location.reload();
68.     }
69. </script>
70. </head>
71. <body>
72. <h1 align="center" class="title">Sensors Configurations</h1>
73. <br>
74. <form id="myform" name="input_form">
75. <table>
76. <script language="JavaScript">
77.     Getinputs();
78.     var tipo;
79.     y="0-5 V,0-10 V,0-20 mA,4-20 mA";
80.     Y=y.split(",",4);
81.     document.write("<tr><td class=\"sub\">Sensor #"+ns+"</td></tr><br>");
82.     document.write("<tr><td class=\"para\">Active</td><td>");
83.     if (check==1)
84.     {
85.         document.write("<input name=check type=checkbox value=1 onclick=\"GetCh
    eck(this.form)\" checked>");
86.     }
87.     else
88.     {
89.         document.write("<input name=check type=checkbox value=0 onclick=\"GetCh
    eck(this.form)\">");
90.     }
91.     document.write("</td></tr>");
92.     document.write("<tr><td class=\"para\">Sensor Type</td><td><select name=typ
    e>");
93.     for(tipo=0; tipo < 4 ; tipo++){
94.         if (tipo == wtipo)
95.         {
96.           document.write("<option value="+tipo+" selected>"+Y[tipo]+"</option>"
    );
97.         }
98.         else
99.         {
100.             document.write("<option value="+tipo+">"+Y[tipo]+"</option>");
101.         }
102.     }
103.     document.write("</select></td></tr>");
104.     document.write("<tr><td class=\"para\">Maximum</td><td><input name=max typ
    e=text size=6 maxlength=6 value="+wmax+"></td></tr>");
105.     document.write("<tr><td class=\"para\">Minimum</td><td><input name=min typ
    e=text size=6 maxlength=6 value="+wmin+"></td></tr><br>");
106.     document.write("<tr><td align=center><input type=button value=\"Change Sen
    sor "+ns+ "\"onClick=\"Setinputs(this.form)\"></td></tr>");
107.     document.write("</table></form>");
108.     GetCheck(document.forms["myform"]);
109. </script>
110. </body>
111. </html>
```

Anexo 11 – tconfig.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Ewen Time Configuration</title>
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
.para {color: #017bb1; font-size: : 12px;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script>
    strInputs = "set";
    function Getinputs()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        period=this.responseXML.getElementsByTagName('per')[0].childNodes[0].nodeValue;
                        freq=this.responseXML.getElementsByTagName('freq')[0].childNodes[0].nodeValue;
                    }
                }
            }
        }
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
    }

    function Settime(Oform)
    {
        strInputs += "per="+Oform.elements["tim"].value;
        strInputs += "&freq="+Oform.elements["frq"].value;
        strInputs +="&";
        Getinputs();
        location.reload();
    }
</script>
</head>
<body>
<h1 align="center" class="title">Time Configurations</h1>
<br>
<form id="myforms2" name="input_forms2">
    <table>
    <tr>
    <td class="para">Time Period of File</td>
    <script language="JavaScript">
        Getinputs();
        var i;
```

```
58.         t="Day,Week,Month";
59.         T=t.split(",",3);
60.         document.write("<td><select name=tim>");
61.         for (i=0; i<3; i++)
62.         {
63.             if (period==i)
64.             {
65.                 document.write("<option value="+i+" selected>"+T[i]+"</option>"
    );
66.             }
67.             else
68.             {
69.                 document.write("<option value="+i+">"+T[i]+"</option>");
70.             }
71.         }
72.         document.write("</select></td></tr>");
73.         document.write("<tr><td class=\"para\">Frequency of Printing</td><td><i
    nput name=frq type=text size=6 maxlength=6 value="+freq+"> (Every "+freq+" seco
    nds)</td></tr>");
74.         document.write("<br><tr><td align=center><input type=button value=\"Cha
    nge Time Parameters\" onClick=\"Settime(this.form)\"></td></tr>");            <
    /script>
75.     </table>
76. </form>
77. </body>
78. </html>
```

Anexo 12 – mconfig.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Ewen Mode Configuration</title>
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-
weight: bold;}
.para {color: #017bb1; font-size: : 12px;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script>
    strInputs = "set";
    function Getinputs()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        mode=this.responseXML.getElementsByTagName('mode')[0].c
hildNodes[0].nodeValue;
                    }
                }
            }
        }
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
    }

    function Setmode(Oform)
    {
        strInputs += "modmode="+Oform.elements["mod"].value;
        strInputs +="&";
        Getinputs();
        location.reload();
    }
</script>
</head>
<body>
<h1 align="center" class="title">Operation Mode Configuration</h1>
<br>
<form id="myforms1" name="input_forms1">
    <table>
    <tr>
    <td class="para">Mode of Operation</td>
    <script language="JavaScript">
        Getinputs();
        var i;
        m="Saving Files,Send to Ewen Server,Both";
        M=m.split(",",3);
        document.write("<td><select name=mod>");
```

```
59.         for (i=0; i<3; i++)
60.         {
61.             if (mode==i)
62.             {
63.                 document.write("<option value="+i+" selected>"+M[i]+"</option>"
    );
64.             }
65.             else
66.             {
67.                 document.write("<option value="+i+">"+M[i]+"</option>");
68.             }
69.         }
70.         document.write("</select></td></tr><br>");
71.         document.write("<tr><td align=center><input type=button value=\"Change
    Mode of Operation\" onClick=\"Setmode(this.form)\"></td></tr>");
72.     </script>
73.     </table>
74. </form>
75. </body>
76. </html>
```

Anexo 13 – network.htm

```
<!DOCTYPE html>
<html>
<head>
<title>Ewen Configuration</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
a:link    {color:blue;}
a:visited {color:blue;}
a:hover   {color:green;}
.title {color: #017bb1; font-family: arial; font-size: : 20px; font-weight: bold;}
.para {color: #017bb1; font-size: : 12px;}
body
{
background-image:url("http://subtlepatterns.com/patterns/old_moon.png");
}
</style>
<script>
    var ip=["0","0","0","0"];
    var dnsg=["0","0","0","0"]
    var mac=["0","0","0","0","0","0"];
    strInputs = "set";
    function Getinputs()
    {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function()
        {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    if (this.responseXML != null) {
                        var count;
                        for (count = 0; count < 6; count++) {
                            if (count<4)
                            {
                                ip[count] = this.responseXML.getElementsByTagName('ip')[count].childNodes[0].nodeValue;
                                dnsg[count] = this.responseXML.getElementsByTagName('dnsg')[count].childNodes[0].nodeValue;
                            }
                            mac[count] = this.responseXML.getElementsByTagName('mac')[count].childNodes[0].nodeValue;
                        }
                    }
                }
            }
        }
        request.open("GET", strInputs, false);
        request.send(null);
        strInputs = "set";
    }

    function Setinputs(Oform)
    {
        strInputs += "ip="+Oform.elements["ip0"].value+".";
        strInputs += Oform.elements["ip1"].value+".";
        strInputs += Oform.elements["ip2"].value+".";
        strInputs += Oform.elements["ip3"].value;
        strInputs += "&d="+Oform.elements["dnsg0"].value+".";
        strInputs += Oform.elements["dnsg1"].value+".";
        strInputs += Oform.elements["dnsg2"].value+".";
```

```
57.         strInputs += Oform.elements["dnsg3"].value;
58.         strInputs += "&mac="+Oform.elements["mac0"].value+".";
59.         strInputs += parseInt(Oform.elements["mac1"].value, 16)+".";
60.         strInputs += parseInt(Oform.elements["mac2"].value, 16)+".";
61.         strInputs += parseInt(Oform.elements["mac3"].value, 16)+".";
62.         strInputs += parseInt(Oform.elements["mac4"].value, 16)+".";
63.         strInputs += parseInt(Oform.elements["mac5"].value, 16);
64.         strInputs +="&"
65.         Getinputs();
66.         location.reload();
67.     }
68. </script>
69. </head>
70. <body>
71. <h1 align="center" class="title">Network Configurations</h1>
72. <br>
73. <form id="myform" name="input_form">
74. <table>
75. <script language="JavaScript">
76.     Getinputs();
77.     document.write("<tr><td class=\"para\">IP address: </td><td><input name=ip0
     type=text size=1 maxlength=3 value="+ip[0]+">.");
78.     document.write("<input name=ip1 type=text size=1 maxlength=3 value="+ip[1]+
    ">.");
79.     document.write("<input name=ip2 type=text size=1 maxlength=3 value="+ip[2]+
    ">.");
80.     document.write("<input name=ip3 type=text size=1 maxlength=3 value="+ip[3]+
    "></td></tr>");
81.     document.write("<tr><td class=\"para\">Gateway/DNS: </td><td><input name=dn
    sg0 type=text size=1 maxlength=3 value="+dnsg[0]+">.");
82.     document.write("<input name=dnsg1 type=text size=1 maxlength=3 value="+dnsg
    [1]+">.");
83.     document.write("<input name=dnsg2 type=text size=1 maxlength=3 value="+dnsg
    [2]+">.");
84.     document.write("<input name=dnsg3 type=text size=1 maxlength=3 value="+dnsg
    [3]+"></td></tr>");
85.     document.write("<tr><td class=\"para\">MAC address: </td><td><input name=ma
    c0 type=text size=1 maxlength=2 value="+mac[0]+">-");
86.     document.write("<input name=mac1 type=text size=1 maxlength=2 value="+mac[1
    ]+">-");
87.     document.write("<input name=mac2 type=text size=1 maxlength=2 value="+mac[2
    ]+">-");
88.     document.write("<input name=mac3 type=text size=1 maxlength=2 value="+mac[3
    ]+">-");
89.     document.write("<input name=mac4 type=text size=1 maxlength=2 value="+mac[4
    ]+">-");
90.     document.write("<input name=mac5 type=text size=1 maxlength=2 value="+mac[5
    ]+"></td></tr><br>");
91.     document.write("<tr><td align=center><input type=button value=\"Change Netw
    ork Settings\" onClick=\"Setinputs(this.form)\"></td></tr>");
92.     document.write("</table></form>");
93. </script>
94. </body>
95. </html>
```