UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE
ESCUELA POLITÉCNICA SUPERIOR DE ORIHUELA

Master Universitario Oficial en Automatización y Telecontrol para la gestión de Recursos Hídricos y Energéticos

MONITORIZACIÓN Y DETECCIÓN DE FALLOS EN UNA INSTALACIÓN SOLAR FOTOVOLTAICA MEDIANTE SISTEMA REMOTO

TRABAJO FIN DE MASTER

AUTOR:

José Sesma Martínez

DIRECTOR/ES:

Antonio Ruiz Canales

José Miguel Molina Martínez

Enero 2015

Índice de contenidos

INTRODUCCIÓN Y OBJETIVOS **5**

1.1. INTRODUCCIÓN 6
1.1.1 Objetivo del proyecto *7*

ENERGÍA SOLAR FOTOVOLTAICA **9**

2.1. INTRODUCCIÓN 10
2.2. CONCEPTOS BÁSICOS 11
2.3. PANELES FOTOVOLTAICOS 14
2.4. INSTALACIÓN SOLAR FOTOVOLTAICA AISLADA 17

MATERIALES Y MÉTODOS **21**

3.1. INTRODUCCIÓN 22
3.2. HARDWARE 22
3.2.1 Raspberry Modelo B *22*
3.2.2 Ordenador para desarrollo web *28*
3.2.3 Maqueta solarfotovoltaica *29*
3.3. SOFTWARE 32
3.3.1 Nodejs *32*
3.3.2 Express js *33*
3.3.3 MongoDB *43*
3.3.4 HighChart *45*
3.3.5 Python *46*
3.3.6 Librería py-spidev *47*
3.4. SERVICIOS CONTRATADOS 48
3.4.1 Servidor VPS *48*
3.4.2 Dominio *48*
3.5. CÓDIGO WEB 49
3.5.1 Modelo de la base de Datos *49*
3.5.1.1 Index 49
3.5.1.2 Datos 50
3.5.1.3 Estación 52
3.5.1.4 Usuarios 53
3.5.2 Aplicación web (Backend) *54*
3.5.2.1 Dependencias 54
3.5.2.2 Configuración servidor y base de datos 55
3.5.2.3 Configuración sockets 58
3.5.2.4 Rutas 59
3.5.3 Aplicación Raspberry-Pi *72*

RESULTADOS OBTENIDOS **79**

4.1. INTRODUCCIÓN. 80
4.2. PANTALLA LOGIN 80
4.3. PANTALLA REGISTRO 81
4.4. PANTALLA USUARIO REGISTRADO 82
4.5. PANTALLA INICIAL 82
4.6. PANTALLA TIEMPO REAL 83
4.7. PANTALLA HISTÓRICO 84
4.8. PANTALLA CREAR ESTACIÓN 85
4.9. PANTALLA MIS ESTACIONES 86
4.10. PANTALLA DATOS DE USUARIO 86

CONCLUSIONES **89**

5.1. CONCLUSIONES 90
5.2. TRABAJOS FUTUROS 91

PRESUPUESTO **93**

ANEXO_A REFERENCIAS **97**

7.1. REFERENCIAS 98

Índice de figuras

FIGURA 2.2.1. MAPA MUNDIAL DE RADIACIÓN SOLAR....13
FIGURA 2.3.1. COMPOSICIÓN PANEL SOLAR....15
FIGURA 2.4.1. ESQUEMA DE UNA INSTALACIÓN FOTOVOLTAICA AUTÓNOMA....17
FIGURA 3.2.1. RASPBERRY PI....22
FIGURA 3.2.2 PINES RASPBERRY PI....25
FIGURA 3.2.3 CIRCUITO BLINK LED....26
FIGURA 3.2.4 CÓDIGO BLINK LED....27
FIGURA 3.2.5. ORDENADOR ASUS....28
FIGURA 3.2.6. PLACA SOLAR....30
FIGURA 3.2.7. REGULADOR DE CARGA....30
FIGURA 3.2.8. RASPBERRY PI....31
FIGURA 3.3.1. NODE.JS....32
FIGURA 3.3.2 EXPRESS JS - AYUDA....34
FIGURA 3.3.3 ESTRUCTURA DE ARCHIVOS EXPRESS....34
FIGURA 3.3.4 PRIMERA APLICACIÓN EXPRESS JS....35
FIGURA 3.3.5 CONFIGURACION EXPRESS....36
FIGURA 3.3.6 CONFIGURACIÓN PERSONALIZADA EXPRESS....36
FIGURA 3.3.7 CONFIGURACIÓN PERSONALIZADA EXPRESS ENVIRONMENT....37
FIGURA 3.3.8 CONFIGURACIÓN BASE DE DATOS EXPRESS....37
FIGURA 3.3.9 CONFIGURACIÓN RUTAS EXPRESS....37
FIGURA 3.3.10 CONFIGURACIÓN RUTA EXPRESS GENERAL....38
FIGURA 3.3.11 EJEMPLO POST EXPRESS....38
FIGURA 3.3.12 EJEMPLO GET EXPRESS....38
FIGURA 3.3.13 DEFINIENDO EL FORMATO EN EXPRESS....39
FIGURA 3.3.14 HELPERS EN EXPRESS....40
FIGURA 3.3.15 SISTEMA DE ERRORES EXPRESS....40
FIGURA 3.3.16 SESIONES EN EXPRESS....41
FIGURA 3.3.17 MULTIPLE RESPONSE EXPRESS....41
FIGURA 3.3.18 MULTIPLE ENVIO RES.SEND EXPRESS....41
FIGURA 3.3.19 RENDERIZAR VISTA EXPRESS....42
FIGURA 3.3.20. MONGODB....43
FIGURA 3.3.21. HIGHCHARTS....45
FIGURA 3.3.22. PYTHON....46
FIGURA 4.2.1 PANTALLA DE LOGIN....81
FIGURA 4.3.1. PANTALLA DE REGISTRO....81
FIGURA 4.4.1. PANTALLA USUARIO REGISTRADO....82
FIGURA 4.5.1. PANTALLA INICIAL....82
FIGURA 4.6.1. PANTALLA TIEMPO REAL....83
FIGURA 4.7.1. PANTALLA HISTÓRICO....84
FIGURA 4.7.2. PANTALLA EXPORTAR GRÁFICA....84
FIGURA 4.8.1. PANTALLA CREAR ESTACIÓN....85
FIGURA 4.9.1. PANTALLA "MIS ESTACIONES"....86
FIGURA 4.10.1. PANTALLA DATOS DE USUARIO....86

1

Introducción y objetivos

En este primer capítulo se expondrá el objetivo del proyecto, así como las metas fijadas para su consecución. Una vez expuesto el objetivo y las metas, se pasará a enumerar los diferentes capítulos de los que consta esta memoria.

1.1. Introducción

Actualmente existen en el mercado una gran variedad de soluciones para aprovechar la energía solar. En este caso, este trabajo se va a centrar en la obtención de energía eléctrica a través de paneles fotovoltaicos.

Uno de los problemas que pueden surgir en la instalación fotovoltaica es que se produzca cualquier alteración en la misma que impida entregar el máximo rendimiento de potencia que debería suministrar. Esto puede ser debido al fallo de algún componente, como puede ser el inversor, la batería, la placa o una mala orientación.

Para localizar este tipo de fallo es necesario desplazarse al lugar de la instalación y realizar las mediciones oportunas para detectar el fallo.

Con este trabajo se pretende ahorrar los costes de desplazamiento del operario, así como la detección del fallo con la mayor brevedad posible. Esto es posible gracias a la monitorización en tiempo real que, mediante gráficas, nos permite visualizar de una forma muy atractiva el funcionamiento de diversos parámetros de nuestra instalación y conseguir así la detección del problema instantáneamente así como la máxima eficiencia energética.

Para el desarrollo de nuestro trabajo, inicialmente se realizará un pequeño análisis sobre los diferentes tipos de placas fotovoltaicas que existen en el mercado con el fin de obtener una comparativa entre ellas y estudiar sus diversos usos.

El software generado se conectará con un servidor externo que será el encargado de almacenar los valores proporcionados por los sensores. Estos datos serán mostrados en una página web, haciéndose visibles desde cualquier parte del mundo, mediante una conexión a internet.

Se va a realizar, a su vez, una maqueta de simulación para representar la instalación solar fotovoltaica. A ésta se le conectarán diversos sensores y una placa Raspberry Pi, o una placa Arduino, que será la encargada de la obtención y procesamiento de los datos y hará de "puente" entre nuestra instalación y nuestro servidor web.

Este proyecto es fácilmente escalable a instalaciones comerciales más grandes, pudiendo proporcionar al usuario un mayor control y una mayor gestión de su instalación.

1.1.1 Objetivo del proyecto

El objetivo de este trabajo es la realización de un estudio de una instalación solar fotovoltaica, así como su monitorización a través de una página web que mostrará los datos más relevantes de la instalación, tales como la tensión, la temperatura, la luminosidad, etc.

Mediante esta combinación de hardware y software se puede monitorizar la instalación solar fotovoltaica remotamente desde cualquier navegador web, permitiendo al usuario tener un control sobre la instalación en tiempo real, así como detectar fallos de funcionamiento en la misma y conseguir mejorar la eficiencia.

Este hardware estará basado en tecnología de bajo coste al usar una Raspberry Pi. Se ha implementado en una placa fotovoltaica de baja potencia instalada en una maqueta de simulación, pudiéndose escalar a placas más grandes para el estudio en una instalación comercial.

La página web nos muestra unas gráficas a tiempo real de la tensión, la temperatura, y la luminosidad, las cuales nos permiten detectar de una manera rápida cualquier fallo que se produzca en nuestra instalación.

A su vez disponemos de un histórico de cada una de las estaciones con el fin de observar la evolución de las variables estudiadas en casa caso. No obstante se dispone de un panel de control mediante el cual se puede controlar las diferentes salidas conectadas a nuestro dispositivo de control.

El usuario es capaz de introducir nuevas instalaciones a su cuenta y posicionarlas dentro de un mapa para tener siempre controlada su posición geográfica, dato muy importante para conocer las horas de luz de las que va a disponer nuestra instalación.

No obstante, cada usuario tiene disponible las estaciones que él haya añadido y de sus gráficas accesibles solo mediante un nombre de usuario y una contraseña creada previamente por el usuario.

2

Energía Solar Fotovoltaica

En el presente capítulo se recopila la información relacionada con la energía solar fotovoltaica así como sus usos y tipos.

2.1. Introducción

La energía es "la medida de la capacidad de un sistema para proporcionar trabajo por medios mecánicos o calor por medios no mecánicos"

El Sol, con una potencia media de 3,7 * 1014 TW, de la que llega a la superficie 173.000 TW (o lo que es lo mismo, 900 W / m2) constituye sin duda alguna una fuente de energía formidable. Tiene un papel fundamental entre las diferentes energías renovables conocidas hoy en día, como lo demuestra la siguiente tabla:

Y teniendo en cuenta el creciente aumento del consumo de energía en el mundo, se puede prever que esta energía es una energía de futuro:

Energía	Recurso (en tep por año)
Hidráulica	$1.7 \cdot 10^9$
Solar	$9.8 \cdot 10^{13}$
Eólica	$1.4 \cdot 10^{10}$
Biomasa	$2.8 \cdot 10^9$
Geotérmica	$2.3 \cdot 10^{16}$
Maremotriz	$1.9 \cdot 10^9$
Maremotérmica	$2.8 \cdot 10^{13}$
Olas	$1.7 \cdot 10^9$

La energía solar es una energía garantizada para los próximos 6.000 millones de años.

El Sol ha brillado en el cielo desde hace unos cinco mil millones de años, y se calcula que todavía no ha llegado a la mitad de su existencia. Es fuente de vida y origen de las demás formas de energía que el hombre ha utilizado desde los albores de la Historia, y puede satisfacer todas nuestras necesidades si aprendemos cómo aprovechar de forma racional la luz que continuamente derrama sobre el planeta. Es una fuente de energía inagotable, por su magnitud y porque su fin será el fin de la vida en la Tierra.

Durante el presente año, el Sol arrojará sobre la Tierra cuatro mil veces más energía que la que vamos a consumir. No sería racional no intentar aprovechar, por todos los medios técnicamente posibles, esta fuente energética gratuita, limpia e inagotable, que puede liberarnos definitivamente de la dependencia del petróleo o de otras alternativas poco seguras o, simplemente, contaminantes.

Es preciso señalar que existen algunos problemas que debemos afrontar y superar.

Aparte de las dificultades que una política energética solar avanzada conllevaría por sí misma, hay que tener en cuenta que esta energía está sometida a continuas variaciones más o menos bruscas. Así, por ejemplo, la radiación solar es menor en invierno, precisamente cuando más la necesitamos.

Es de vital importancia proseguir con el desarrollo de la incipiente tecnología de captación, acumulación y distribución de la energía solar, para conseguir las condiciones que la hagan definitivamente competitiva, a escala planetaria.[1] (2009, Vegas Portero, A.)

2.2. Conceptos básicos

La energía solar fotovoltaica es aquella que se obtiene por medio del proceso directo de transformación de la energía del sol en energía eléctrica.

El proceso de transformación de la energía del sol se puede llevar a cabo de dos maneras:

- En la primera, se utiliza una parte del espectro electromagnético de la energía del sol para producir calor. A la energía obtenida se le

llama energía solar térmica. La transformación se realiza mediante el empleo de colectores térmicos.

- En la segunda se utiliza la otra parte del espectro electromagnético de la energía del sol para producir electricidad. A la energía obtenida se le llama energía solar fotovoltaica. La transformación se realiza por medio de módulos o paneles solares fotovoltaicos.

La energía solar fotovoltaica se puede utilizar para hacer funcionar lámparas eléctricas, para iluminación o para hacer funcionar radios, televisores y otros electrodomésticos de bajo consumo energético, generalmente, en aquellos lugares donde no existe acceso a la red eléctrica convencional. No obstante para instalaciones para instalaciones de tamaño mediano o grande, se puede plantear la utilización de la energía solar fotovoltaica de cara a la producción de energía eléctrica para su introducción en las redes de distribución y transporte eléctrico, empleando en ese caso una fuente de energía renovable y absolutamente limpia.

Es necesario disponer de un sistema forzado por equipos especialmente construidos para realizar la transformación de la energía solar en energía eléctrica. Este sistema recibe el nombre de **sistema fotovoltaico** y los equipos que lo forman reciben el nombre de componentes fotovoltaicos. Los componentes que forman el sistema fotovoltaico van a ser distintos en función del uso de la instalación.

La energía solar se encuentra disponible en todo el mundo. Algunas zonas del planeta reciben más radiación solar que otras lo que convierte a estas en preferentes. Sin embargo, los sistemas fotovoltaicos tienen muchas aplicaciones, tales como la alimentación de sistemas de emergencia o alumbrado aislados, que son factibles en cualquier lugar.

En el caso particular de España, los sistemas fotovoltaicos son una alternativa muy interesante, desde la perspectiva técnica, pues la región dispone durante todo el año de abundante radiación solar.

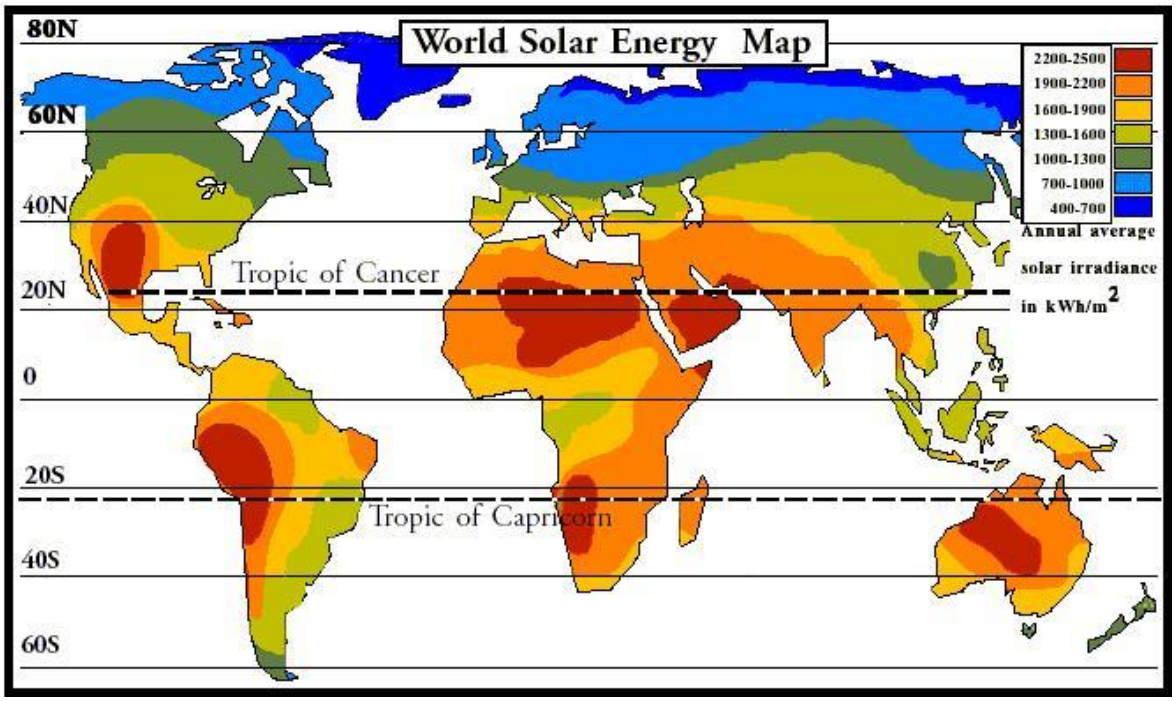

Figura 2.2.1. Mapa mundial de radiación solar

En la célula fotovoltaica comienza la generación de corriente continua, tan pronto como la luz del sol incide sobre su superficie. Estas células se basan en las propiedades de los materiales semiconductores como el silicio.

La mencionada generación eléctrica tiene lugar sin que sea necesaria la intervención de ningún tipo de componente mecánico o móvil, ni de ningún proceso o tipo químico o termodinámico, por lo que las células disfrutan de una vida útil muy prolongada, de hasta más de 30 años.

Muy generalmente, al incidir la luz del sol sobre la superficie de la célula fotovoltaica, los fotones de luz van a transmitir su energía a los electrones del semiconductor para que así puedan circular dentro del sólido. La tecnología fotovoltaica consigue que parte de estos electrones salgan al exterior del material semiconductor generándose así una corriente eléctrica capaz de circular por un circuito externo.

Las células solares se unen eléctricamente unas con otras, y, tras realizar un posterior encapsulado y enmarcado sobre el conjunto con objeto de proporcionar la necesaria resistencia a la intemperie, se llega a la obtención de los conocidos paneles o módulos fotovoltaicos. [2]

Los módulos se pueden conectar a su vez entre sí, en lo que se denomina sistemas de energía solar, formando un número conveniente de ramas o cadenas. De este modo es posible realizar, tanto pequeños equipos de baja potencia (con solo unos pocos watios) como instalaciones de mayor tamaño (Centrales de varios megawatios).

El rendimiento de un panel fotovoltaico depende de algunas variables externas como la radiación solar, la temperatura de funcionamiento y la orientación de este panel frente al sol y, además, de la calidad con la que esté fabricado el panel y parámetros de mantenimiento como la suciedad, envejecimiento, etc.

Los sistemas de energía solar fotovoltaica se sustentan en una tecnología de vanguardia, tomando como base en conglomerado industrial que, en el caso de España, está a la cabeza mundial en el campo de fabricación tanto de los paneles como del resto de componentes y de las aplicaciones.

2.3. Paneles fotovoltaicos

El silicio se emplea como materia prima en aproximadamente el 90% de los módulos fotovoltaicos, tanto para paneles fabricados con la tecnología cristalina, como de forma esporádica en los receptores de lámina delgada basados en el silicio amorfo.

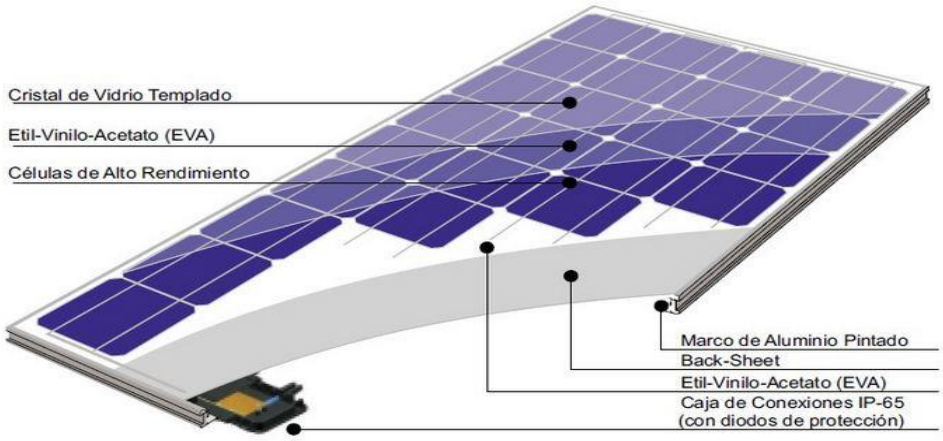

Figura 2.3.1. Composición panel solar

El uso del silicio cristalino se encuentra más extendido ya que, a pesar de que su proceso de elaboración sea más costoso y complicado, se obtiene una mayor eficiencia.

Un aspecto a tener en cuenta sobre la utilización de tecnologías de silicio es la obtención de la materia prima. Para la fabricación de los paneles fotovoltaicos, es el silicio que la industria electrónica no utiliza, el que sirve como materia prima para producir silicio cristalino de grado solar. De este modo, el desarrollo de la industria fotovoltaica viene condicionado por la industria de la electrónica, ya que la materia prima que emplea ésta es un subproducto de menor pureza a un coste sensiblemente inferior.

Hay que considerar al panel fotovoltaico como un elemente de producción de energía, ya que, a lo largo de su vida útil, produce muchas más energía de la que consume, y, además, la obtiene a partir de una fuente inagotable y no contaminante. Los principales consumos energéticos se producen en la fabricación del módulo y de la estructura de montaje, considerándose que resulta favorable su balance energético con un periodo de recuperación de la energía consumida o "pay-back" que actualmente es de 3 a 5 años.

Componentes:

Un panel solar está constituido por varias células iguales conectadas eléctricamente entre sí, en serie y/o en paralelo, de forma que la tensión y corriente suministrada por el panel se incrementa hasta ajustarse al valor

deseado. La mayor parte de los paneles solares se construyen asociando primero células en serie hasta conseguir el nivel de tensión deseado, y luego asociando en paralelo varias asociaciones serie de células para alcanzar el nivel de corriente deseado. Además, el panel cuenta con otros elementos a parte de las células solares, que hacen posible la adecuada protección del conjunto frene a los agentes externos; asegurando una rigidez suficiente, posibilitando la sujeción a las estructuras que lo soportan y permitiendo la conexión eléctrica.

Estos elementos son:

- Cubierta exterior de cara al Sol. Es de vidrio que debe facilitar al máximo la transmisión de la radiación solar. Se caracteriza por su resistencia mecánica, alta transmisividad y bajo contenido en hierro.

- Encapsulante. De silicona o más frecuentemente EVA (etilen-vinil-acetato). Es especialmente importante que no quede afectado en su transparencia por la continua exposición al sol, buscándose además un índice de refracción similar al del vidrio protector para no alterar las condiciones de la radiación incidente.

- Protección posterior. Igualmente debe dar rigidez y una gran protección frente a los agentes atmosféricos. Usualmente se emplean láminas formadas por distintas capas de materiales, de diferentes características.

- Marco metálico. De Aluminio, que asegura una suficiente rigidez y estanqueidad al conjunto, incorporando los elementos de sujeción a la estructura exterior del panel. La unión entre el marco metálico y los elementos que forman el modulo está realizada mediante distintos tipos de sistemas resistentes a las condiciones de trabajo del panel.

- Cableado y bornas de conexión. Habituales en las instalaciones eléctricas, protegidos de la intemperie por medio de cajas estancas.

- Diodo de protección. Su misión es proteger contra sobre-cargas u otras alteraciones de las condiciones de funcionamiento de panel.

Los Panel solares tienen entre 28 y 40 células, aunque lo más típico es que cuenten con 36. La superficie del panel o modulo puede variar entre 0.1 y 0.5m2 y presenta dos bornas de salida, positiva y negativa, a veces tienen alguna intermedia para colocar los diodos de protección.

2.4. Instalación solar fotovoltaica aislada.

Una instalación fotovoltaica está compuesta por un grupo generador, formado por una extensión de paneles solares fotovoltaicos, un regulador de carga, un grupo acumulador y un inversor.

Durante las horas de insolación, los paneles fotovoltaicos producen energía eléctrica en forma de corriente continua que es almacenada en los acumuladores. En los momentos de consumo energético, los acumuladores suministran a los receptores esta electricidad, que es transformada en corriente alterna por el inversor. [3]

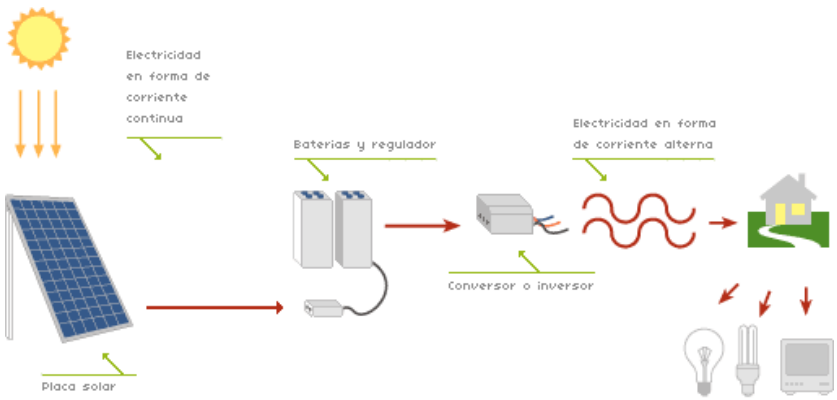

Figura 2.4.1. Esquema de una instalación fotovoltaica autónoma

Placas solares

La energía solar se encuentra almacenada en partículas de luz: los fotones.

Las placas o módulos solares fotovoltaicos usan ciertos materiales semiconductores, como el silicio, que absorben los fotones y los convierten en una corriente continua de electrones, es decir, en electricidad. Esta electricidad se recoge mediante unos hilos metálicos que al final la conducen hacia el regulador.

Regulador de carga

Controla la entrada de electricidad en la batería y la protege de sobrecargas o bajadas de tensión que podrían dañarla. Los modelos avanzados ponen en marcha el grupo electrógeno ara producir electricidad cuando la batería corre riesgos.

Baterías y cargador

Es necesario inyectar la energía en baterías para que se encuentre disponible cuando haga falta, generalmente por la noche. Es lo que ocurre con las instalaciones fotovoltaicas en viviendas unifamiliares, o en circunstancias en las que existe más demanda de potencia de la que dan las placas.

Principalmente, existen dos tipos de baterías:

- Monobloque: similares a las del automóvil; son más económicas pero tienen un mantenimiento más complejo y una menor duración
- Estacionarias: resultan más apropiadas para estos sistemas ya que su durabilidad y versatilidad es mayor. Se pueden conectar en serie.

Antes de que llegue al 80% de descarga, para evitar que se estropee, se debe de recargar la batería.

Ondulador, convertidor o inversor

Sirve para convertir la corriente continua producida por el campo fotovoltaico en corriente alterna de onda senoidal, que es la única que se puede usar en la alimentación de electrodomésticos convencionales. Con él, se suele poner en marcha el grupo electrógeno.

3

Materiales y Métodos

En este capítulo se describe el material utilizado para el desarrollo de este trabajo así como los métodos y software desarrollados.

3.1. Introducción

Para llevar a cabo este proyecto se ha hecho una investigación previa de las tecnologías disponibles actualmente en materia de paneles fotovoltaicos así como de las tecnologías en desarrollo web y almacenamiento de datos.

En la actualidad hay varias empresas que proporcionan servicios parecidos al que se pretende ofrecer en este proyecto, no obstante requiere un hardware caro (PLCs) y de una estación de procesado dentro de la instalación fotovoltaica.

Con esta aplicación se pretende solventar este problema y poder estar al alcance de todos la monitorización de las instalaciones.

Una vez que se ha diseñado la estructura de la web y de la maqueta y se ha validado su uso, se han comenzado el desarrollo y la contratación de los servicios necesarios.

3.2. Hardware

3.2.1 Raspberry Modelo B

Figura 3.2.1. Raspberry Pi

Raspberry Pi es una placa computadora (SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos Turbo para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM aunque originalmente al ser lanzado eran 256 MB. El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa.

Características Técnicas:

Característica	Modelo B
SoC:	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11)
Juego de instrucciones:	RISC de 32 bits
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM):	512 MiB (compartidos con la GPU)[4] desde el 15 de octubre de 2012
Puertos USB 2.0:	2 (vía hub USB integrado)

Entradas de vídeo:	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
Salidas de vídeo:	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
Salidas de audio:	Conector de 3.5 mm, HDMI
Almacenamiento integrado:	MicroSD
Conectividad de red:[5]	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel:	8 x GPIO, SPI, I²C, UART
Reloj en tiempo real:	Ninguno
Consumo energético:	700 mA, (3.5 W)
Fuente de alimentación:[5]	5 V vía Micro USB o GPIO header
Dimensiones:	85.60mm × 53.98mm (3.370 × 2.125 inch)
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS

Pines:

Los pines de la raspberry pi están organizados de la siguiente forma:

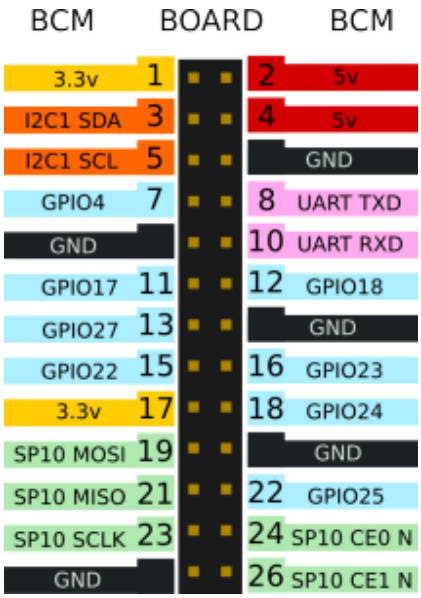

Figura 3.2.2 Pines Raspberry Pi

GPIO

Para utilizar los pines de entrada/salidas de la raspberry pi es necesario configurar y activar las salidas de la siguiente forma [5]:

El primer paso será comprobar que se dispone de la librería RPi.GPIO instalada y en la última versión.

Abrir la Terminal y escribir:

sudo python

Dentro de la consola de Python comprobamos la versión

import RPi.GPIO

RPi.GPIO.VERSION

Se actualizan las dependencias:

sudo apt-get update

sudo apt-get upgrade

Si no tenemos instalada RPi.GPIO, la descargamos e instalamos. Estos comandos son para la instalar version actual.

wget https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.4.tar.gz

tar zxf RPi.GPIO-0.5.4.tar.gz

cd RPi.GPIO-0.5.4.tar.gz

sudo python setup.py install

Ejemplo blink LED

Se monta el siguiente circuito:

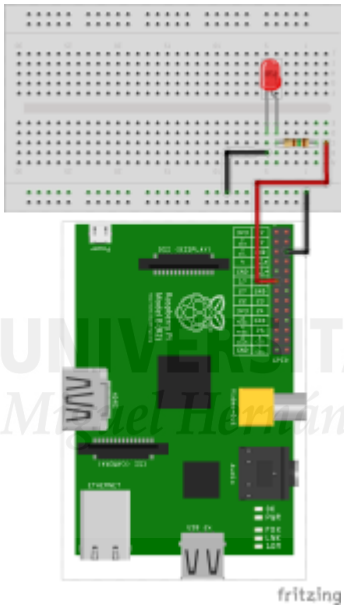

Figura 3.2.3 Circuito blink LED

Creamos el archivo blink.py con nuestro editor de texto favorito:

nano blink.py

El código es el siguiente:

```
import RPi.GPIO as GPIO #importamos la libreria y cambiamos su nombre
import time #necesario para los delays

#establecemos el sistema de numeracion que queramos, en mi caso BCM
GPIO.setmode(GPIO.BCM)

#configuramos el pin GPIO17 como una salida
GPIO.setup(17, GPIO.OUT)

#encendemos y apagamos el led 5 veces
for i in range(0,5):

    GPIO.output(17, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(17, GPIO.LOW)
    time.sleep(1)

GPIO.cleanup()  #devuelve los pines a su estado inicial
```

Figura 3.2.4 Código blink LED

Ejecutar el programa

sudo python blink.py

Si está todo bien conectado, deberíamos ver como se apaga y se enciende el LED en intervalos de un segundo. Esto significa que la librería funciona correctamente y está todo preparado para realizar proyectos de mayor complejidad.

3.2.2 Ordenador para desarrollo web

Figura 3.2.5. Ordenador ASUS

Características

- Procesador: Intel Core i5-430M (2,26 Ghz, 3 Mb. de cache)
- Memoria: 4 Gb. DDR3 a 1066 Mhz. (Ampliable a 8 Gb.)
- Disco Duro: 500 Gb. a 5400rpm.
- Sistema Operativo: Windows 7 Home Premium (64 bits)
- Red: Gigabit Ethernet y Wifi-N
- Tarjeta Gráfica: Intel GMA HD (integrada)
- Pantalla: Pantalla de 15,6" HD (1366x768) con tecnología LED
- Dimensiones: 380x255x3.57 cm (W x D x H)
- Peso: 2,62 Kg.
- Otros: 3 puertos USB, salida VGA, salida HDMI, teclado numérico, lector de tarjetas 5 en 1, touchpad multi-táctil, altavoces Altec Lansing y webcam.

3.2.3 Maqueta solarfotovoltaica

En este apartado se va a mostrar el diseño de la maqueta realizada para la adquisición y envío de datos a nuestra página web.

Para la maqueta de la estación fotovoltaica se han utilizado los siguientes componentes:

- Placa fotovoltaica monocristalina 12V - 20W
- Regulador de tensión 12V
- Raspberry Pi Modelo B
- Sensor de temperatura LM35
- Sensor LDR
- Convertidor ADC MCP3008
- Cables
- Webcam

Se han conectado los diferentes sensores utilizados (sensor LDR y sensor de temperatura LM35) al convertidor MCP3008 para poder adaptar las señales a un voltaje válido para la Raspberry Pi y que puedan ser leídas.

- Placa Solar:

Figura 3.2.6. Placa solar

- Regulador de carga:

Figura 3.2.7. Regulador de carga

- Raspberry Pi

Figura 3.2.8. Raspberry Pi

3.3. Software

3.3.1 Nodejs

Figura 3.3.1. Node.JS

Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación Javascript, asíncrono, con entrada/salida de datos en una arquitectura orientada a eventos y basado en el motor Javascript V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent, que además tiene contratado a Dahl en plantilla.

Node.js es similar en su propósito a Twisted o Tornado de Python, Perl Object Environment de Perl, React de PHP, libevent o libev de C, EventMachine de Ruby, vibe.d de D y de Java existe Apache MINA, Netty, Akka, Vert.x, Grizzly o Xsocket. Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor. Node.js implementa algunas especificaciones de CommonJS. Node.js incluye un entorno REPL para depuración interactiva.

Módulos

Node.js incorpora varios "módulos básicos" compilados en el propio binario, como por ejemplo el módulo de red, que proporciona una capa para programación de red asíncrona y otros módulos fundamentales, como por ejemplo Path, FileSystem, Buffer, Timers y el de propósito más general Stream. Es posible utilizar módulos desarrollados por terceros, ya sea como archivos ".node" precompilados, o como archivos en javascript plano. Los módulos Javascript se implementan siguiendo la especificación CommonJS para módulos, utilizando una variable de exportación para dar a estos scripts acceso a funciones y variables implementadas por los módulos.

Los módulos de terceros pueden extender node.js o añadir un nivel de abstracción, implementando varias utilidades middleware para utilizar en aplicaciones web, como por ejemplo los frameworks connect y express. Pese a que los módulos pueden instalarse como archivos simples, normalmente se instalan utilizando el Node Package Manager (npm) que nos facilitará la compilación, instalación y actualización de módulos así como la gestión de las dependencias. Además, los módulos que no se instalen el directorio por defecto de módulos de Node necesitarán la utilización de una ruta relativa para poder encontrarlos. El wiki Node.js proporciona una lista de varios de los módulos de terceros disponibles.

3.3.2 Express js

Figura 3.3.2. Express js

Express es sin duda el framework más conocido de node.js, es una extensión del poderoso connect y esta inspirado en **sinatra**, además es robusto, rápido, flexible, simple…

Sin duda el exito de "express" radica en lo sencillo que es usarlo, y además abarca un sin número de aspectos que muchos desconocen pero son necesarios.

De entre las tantas cosas que tiene este framework podemos destacar:

- Session Handler
- 11 middleware poderosos asi como de terceros.
- cookieParser, bodyParser…
- vhost
- router, y contando…

Para comenzar, lo único que es necesario hacer es lo siguiente:

npm install -g express

A través de esta linea, se está instalando globalmente express, el cual expone una interfaz interactiva en la linea de comandos. Por ejemplo si escribimos "express –h" lo siguiente sera mostrado:

```
  Usage: express [options] [dir]

  Options:

    -h, --help          output usage information
    -V, --version       output the version number
    -e, --ejs           add ejs engine support (defaults to jade)
        --hbs           add handlebars engine support
    -H, --hogan         add hogan.js engine support
    -c, --css <engine>  add stylesheet <engine> support (less|stylus|compass) (d
efaults to plain css)
    -f, --force         force on non-empty directory
```

Figura 3.3.2 Express js - Ayuda

Una vez instalado express puedes utilizar "*express NOMBRE-DEL-APP*" lo cual te creara una estructura personalizada y los archivos necesarios para comenzar a trabajar con el mismo. Ejecutando: "*cd NOMBRE-DEL-APP && npm install*"para instalar automaticamente las dependencias, la estructura generada seria como la siguiente:

```
NOMBRE-DEL-APP
\
 |- package.json
 |- app.js
 |- public
    \
     |- javascripts
     |- stylesheets
     |- images
 |- routes
    \
     |- index.js
 |- views
    \
     |- layout.jade
     |- index.jade
```

Figura 3.3.3 Estructura de archivos Express

Como podemos ver, la estructura que se genera es muy útil y sencilla, se ha creado unpackage.json el cual contiene todas las dependencias necesarias de la aplicación. [5]

app.js

Una aplicación escrita con "express" posee ciertos rasgos y cosas que no se pueden obviar, en "app.js" existe una estructura interna bien definida y es como sigue:

```
// app.js
var express = require('express')
  , routes = require('./routes')

var app = module.exports = express.createServer();
```

Figura 3.3.4 Primera aplicación express js

En esta primera parte se "**requieren"** los modúlos o archivos externos necesarios para ejecutar la aplicación, en este caso el mismo express y routes que es ni más ni menos un archivo donde se encuentran cada una de las rutas disponibles,routes es algo opcional, ya que tu mismo puedes ir generando las rutas a medida que vas avanzando. De igual forma, en esta parte se **crea** el servidor, mediante: *express.createServer()*, y es asignado a la variable app y de igual forma se "exporta" para que este disponible en caso de que sea necesario ejecutarla como secundario.

Configuración

La segunda parte y una muy importante es la configuración:

```
// Configuration

app.configure(function(){
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(app.router);
  app.use(express.static(__dirname + '/public'));
});
```

Figura 3.3.5 Configuracion express

Es aqui donde se definen aspectos muy importantes para el funcionamiento correcto de nuestra aplicación, comumente conocido como middleware. Como se puede ver se define que el directorio donde se encuentran las vistas (templates) de nuestra aplicación, además se define en que lenguaje o motor esta escrito *app.set("view engine", "jade");*en este caso en jade, además se define bodyParser y methodOverride mediante *app.use().*

El **bodyParser** en palabras cortas, se encarga de decodificar la información que recibimos de un socket-cliente y lo expone en cada una de las requests mediante *request.body.**methodOverride*** proveee soporte par el metodo faux HTTP. Como podemos ver se hace uso tambien de el router de express, el cual nos proporciona la habilidad para estructurar nuestro código de una manera más sencilla mediante: *app.get('/path', function)*, y por último se define el middleware de archivos estáticos con la carpeta donde se encuentran los archivos que son "publicos".

En el código anterior se define una configuración "**global"**si tú quieres una configuración más especifica, asignale un nombre a tu ambiente mediante:

```
app.configure('personalizada', function(){

});
```

Figura 3.3.6 Configuración personalizada express

personalizada es una variable global que es pasada al momente de ejecutar tu programa mediante "NODE_ENV=personalizada" y es así como se origina la siguiente parte del código:

```
app.configure('development', function(){
  app.use(express.errorHandler({ dumpExceptions: true, showStack: true }))
});

app.configure('production', function(){
  app.use(express.errorHandler());
});
```

Figura 3.3.7 Configuración personalizada express environment

Es aqui donde se definen los ambientes más comunes, el de desarrollo (que es el default) y el de producción, en este ejemplo, se hace uso del middleware "errorHandler", el cual te muestra las *excepciones* que el express encuentra, al habilitar showStack tu aplicación al encontrar un error te muestra un mensaje vistoso en el navegador (lo cual solo es útil para el creador, por lo tanto no es recomendable mostrarlo en producción).

Por ejemplo el uso y la configuración de las bases de datos:

```
// app.configure('development' ...
  ...
  app.set('db-uri','http://localhost:5984/DB')
  ...
// app.configure('production' ...
  ...
  app.set('db-uri', 'http://USR.iriscouch.com/DB')
  ...
```


Figura 3.3.8 Configuración base de datos express

Rutas

Las rutas son definitivamente la parte más importante de la aplicación, porque si éstas no están definidas, no existiria una interfaz para el cliente. En el ejemplo se generó esta dirección automaticamente, "routes" fueron definidas en la primer parte del archivo, y está definido como sigue:

```
app.get('/', routes.index);
```

Figura 3.3.9 Configuración rutas express

Como se puede ver una ruta esta especificada de la siguiente forma:

```
app.VERBO(PATH, ACCIÓN);
```

Figura 3.3.10 Configuración ruta express general

- **app** ya la conocemos.
- **VERBO** puede ser:
  - GET
  - POST
  - PUT
  - DELETE
  - y así para cada uno de los verbos HTTP
- **PATH**: define la dirección de acceso.
- **ACCION**: que es lo que se tiene que hacer.

Por ejemplo:

```
app.post('/user/new', function(request,respone){
  var body = request.body; // accede a la información enviada por el socket
  // Guarda la información, haz lo que tengas que hacer
});
```

Figura 3.3.11 Ejemplo post express

O:

```
app.get('/blog/:id', function(req,res){ /
  var id = req.params.id; // accede al id
})
```

Figura 3.3.12 Ejemplo get express

Definiendo el formato:

```
app.get('/u/:id/mensajes.:format', function(req,res){
  var id = req.params.id;
  var format = req.params.format;
  if (format === 'json'){
    res.json({status:200, id:id});
  } else {
    res.json({status:500});
  }
});
```

Figura 3.3.13 Definiendo el formato en express

Además de los verbos mencionados tambien se expone:

app.all(,function);*

Por la cual pasan todas las request, es muy útil en el caso que quieras detectar si es de un movil, o cualquier cosa que tú quieras. Un aspecto interesante de las rutas es que si bien es cierto responden a una estructura definida, se pueden definir ciertas acciones antes de dar una respuesta al usuario que hizo la petición.

Listen

Por último es importante que tu aplicación este disponible en algún puerto (doh!), y es aqui:

app.listen(3000);

Por defecto es el puerto **3000** que queda habilitado para la aplicación, se puede cambiar por algo como lo siguiente:

app.listen(process.env.PORT || 3000);

process.env.PORT es utilizada por procesos globales de los proveedores de hosting, para especificar el puerto a utilizar por las aplicaciones. Por ejemplo en "cloud9" tienes que especificar: *process.env.C9_PORT*.

Básicamente esto es lo que te provee "express" al hacer "*express NOMBRE-DEL-APP*", pero aparte de esto a se pueden especificar otras cosas más. Por ejemplo:

Helpers

Los *dynamicHelpers* te ayudan a definir un puente entre el cliente y el servidor, para crear variables o funciones que esten disponibles en ambos lafos. Hay dos formas de habilitar este middleware, uno de ellos es:

Dentro de app.configure():

app.helpers(require('./path/to/helpers'));

En este caso "./path/to/helpers" podria ser un archivo. O en cualquier otra parte de "app.js".

```
app.dynamicHelpers({
  sitename: function(){
    return 'NOMBRE-DEL-APP'
  }
});
```

Figura 3.3.14 Helpers en express

Los *dynamicHelpers* es un objeto pero cada uno de sus miembros son siempre funciones, por eso es necesario el *return 'NOMBRE'*.

Error Handler

Express provee tambien de un sistema de errores:

```
app.error(function(err, req, res, next){
  if (err) {
    // Hacer algo con el error
  } else {
    next();
  }
});
```

Figura 3.3.15 Sistema de errores express

Funciona tambien mediante: *app.use(function(err,req,res,next){})*

Sessiones

Para habilitar las sessiones es necesario hacer lo siguiente:

```
app.configure(function(){
  app.use(express.bodyParser())
  app.use(express.cookieParser('nhispano'))
  app.use(express.session({secret: 'SECRET', store: store }))
});
```

Figura 3.3.16 Sesiones en express

Respuesta

El método (?) response, tiene una gran cantidad de funciones y otros métodos disponibles. Por ejemplo:

res.sendfile('/path/to/archivo')

Envia un archivo directamente al usuario, muy útil para enviar archivos html directamente.

res.json(DATA)

Envia como respuesta un documento en formato json.

res.write()

Escribe información al cliente que hizo la petición, se puede utilizar multiples veces en una misma petición:

```
req.write('<h1>Hola</h1>');
req.write('<h2>mundo</h2');
```

Figura 3.3.17 Multiple response express

res.send()

Es un método primario y se puede utilizar múltiples veces tambien.

```
res.send(); // 204
res.send(new Buffer('wahoo'));
res.send({ some: 'json' });
res.send('<p>some html</p>');
res.send('Sorry, cant find that', 404);
res.send('text', { 'Content-Type': 'text/plain' }, 201);
res.send(404);
```

Figura 3.3.18 Multiple envio res.send express

res.writeHeader(CODE,Content-type)

Define el tipo de datos de la respuesta por ejemplo si se usa res.write() con html como el ejemplo anterior, se tiene que definir el tipo de contenido. {"Content-type","text/html"}

res.contentType(type)

Al igual que res.writeHeader() nada más que en este caso solo pasas él Content-type.

res.redirect()

Redirige al usuario a otra ruta

res.render

Define y renderiza un template o layout, el uso más comun es el siguiente:

```
res.render('TEMPLATE', {
  // variables locales de TEMPLATE
  layout: false, // por default busca dentro de views archivo llamado `layout.jade|ejs`,
  nombre: 'NOMBRE'
})
```

Figura 3.3.19 Renderizar vista express

res.end()

Termina la respuesta. Necesaria al utilizar res.write, res.send.

3.3.3 MongoDB

Figura 3.3.20. mongoDB

MongoDB (de la palabra en inglés "humongous" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. [7]

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

El desarrollo de MongoDB empezó en octubre de 2007 por la compañía de software 10gen. Ahora MongoDB es una base de datos lista para la producción de uso y con muchas características (features). Esta base de datos es altamente utilizada en las industrias y MTV Network, Craiglist y Foursquare4 son algunas de las empresas que utilizan esta base de datos.

El código binario está disponible para los sistemas operativos Windows, Linux, OS X y Solaris.

Características principales

Consultas Ad hoc

MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento pero también puede ser una función JavaScript definida por el usuario.

Indexación

Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios. El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales.

Replicación

MongoDB soporta el tipo de replicación maestro-esclavo. El maestro puede ejecutar comandos de lectura y escritura. El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura o para copia de seguridad, pero no se pueden realizar escrituras. El esclavo tiene la habilidad de poder elegir un nuevo maestro en caso de que se caiga el servicio con el maestro actual.

Balanceo de carga

MongoDB se puede escalar de forma horizontal usando el concepto de "shard". El desarrollador elige una llave shard, la cual determina cómo serán distribuidos los datos en una colección. Los datos son divididos en rangos (basado en la llave shard) y distribuidos a través de múltiples shard. Un shard es un maestro con uno o más esclavos. MongoDB tiene la capacidad de ejecutarse en múltiple servidores, balanceando la carga y/o duplicando los datos para poder mantener el sistema funcionando en caso que exista un fallo de hardware. La configuración automática es fácil de implementar bajo MongoDB y nuevas máquinas pueden ser agregadas a MongoDB con el sistema de base de datos corriendo.

Almacenamiento de archivos

MongoDB puede ser utilizado con un sistema de archivos, tomando la ventaja de la capacidad que tiene MongoDB para el balanceo de carga y la replicación de datos utilizando múltiples servidores para el almacenamiento de

archivos. Esta función (que es llamada GridFS) está incluida en los drivers de MongoDB y disponible para los lenguajes de programación que soporta MongoDB. Esta base de datos expone funciones para la manipulación de archivos y contenido a los desarrolladores. En un sistema con múltiple servidores, los archivos pueden ser distribuidos y copiados entre los mismos varias veces y de una forma transparente, de esta forma se crea un sistema eficiente que maneja fallos y balanceo de carga.

Agregación

La función MapReduce puede ser utilizada para el procesamiento por lotes de datos y operaciones de agregación. Esta función permite que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL "group-by".

Ejecución de JavaScript del lado del servidor

MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

3.3.4 HighChart

Figura 3.3.21. Highcharts

HighCharts es una librería escrita en Javascript que permite la creación de gráficas. La librería ofrece un método fácil e interactivo para insertar graficas en su sitio web o aplicación web. [8]

Características

La librería es compatible con todos los navegadores modernos incluyendo iPhone/iPad e Internet Explorer desde su versión 6.

No es comercial, no se necesita el permiso de los autores para su implementación en sitios web personales o sin fines de lucro.

Es abierto, todas las características pueden ser personalizadas permitiendo una gran flexibilidad además HighCharts está escrito solamente con código Javascript, sólo se requiere incluir el archivo highcharts.js y cualquiera de los tres frameworks más populares de Javascript (jQuery, MooTools o Prototype).

3.3.5 Python

Figura 3.3.22. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es

compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y conteo de referencias para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse, en algunas situaciones, hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

3.3.6 Librería py-spidev

Se ha hecho uso de la librería py-spidev del desarrollador *doceme* para la utilización de dispositivos SPI en la raspberry pi.

Py-spidev es una librería escrita en Python que hace de interface entre el driver del kernel de Linux y los dispositivos conectador por protocolo SPI.

3.4. Servicios contratados

Para la realización de este proyecto se ha necesitado la contratación de los siguientes servicios:

3.4.1 Servidor VPS

Se ha contratado un Servidor Privado Virtual (VPS) para el alojamiento del servidor web. Este servicio se ha contratado con la empresa CrownCloud (www.crowncloud.net), la cual nos proporciona un servidor con las siguientes características:

- Tipo de servidor: OpenVZ VPS-3072
- Localización: Los Ángeles, California
- RAM: 3072 MB
- vSWAP: 3072 MB
- HDD: 50 GB
- 3 TB bandwidth
- 1 IPv4 and 10 IPv6

3.4.2 Dominio

Para poder acceder a la página web con el nombre www.monitorizatuplaca.com en vez de con la ip del servidor (VPS) es necesario la contratación de un dominio, que se asocia a esa ip para que se haga la redirección.

Este servicio se ha contratado con 1&1 (www.1and1.es) el cual redirigiremos a nuestra web.

3.5. Código web

En el siguiente bloque se va a detallar las funciones utilizadas en Nodejs para el desarrollo de la aplicación web

3.5.1 Modelo de la base de Datos

En una primera sección se explicarán las diferentes tablas que dispone el proyecto.

3.5.1.1 Index

Este primer archivo es necesario para que la aplicación reconozca las diferentes tablas de las que dispone la web así como del nombre de la base de datos.

```
// models/index.js
if (!global.hasOwnProperty('db')) {

  var mongoose = require('mongoose');

  var dbName = 'monitoriza'

  mongoose.connect('mongodb://localhost/' + dbName);


  global.db = {
```

```
    mongoose: mongoose,


    //models
    User:          require('./user')(mongoose),
    Data:          require('./data')(mongoose),
    Estacion:      require('./estacion')(mongoose)
    // agregar más modelos aquí en caso de haberlos
  };


}


module.exports = global.db;
```


3.5.1.2 Datos

En esta tabla es donde se van a almacenar todos los datos de monitorización relevantes de las estaciones como son el ID de la estación, la tensión, la temperatura, la iluminación, la fecha y el usuario que envía esta información.

```
// models/Data.js


module.exports = function(mongoose) {


  var Schema = mongoose.Schema;
```

```
// Objeto modelo de Mongoose
var DataSchema = new Schema({



  estacionId   : String,
  tension      : Number,
  temp         : Number,
  ilum         : Number,
  fecha        : Date,
   userId      : String
});


return mongoose.model('Data', DataSchema);
}
```

3.5.1.3 Estación

En esta tabla es donde se almacenan los datos de las estaciones como el id, el nombre, la posición geográfica (latitud, longitud), y los estados de las salidas.

```
// models/Estacion.js


module.exports = function(mongoose) {


  var Schema = mongoose.Schema;


  // Objeto modelo de Mongoose
  var EstacionSchema = new Schema({


        estacionId    : String,
        latitud       : String,
        longitud      : String,
        userId                : String,
        name          : String,
        salida1               : Boolean,
        salida2               : Boolean,
        salida3               : Boolean


  });
    return mongoose.model('Estacion', EstacionSchema);
}
```

3.5.1.4 Usuarios

En esta tabla se guardarán los datos referentes a los usuarios, como su nombre de usuario, nombre real, email y contraseña.

```
// models/User.js

module.exports = function(mongoose) {

  var Schema = mongoose.Schema;

  // Objeto modelo de Mongoose
  var UserSchema = new Schema({

    username    : String,
    password    : String,
    nombre      : String,
    email       : String,

  });

  return mongoose.model('User', UserSchema);
}
```

3.5.2 Aplicación web (Backend)

En esta sección se va a explicar el código de la web desde que el usuario accede a la página hasta que se muestran los datos por pantalla.

El archivo principal es ***app.js*** que es el que se encarga de gestionar todas las peticiones y redirigir a una parte de código u otro dependiendo de la sección de la web en la que nos encontremos.

Para comenzar se empieza definiendo las dependencias (módulos externos y variables) que se vayan a utilizar es la aplicación:

3.5.2.1 Dependencias

```
//**********************************************************************//

//**********************************************************************//

//                    Dependencias

//**********************************************************************//

//**********************************************************************//


var http = require('http');

require('./models');

var bodyParser = require('body-parser')

var passport = require('passport')

, LocalStrategy = require('passport-local').Strategy;

var ensureLoggedIn = require('connect-ensure-login').ensureLoggedIn,

ensureNotLoggedIn = require('connect-ensure-login').ensureNotLoggedIn;

var cookieSession = require('cookie-session');
```

```
var express = require('express')
, partials = require('express-partials')
, app = express();
```

3.5.2.2 Configuración servidor y base de datos

Una vez que se han definido las dependencias se procede a configurar el servidor y la base de datos:

```
//*****************************************************************//
//*****************************************************************//
//            Configuracion Servidor y Base de datos
//*****************************************************************//
//*****************************************************************//

app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.static(__dirname + '/public'));
app.use( bodyParser.json() );       // to support JSON-encoded bodies
app.use( bodyParser.urlencoded({extended:true}) ); // to support URL-encoded
bodies

// load the express-partials middleware
```

```
app.use(partials());


app.use(cookieSession({
  secret : "CLAVEULTRASECRETA"}))



app.use(passport.initialize());
app.use(passport.session());



passport.use(new LocalStrategy(function(username, password, done) {


  process.nextTick(function() {
    db.User.findOne({
      'username': username,
    }, function(err, user) {
      if (err) {
        return done(err);
      }
      if (!user) {
        return done(null, false, { message: 'Incorrect username.' });
      }
      if (user.password != password) {
```

```
      return done(null, false, { message: 'Incorrect password.' });
    }
    return done(null, user);
  });
});
}));


passport.serializeUser(function(user, done) {
  done(null, user);
});

passport.deserializeUser(function(user, done) {
  done(null, user);
});

//Asignamos a server la creación del servidor http.

var server = http.createServer(app).listen(app.get('port'), function(){
  console.log("Express server listening on port " + app.get('port'));
});
```

Como se puede observar el servidor va a estar activo en el puerto 3000 de la máquina, que más tarde se tendrá que redirigir el tráfico web del puerto 80 a ese puerto para que funcione correctamente la web.

3.5.2.3 Configuración sockets

Un socket es una conexión TCP por una canal de comunicación bidireccional y full dúplex, en este caso, entre el navegador y nuestro servidor. Esto nos va a permitir enviar información en tiempo real al navegador para poder actualizar, por ejemplo, las gráficas.

```
//*************************************************************//
//*************************************************************//
//                Configuracion Sockets
//*************************************************************//
//*************************************************************//


//Importamos socket.io utilizando el servidor creado anteriormente.
var io = require('socket.io').listen(server);


//Iniciamos la conexión.
io.sockets.on('connection', function(socket){


  socket.on('create', function(room) {
    socket.join(room);
  });
```

```
 socket.on('disconnect', function(){
  console.log('user disconnected');
 });
});
```

Como se puede observar se está conectando a una sala (room) en la que el único "oyente" es el usuario al que pertenece esa estación. Esto se hace para que a cada usuario le lleguen solo los datos que son de sus estaciones y no se produzcan envíos de datos erróneos a usuarios que no les corresponte.

3.5.2.4 Rutas

En esta sección se van a mostrar los diferentes comportamientos de la aplicación según el usuario navegue por una zona de la web u otra, explicando en cada caso el código programado. [10]

3.5.2.4.1 Inicial

En esta ruta, la ruta inicial, se comprueba si el usuario está logeado y en caso de que no lo esté se le redirige a la ruta de login. En caso de que esté logeado se le muestra la pantalla principal (home)

```
app.get('/',ensureLoggedIn('/login'), function(req, res){

 res.render('home',
 {
  user : req.user
 });
```

```
});
```

3.5.2.4.2 *Login / registro*

En estas rutas se comprueba que no esté logeado y se renderiza la página correspondiente, ya que en caso de estar logeado se redireccionará a la pantalla principal.

```
app.get('/login',ensureNotLoggedIn('/'), function(req, res){

  res.render('login.ejs' , { registrado: false});

});



app.get('/register',ensureNotLoggedIn('/'), function(req, res){

  res.render('register.ejs', { error: false} );

});
```

3.5.2.4.3 *Datos a tiempo real*

Como en los demás casos inicialmente se comprueba que el usuario esté registrado y logeado.

```
app.get('/real',ensureLoggedIn('/login'), function(req, res){
```

```
 db.Estacion.find({
  'userId': req.user._id,
}, function(err, data) {
  if (err) {
    var datos={};
  }else{
    var estaciones=data;
  }

  var ahora = new Date().getTime();

//Ultimao cuarto de hora
 var antes = ahora - (0.25*60*60*1000) ;

  db.Data.find({
    'userId': req.user._id,
     'fecha' : {$gte: antes, $lt: ahora}
  }).sort('fecha').exec(function (err, data) {
    if (err) {
      var datos={};
    }else{
      var datos=data;
    }
```

```
 res.render('real',
 {
   user : req.user,
   estaciones: estaciones,
   datos: datos
 });



});


});


});
```

Después lo que se hace es buscar en la base de datos las estaciones que dispone el usuario y, sabiendo el ID de la estación, se hace una búsqueda de los datos de las estaciones con fecha no posterior a un cuarto de hora antes de la fecha actual. Esta comprobación se realiza para evitar los datos excesivos en la gráfica.

Para hacer el envío de los datos a la gráfica en tiempo real debemos irnos al controlador de obtención de datos:

```
app.post('/saveData', function(req, res){


  var form = req.body;
  estacionId = form.estacionId;
  tension = form.tension;
  userId = form.userId;
  name = form.name;


  fecha = new Date().getTime();
  var params = {
    estacionId  : estacionId,
    tension     : tension,
    userId      : userId,
    name        : name,
    fecha       : fecha
  }


 db.Estacion.findOne({
  'estacionId': estacionId,
}, function(err, data) {
  if (err) {
    var datos={err:"No hay estacion"};
  }else{
```

```
  if(data){
    var estacion = JSON.parse(JSON.stringify(data));
    estacion['err'] = "No error"
  }
  else
    var estacion={err:"No hay estacion"};
 }

 addData(params, function (err, result){
  if(err)
    res.send("ERROR")
  else
    res.send(estacion)
 });
});
});
```

Este controlador es el que llama la Raspberry Pi para almacenar los datos en la base de datos. En él se procesan los datos que llegan al servidor y se añaden a la base de datos mediante la función ***addData():***

```
function addData(params, callback){

 var newData = new db.Data({
```

```
    estacionId  : params.estacionId,
    tension     : params.tension,
    temp        : params.temp,
    ilum        : params.ilum,
    fecha       : params.fecha,
    userId      : params.userId,
    name        : params.name
  });

  newData.save(function(error, data) {
    if (error) callback(true,error);

sendData(data);

    callback(null,data);
  });

}
```


Esta función lo añade a la base de datos y lo manda a las gráficas del usuario con la función ***sendData()***

```
function sendData (data) {

 io.sockets.in(""+data.userId).emit('otherClick'+data.estacionId, data);

}
```

Esta función, tal y como se ha comentado anteriormente manda el id de la estación como identificador para que solo actualice la gráfica correspondiente a cada estación.

3.5.2.4.4 Histórico

En este controlador se encarga de obtener los datos necesarios de la base de datos según la escala de tiempos requerida y los muestra por pantalla:

```
app.get('/historico/:tiempo/:id',ensureLoggedIn('/login'), function(req, res){


 var tiempo = req.params.tiempo ? req.params.tiempo:'dia';

 var estacionId = req.params.id?req.params.id:0;

 if(!tiempo || tiempo=="" || tiempo==undefined)

  tiempo='dia';


 db.Estacion.find({

  'userId': req.user._id,

 }, function(err, data) {

  if (err) {

   var datos={};
```

```
 }else{
   var estaciones=data;
 }
 var ahora = new Date().getTime();
 if(tiempo=='dia')
  var antes = ahora - (24*60*60*1000) ;
else if(tiempo=='mes')
  var antes = ahora - (24*31*60*60*1000);
else if(tiempo=='anyo')
  var antes = ahora - (24*365*60*60*1000);
else if(tiempo=='total')
  var antes = ahora - (100*24*365*60*60*1000) ;
else{
  var antes = ahora - (24*60*60*1000);
}
db.Data.find({
 'userId': req.user._id,
 'estacionId': estacionId,
 'fecha' : {$gte: antes, $lt: ahora}
}).sort('fecha').exec(function (err, data) {
 if (err) {
   var datos={};
 }else{
```

```
    var datos=data;
  }
  var resultado = [];


  if(datos.length > 1000)
    {
      ratio = parseInt(datos.length/999)+1;
      for(var i=0; i<datos.length; i=i+ratio)
      {
        resultado.push(datos[i])
      }
    }else
    resultado=datos;


  res.render('historico',
  {
    user : req.user,
    estaciones: estaciones,
    datos: resultado,
    tiempo: tiempo,
    estacionId: estacionId
  });
});
```

```
});

});
```

Se puede observar que se mandan un máximo de 1000 muestras para evitar la sobrecarga de la web, para ello se escalan las gráficas para que los 1000 puntos sean equidistantes en el tiempo.

3.5.2.4.5 Estación

El código de esta ruta es muy similar a los anteriores. Inicialmente se cogen los datos de la base datos y después se renderiza la página web con los datos obtenidos:

```
app.get('/estaciones',ensureLoggedIn('/login'), function(req, res){


 db.Estacion.find({
  'userId': req.user._id
 },
 function(err, data) {
  if (err) {
   var datos={};
  }else{
   var estaciones=data;
  }


  res.render('estaciones',
  {
```

```
    user : req.user,
    estaciones: estaciones
  });
 });
});
```

Esta vista tiene la particularidad de que se pueden actualizar los datos de las salidas de la Raspberry Pi a tiempo real, para ello se ha habilitado el siguiente controlador:

```
app.post('/cambiarSalida',ensureLoggedIn('/login'), function(req, res){

 var query = {"_id": req.body.estacion};

 var salida = req.body.salida;
 var estado = req.body.estado;

 if(salida == "salida1")

  var update = { 'salida1' : estado };

 if(salida == "salida2")
  var update = { 'salida2' : estado };
```

```
 if(salida == "salida3")

  var update =  { 'salida3' : estado  };


 var options = {new: true};


 db.Estacion.findOneAndUpdate(query, update, options, function(err,
person) {

  if (err) {

   res.send("error");

  }else{

   res.send("ok");

  }


 });

});
```

3.5.3 Aplicación Raspberry-Pi

```
#!/usr/bin/python
#--------------------------------------
# Este script lee los datos del
# MCP3008 ADC usando el bus SPI y manda
# los datos a un servidor web.
#
# Author : Jose Sesma
# Date   : 11/11/2014
#
# http://www.monitorizatuplaca.com
#
#--------------------------------------


import spidev
import time
import os
import requests
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
salida1 = 17 ## PIN 17 como salida1
salida2 = 27 ## PIN 27 como salida1
```

```
salida3 = 12 ## PIN 22 como salida1


GPIO.setup(salida1, GPIO.OUT) ## GPIO 17 como salida1

GPIO.setup(salida2, GPIO.OUT) ## GPIO 27 como salida2

GPIO.setup(salida3, GPIO.OUT) ## GPIO 22 como salida3



# Abrir bus SPI

spi = spidev.SpiDev()

spi.open(0,0)


# Funcion para leer los datos del bus SPI del MCP3008

# El canal es un entero entre 0-7

def LeerCanal(canal):

  adc = spi.xfer2([1,(8+canal)<<4,0])

  data = ((adc[1]&3) << 8) + adc[2]

  return data


# Funcion para convertir data en voltaje,

# redondeando al numero de decimales establecidos.

def ConvertirVoltios(data,places):

  volts = (data * 5) / float(1023)

  volts = round(volts,places)
```

```
  return volts


# Funcion para calcular la temperatura

# del sensor LM35, con redondeo.

def ConvertirTemp(data,places):


  temp = (5 * data * 100)/float(1023)

  temp = round(temp,places)

  return temp


# Definir funcion para enviar datos a la web

def EnviarData(temp,ilum,tension,userId,estacionId):

  payload = { 'tension': tension, 'ilum':ilum , 'temp':temp, 'userId': userId,
'estacionId': estacionId}

  r = requests.post('http://204.152.203.117:3000/saveData', data=payload)

  respuesta = r.json()

  print (r.text)

  print (respuesta)

        if(respuesta['err']!="No hay estacion"):

              if(respuesta['salida1']=="true"):

                    respuesta1=True

              else:

                    respuesta1=False
```

```
            if(respuesta['salida2']=="true"):
                respuesta2=True
            else:
                respuesta2=False

            if(respuesta['salida3']=="true"):
                respuesta3=True
            else:
                respuesta3=False

            GPIO.output(salida1, respuesta1) # Enciendo la salida1
            GPIO.output(salida2, respuesta2) # Enciendo la salida2
            GPIO.output(salida3, respuesta3) # Enciendo la salida2
            print( "Salida 1: ",respuesta1)
            print( "Salida 2: ",respuesta2)
            print( "Salida 3: ",respuesta3)
            #print (respuesta['fecha'])
            #print (respuesta['_id'])
            #print (respuesta['estacionId'])
        else:
          print("ERROR: " , respuesta['err'])
    return 'ok'
```

```
# Definir canales de los sensores
light_channel = 0
temp_channel  = 1
voltaje_channel = 2

# Definir retardo entre lecturas 5 segundos
delay = 5

while True:

 # Leer el sensor de luz
 light_level = LeerCanal(light_channel)
 light_volts = ConvertirVoltios(light_level,2)

 # Leer el sensor de temperatura
 temp_level = LeerCanal(temp_channel)
 temp_volts = ConvertirVoltios(temp_level,2)
 temp       = ConvertirTemp(temp_level,2)

 #Leer el sensor de voltaje
 tension = LeerCanal(voltaje_channel)

 # Mostrar resultados
```

```
  print "-----------------------------------------"

  print("Light : {} ({}V)".format(light_level,light_volts))

  print("Temp  : {} ({}V) {} grados C".format(temp_level,temp_volts,temp))


  try:

        #EnviarData(temp, 10,light_volts, 'jose')

        EnviarData(temp, light_volts, tension, userId, estacionId)

  except requests.exceptions.RequestException as e:    # This is the correct
syntax

    print e


  # Esperar hasta el siguiente bucle

  time.sleep(delay)
```

4

Resultados Obtenidos

En este capítulo se exponen los resultados obtenidos tras el desarrollo del trabajo, así como las características más destacables.

4.1. Introducción.

Como resultado, la aplicación web muestra en diferentes pestañas el estado de nuestras estaciones fotovoltaicas, que previamente han sido introducidas en el sistema mediante el formulario correspondiente.

Mediante esta web se puede ver a tiempo real los datos de temperatura, tensión y luminosidad de cada estación así como si se encuentra en un estado crítico. Lo que nos permite actuar en un intervalo de tiempo muy pequeño para poder solventar el error. Y todo ello con la comodidad que ofrece el servicio online, sin necesidad de desplazarse al lugar para obtener estadísticas de las estaciones.

Por otro lado se puede ver un histórico general organizado por periodos de tiempo (un día, un mes, un año e histórico total) de las mismas variables que se visualizan a tiempo real (temperatura, tensión y luminosidad), lo cual nos permite ver una visión general del funcionamiento de nuestra instalación y de la evolución de la misma.

La web nos permite también el control de salidas a tiempo real en la estación. Una característica muy importante para evitar futuros problemas en caso de detección de errores.

Las gráficas son exportables para una visualización offline o el posterior uso de las mismas en otras aplicaciones, tales como conferencias, resumen a equipo técnico o directivos.

Todo ello con personalizado y con un registro de usuario y contraseña para una mayor privacidad de los datos.

4.2. Pantalla login

En esta pantalla inicial es donde el usuario debe introducir los datos correspondientes a su registro.

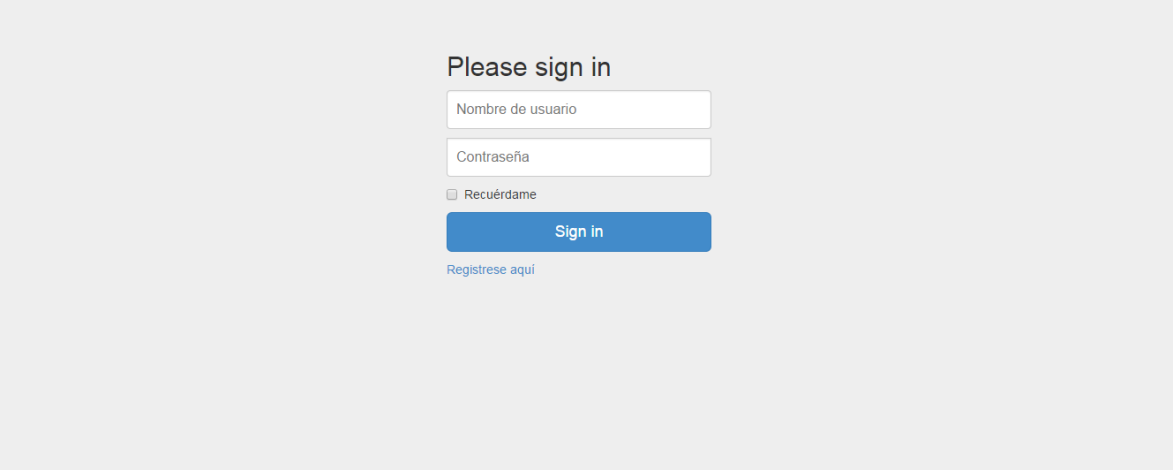

Figura 4.2.1 Pantalla de login

Como se puede observar la pantalla está compuesta por un formulario en el que se debe introducir el usuario y la contraseña.

4.3. Pantalla registro

El usuario en esta pantalla deberá rellenar los datos solicitados (nombre, usuario, email y contraseña) para poder darse de alta en el servicio.

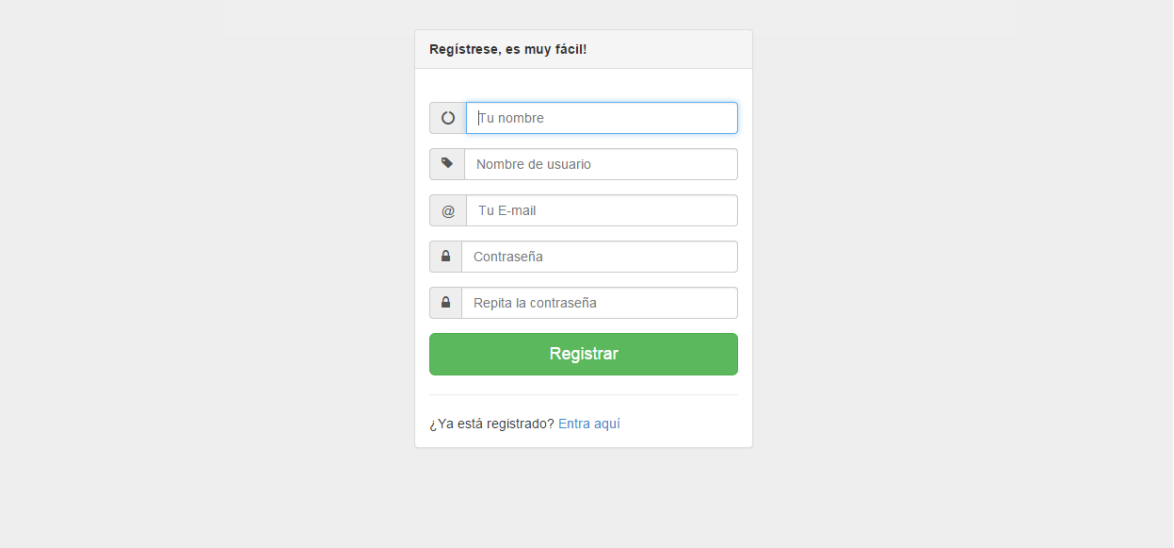

Figura 4.3.1. Pantalla de registro

4.4. Pantalla usuario registrado

Una vez el usuario se ha registrado es redirigido a la pantalla de login.

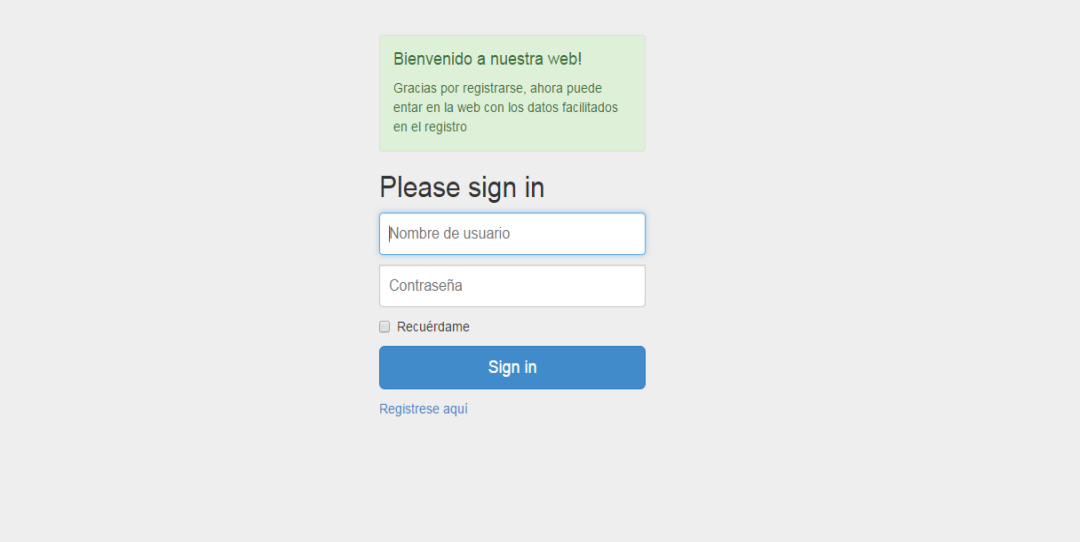

Figura 4.4.1. Pantalla usuario registrado

4.5. Pantalla inicial

Ésta es la pantalla inicial de bienvenida al usuario, en ella se puede ver un menú superior que será común a toda la web y un mensaje de bienvenida que te insta a crear una nueva estación.

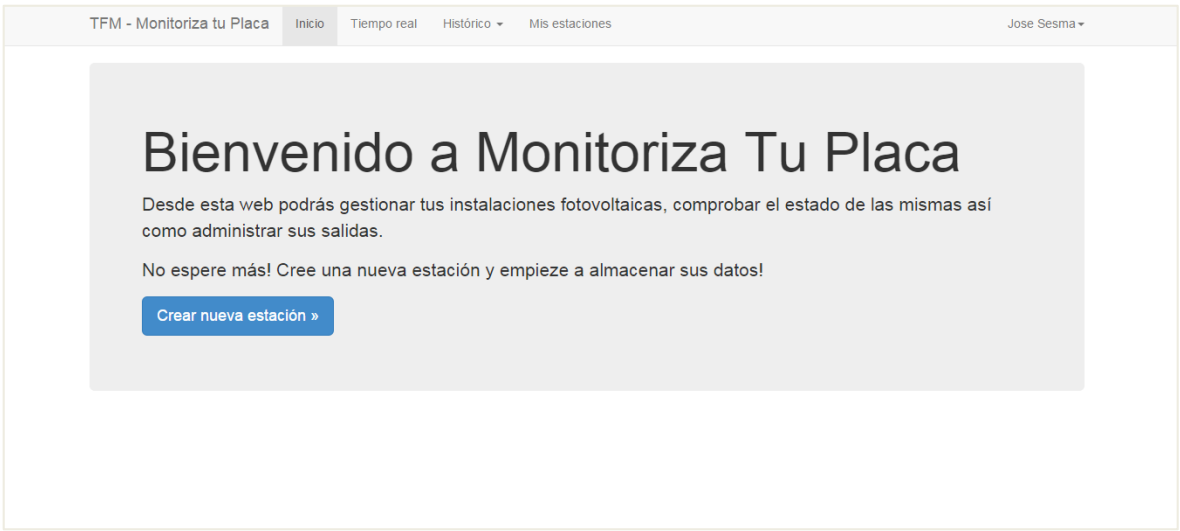

Figura 4.5.1. Pantalla Inicial

4.6. Pantalla tiempo real

En esta pantalla se muestran los datos con gráficas a tiempo real de diferentes medidas como la temperatura, tensión y luminosidad. Se puede observar que en el campo de tensión, hay una franja de sobretensión marcada en rojo y avisa con un icono rojo que se ha sobrepasado el límite establecido.

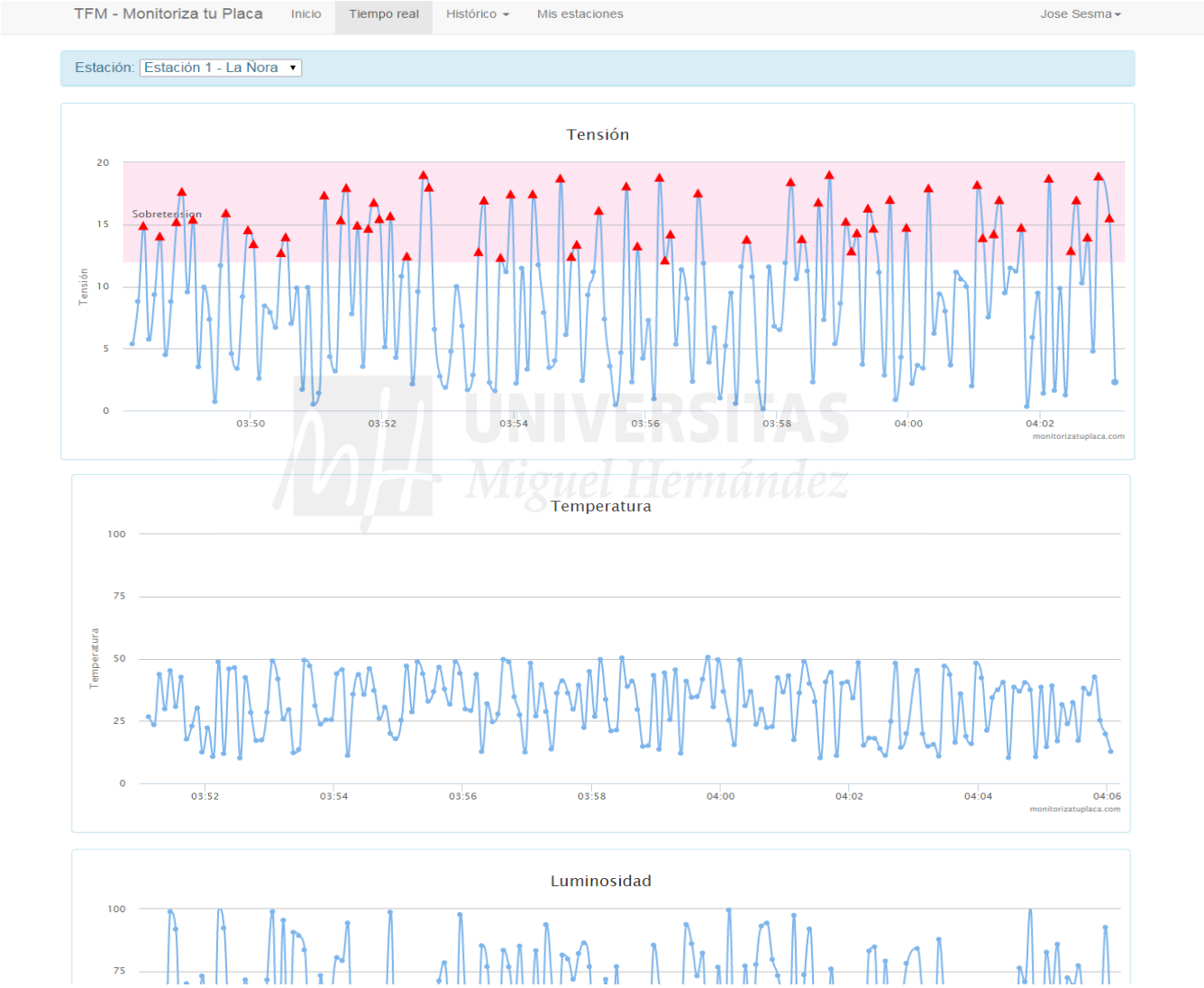

Figura 4.6.1. Pantalla tiempo real

4.7. Pantalla histórico

En esta pantalla se puede observar la evolución de los valores de las diferentes variables a estudiar en nuestra estación.

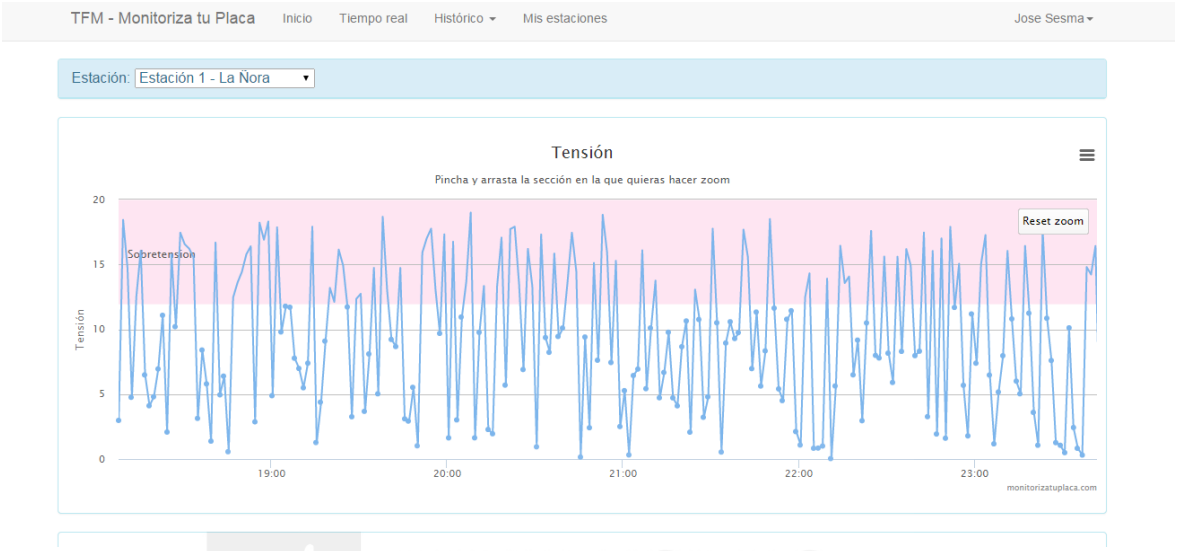

Figura 4.7.1. Pantalla histórico

Las gráficas pueden ser exportadas a formato: PNG, JPEG, PDF o SVG permitiendo un posterior uso.

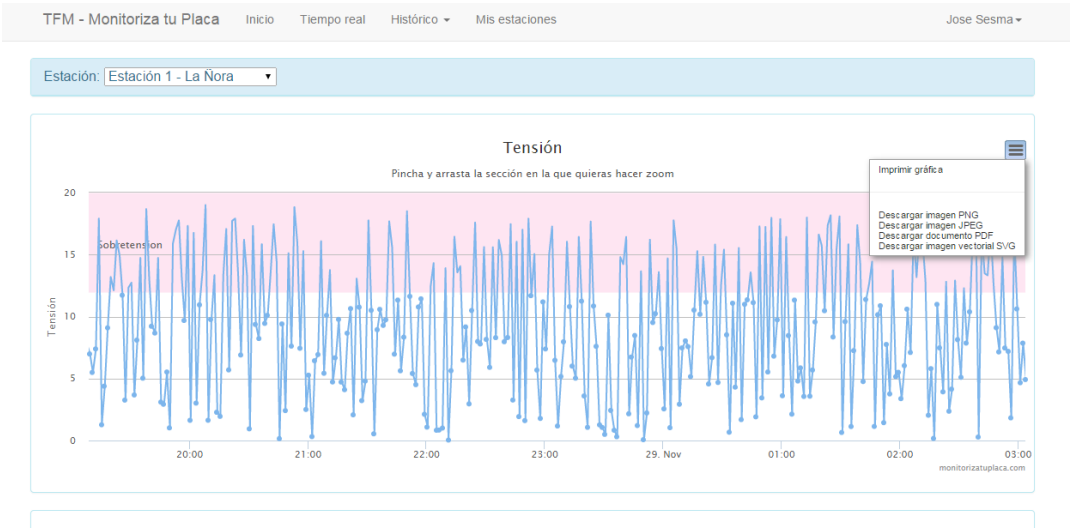

Figura 4.7.2. Pantalla exportar gráfica

4.8. Pantalla crear estación

En esta pantalla se deben introducir los datos de la estación como son:

- Nombre de la estación
- Latitud
- Longitud

La latitud y la longitud pueden ser puestas automáticamente en caso de que se seleccione una zona del mapa incluido en la web.

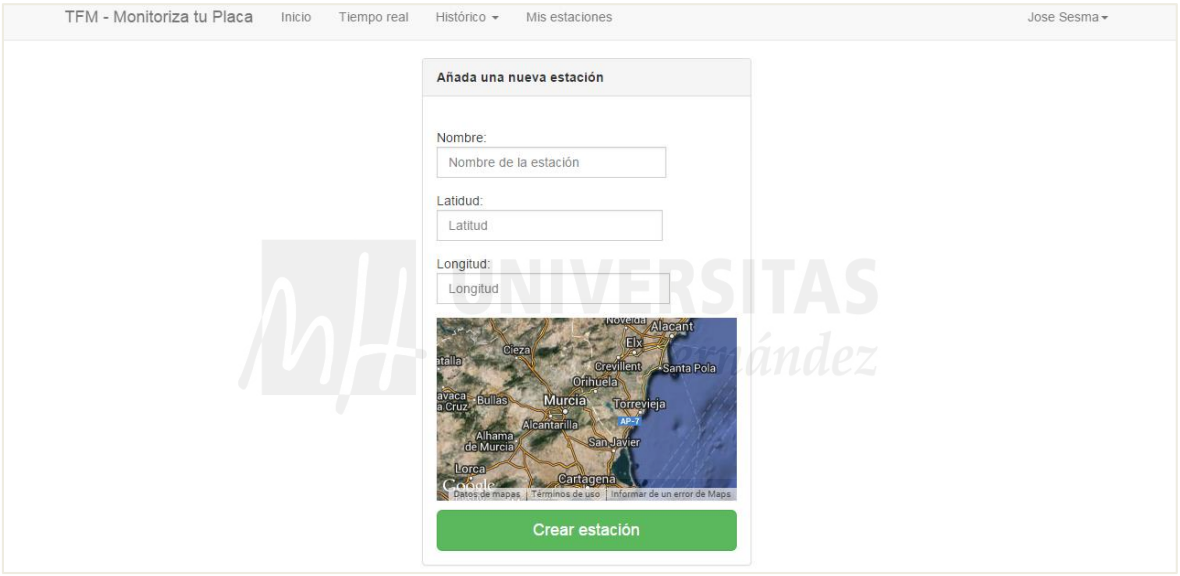

Figura 4.8.1. Pantalla crear estación

4.9. Pantalla mis estaciones

Desde esta pestaña se puede gestionar las salidas de nuestra estación así como ver el estado de las mismas.

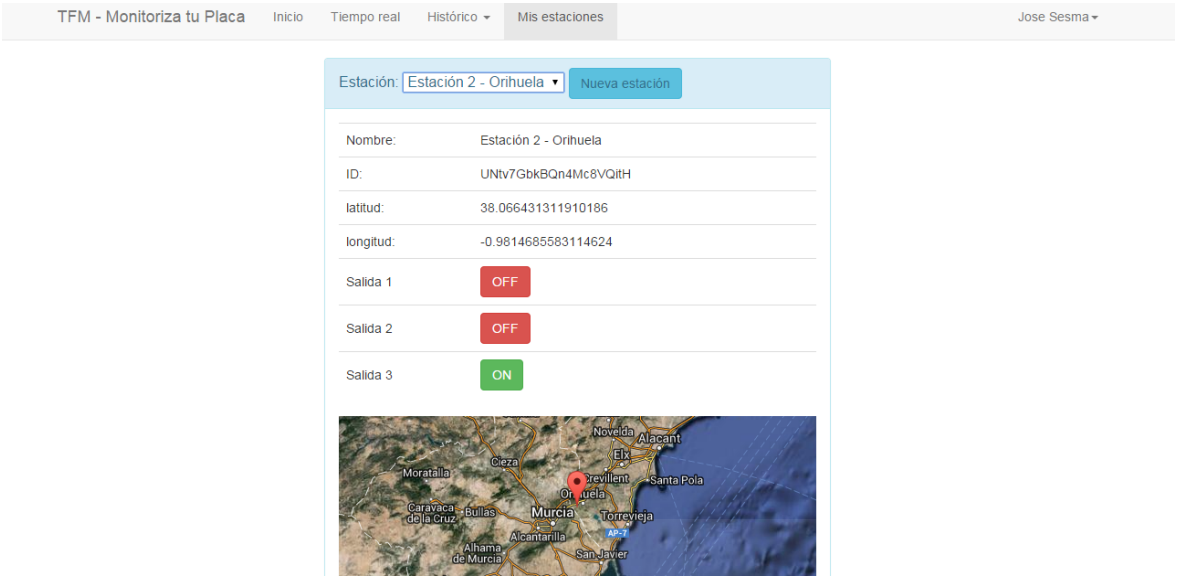

Figura 4.9.1. Pantalla "mis estaciones"

4.10. Pantalla datos de usuario

Aquí se pueden ver los datos de perfil del usuario registrado en nuestro sistema.

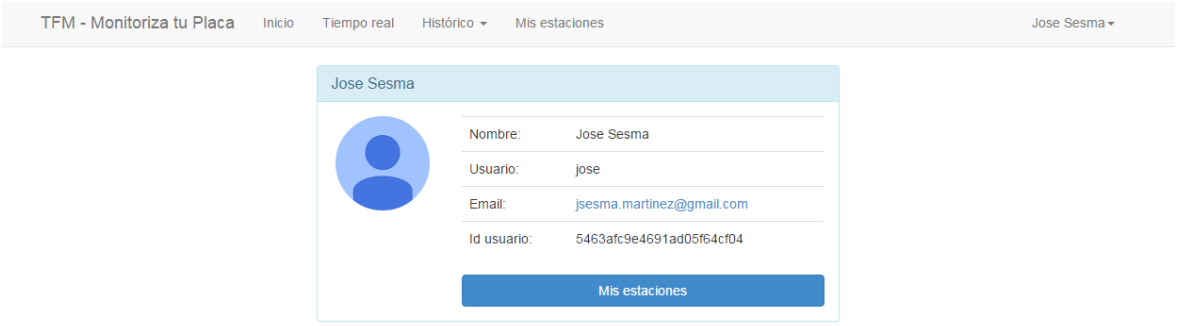

Figura 4.10.1. Pantalla Datos de usuario

5

Conclusiones

En el presente apartado se expone la conclusión del proyecto, se hace una recapitulación de los resultados obtenidos en cada una de las metas fijadas para la consecución del objetivo del proyecto y, para finalizar, se sugieren trabajos futuros relacionados con la materia aquí tratada.

5.1. Conclusiones

La web permite realizar una monitorización y control a tiempo de real sobre nuestras estaciones fotovoltaicas, sin movernos de casa y mediante una interfaz vistosa e intuitiva.

Esta web nos proporciona la comodidad de tener todos los datos requeridos de todas nuestras estaciones en la misma web y todo ello sin movernos de nuestro puesto de trabajo, lo que nos proporciona una comodidad y un ahorro bastante importante ya que no tenemos que desplazarnos a las diferentes instalaciones.

Se puede controlar las diferentes salidas de las que dispone el hardware de control de cada estación, pudiendo detener cualquier dispositivo o elemento conectado a la estación con el fin de evitar daños en la misma o cualquier otro motivo.

Permite monitorizar tanto los valores actuales de temperatura, tensión e iluminación como los históricos almacenados.

Además las gráficas se pueden exportar a diferentes formatos de archivos de salida para poder utilizarlos en cualquier otro proyecto o documento.

En definitiva, una web que nos permite ahorrar mucho tiempo y dinero y nos proporciona una gran comodidad para la monitorización y el control de las instalaciones fotovoltaicas.

5.2. Trabajos futuros

Como futuros trabajos se podría implementar la adquisición de más parámetros así como la tensión a la salida del inversor, la potencia, o el rendimiento de la instalación

Por otro lado, se podría implementar el control de la posición de las placas solares en función de la localización y de los resultados obtenidos por la temperatura, la luminosidad y la potencia de salida.

Otra mejora sería la visualización del valor medio de las diferentes medidas para poder tener un mayor control sobre la placa y ver comportamientos anormales de la misma.

Además se podría añadir el sensor GPS dentro del hardware para tener en todo momento la localización exacta y que no haga falta que el usuario la tenga que introducir a mano.

6

Presupuesto

A continuación se detalla el presupuesto del presente proyecto.

Descripción	Coste Unitario (€)	Nº unidades	Coste (€)
Horas de programación	30	560	16800
Raspberry Pi Modelo B	35	1	35
Webcam	20	1	20
Portátil Asus	699	1	699
Placa fotovoltaica monocristalina 12V - 20W	55	1	55
Regulador de tensión 12V	12	1	12
Sensor de temperatura LM35	0.5	1	0.5
Sensor LDR	1.20	1	1.20
Convertidor ADC MCP3008	3	1	3
Cables	4	1	4
Servidor VPS	22	1	22
Dominio	1	1	1
Coste total del desarrollo (IVA incluido)			**17652.7 €**

El total del presupuesto asciende a la cantidad total (IVA incluido) de:

Diecisiete mil seiscientos cincuenta y dos euros con setenta céntimos.

7

Anexo_A Referencias

A continuación se detallan las referencias utilizadas en el presente proyecto.

7.1. Referencias

[1]Vegas Portero, A. (2009) *Diseño de una instalación fotovoltaica de 200 kW en un edificio*. Universidad Carlos III de Madrid

[2]Blanco Sardinero, Israel. (2008) *Instalación solar fotovoltaica conectada a red sobre la azotea de una nave industrial*. Universidad Carlos III de Madrid

[3] Sheikh Ferdoush, Xinrong Li. (2014) *Wireless Sensor Network System Design using Raspberry Pi and Arduino for Environmental Monitoring Applications.* University of North Texas, Denton, Texas

[4] Página web solclima

http://www.soliclima.es/fotovoltaica-aislada

[5]Componentes de una instalación solar fotovoltarica, McGrawHill

http://www.mcgraw-hill.es/bcv/guide/capitulo/8448171691.pdf

[6] Página web GPIO Raspberry Pi

http://robologs.net/2014/04/12/tutorial-de-raspberry-pi-gpio-y-python-i/

[7] Express JS, primeros pasos

http://www.nodehispano.com/2012/01/express-el-framework-web-para-nodejs/

[8] Documentación API Express

http://expressjs.com/api.html

[9] MongoDB

http://docs.mongodb.org/

[10] Página web HighCharts

www.highcharts.com

[11] Página web Stackoverflow.

http://www.stackoverflow.com/

Blas Martínez, David. (2011) *Instalacion de paneles fotovoltaicos en bosal S.A.* Universidad de Zaragoza

Página web NodeJS

http://nodejs.org/

Dejan Mitrovic, Mirjana Ivanovic, Zoran Budimac, Milan Vidakovic (2014). *Radigost: Interoperable web-based multi-agent platform* University of Novi Sad, Serbia

Green, Martin A. (1982*). Solar Cells, operating principles, technology and system applications*, Prentice-Hall

Página web Google Maps

http://maps.google.es