

**UNIVERSIDAD SAN PABLO - CEU**

**GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN**

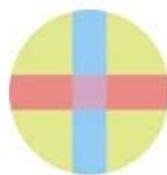


**TRABAJO DE FINAL DE GRADO**

**Control inteligente del frigorífico en el hogar**

Autor: Miguel González Gómez  
Director: Raúl García García

Septiembre 2014



UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería Informática y de Telecomunicación

## Calificación del Proyecto Fin de Carrera

### Datos del alumno

NOMBRE:

### Datos del Proyecto

TÍTULO DEL PROYECTO:

### Tribunal calificador

PRESIDENTE:

FDO.:

SECRETARIO:

FDO.:

VOCAL:

FDO.:

Reunido este tribunal el \_\_\_\_/\_\_\_\_/\_\_\_\_, acuerda otorgar al Proyecto de Fin de  
Carrera presentado por Don \_\_\_\_\_ la calificación de  
\_\_\_\_\_.

Dedicated to my parents and sister  
for their love, endless support  
and encouragement.

## Agradecimientos

---

Quisiera agradecer en primer lugar a mis padres y a mi hermana Beatriz. Gracias por haberme animado desde pequeño a perseguir aquello que más me gustara, por vuestro apoyo constante y por soportar todas aquellas horas que he pasado encerrado trasteando con el ordenador.

A mis compañeros de curso por la ayuda que me han brindado a lo largo de cuatro años, así como los buenos momentos que hemos pasado y seguiremos pasando. En especial a Jaime, Felipe, Javier y Yahi por los findes de semana que hemos estado encerrados estudiando en Villalba, fines de los que tengo grandes recuerdos. También, a Jorge, con quien he descubierto nuevas tecnologías, he asistido al evento Codemotion Madrid donde aprendimos un montón y en especial, por sus ánimos cuando he encontrado dificultades a la hora de realizar este proyecto.

Agradecer a mis compañeros de piso; Diane, Lydia, Raúl y Rubén, por aguantarme en el último año con el estrés de los trabajos y exámenes finales, y en especial, por soportar mis desorden debido a este proyecto. También, daros las gracias por haberme hecho disfrutar de los momentos libres que he tenido, así como de las vacaciones de navidad y verano.

A mis amigos de Logroño, quienes me han dado siempre su confianza a la hora de afrontar esta aventura. Y en especial, a mi amigo Álvaro por haber aguantado todas mis conversaciones sobre la carrera y los trabajos que he tenido que realizar.

Por último, me gustaría agradecer a la fundación Ramón Areces (El Corte Inglés) y su sistema de becas, así como a la Universidad CEU San Pablo, por darme la oportunidad de realizar este Grado en Ing. de Sistemas de Información y obtener una formación en un puesto de trabajo en el grupo de Informática El Corte Inglés, en concreto en Telecor, donde he podido formarme y crear amistades que no se van a olvidar.

## Resumen

---

Compañías como LG han presentado en los últimos años frigoríficos inteligentes; frigoríficos que se pueden controlar a través del móvil y permiten conocer los productos que hay en su interior a través de etiquetas inteligentes.

El objetivo del proyecto es traer parte de esta tecnología al usuario sin tener que realizar una gran inversión. No será necesario cambiar de frigorífico, para ello se ha creado un componente gracias a *Arduino*, el cual permite escanear los productos que se introduzcan o se consuman del frigorífico.

Existen aplicaciones que permiten llevar el gasto en el hogar, pero a diferencia de este proyecto, están enfocadas para insertar el gasto total de una compra. Este proyecto permite escanear producto por producto, y gracias a una base de datos colaborativa, obtener el gasto total de la compra; distinguir por tipo de alimento... También se podrá consultar los hábitos de alimentación, consultar que productos faltan, generar una lista de la compra, etc.

Esta solución se divide en tres bloques principales:

### Componente electrónico

Es la parte del proyecto más física donde se ha creado un aparato que permite la lectura de productos.

Para ello se cuenta con dos modos de entrada de productos; a través del código de barras ó de una etiqueta RFID, y de una salida Wireless para comunicar al servidor la entrada de los productos.

### Servidor centralizado

Es la parte lógica del proyecto, el encargado de recibir la información del componente electrónico, almacenarla y operar con ella para ofrecer a las aplicaciones cliente acceso a toda la información.

### Aplicación cliente

Es la parte de comunicación donde toda la información es consumida por el usuario a través de gráficas, resúmenes, listados, etc.

Mediante el cruce de datos generado por el componente electrónico y toda la información contenida en el servidor se le mostrará al cliente la siguiente información:

- Stock actual del frigorífico
- Estadísticas de consumo; listados y estadísticas con los productos que se han comprado mediante el uso de filtros como; tipo de productos, fechas, precios...
- Estadísticas de gastos

Al emplear un servidor centralizado se va a permitir la creación de herramientas de colaboración entre los usuarios de la aplicación para la administración y creación de:

- Identificación de productos; nombre, precio, imagen, etc.
- Planes de compra; permitirá compartir planes de compra que pueden orientarse a distintos fines como el conseguir un ahorro económico.
- Recetas; de esta manera se podrá consultar en base al stock que se dispone en el frigorífico qué recetas de cocina se pueden preparar.

## Abstract

---

Companies like LG have introduced smart fridges in the last years; fridges that can be controlled via smartphone and allow know the information about the products what contains inside thanks to smart labels.

The project aims to bring some of this technology to the user without having to make a large investment. No need to change the fridge, it has been built thanks to a *Arduino component*, which allows scanner the products.

There are applications that allow control the spending at home, but unlike this project, they are focused to include the total cost of a purchase. This project allows you to scan product by product, and thanks to a collaborative data base, obtain the total cost of the purchase; distinguished by type of food... Also eating habits, create a shopping list, etc.

This solution was divided into three main blocks:

### Electronic Component

It is part of the more physical project which has created a device that allows reading the products.

To do this it has two input modes; through barcode or an RFID tag, and a wireless output to communicate to the server.

### Centralized Server

It is the logic of the project, responsible for receiving the data from the electronic component, store it and server it to client applications.

### Client Application

It is the communication part where all the information is consumed by the user through graphical summaries, lists, etc.

By crossing data generated by the electronic component and all information contained on the server will show to the customer the following information:

- Current Stock of the fridge
- Consumption statistics; lists and statistics with the products purchased using filters like; product, dates, prices...

- Expenditure statistics

By using a centralized server will allow the creation of tools for collaboration between users of the application for the creation and management:

- Product identification; name, price, image, etc.
- Purchase plans; allow share purchase plans that can be adjusted to achieve different purposes like saving money.
- Recipes; this way you can see based on the stock that you have in the fridge which recipes can be prepared.

## Índice general

---

<b>Agradecimientos</b>	<b>3</b>
<b>LISTA DE FIGURAS</b>	<b>13</b>
<b>1. INTRODUCCIÓN</b>	<b>15</b>
<b>2. ESTADO DE LA CUESTIÓN</b>	<b>16</b>
2.1. DOMÓTICA . . . . .	16
2.1.1. Historia . . . . .	16
2.1.2. X10 . . . . .	17
2.1.3. CEBus . . . . .	17
2.1.4. EIB . . . . .	18
2.1.5. KNX . . . . .	18
2.2. ARDUINO . . . . .	19
2.2.1. Hardware . . . . .	19
2.2.2. Arduino Yún . . . . .	22
2.2.3. Pantalla TFT . . . . .	24
2.2.4. RFID . . . . .	27
2.2.5. Lector de código de barras . . . . .	29
2.3. COMPARATIVA CON OTROS COMPONENTES . . . . .	29

2.3.1. Raspberry PI . . . . .	29
2.3.2. Z-Wave . . . . .	30
2.4. CONCLUSIONES . . . . .	31
<b>3. REQUISITOS DEL SISTEMA</b>	<b>33</b>
3.1. Introducción . . . . .	33
3.2. Propósito de este capítulo . . . . .	33
3.3. Ámbito del sistema . . . . .	33
3.4. Definiciones acrónimos y abreviaturas . . . . .	33
3.5. Referencias . . . . .	34
3.6. Descripción general . . . . .	34
3.7. Perspectiva del producto . . . . .	34
3.8. Funciones del sistema . . . . .	35
3.9. Características de los usuarios . . . . .	35
3.10. Restricciones . . . . .	35
3.11. Suposiciones y dependencias . . . . .	35
3.12. Requisitos específicos . . . . .	35
3.13. Requisitos funcionales . . . . .	36
3.13.1. Caso de Uso: Alta cuenta usuario . . . . .	36
3.13.2. Caso de Uso: Alta de un producto . . . . .	36
3.13.3. Caso de Uso: Añadir un producto al frigorífico vía web . . . . .	37

3.13.4. Caso de Uso: Vincular el lector a la cuenta de un usuario . . . . .	37
3.13.5. Caso de uso: Seleccionar un frigorífico en el lector . . . . .	38
3.13.6. Caso de uso: Añadir un producto al frigorífico vía lector . . . . .	38
3.13.7. Caso de uso: Sacar un producto del frigorífico vía lector . . . . .	39
3.13.8. Caso de uso: Obtener información del frigorífico . . . . .	40
3.13.9. Caso de uso: Desvincular el lector . . . . .	40
3.14. Requisitos de interfaces externas . . . . .	41
3.14.1. Interfaces de usuario . . . . .	41
3.14.2. Interfaces hardware . . . . .	41
3.14.3. Interfaces software . . . . .	41
3.14.4. Interfaces de comunicación . . . . .	41
3.15. Requisitos de rendimiento . . . . .	41
3.16. Requisitos tecnológicos . . . . .	42
<b>4. ARQUITECTURA DEL SISTEMA</b>	<b>43</b>
4.1. Diagrama general . . . . .	43
4.2. Componente hardware . . . . .	43
4.3. Componente servidor . . . . .	44
4.4. Componente cliente . . . . .	45
<b>5. DISEÑO E IMPLEMENTACIÓN</b>	<b>46</b>

5.1. COMPONENTE HARDWARE . . . . .	46
5.1.1. Configurando el sistema . . . . .	47
5.1.2. Problemas detectados . . . . .	50
5.1.3. Decodificando el lector de código de barras . . . . .	53
5.1.4. Vinculación del lector . . . . .	56
5.1.5. Opciones del lector . . . . .	56
5.2. COMPONENTE SERVIDOR . . . . .	61
5.2.1. Configurando el sistema . . . . .	61
5.2.2. Control de Versiones, tests unitarios e integración continua . . . . .	64
5.2.3. Problemáticas para la creación de tests unitarios . . . . .	65
5.2.4. Esquema de la base de datos . . . . .	69
5.2.5. Esquema del servidor . . . . .	70
5.2.6. Vinculación del lector . . . . .	77
5.2.7. Generación de código de barras . . . . .	78
5.3. COMPONENTE CLIENTE . . . . .	80
5.3.1. Flujo de la aplicación . . . . .	82
5.3.2. Secciones de la página web . . . . .	82
5.3.3. Sistema de rutas . . . . .	88
<b>6. CONCLUSIONES</b>	<b>91</b>
6.1. Primer paso . . . . .	91

---

6.2. Segundo paso . . . . .	92
6.3. Tercer paso . . . . .	92
6.4. Cuarto paso . . . . .	93
<b>7. LÍNEAS FUTURAS</b>	<b>94</b>
7.1. COMPONENTE HARDWARE . . . . .	94
7.2. COMPONENTE SERVIDOR Y SOFTWARE . . . . .	95
<b>8. BIBLIOGRAFÍA</b>	<b>96</b>

## Índice de figuras

---

2.1. Arduino uno . . . . .	20
2.2. IDE Arduino . . . . .	21
2.3. Frontal arduino . . . . .	22
2.4. Componentes de arduino . . . . .	23
2.5. Pantalla TFT . . . . .	24
2.6. TFT - Dibujando . . . . .	26
2.7. TFT - Imagen . . . . .	27
2.8. Lector de código de barras . . . . .	29
2.9. Z-Wave . . . . .	31
5.1. Lector Arduino . . . . .	46
5.2. Arduino Yún Web Password . . . . .	47
5.3. Arduino Yún Web Diagnostic . . . . .	48
5.4. Arduino Yún Web Config . . . . .	49
5.5. Arduino Yún Web Rebooting . . . . .	50
5.6. Vincular lector . . . . .	56
5.7. Vincular lector . . . . .	57
5.8. Pantalla de inicio del lector . . . . .	57
5.9. Selector de frigoríficos . . . . .	58

5.10. No dispone de frigoríficos . . . . .	58
5.11. Entrar productos en el frigorífico . . . . .	59
5.12. Sacar productos del frigorífico . . . . .	59
5.13. Confirmar desvincular lector . . . . .	60
5.14. Vincular lector . . . . .	60
5.15. Tets - Travis CI . . . . .	64
5.16. Esquema de la base de datos . . . . .	70
5.17. Flujo de trabajo de CodeIgniter . . . . .	71
5.18. Flujo de la vinculación de un lector . . . . .	78
5.19. Flujo de la aplicación web . . . . .	82
5.20. Web - Home pública . . . . .	83
5.21. Web - Sección frigorífico . . . . .	84
5.22. Web - Sección compras . . . . .	85
5.23. Web - Sección compra . . . . .	86
5.24. Web - Sección supermercados . . . . .	86
5.25. Web - Sección de un supermercado . . . . .	87
5.26. Web - Añadir supermercado . . . . .	87
5.27. Web - Alta de un lector . . . . .	88

## 1. INTRODUCCIÓN

---

El Internet de las cosas (IoT) es un concepto que nació en el Instituto de Tecnología de Massachusetts (MIT) y es considerado la siguiente evolución de Internet. El propósito es sencillo, que todo esté conectado a Internet; electrodomésticos que se pueden controlar remotamente; vehículos que avisen al taller cuando surja un problema mecánico, etc.

La previsión para el 2020 es de 24 mil millones de dispositivos conectados a Internet. Esta evolución en una Red de objetos intercomunicados permitirá recibir una ingente cantidad de información de todo lo que nos rodea e interactuar con el mundo físico.

Todos estos nuevos conceptos han sido el detonante de este proyecto, intentar acercar nuestros frigoríficos a esta evolución en la tecnología. Para ello se ha desarrollado un componente, el cual se puede colocar en el frigorífico, para poder registrar los productos que compramos. Este componente se conecta a Internet para transmitir toda la información a un servidor permitiendo su análisis; productos que disponemos en el frigorífico, estadísticas de consumo, calcular listas de la compra, etc.

## 2. ESTADO DE LA CUESTIÓN

---

En este capítulo se va a hacer un estudio de la tecnología existente para llevar las cosas a Internet en el hogar. En concreto se profundizará en Domótica para casas inteligentes y en los distintos componentes que nos permiten desarrollar nuevos conceptos de forma sencilla.

### 2.1. DOMÓTICA

El término domótica proviene de la unión de las palabras *domus* (que significa casa en latín) y *tica* (de automática, palabra en griego). La Real Academia Española lo define como el «conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda».

En este proyecto se utilizará el concepto de domótica como la tecnología aplicada al Internet de las Cosas en el hogar, es decir, conectar nuestro hogar a la nube.

#### 2.1.1. Historia

El hogar inteligente y automático comenzó a imaginarse en historias de ciencia ficción a inicios del siglo XX. En 1956 se emitió la película *Design for Dreaming* 2.1.1 donde una mujer cae dormida y tiene una serie de sueños futuristas, incluyendo a *Frigidaire*, la cocina automatizada del futuro. De ahí que no se pueda establecer una fecha concreta donde establecer el inicio de la domótica.



No obstante, si hemos de establecer una fecha de importante tenemos que hablar de 1975 con la aparición de *X10*, un protocolo de comunicación que básicamente permite el control de las luces del hogar, para ello, utiliza la línea eléctrica como medio de comunicación. Debido a las limitaciones del protocolo de comunicación *X10* se han desarrollado nuevas tecnologías para suplir estas carencias; CEBus, EIB, KNX...

A continuación se va a dar una breve explicación de cada una de estas tecnologías existentes en el mercado.

### 2.1.2. X10

X10 es un protocolo de comunicación para el control remoto de dispositivos eléctricos. Fue desarrollado en 1978 por *Pico Electronics of Glenrothes*, Escocia, siendo la primera tecnología domótica en aparecer.

Utiliza el cableado eléctrico como medio de comunicación que, debido a la limitación en ancho de banda que proporciona, solo es capaz de soportar hasta 256 dispositivos. A pesar de ello, es una tecnología ampliamente utilizada en Norteamérica y Europa por su característica de autoinstalable, sin necesidad de cableado adicional.



Las señales de control de X10 se basan en la transmisión de ráfagas de pulsos de RF (120 kHz) que representan información digital. Estos pulsos se sincronizan en el cruce por cero de la señal de red (50 Hz ó 60 Hz). Con la presencia de un pulso en un semicírculo y la ausencia del mismo en el semicírculo siguiente se representa un '1' lógico y a la inversa se representa un '0'. A su vez, cada orden se transmite 2 veces, con lo cual toda la información transmitida tiene cuádruple redundancia. Cada orden involucra 11 ciclos de red (220 ms para 50 Hz y 183,33, para 60Hz).

Primero se transmite una orden con el Código de Casa y el Número de Módulo que direccionan el módulo en cuestión. Luego se transmite otro orden con el código de función a realizar (Function Code): Apagar/Encender todas las unidades, apagar/encender una unidad, comprobar el estado de una o varias unidades...

### 2.1.3. CEBus

Bajo la necesidad de mejorar la limitada capacidad del protocolo X10 surgió CEBus. El estándar CEBus se desarrolló por la EIA (Electronic Industries Allianza) y su primera especificación salió en 1992.

CEBus, a diferencia de X10, se comunica sobre varias tecnologías; cableado eléctrico; cable coaxial; infrarrojos; radio frecuencia y cable óptico. Y además, se prescinde de un controlador central distribuyéndose la inteligencia entre todos los componentes del sistema.

Para la comunicación entre distintos medios físicos se necesita utilizar un router intermedio. La información se transmite en forma de mensajes a través de una dirección broadcast o una específica al dispositivo receptor (cada aparato *CEBus* tiene una dirección única asignada).

Por último, se definió un lenguaje orientado a objetos para el control de los dispositivos *CEBus* llamado *CAL* (Common Appliance Language).

#### 2.1.4. EIB

El European Installation Bus (*EIB*) es un sistema domótico impulsado por la Unión Europea, en concreto, por la *EIBA*, una asociación de empresas líderes en instalaciones eléctricas. El objetivo era crear un estándar europeo, y de esta manera, competir contra la importación de productos norteamericanos y japoneses.

*EIB* utiliza un solo bus de comunicación, reduciendo el tiempo de instalación, aunque a diferencia de *CEBus* es necesaria la instalación de su propio cableado. Permite unir hasta 64 dispositivos bajo una sola línea, y mediante acopladores un total de 256 aparatos. Al igual que *CEBus* no es necesario un puesto de control central ya que el sistema trabaja de forma centralizada comunicándose a través del modelo OSI.

Debido al impulso de la Unión Europea y al tratarse de un estándar existen muchos fabricantes de productos *EIB* siendo compatibles entre ellos. Se ha creado una asociación de más de cien miembros y existiendo unas veinte empresas que suministran productos, siendo las más importantes *Siemens*, *ABB*, *Temper*, *Grasslin* y *Niessen*.

#### 2.1.5. KNX

*KNX* es el sucesor y la convergencia de tres estándares previos: el European Home System Protocol (EHS), BâtiBUS, y el European Installation Bus (*EIB*). En 2002 se publicó la especificación *KNX*, permitiendo competir en calidad, prestaciones y precios con otros sistemas existentes, ya que reúne las mejores características de sus predecesores.

El estándar *KNX* permite el uso de cuatro medios de comunicación; TP (*Twisted Pair*), PL (*Power Line*), RF (*Radio Frequency*) e IP (*Ethernet*). Al igual que sus pre-

decesores es un sistema descentralizado por lo cual no es necesario la utilización de un controlador principal.

## 2.2. ARDUINO

Una buena definición de lo que es Arduino lo da la Wikipedia; «Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.».

Para este proyecto de fin de grado se va a utilizar *Arduino* para desarrollar el hardware que permita la lectura de códigos de barras de los productos. Para ello se va a contar con una placa *Arduino Yún* y distintos componentes como una pantalla TFT, un lector de código de barras, resistencias, pulsadores, etc.

A continuación se va a explicar brevemente el hardware *Arduino*, *Arduino Yún*, una pantalla TFT desarrollada para *Arduino*, un lector RFID y un lector de código de barras.

### 2.2.1. Hardware

El hardware consiste en una placa con un microcontrolador *Atmel AVR* y puertos de entrada/salida. Los microcontroladores más usados son el *Atmega168*, *Atmega328*, *Atmega1280*, *ATmega8* por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños.

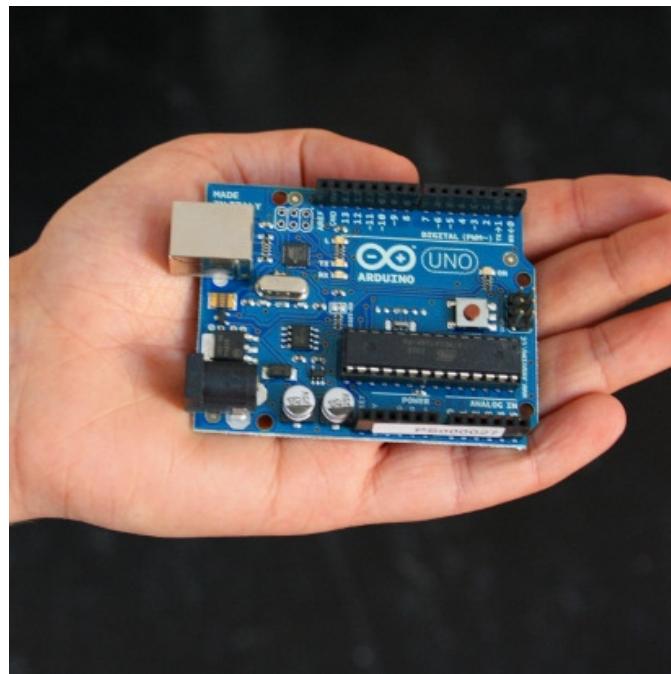


Figura 2.1: Arduino uno

Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación *Wiring* y el cargador de arranque que es ejecutado en la placa.

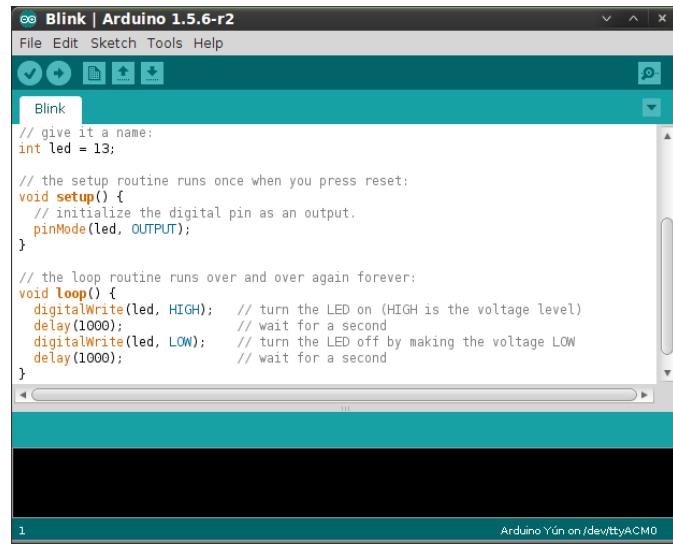


Figura 2.2: IDE Arduino

El lenguaje de alto nivel *Wiring*, implementado en C/C++, permite una programación sencilla del hardware. Comparte todas las funciones estándar de C, algunas de C++ y añade las propias para el manejo de puertos (digitales, analógicos).

A continuación se muestra un código sencillo para hacer parpadear un diodo led cada segundo, se utiliza el puerto 8 digital de la placa *Arduino*:

```
1 void setup() {  
2     pinMode(8, OUTPUT);  
3 }  
  
4  
5     boolean encendido = false;  
6 void loop() {  
7     if(!encendido) {  
8         digitalWrite(8, HIGH);  
9     } else {  
10        digitalWrite(8, LOW);  
11    }  
12    encendido!=encendido;  
13    delay(1000);  
14 }
```

### 2.2.2. Arduino Yún

Existen una gran variedad de placas Arduino, tanto oficiales como no oficiales; Arduino uno, Arduino Mega, Arduino Leonardo... Y una de las más recientes, la cual usará en este proyecto, es la placa *Arduino Yún*.

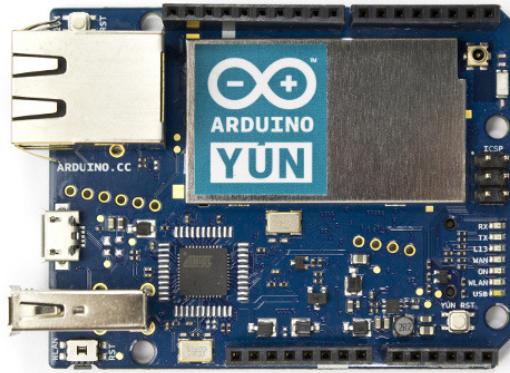


Figura 2.3: Frontal arduino

La ventaja principal, y por la que me he decantado por comprar esta placa, es la combinación de Arduino (*microcontrolador ATmega 32U4*) con un sistema operativo Linux (*microprocesador AR9331*) con conectividad WiFi. Esto permite utilizar todo el potencial de Arduino en el desarrollo de nuestro hardware y obtener todo el potencial de una máquina Linux, a través del sistema operativo *Linino*, para ejecutar comandos, scripts, aplicaciones y cualquier cosa que se nos ocurra.

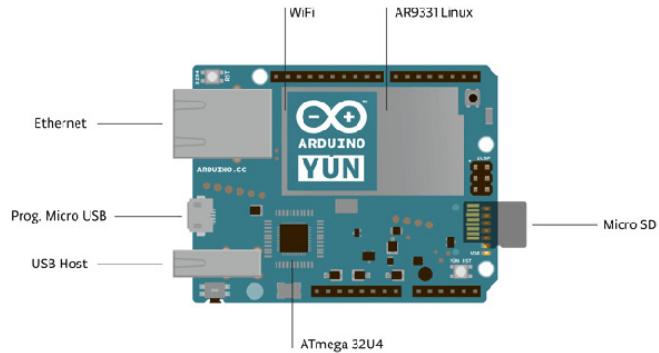


Figura 2.4: Componentes de arduino

La comunicación entre Arduino y Linux se realiza a través de la librería *Bridge*, facilitando el intercambio de información y órdenes entre ambos micros. La librería da el potencial al microcontrolador de ejecutar scripts, comunicarse con interfaces de red, tener acceso a una memoria compartida entre el microprocesador y el microcontrolador, acceder al sistema de ficheros...

Las características de *Arduino Yún* son las siguientes:

#### **AVR Arduino microcontroller**

Microcontroller	ATmega32u4
Operating Voltage	5V
Input Voltage	5V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

### Linux microprocessor

Processor	Atheros AR9331
Architecture	MIPS @400MHz
Operating Voltage	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11b/g/n
USB Type-A	2.0 Host/Device
Card Reader	Micro-SD only
RAM	64 MB DDR2
Flash Memory	16 MB

#### 2.2.3. Pantalla TFT

Uno de los componentes para dotar de comunicación a nuestra placa Arduino es utilizar una pantalla TFT. Existen variedad de pantallas TFT; diferentes dimensiones, con soporte táctil, mayor rapidez de refresco... Para el proyecto he optado por comprar la oficial de Arduino:

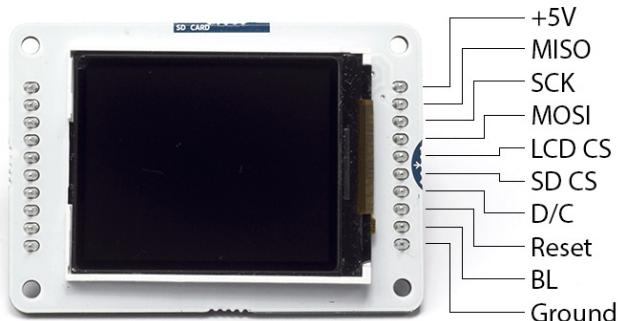
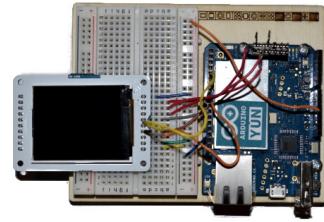


Figura 2.5: Pantalla TFT

Es una pantalla de 1,77" de diagonal, con 160x128 pixels de resolución, y con un lector de tarjetas microSD para cargar imágenes. La pantalla está pensada para encajar con las placas *Arduino Esplora* ó *Arduino Robot*, con el resto de placas Arduino se pueden realizar las conexiones a mano.



A través de la librería *TFT* se pueden dibujar elementos en la pantalla o, mediante una memoria microSD, cargar imágenes en la pantalla. A continuación se expone una manera para dibujar un círculo y un cuadrado en la pantalla:

```

1   ...
2
3   void setup() {
4       TFTscreen.begin();
5       TFTscreen.background(255, 255, 255);
6
7   }
8
9   void loop() {
10      TFTscreen.stroke(0,0,0);
11      TFTscreen.fill(127,127,127);
12      TFTscreen.circle(TFTscreen.width()/2, TFTscreen.height()/2,
13                      30);
14
15      TFTscreen.fill(50,40,30);
16      TFTscreen.rect(TFTscreen.width()/2, TFTscreen.height()/2,
17                      30, 30);
18
19      delay(1500);
20  }
```

A continuación se muestra una captura de como se visualiza el dibujo en la pantalla:

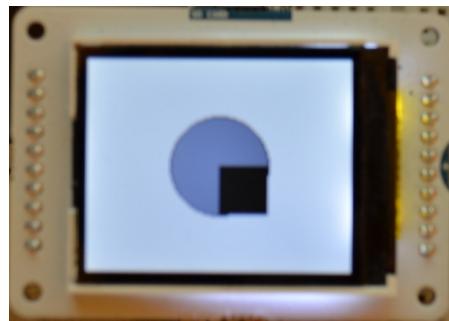


Figura 2.6: TFT - Dibujando

En cambio, para cargar una imagen desde la tarjeta de memoria y mostrarla en la pantalla TFT, primero se tiene que esperar a que la memoria SD esté inicializada y a continuación, leer la imagen y renderizarla en pantalla:

```
1     ...
2
3     void setup() {
4         TFTscreen.begin();
5         TFTscreen.background(255, 255, 255);
6
7         if (!SD.begin(sd_cs)) {
8             return; // Error SD
9         }
10    }
11
12    void loop() {
13        logo = TFTscreen.loadImage("f1.bmp");
14        TFTscreen.image(logo, 0, 0);
15        delay(1500);
16    }
```

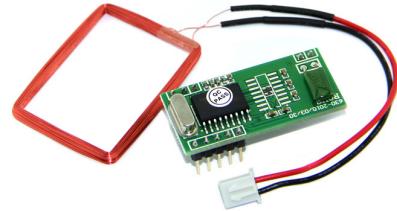
A continuación se muestra una captura de como se visualiza la imagen en la pantalla:



Figura 2.7: TFT - Imagen

#### 2.2.4. RFID

El lector *RFID* es uno de los componentes empleados para poder escanear productos. En concreto se ha utilizado para el proyecto un lector *RFID* de 125K con las siguientes características:



- Working voltage: 3.5-5V
- Frequency: 125Khz
- Baud rate: 9600
- Interface Type: TTL Level RS232 format
- Reception range: 30mm 50mm

El código para trabajar con el lector *RFID* es muy sencillo, la única "complicación" es tener que utilizar la librería *SoftwareSerial* para simular dos pines como puerto serie (ya que *Arduino Yún* tiene reservados los pines 0 y 1 de puerto serie para la comunicación con Linux a través de la librería *bridge*).

No se pueden utilizar los pines dedicados de puerto serie (0 y 1) ya que *Arduino Yún* los utiliza para la comunicación entre el microcontrolador *ATmega32U4* y el microprocesador *Atheros AR9331* a través de la librería *Bridge*.

El código utilizado para escanear tarjetas *RFID* es el siguiente:

```
1 //Lector RFID
2 #include <SoftwareSerial.h>
3 #define rxPin 11
4 #define txPin 12
5 SoftwareSerial RFID = SoftwareSerial(rxPin, txPin); // RX and TX
6 char arrayTag[10];
7
8 void setup() {
9     RFID.begin(9600);
10 }
11
12 void loop() {
13     if(leerYProcesarEtiquetaRFID()) {
14         // Hacer algo
15     }
16 }
17
18 boolean leerYProcesarEtiquetaRFID() {
19     int leidoTag = 0;
20
21     String tag;
22     while(RFID.available() > 0) {
23         leidoTag = 1;
24         int rfid = RFID.read();
25
26         if(rfid != 13 && rfid != 10) {
27             tag += (char)rfid;
28         }
29     }
30     if(leidoTag) {
31         tag.toCharArray(arrayTag, 10);
32     }
33
34     return leidoTag != 0;
35 }
```

### 2.2.5. Lector de código de barras

Además de utilizar como método de entrada un lector *RFID* se ha utilizado otro método tradicional, el lector de código de barras. Para ello se ha buscado un lector que sea *USB HID*, es decir, que no necesite la instalación de drivers para funcionar.



© THOMAS-IT

Figura 2.8: Lector de código de barras

En la sección *COMPONENTE HARDWARE* se explica la configuración de *Arduino Yún* y como se decodifica la entrada de dígitos del lector con *Python*.

## 2.3. COMPARATIVA CON OTROS COMPONENTES

La elección de *Arduino Yún* se ha realizado teniendo en cuenta las alternativas existentes en el mercado, encontrando que es la más adecuada para el tipo de proyecto que se va a realizar. A continuación se muestran algunos de los componentes que se han tenido en cuenta.

### 2.3.1. Raspberry PI

La *Raspberry Pi* es un computador de bajo costo y del tamaño de una tarjeta de crédito, el cual se puede conectar a un monitor o aun ordenador, junto a un teclado y un ratón. Permite realizar todo lo que puede hacer un ordenador de sobremesa; navegar por internet, reproducir vídeo en alta definición, programas de ofimática e incluso correr videojuegos. Además, la placa de *Raspberry Pi* cuenta con 8 pines GPIO que permiten controlar componentes electrónicos como leds, pulsadores, etc.



A diferencia de *Arduino Yún* el propósito de la *Raspberry PI* es ser un ordenador en miniatura, corriendo un sistema Linux, y aunque pueda comunicarse a través de sus pines GPIO ese no es su propósito. *Arduino Yún* en cambio es un microcontrolador especializado en controlar componentes, además de contar con la ventaja de poseer un microprocesador con una versión de Linux. Para ello cuenta con un mayor número de pines entrada salida; analógicos, comunicación SPI... Que serán útiles para añadir componentes como un lector RFID, una pantalla TFT y botones para controlarla; así como se pueden añadir sensores de temperatura, distancia, etc. para poder controlar parámetros como la temperatura del frigorífico, la hora y el tiempo que ha estado abierto el frigorífico, etc.

### 2.3.2. Z-Wave

Z-Wave es el estándar internacional para la interconexión inalámbrica de los sistemas domóticos de su casa. Puede gestionar la iluminación, la electricidad, las persianas, las alarmas, etc. Los productos Z-Wave de diferentes fabricantes trabajan bien juntos permitiendo el control completo de su hogar.



Figura 2.9: Z-Wave

La integración de *Arduino* con el estándar *Z-Wave* no es sencillo. Es posible encontrar módulos wireless que son compatibles con la frecuencia que emiten los componentes *Z-Wave* pero hay que implementarse todo el protocolo *Z-Wave* de comunicación.

Además, si se optara por realizar esta integración, sería necesario contar un servidor *Z-Wave* para realizar la comunicación. Y los productos de *Z-Wave* tienen un precio bastante elevado, por lo que el coste final del proyecto se incrementaría notablemente, lo cual no cumple con uno de los objetivos de realizar un componente de bajo coste.

## 2.4. CONCLUSIONES

En resumen, se puede observar que a lo largo de la historia se han desarrollado múltiples tecnologías para facilitar y mejorar las condiciones de vida en el hogar a través de tecnología.

A día de hoy existen proyectos para dotar de inteligencia a los frigoríficos, sin embargo, solamente son prototipos debido a su alto coste de fabricación.

Para traer al hogar parte de esta tecnología se ha aprovechado de los avances en hardware libre (a través de *Arduino*) para desarrollar un prototipo de frigorífico inteligente. Este prototipo tendrá como objetivo dotar de inteligencia al



frigorífico sin tener que realizar una gran inversión en hardware.

### 3. REQUISITOS DEL SISTEMA

---

#### 3.1. Introducción

Esta especificación se ha elaborado inspirándose en las directrices dadas en la norma ANSI/IEEE 830-1998 [IEEE, 98].

#### 3.2. Propósito de este capítulo

El objeto de la especificación es definir de manera clara y precisa todas las funciones y restricciones del sistema que se desea construir.

#### 3.3. Ámbito del sistema

El objetivo de este proyecto es el de elaborar un sistema que permita dotar de inteligencia al frigorífico del hogar utilizando para ello una placa Arduino. El sistema deberá ser capaz de leer productos y transmitirlos a un servidor vía WiFi. El usuario entonces podrá consultar los productos que contiene su frigorífico a través de una aplicación.

#### 3.4. Definiciones acrónimos y abreviaturas

HID - *Human Interface Device*

USB - *Universal Serial Bus*

PHP - *Hypertext Preprocessor*

LAMP - *Linux, Apache, MySQL, PHP*

MVC - *Modelo, Vista, Controlador*

JSON - *JavaScript Object Notation*

SMTP - *Simple Mail Transfer Protocol*

SQL - *Structured Query Language*

EAN - *European Article Number ó International Article Number*

HTML - *HyperText Markup Language*

W3C - *World Wide Web Consortium*

CSS - *Cascading Style Sheets*

### 3.5. Referencias

Todas las referencias bibliográficas se muestran en el capítulo correspondiente de la presente memoria.

### 3.6. Descripción general

En esta sección se presentan los aspectos generales, las restricciones y otros factores que afecten al desarrollo de éste.

### 3.7. Perspectiva del producto

El componente hardware identificará los productos que se escaneen y se comunicará con el servidor para notificarle la entrada / salida de los mismos. La aplicación cliente se comunicará con el servidor para mostrarle al usuario la información.

### 3.8. Funciones del sistema

La función del componente hardware es la identificación de la entrada / salida de productos del frigorífico y la comunicación de dicha información al servidor. La aplicación cliente, en cambio, será la encargada de recoger esa información del servidor y mostrársela al cliente.

### 3.9. Características de los usuarios

Los usuarios no deben tener ningún conocimiento técnico para ser capaces de manejar el sistema. El único requisito es que dispongan de un router WiFi con conexión a Internet.

### 3.10. Restricciones

El componente hardware deberá estar al alcance de una red WiFi para poder comunicarse con el servidor.

### 3.11. Suposiciones y dependencias

En esta sección se presentan las suposiciones y dependencias del sistema.

### 3.12. Requisitos específicos

En este apartado se presentan los requisitos funcionales que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son esenciales, es decir, no sería aceptable un sistema que no satisfaga alguno de los requisitos aquí presentados.

### 3.13. Requisitos funcionales

Los requisitos funcionales se expondrán en forma de casos de uso.

#### 3.13.1. Caso de Uso: Alta cuenta usuario

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El usuario debe acceder a la página web a través de un navegador.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** El usuario se da de alta en el sistema a través de un formulario de registro.

**Escenario alternativo.** El usuario introduce valores no válidos en el formulario y el sistema le avisa de ellos para que sean corregidos.

**Cómo probarlo.** Una vez el usuario se ha registrado accede al panel de usuario a utilizando para ello el formulario de logueo y sus credenciales.

#### 3.13.2. Caso de Uso: Alta de un producto

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El usuario debe estar logueado en el sistema.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** El usuario accede a un supermercado en la zona de *Supermercados* y da de alta un producto rellenando para ello todos los datos requeridos.

**Escenario alternativo.** El sistema le avisa de que hay datos mal introducidos que necesitan ser corregidos.

**Cómo probarlo.** El nuevo producto debe aparecer en la web para el supermercado elegido.

### 3.13.3. Caso de Uso: Añadir un producto al frigorífico vía web

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El usuario debe estar logueado en el sistema y tener creado un frigorífico.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** El usuario accede a uno de sus frigoríficos, clicka en el botón para añadir un producto e introduce su código de barras seleccionando finalmente el producto a introducir.

**Escenario alternativo.** El sistema no encuentra el producto con el código de barras seleccionado, se debe añadir el producto al supermercado que corresponde y volver a intentar.

**Cómo probarlo.** El producto añadido debe aparecer en el frigorífico.

### 3.13.4. Caso de Uso: Vincular el lector a la cuenta de un usuario

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El usuario debe estar logueado en el sistema y tener creado un lector sin vincular. El lector debe estar sin vincular y encendido.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** El usuario genera dos códigos de barras de vinculación, los escanea con el lector, una vez termina la vinculación debe mostrar la pantalla de opciones (Introducir un producto, sacar un producto, desvincular el lector...).

**Escenario alternativo.** Se produce un error al iniciar la vinculación y el lector lo indica con una pantalla de Error y un mensaje de volver a intentar.

**Cómo probarlo.** Se pueden acceder a las opciones de funcionamiento del lector, desapareciendo así la pantalla de vinculación.

#### 3.13.5. Caso de uso: Seleccionar un frigorífico en el lector

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El componente hardware debe estar configurado y operativo.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** Se accede a la opción de menú para seleccionar un frigorífico, aparece un listado de frigoríficos y se selecciona el deseado.

**Escenario alternativo.** El usuario no tiene ningún frigorífico creado por lo que se da un aviso para que primero se añada uno.

**Cómo probarlo.** Al escanear un producto para añadirlo a la aplicación web este aparece en el frigorífico seleccionado.

#### 3.13.6. Caso de uso: Añadir un producto al frigorífico vía lector

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El producto debe tener un código de barras legible. El componente hardware debe estar configurado y operativo.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** Se accede a la opción de menú para introducir un producto en el frigorífico, se escanea el código de barras y la información se envía vía WiFi al servidor.

**Escenario alternativo.** El sistema no reconoce el código de barras (ilegible) por lo que se debe añadir vía web el producto. El producto no está en el servidor por lo que se debe crear primero.

**Cómo probarlo.** Al acceder al frigorífico del usuario vía web se comprueba que se ha añadido el producto al frigorífico.

### 3.13.7. Caso de uso: Sacar un producto del frigorífico vía lector

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El producto debe tener un código de barras legible. El componente hardware debe estar configurado y operativo.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** Se accede a la opción de menú para sacar un producto del el frigorífico, se escanea el código de barras y la información se envía vía WiFi al servidor.

**Escenario alternativo.** El sistema no reconoce el código de barras (ilegible) por lo que se debe añadir vía web el producto. El producto no está en el frigorífico por lo que se muestra un aviso.

**Cómo probarlo.** Al acceder al frigorífico del usuario vía web se comprueba que el producto no aparece.

### 3.13.8. Caso de uso: Obtener información del frigorífico

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** Tener acceso a la aplicación cliente y estar logueados en el sistema.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** El sistema recupera la información de productos del servidor y nos muestra la información.

**Cómo probarlo.** El usuario se identifica en la aplicación y debe ser capaz de ver la información del frigorífico.

### 3.13.9. Caso de uso: Desvincular el lector

**Actor principal.** Usuario.

**Actores de apoyo.** Ninguno.

**Precondiciones.** El componente hardware debe estar configurado y operativo.

**Postcondiciones.** Ninguna.

**Escenario principal de éxito.** Se accede a la opción de menú para desvincular el lector y se confirma la opción.

**Escenario alternativo.** Ninguno.

**Cómo probarlo.** El lector muestra la pantalla de vinculación y desde la página web el lector creado aparece para ser vinculado.

## 3.14. Requisitos de interfaces externas

### 3.14.1. Interfaces de usuario

Los usuarios interactuarán con el sistema a través de un componente hardware arduino y de una o varias aplicaciones cliente basadas en la nube.

### 3.14.2. Interfaces hardware

La interfaz hardware que se ha diseñado está basado en Arduino Yún. Se comunicará con el servidor vía WiFi. El componente puede utilizar una batería como sistema de alimentación, recargable mediante cable usb, ó estar conectado directamente la corriente.

### 3.14.3. Interfaces software

El cliente se comunicará mediante el sistema a través de una aplicación cliente.

### 3.14.4. Interfaces de comunicación

Todo el sistema está basado en un servidor en la nube por lo que el protocolo de comunicación será HTTP.

## 3.15. Requisitos de rendimiento

El componente hardware debe reconocer el producto y comunicarlo al servidor en la mayor brevedad posible, en menos de 5 segundos. En caso de que haya problemas de comunicación con el servidor deberá mostrar información acerca del problema al usuario.

La aplicación cliente debe ser capaz de obtener la información del servidor de forma instantánea (menos de un segundo), siempre que las condiciones de Internet sean

adecuadas.

Estos requisitos deberán cumplirse si lo hacen los requisitos tecnológicos especificados a continuación.

### **3.16. Requisitos tecnológicos**

El componente hardware estará basado en Arduino Yún, y, para realizar la comunicación con el cliente se le dotará de una pantalla TFT.

La aplicación cliente web deberá correr en un navegador con soporte HTML 5, tener habilitado JavaScript y las Cookies.

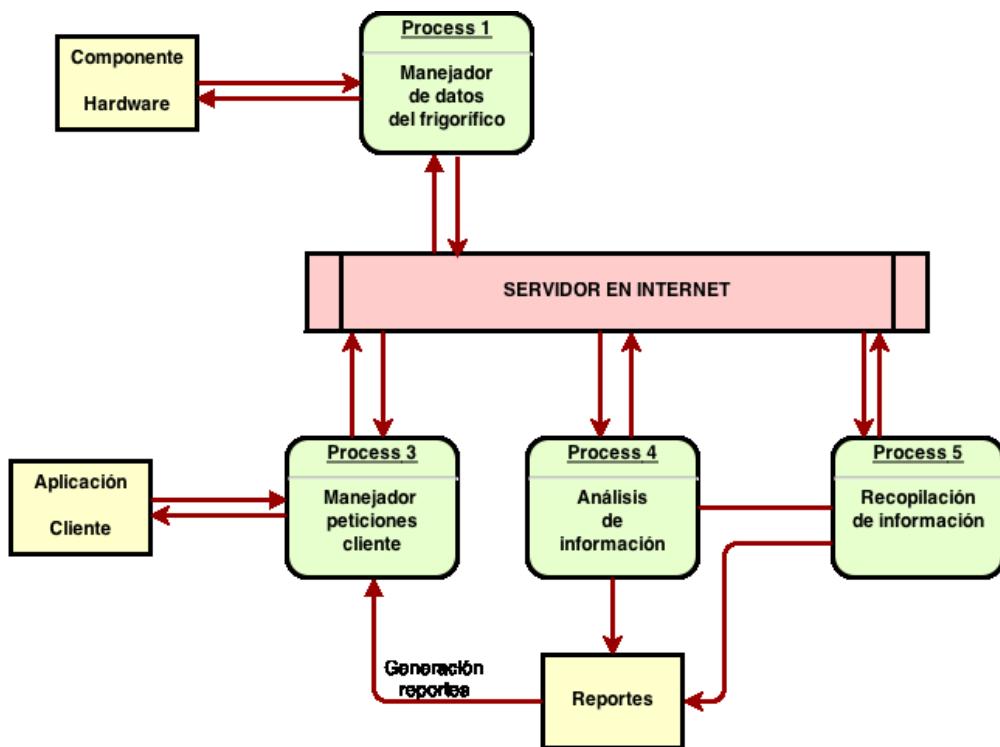
El servidor estará basado en un sistema Linux, con Apache y un entorno de funcionamiento PHP 5.

## 4. ARQUITECTURA DEL SISTEMA

### 4.1. Diagrama general

Siguiendo los requisitos anteriormente explicados, se desarrolla un sistema basado en tres partes: componente hardware, servidor y aplicación cliente.

El diagrama general del sistema es el que se muestra en la siguiente figura:



### 4.2. Componente hardware

Como se observa en el diagrama general del sistema el componente hardware se encarga de registrar los productos que se introducen o se sacan del frigorífico, comunicándose con el servidor para registrar la información.

Para esta tarea este componente se compone de las siguientes tres partes:

- **Lector de código de barras:** Permite identificar los productos que se quieren introducir o sacar del frigorífico a través del código de barras.
- **Lector de código RFIDs:** Permite identificar los productos que se quieren introducir o sacar del frigorífico a través de una etiqueta RFID.
- **Pantalla TFT:** Permite controlar las distintas opciones del componente hardware.
- **Comunicación WiFi:** Permite la comunicación con el servidor.

El hardware se ha desarrollado gracias a *Arduino Yún* y se ha programado en Python (*AR9331* bajo Linino) y Wiring (*ATmega 32U4*), realizando la comunicación entre los dos procesadores a través de la librería *Bridge*.

### 4.3. Componente servidor

El servidor es la pieza central de toda la arquitectura del proyecto, es el nexo común entre todas las interfaces del proyecto.

La comunicación entre las distintas interfaces se realiza a través de una API *RESTful* que se caracteriza por:

- Está basada en URIs, por ej: `http://example.com/resources`.
- Tipos de datos de Internet como *JSON*.
- Métodos estándar HTTP: *GET*, *POST*, *PUT* ó *DELETE*.
- Uso de hipermedios

El lenguaje utilizado para el desarrollo del servidor es *PHP* debido a que en prácticamente la totalidad de los servidores viene por defecto. Mientras que el empleo de otros lenguajes de servidor, obligaría a contratar un servidor dedicado para poder realizar la instalación del mismo, aumentando los costes y la complejidad del proyecto.

## 4.4. Componente cliente

Dentro de toda la arquitectura del proyecto esta es la parte que cobra más importancia de cara al usuario. Se encarga de procesar toda la información contenida por el servidor y de presentarla en un formato amigable.

La comunicación con el servidor se realizará a través de la *API RESTful* que se ha desarrollado. Gracias a esta API se pueden realizar aplicaciones cliente para distintas plataformas de manera sencilla.

Se ha desarrollado una aplicación cliente *web*, utilizando para ello la *API RESTful* proporcionada por el servidor, un sistema de plantillas basadas en *handlebars* para renderizar las diferentes pantallas de presentación al cliente y *JavaScript* para la programación desde el lado del cliente.

## 5. DISEÑO E IMPLEMENTACIÓN

---

Para la implementación del sistema se han utilizado diferentes lenguajes de programación y tecnologías que se irán comentando y describiendo a lo largo del capítulo.

### 5.1. COMPONENTE HARDWARE

El componente hardware se ha implementado utilizando dos lenguajes de programación; *Wiring* para el microprocesador ATmega32u4, Python para el procesador *Atheros AR9331*.

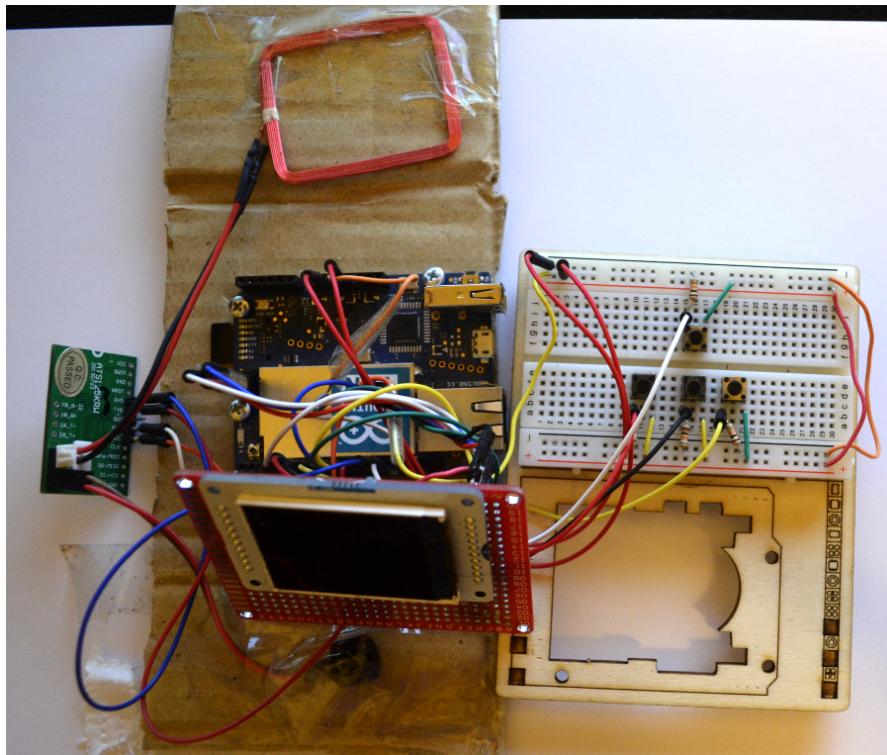


Figura 5.1: Lector Arduino

### 5.1.1. Configurando el sistema

El primer paso es configurar la placa de Arduino Yún para empezar a trabajar sobre ella. Para ello Yún corre una distribución de Linux llamada *OpenWrt-Yun*, basado en *OpenWrt*, la cual nos permite acceder vía *ssh* o vía web para poder configurarla.

La primera vez que se conecta Arduino Yún se crea una red sin encriptar para poder conectarnos a ella y poder configurarla, el nombre de la red tiene el formato *Arduino Yun-XXXXXXXXXXXXXX*. Una vez conectados se accede al panel web de configuración a través de la url *http://arduino.local* y se nos muestra la siguiente ventana:

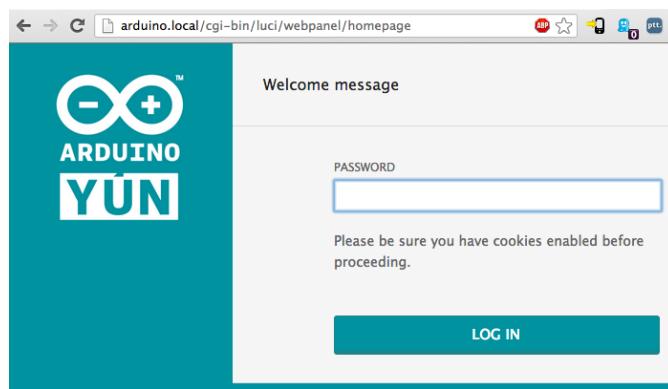


Figura 5.2: Arduino Yún Web Password

La contraseña por defecto es *arduino*; permite acceder al panel de diagnóstico donde se muestra información sobre la red WiFi y la conexión Ethernet. Además, permite acceder al panel de configuración:

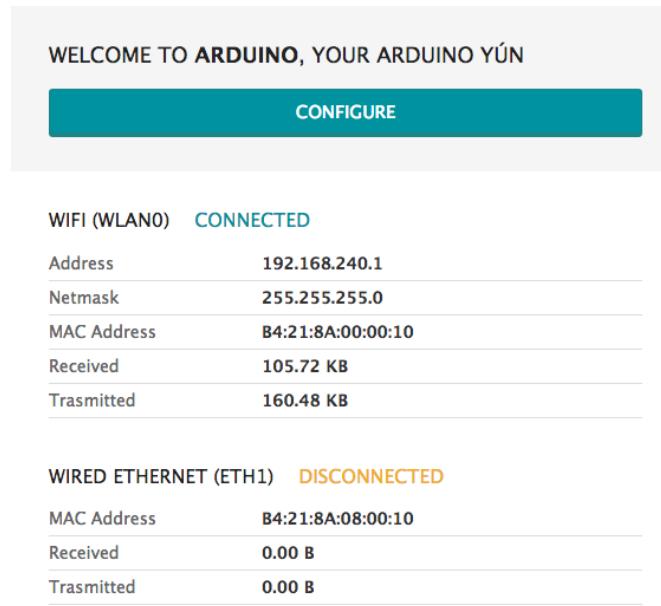


Figura 5.3: Arduino Yún Web Diagnostic

El panel de configuración permite nombrar nuestra placa Arduino, establecer una contraseña de acceso, configurar la zona horaria y por último, y lo más importante, decirle a que red WiFi debe conectarse.

YUN BOARD CONFIGURATION ⓘ

YUN NAME \*

PASSWORD

CONFIRM PASSWORD

TIMEZONE \*

## WIRELESS PARAMETERS ⓘ

CONFIGURE A WIRELESS NETWORK 

WIRELESS NAME \*

SECURITY 

PASSWORD \*

DISCARDCONFIGURE & RESTART

Figura 5.4: Arduino Yún Web Config

Una vez se guarda la nueva configuración la distribución Linux se reinicia y automáticamente se conectará a la red WiFi que hayamos establecido, indicando a través de un mensaje que se debe volver conectar a la WiFi del usuario:

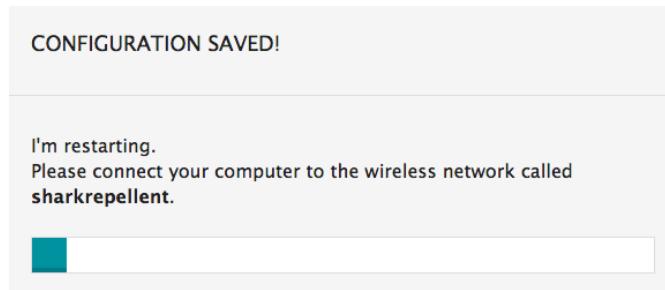


Figura 5.5: Arduino Yún Web Rebooting

Pasado varios segundos automáticamente se nos volverá a conectar a la página de configuración de Arduino donde se tendrá que introducir la nueva contraseña para acceder.

### 5.1.2. Problemas detectados

La versión de Linux que trae la placa Arduino no viene con las librerías para utilizar el puerto USB en modo HID (*Human Interface Device*) por lo que no reconoce el lector de códigos de barras USB. Para configurarlo se han tenido que bajar las librerías e instalarlas a través de SSH.

1. Descargar y guardar en la memoria micro-sd (*/mnt/sda1/*) las dos siguientes librerías:

- kmod-hid-generic\_3.8.3-1\_ar71xx.ipk
- kmod-hid\_3.8.3-1\_ar71xx.ipk

2. Actualizar el sistema de paquetes e instalar las dos siguientes librerías para permitir la instalación de las librerías anteriores:

```
1 opkg update
2 opkg install kmod-input-core
3 opkg install kmod-input-evdev
```

3. Instalar las librerías de la micro-sd:

```
1 rm /tmp/opkg-lists/*
2 opkg install /mnt/sda1/kmod-hid\_\_3.8.3-1\_\_ar71xx.ipk
3 opkg install /mnt/sda1/kmod-hid-generic\_\_3.8.3-1\_\_ar71xx.ipk
```

4. Instalar el driver HID:

```
1 opkg update
2 opkg install kmod-usb-hid
```

5. Cargar el driver HID en el Kernel:

```
1 insmod hid-generic
2 echo "hid-generic" >>/etc/modules.d/62-hid-generic
```

6. Reiniciar Linux:

```
1 reboot
```

7. Enchufar el lector de tarjetas USB y comprobar que funciona:

```
1 cat /dev/input/event1 hexdump
```

Con esta configuración Linux es capaz de cargar el dispositivo USB, pero, si se desenchufa y se vuelve a enchufar a veces no carga y da problemas. Por suerte, el *23 de Abril de 2014* han sacado una nueva versión de *OpenWrt-Yun* que corrige estos errores, e introduce las siguientes mejoras:

- OpenWrt
  - wget now supports ssl
  - Fixed fuses values in run-avrdude
  - nano is now built-in (no need to become "vi" experts any more)
  - Updated ruby to 1.9.3-p429

- Fixed USB port stability (when using it both with a pen drive or as a serial device)
  - Patched dhcp script to fix ethernet routing issue
  - Added lots of webcam modules
  - Fixed USB keyboard support
  - Heartbleed <http://heartbleed.com/>
  - Linux side ready visual notification: when linux boot completes, the usb led lights up (it's bright white)
  - Disk space expander tutorial: using an micro SD card to have gigabytes of disk space
  - Added "upgrade-all" script for easier massive upgrade of packages (available only after having followed the ExpandingYunDiskSpace tutorial)
  - Node.js is now available as an optional package: other native nodejs packages include bleno, noble, serialport, socket.io.
- Web panel
    - Previously, jsonp calls were triggered by the "jsonp" query string parameter only. Now, you can use "callback" as well.
    - Added Mailbox support
    - Various fixes
  - Bridge
    - File.size() now implemented
    - PHP bridge client (thx Luca Saltoggio)
    - Bridge is now run with "-u" python flag, preventing some random lockups in the Bridge
    - Resolved conflict with python "json" module

Una vez actualizado la versión de *OpenWrt-Yun* siguiendo las instrucciones de la página de Arduino (<http://arduino.cc/en/Tutorial/YunSysupgrade>) solamente se tiene que instalar la librería USB HID por SSH (ya que sigue sin venir por defecto):

```

1 opkg update
2 opkg install kmod-usb-hid

```

Y automáticamente detectará el lector de códigos de barras USB sin ningún problema, aunque se desenchufe y se vuelva a enchufar.

### 5.1.3. Decodificando el lector de código de barras

El siguiente problema a resolver es la creación de un script en *python* que permita la lectura de códigos de barras. Para ello se ha tenido que decodificar los caracteres que envía al sistema operativo, y una vez hecho, comprender como funciona.

El funcionamiento es muy sencillo. Cuando el lector de código de barras encuentra un código válido lo envía al sistema por letras y por último, y es lo que nos ayuda a comprender que ha terminado de enviarnos el código de barras, nos envía dos caracteres seguidos (*Control + j*) que simbolizan el salto de línea (*Line Feed*).

El siguiente script permite leer los caracteres escaneados e imprimirlos por consola o enviarlos al microcontrolador a través de la librería *Bridge*.

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import struct
5 import time
6 import sys
7 sys.path.insert(0, '/usr/lib/python2.7/bridge/')
8
9 from bridgeclient import BridgeClient as bridgeclient
10
11 # Variables
12 infile_path = "/dev/input/event1"
13
14 #long int, long int, unsigned short, unsigned short, unsigned int
15 FORMAT = 'llHHI'
16 EVENT_SIZE = struct.calcsize(FORMAT)

```

```
17
18     teclasPulsadas = dict()
19     escribiendoLetras = False
20     codigoBarras = ''
21     bridgeCliente = bridgeclient()
22     in_file = None
23
24     # Funciones
25
26     def esTeclaNumerica(value):
27         return value >= 458782 and value <= 458791
28
29     def getNumero(value):
30         return (value - 458781) % 10
31
32     def esTeclaLetra(value):
33         return (value >= 458756 and value <= 458781)
34
35     def getLetra(value):
36         posicionValue = value - 458756
37         asciiValue = posicionValue + 65
38         return str(unichr(asciiValue))
39
40     # 458792 -> Intro
41     # 458976 + 458765 (Control + j => Line Feed)
42     def esTeclaControl(value):
43         return value == 458792 or \
44             value == 458976 or \
45             value == 458765
46     def procesarEvento(event):
47         global teclasPulsadas, escribiendoLetras, codigoBarras,
48             bridgeCliente
49         (tv_sec, tv_usec, type, code, value) = struct.unpack(FORMAT,
50             event)
51
52         # Importa este orden de la condicion, si se altera puede
53             # imprimir
54         # caracteres escritos muy deprisa en otro orden
55         if value not in teclasPulsadas:
56             teclasPulsadas[value] = 'S'
57         else:
```

```
55     if esTeclaControl(value) and escribiendoLetras:
56         escribiendoLetras = False
57         print codigoBarras
58         #bridgeCliente.put('codebar', codigoBarras)
59
60         codigoBarras = ''
61     elif not esTeclaControl(value) and \
62         (esTeclaNumerica(value) or \
63          esTeclaLetra(value)):
64         escribiendoLetras = True
65         if esTeclaNumerica(value):
66             codigoBarras += str(getNumero(value))
67         elif esTeclaLetra(value):
68             codigoBarras += getTeclaLetra(value)
69     del teclasPulsadas[value]
70
71 def conectarUsb():
72     global in_file
73     in_file = None
74     conectado = False
75     while not conectado:
76         try:
77             # Intentamos abrir el fichero
78             in_file = open(infile_path, "rb")
79             conectado = True
80         except IOError:
81             time.sleep(1) # Pausamos 1seg
82
83 # Inicio programa
84 conectarUsb()
85 event = in_file.read(EVENT_SIZE)
86 while event:
87     try:
88         procesarEvento(event)
89         event = in_file.read(EVENT_SIZE)
90     except IOError:
91         conectarUsb()
92         event = in_file.read(EVENT_SIZE)
```

### 5.1.4. Vinculación del lector

Tras conectar el lector a nuestra red WiFi como se explica anteriormente, lo que se muestra en la pantalla de nuestro lector *Arduino* es un mensaje indicando que se necesita vincular el dispositivo.

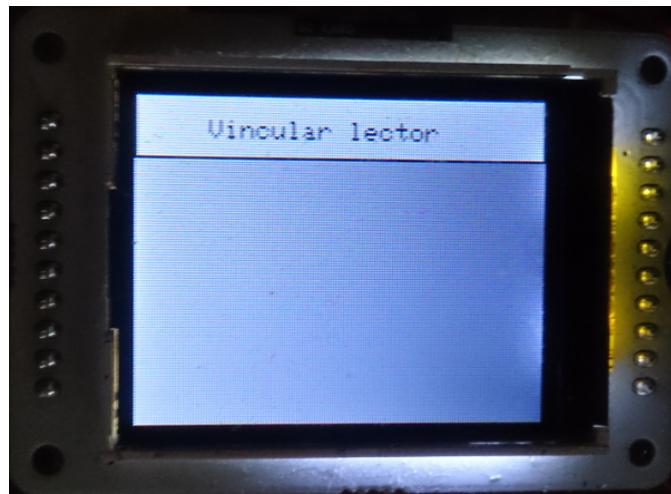


Figura 5.6: Vincular lector

A través de la aplicación web se debe añadir un lector y generar los dos códigos de barras, los cuales se imprimirán para que puedan ser leídos por el lector de código de barras.

Se escanearán con el lector en el orden que se indica en la aplicación web, una vez se han escaneado el lector lanzará una petición para vincularse con nuestra cuenta de usuario. Si el servidor responde correctamente, se guarda en la memoria *MicroSD* la clave de acceso pública y se muestra la Home del dispositivo con las opciones disponibles.

En caso de producirse un error se muestra un mensaje para que lo vuelva a intentar.

### 5.1.5. Opciones del lector

El lector cuenta con una pequeña *pantalla TFT* y unos pocos botones para su manejo. La primera opción que se da al lector es su vinculación con la cuenta del usuario y para

ello se muestra el siguiente aviso en la *pantalla TFT*:

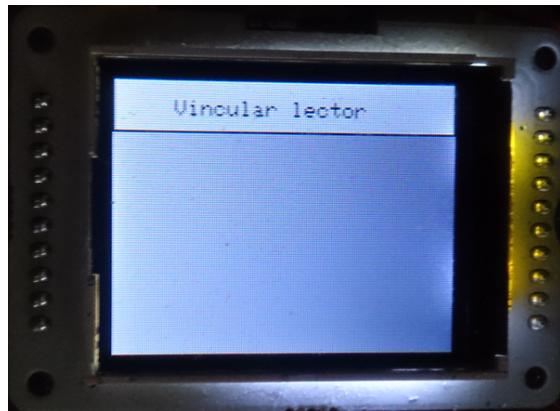


Figura 5.7: Vincular lector

Una vez se ha vinculado, escaneando para ello los dos códigos de barras generados por la aplicación web, se muestra la siguiente pantalla con las opciones disponibles:



Figura 5.8: Pantalla de inicio del lector

Para acceder a cada opción de menú hay que pulsar cada botón correspondiente; superior para listar los frigoríficos, inferior para desvincular el producto, y laterales para añadir o quitar productos del frigorífico.

- Selección del frigorífico activo

Esta opción de menú permite elegir el frigorífico sobre el que se realizarán las acciones (entrada / salida) de productos.

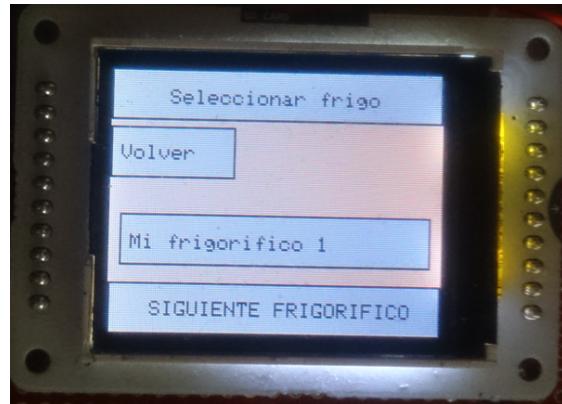


Figura 5.9: Selector de frigoríficos

Pulsando el botón inferior se puede uno desplazar entre los frigoríficos que los cuales se disponen, para ello se han traído a través de una petición *RESTful*. Una vez se está sobre el frigorífico que se quiere dejar configurado, basta con pulsar en la tecla *Volver* para que el lector almacene dicho frigorífico como activo.

En el caso de no contar con frigoríficos se muestra el siguiente mensaje en pantalla:

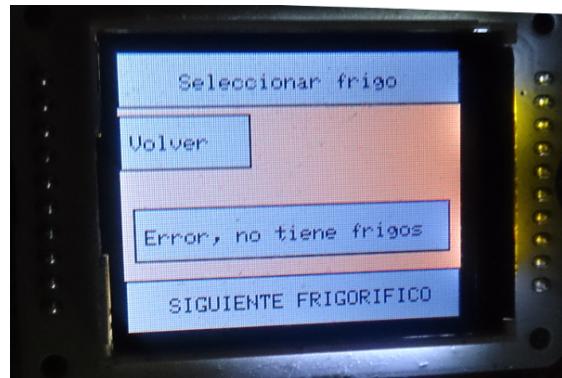


Figura 5.10: No dispone de frigoríficos

- Modo entrada de productos

Para introducir productos al frigorífico se selecciona la opción de menú *Añadir productos al carrito*, donde se activará la lectura de productos por *código de barras* o a través del *lector RFID*. En la pantalla se muestra la siguiente información:



Figura 5.11: Entrar productos en el frigorífico

Al escanear un producto se realiza la petición *RESTful* para añadirse al frigorífico activo.

Al pulsar el botón izquierdo se regresa a la pantalla anterior para acceder a las opciones de menú.

- Modo salida de productos

Para sacar productos del frigorífico hay que seleccionar la opción de menú *Sacar productos*, activándose la lectura de productos por *código de barras* o a través del *lector RFID*.



Figura 5.12: Sacar productos del frigorífico

Al pulsar el botón izquierdo se regresa a la pantalla anterior para acceder a las opciones de menú.

- Desvinculación del aparato

Esta opción de menú borrará de la memoria *MicroSD* la clave pública de acceso, con lo cual se quedará desvinculado el lector de la cuenta de usuario. Además se lanza una petición *RESTful* para indicar al servidor que se quiere desvincular el lector.

Antes de iniciar la desvinculación se muestra una pantalla de confirmación al usuario, debiendo pulsar el botón inferior para *Aceptar* la opción:

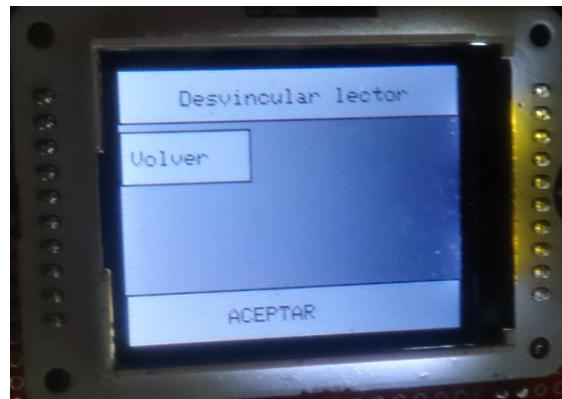


Figura 5.13: Confirmar desvincular lector

Una vez se confirma, se lanza la petición al servidor y se procede a eliminar la clave pública, una vez desvinculado se muestra la pantalla para volver a iniciar la vinculación del lector:

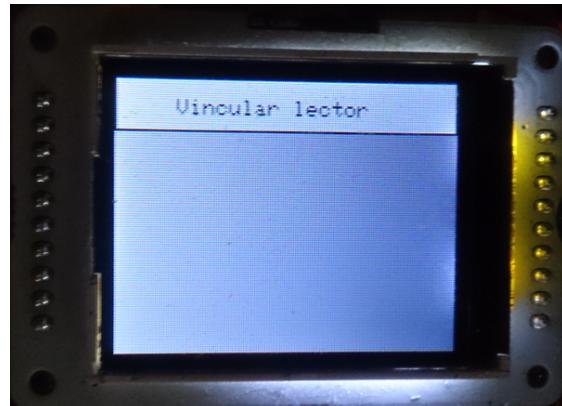


Figura 5.14: Vincular lector

## 5.2. COMPONENTE SERVIDOR

El componente servidor se ha implementado utilizando el lenguaje PHP (*Hypertext Preprocessor*). Se ha utilizado este lenguaje de programación debido a que es uno de los lenguajes de desarrollo web más utilizados, por lo que ha sido elegido ya que cuenta con las siguientes ventajas:

- Gran soporte por las empresas de servicios web
- Una comunidad de desarrolladores muy activa
- Amplia documentación y una comunidad activa y colaboradora
- Multitud de librerías y frameworks

Como base de desarrollo se ha utilizado el framework *CodeIgniter*, una potente herramienta de desarrollo que permite realizar aplicaciones de manera sencilla y rápidamente. Algunas de las características que posee son:

- Posee un gran rendimiento
- Tiene una alta compatibilidad entre distintas versiones y configuraciones de servidores PHP
- No necesita una gran configuración para empezar a funcionar
- Evita introducirse en una gran complejidad de desarrollo
- Tiene una gran documentación

### 5.2.1. Configurando el sistema

El programa necesita funcionar bajo un servidor LAMP (*Linux, Apache, MySQL, PHP*):

- Linux: Es el término cotidiano de nombrar al Sistema Operativo *GNU/Linux*, es muy ligero y es capaz de ejecutarse sobre multitud de dispositivos; ordenadores; móviles; teléfonos; routers, etc. Una de sus aplicaciones es la de ejercer de Servidor Web.
- Apache: Es un servidor web HTTP de código abierto. Su objetivo es proveer seguridad, eficiencia y un servidor extensible que provea servicios HTTP cumpliendo con los estándares HTTP.
- MySQL: Es la base de datos más popular de código abierto
- PHP: Es un lenguaje de scripting que se ejecuta desde el lado del servidor, diseñado para desarrollo web pero también usado como lenguaje de propósito general.

Opcionalmente se puede configurar un servidor Windows para ejecutar la aplicación, aunque no es lo usual, ya que un servidor Windows se caracteriza por servir aplicaciones web desarrolladas bajo *.NET*.

Con la disposición de un servidor LAMP en funcionamiento se tienen que realizar las siguientes configuraciones para que la aplicación web sea capaz de servir páginas a los clientes:

- Apache:

Tendremos que configurar Apache para que sea capaz de servir nuestro sitio indicándole en qué directorio se encuentra instalada, el nombre del servidor bajo el que se va a servir y algunas otras opciones opcionales. A continuación se pone el ejemplo de un fichero de configuración:

```
1 <VirtualHost *:80>
2   <Directory />
3     Options Indexes FollowSymLinks MultiViews
4     AllowOverride All
5     Order allow,deny
6     allow from all
7   </Directory>
8
9   ServerName fribone.es
10
```

```

11     ServerAdmin correo_administrador@gmail.com
12     DocumentRoot /var/www/fribone-php-server/application
13     LogLevel info
14
15     ErrorLog ${APACHE_LOG_DIR}/error.local.log
16     CustomLog ${APACHE_LOG_DIR}/access.local.log combined
17 </VirtualHost>

```

- MySQL

Una vez instalado es necesario crear una base de datos con codificación *utf8\_general\_ci*, y para dar mayor seguridad, crear un usuario específico con permisos para acceder a esa base de datos.

Una vez creada la base de datos es necesario ejecutar el script sql de instalación que se encargará de crear las tablas necesarias, así como inserción de datos, para que la aplicación sea capaz de funcionar.

- PHP

Hay que encargarse de comprobar que se tiene habilitado el módulo *mod\_rewrite*, es un modulo que a través del fichero *.htaccess* permite a base de reglas reescribir la URL de una petición al vuelo. Es ampliamente utilizado para permitir poner urls amigables al cliente, siendo el módulo quien lo convierte a una url parametrizada que entiende el lenguaje.

- Aplicación PHP

Hay que editar dos ficheros contenidos en la carpeta *conf*.

- config.php

El parámetro más destacado que es de obligación cambiar es la *encryption\_key*, clave que se utiliza para la encriptación de la sesión en una cookie que se envía al cliente (se utiliza para controlar que un usuario está logueado).

- database.php

En este fichero de configuración hay que establecer los distintos parámetros de la configuración de nuestra base de datos, siendo los más importantes el nombre de usuario, la contraseña y el nombre de la base de datos.

### 5.2.2. Control de Versiones, tests unitarios e integración continua

Desde el inicio del proyecto se ha desarrollado utilizando *GIT* como control de versiones, utilizando la plataforma web *GitHub*, una forja que sirve para alojar proyectos de forma pública y de manera gratuita, aunque se puede crearse una cuenta de pago para tener repositorios privados.

Además, se han realizado tests unitarios utilizando para ello la herramienta *PHPUnit*, lo cual nos permite tener una aplicación sólida y poco propensa a errores, ya que con cada modificación que se realiza se vuelven a pasar los tests para comprobar la estabilidad del código. Otra de las ventajas de realizar tests es que si se encuentra un error, solamente hay que escribir un test para ello y corregirlo, por lo que será imposible que el mismo error se vuelva a repetir en un futuro ya que estará cubierto bajo ese test.

Y por último, por encima de estos dos pilares, se ha realizado integración continua con *Travis CI*, una aplicación web que se sincroniza con la cuenta de *GitHub*, construyendo la aplicación y realizando los tests unitarios cada vez que se realiza un *push* al repositorio.

```

1 Using worker: worker-linux-3-2.bb.travis-ci.org:travis-linux-15
2
3 $ git clone -depth=50 --branch=master git://github.com/MiguelGonzalez/fribone-php-server.git MiguelGonzalez/fribone-php-server
4 $ cd MiguelGonzalez/fribone-php-server
5 $ git checkout -qf 271adca174db03548d0e20b726fcf27ffaf24722
6 $ phpenv global 5.5
7 $ php --version
8 PHP 5.5.13 (cli) (built: Jun 12 2014 13:54:54)
9 Copyright (c) 1997-2014 The PHP Group
10 Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
11     with Zend OPcache v7.0.4-dev, Copyright (c) 1999-2014, by Zend Technologies
12     with Xdebug v2.2.5, Copyright (c) 2002-2014, by Derick Rethans
13 $ composer --version
14 Composer version ac497feabaa0d247c441178b7b4aaa4c61b07399 2014-06-10 14:13:12
15 $ mysql -e 'create database fribone_test;' -uroot
16 $ phpunit .
17 PHPUnit 4.1.3 by Sebastian Bergmann.
18
19 Configuration read from /home/travis/build/MiguelGonzalez/fribone-php-server/phpunit.xml
20
21 .....
22
23 Time: 3.02 seconds, Memory: 7.25Mb
24
25 [x] Tests: 07 Exceptions
26
27 The command "phpunit ." exited with 0.
28
29 Done. Your build exited with 0.
30
31
32
33
34
35
36
37
```

Figura 5.15: Tets - Travis CI

### 5.2.3. Problemáticas para la creación de tests unitarios

CodeIgniter viene con su propia herramienta para generar tests unitarios, la cual no nos permite su integración con *Travis CI*, por ello se han realizado una serie de modificaciones para permitir el uso de PHPUnit.

Se han tenido que tocar los siguientes módulos y realizar los siguientes cambios:

- Modificación del fichero *DB\_driver.php*

Si se produce un error se llama al método `display_error`, el cual lo imprime por pantalla. Para correr tests se ha modificado para lanzar una excepción en caso de que se esté en el entorno de testing.

```

1 if (defined('ENVIRONMENT') && ENVIRONMENT == 'testing') {
2     $message = $error . ' ' . $swap;
3     throw new Exception($message);
4 } else {
5     ...
6     ...
7 }
```

- Modificación del fichero *Utf8.php*

Se ha modificado la línea donde se declara la variable global `$CFG` por el siguiente código para evitar un error al lanzarse los tests:

```

1 //global $CFG; // Fix PHPUnit Error
2 $CFG =& load_class('Config', 'core');
```

- Modificado fichero *mysql\_driver.php* para evitar un error al utilizar la función *mysql\_escape\_string*:

```

1 function escape_str($str, $like = FALSE) {
2     if (is_array($str)) {
3         foreach ($str as $key => $val) {
4             $str[$key] = $this->escape_str($val, $like); }
5
6     return $str;
```

```

7      }
8
9      $str = is_resource($this->conn_id) ?
10     mysql_real_escape_string($str, $this->conn_id) :
11     addslashes($str);
12
13     // escape LIKE condition wildcards
14     if ($like === TRUE) {
15         return str_replace(
16             array($this->_like_escape_chr, '%', '_'),
17             array($this->_like_escape_chr.$this->
18                 _like_escape_chr,
19                 $this->_like_escape_chr.'%', $this->
20                 _like_escape_chr.'_'),
21             $str);
22     }
23
24     return $str;
25 }
```

Por último, para ser capaces de lanzar los tests se han tenido que añadir las siguientes configuraciones a la aplicación:

- Creación del fichero de configuración *phpunit.xml*

Este fichero XML permite indicarle a PHPUnit los parámetros para ejecutar los tests. Consta de la siguiente configuración:

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <phpunit bootstrap="application/application/tests/bootstrap.
3   php"
4     colors="true"
5     convertErrorsToExceptions="true"
6     convertNoticesToExceptions="true"
7     convertWarningsToExceptions="true"
8     processIsolation="false"
9     stopOnFailure="false"
10    syntaxCheck="false"
11    verbose="true">
```

```
11 <testsuites>
12   <testsuite name="TestUserModel">
13     <file>application/application/tests/models/
14       UserTest.php</file>
15   </testsuite>
16   <testsuite name="TestLectorModel">
17     <file>application/application/tests/models/
18       LectorTest.php</file>
19   </testsuite>
20   <testsuite name="TestFridgeModel">
21     <file>application/application/tests/models/
22       FridgeTest.php</file>
23   </testsuite>
24   <testsuite name="TestSupermercadoModel">
25     <file>application/application/tests/models/
26       SupermercadoTest.php</file>
27   </testsuite>
28   <testsuite name="TestCompraModel">
29     <file>application/application/tests/models/
30       CompraTest.php</file>
31   </testsuite>
32   <testsuite name="TestLogin_auth">
33     <file>application/application/tests/libraries/
34       Login_authTest.php</file>
35   </testsuite>
36   <testsuite name="TestMy_PHPMailer">
37     <file>application/application/tests/libraries/
38       My_PHPMailerTest.php</file>
39   </testsuite>
40   <testsuite name="TestLector_library">
41     <file>application/application/tests/libraries/
       Lector_libraryTest.php</file>
```

```

42     <testsuite name="TestCompra_library">
43         <file>application/application/tests/libraries/
44             Compra_libraryTest.php</file>
45     </testsuite>
46 </testsuites>
47     <filter>
48         <blacklist>
49             <directory suffix=".php">application/application/
50                 config</directory>
51             <directory suffix=".php">application/application/
52                 controllers</directory>
53             <directory suffix=".php">application/application/
54                 core</directory>
55             <directory suffix=".php">application/application/
56                 libraries/PHPMailer</directory>
57             <directory suffix=".php">application/application/
58                 tests/mockups</directory>
59             <directory suffix=".php">application/system</
60                 directory>
61             <file>application/application/tests/bootstrap.php
62                 </file>
63             <file>application/application/tests/
64                 database_inflater.php</file>
65             <file>application/application/tests/PHPTest_Unit.
66                 php</file>
67         </blacklist>
68     </filter>
69     <php>
70         <const name="PHPUnit_TEST" value="1" />
71         <const name="PHPUnit_CHARSET" value="UTF-8" />
72         <const name="REMOTE_ADDR" value="217.0.0.1" />
73     </php>
74     <logging>
75         <log type="coverage-text" target="php://stdout"
76             showUncoveredFiles="true"/>
77         <log type="coverage-clover" target="coverage/clover.
78             xml"/>
79     </logging>
80 </phpunit>

```

Ejecutará la aplicación a partir del fichero *bootstrap.php* y lanzará los tests defini-

dos.

- Creación del fichero *bootstrap.php*

Este fichero es una copia del *index.php*, se ha modificado el entorno a testing, la dirección remota del servidor, el reporte de errores del sistema y el path de las carpetas:

```

1 define('ENVIRONMENT', 'testing');

2
3 $_SERVER["REMOTE_ADDR"]      = array_key_exists( 'REMOTE_ADDR',
4                                                 $_SERVER) ? $_SERVER['REMOTE_ADDR'] : '127.0.0.1';

5 error_reporting(E_ALL ^ E_NOTICE);
6 $system_path = '../..//system';
7 $application_folder = '...//..//application';

```

- En la definición de controlador global se ha añadido una condición para comprobar si se está en modo testing. En modo testing no se carga ninguna librería y se añade en su lugar la carpeta de *tests/mockups* como paquete, de esta manera se cargarán automáticamente primero las librerías que se encuentren ahí por defecto, facilitando la creación de *mockups* en el entorno de test.

```

1 if (defined('ENVIRONMENT') && ENVIRONMENT == 'testing') {
2     $this->load->add_package_path(APPPATH.'tests/mockups');
3 } else {
4     $this->load->library('login_auth');
5 }

```

#### 5.2.4. Esquema de la base de datos

A continuación se muestra el esquema de la base de datos donde se observan las relaciones que hay entre las distintas entidades. En su diseño se ha tenido en cuenta que hay dos partes diferenciadas que necesitan un punto de unión:

- Zona del usuario

Un usuario tendrá sus propios frigoríficos y productos asociados a él

- Zona comunitaria

En la zona comunitaria se crearán los supermercados y productos asociados a cada uno. Así, un usuario agregará a su frigorífico los productos que de forma colaborativa se han guardado en la plataforma.

- Punto de unión

La tabla *my\_compra\_producto* enlaza entre un producto agregado por un usuario a una compra, al supermercado al que pertenece ese producto y que producto del supermercado es.

Además, en esta tabla, se duplican todos los datos del producto del supermercado. De esta manera, se puede editar el precio del producto del supermercado y para un usuario; para tal compra; ese precio se mantiene.

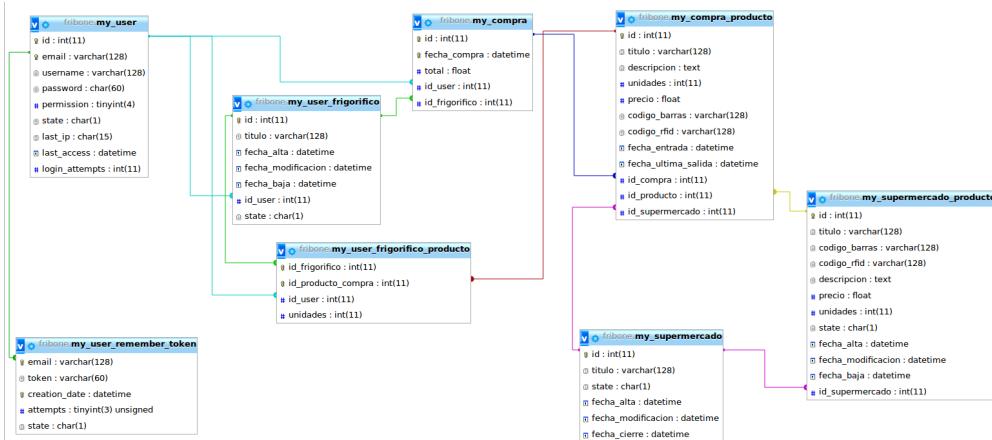


Figura 5.16: Esquema de la base de datos

### 5.2.5. Esquema del servidor

El esquema de servidor se basa en el patrón *MVC* (Modelo, Vista, Controlador) que provee *CodeIgniter* con su framework.

A continuación se muestra un ejemplo del funcionamiento del patrón *MVC* que provee *CodeIgniter*:

- Un usuario lanza una petición contra la siguiente url

[http://www.example.com/clientes/get\\_cliente/1](http://www.example.com/clientes/get_cliente/1)

- A través del modulo mod\_rewrite se redirige la petición al fichero *index.php* de CodeIgniter, el cual se encarga de analizar la petición siguiendo el siguiente flujo de trabajo:

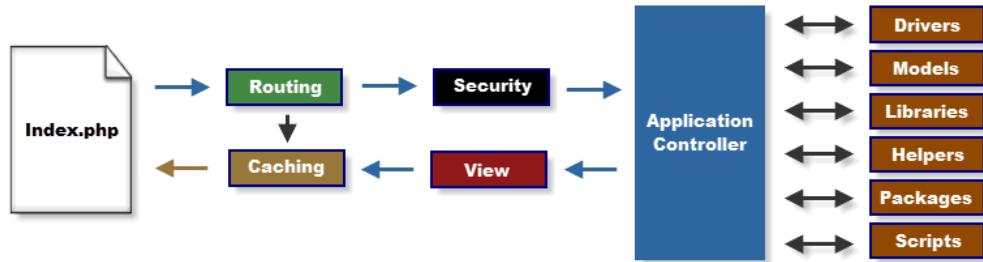


Figura 5.17: Flujo de trabajo de CodeIgniter

- CodeIgniter descompone la URL en la siguiente estructura para conocer a qué controlador tiene que pasar la petición, a qué función llamar y qué parámetros debe pasar.

[http://example.com/\[controller-class\]/\[controller-method\]/\[arguments\]](http://example.com/[controller-class]/[controller-method]/[arguments])

Siendo para nuestro ejemplo los siguientes valores:

- Controller class: clientes
- Controller method: get\_cliente
- Arguments: 1

Los controladores son los encargados de procesar la petición, cargar tantas librerías, modelos... como necesiten para finalmente devolver un resultado.

### Extendiendo el core de CodeIgniter

CodeIgniter al cargarse intenta buscar los siguientes ficheros de sistema en la carpeta *application/core*, esta funcionalidad permite sobreescibir algunas de las funcionalidades para ampliarlas.

En el proyecto se han creado dos nuevos controladores para ampliar la funcionalidad del controlador por defecto de CodeIgniter:

- MY\_Controller

Extiende de *CI\_Controller* y se encarga de:

- Cargar la librería *login\_auth* si no se están ejecutando los tests unitarios
- Ofrecer los siguientes métodos para renderizar la respuesta (*\_renderJson*, *\_renderXml*, *\_renderToVar*, *\_render*)

- MY\_Controller\_User

Extiende de *MY\_Controller* y únicamente se encarga de validar en el constructor que el usuario está logueado en el sistema, en caso de no estarlo, redirige a la Home pública.

### Esquema de controladores

El esquema de controladores se pueden dividir en dos partes; aquellos que permiten el acceso a la web (la aplicación web los utiliza para renderizar la página) y controlan la sesión del usuario; y aquellos que funcionan como una API *RESTful*, solamente devuelven objetos *JSON*.

#### Controladores de acceso web

- home.php

Se encarga de renderizar la parte pública de la web y controlar las acciones de registro de usuario, recordatorio de contraseña y acceso a la zona privada de usuario.

- tablon.php

Se encarga de renderizar el esqueleto de la parte privada de usuario, de controlar que está logueado en el sistema y de permitir salir de su sesión.

#### Controladores de API *RESTful*

- fridge.php

Se encarga de procesar las solicitudes para manejar los frigoríficos del usuario y los productos que contienen, así como sus compras.

- **lector.php**

Se encarga de procesar las solicitudes para manejar los lectores del usuario, así como la generación de códigos de barras para vincularlos a su cuenta.

- **supermercado.php**

Se encarga de procesar las solicitudes para manejar los supermercados y los productos que contienen.

- **codigobarras.php**

Permite la generación de una imagen de un código de barras.

- **rest.php**

Proporciona una serie de funciones accesibles desde el exterior para permitir al lector comunicarse con el servidor. Por ejemplo, añadir un producto al frigorífico del usuario.

### Esquema de librerías

La única restricción que se ha encontrado en CodeIgniter para hacer tests unitarios es que los controladores no se pueden cargar para hacerles pruebas unitarias. Por ello, toda la funcionalidad compleja se traslada a una librería específica la cual tiene sus tests unitarios.

De esta manera el controlador, no necesita de tests unitarios ya que actúa como mediador entre el cliente y las librerías.

Se han implementado las siguientes librerías en el sistema:

- **Login\_auth\_library.php**

Esta librería es la más crucial a la hora de aportar seguridad ya que se encarga de validar los datos de un usuario cuando intenta acceder al sistema; además de controlar el registro de los usuarios, la recuperación de contraseñas y la opción para salir de una cuenta; destruyendo la sesión del usuario.

Se ha utilizado la función nativa *password\_hash* que ofrece *PHP* para crear el Hash de la contraseña que guardaremos en base de datos. Esta función solo funciona en versiones de *PHP 5.5.0* o superiores, si es inferior se utiliza la función *crypt*.

El algoritmo empleado para hashear con la función *password\_hash* es *BCrypt*. Este algoritmo se considera seguro y perdurable en el tiempo ya que puede establecer el coste de computación para obtener el hash, por lo que aunque la potencia de cálculo de las máquinas aumente se puede aumentar el coste de computación, esto hace que en un futuro siga siendo un algoritmo fuerte contra ataques de fuerza bruta.

Por el contrario, el algoritmo *DES* que utiliza la función *crypt* se considera insegura, ya que el tamaño de clave que emplea solo es de 56 bits,

La función para obtener el Hash se encarga de comprobar la versión de *PHP*, si es inferior se utiliza la función *crypt* con un *salt*; aún así a día de hoy ya se empieza a considerar inseguro porque utiliza un algoritmo DES *Data Encryption Standard* con un tamaño de clave de 56 bits que permite realizar ataques de fuerza bruta para obtener la contraseña en un tiempo relativamente corto.

A continuación se expone la función para obtener el hash de una contraseña, donde se compara la versión de *PHP* para utilizar uno u otro:

```
1 private function get_password_hash($password) {
2     $password_hashed = NULL;
3
4     if (strnatcmp(phpversion(), '5.5.0') >= 0) {
5         $password_hashed = password_hash($password,
6             PASSWORD_BCRYPT);
7     } else {
8         // Original PHP code by Chirp Internet: www.chirp.com.
9         // Please acknowledge use of this code by including
10        // this header.
11        $salt = "";
12        $salt_chars = array_merge(range('A','Z'), range('a','z'),
13            range(0,9));
14        for($i=0; $i < 22; $i++) {
15            $salt .= $salt_chars[array_rand($salt_chars)];
16        }
17        $password_hashed = crypt($password, sprintf('$2a$%02d$',
18            7) . $salt);
19    }
20
21    return $password_hashed;
```

18 | }

Además, para evitar ataques de fuerza bruta, se ha implementado un sistema que registra los intentos para acceder a una cuenta. Incrementando hasta un máximo de 45 segundo el tiempo entre intentos, y además, al llegar a este máximo a partir del cual se considera que es un ataque de fuerza bruta, en el formulario se incluye un campo de seguridad para que el usuario resuelva (una operación matemática).

- My\_PHPMailer.php

Esta librería permite el envío de correos electrónicos a través de SMTP ó, si el servidor no lo soporta, utilizando el método *send\_mail*. Para funcionar se carga la librería *PHPMailer*, esta librería implementa funcionalidades para el envío de email como el soporte para el envío de adjuntos o el disponer de una implementación de un servidor *SMTP* para entornos donde no se dispone de un servidor de correos local.

- Lector\_library.php

Esta librería nos permite crear y asociar los lectores físicos que se dispongan.

Para asociar un lector se generan dos códigos de barras que deben ser escaneados en el tiempo de una hora; tiempo en el que están activos. El formato de los códigos de barras es el siguiente:

- Primer código de barras

Está formado por la suma del identificador del usuario más la fecha unix actual. Se rellena con ceros a la izquierda hasta completar doce dígitos.

Nota\*: Si el código de barras empieza en cero el lector automáticamente desecha ese dígito por lo que se establece siempre el primer dígito a 1.

- Segundo código de barras

Está formado por doce caracteres numéricos generados de forma aleatoria.

Cuando un lector lanza una petición se realiza la comprobación de los dos códigos de barra, y si son válidos, se le devuelve un *token* al lector, el cual será utilizado a modo de identificador para lanzar todas las peticiones.

- Fridge\_library.php

Esta librería, a través del modelo *fridge*, permite la creación de frigoríficos a un usuario, así como añadir productos a la compra utilizando para ello la librería *Compra\_library.php* y el modelo *fridge\_model*.

- Compra\_library.php

Esta librería se encarga de añadir productos al frigorífico, gestionando las compras, para ello utiliza las librerías *supermercado\_model* y *compra\_model*.

Su funcionamiento al añadir un producto es buscar una compra creada en la última media hora, si no se encuentra ninguna se crea una compra, y si ya existe una, el producto se añade a dicha compra. De esta manera los productos que se añaden se agrupan en compras.

- Supermercado\_library.php

Esta librería permite gestionar los supermercados y sus productos utilizando para ello el modelo *supermercado\_model*.

## Esquema de modelos

Antes de explicar el esquema de modelos es interesante comentar como funciona el acceso a base de datos.

CodeIgniter utiliza una versión modificada del patrón *Active Record Database*, lo que nos permite obtener, insertar y actualizar la base de datos con una mínima cantidad de código.

Además, este patrón nos permite abstraernos del motor de base de datos, para cada motor de base de datos hay un adaptador que se encarga de generar las consultas SQL para funcionar contra él.

A continuación se muestra una consulta generada con la *Active Record class* de CodeIgniter:

```

1 private function get_total_compra($id_compra) {
2     $this->db->select_sum('compra_producto.precio', 'total_compra
3         ');
4     $this->db->from('compra_producto');
5     $this->db->where('compra_producto.id_compra', $id_compra);
6
7     $query = $this->db->get();
8
9     if ($query->num_rows() === 1) {

```

```
9         return $query->row()->total_compra;
10    }
11    return 0;
12 }
```

Los siguientes modelos permiten manejar la información en base de datos a través del *Active Record Database* explicado anteriormente. Se han separado lógicamente, cada modelo engloba unas pocas tablas para especializarse al acceso de esos datos.

- user\_model.php
- fridge\_model.php
- compra\_model.php
- supermercado\_model.php
- lector\_model.php

Así, permiten recuperar una o varios registros en base de datos, modificar información o incluso realizar un borrado lógico.

Un borrado lógico es actualizar un registro para marcarlo como borrado, sin llegar a borrarlo físicamente de la base de datos, esto es útil para poder recuperar información (análisis de datos) en un futuro, o incluso, poder recuperar registros en un futuro.

#### 5.2.6. Vinculación del lector

La vinculación del lector se realiza a través de la generación de dos códigos de barras que funcionan a modo de tokens. Al generarse los dos códigos; estos quedan asociados al identificador del usuario.

El flujo de la vinculación de un lector con la cuenta de un usuario es el siguiente:

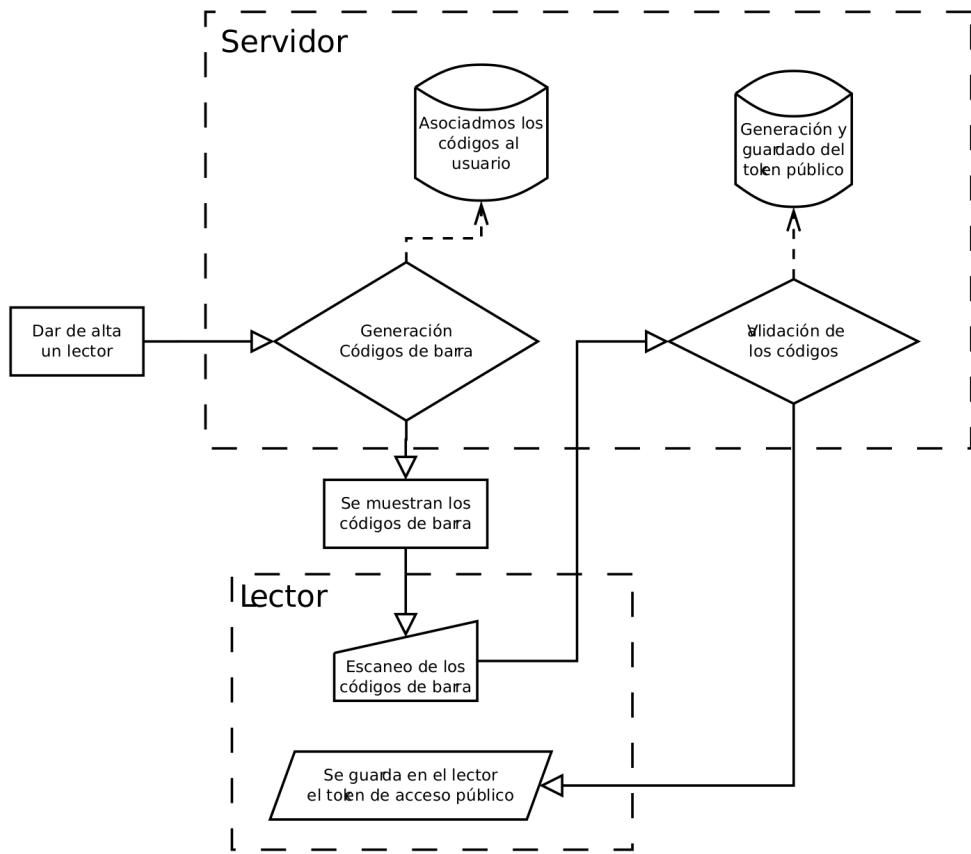


Figura 5.18: Flujo de la vinculación de un lector

El servidor se encarga de generar y verificar los códigos de barra, asociando si la vinculación es correcta una clave de acceso pública al lector. Así, en las siguientes peticiones (añadir productos, sacar...) el lector deberá enviar junto con la información del producto, la clave pública.

### 5.2.7. Generación de código de barras

Para enlazar el lector con la aplicación se generan dos códigos de barras que deben ser escaneados por el lector. Se ha ideado esta forma debido a que es la forma más sencilla que se tiene para introducir información en el aparato.

Para generar los códigos de barra se ha buscado una librería que cumpla con la codificación EAN-13 (*European Article Number* ó *International Article Number*). Es un sistema de codificación adoptado por más de 100 países compuesto por 13 dígitos que siguen el siguiente formato:

- Código del país (3 dígitos)
- Código de empresa (4 o 5 dígitos)
- Código del producto (completa los 12 primeros dígitos)
- Dígito de control

Tras realizar una búsqueda de las librerías existentes en *PHP* que cumplan estos requisitos, finalmente, se ha encontrado un proyecto en *GitHub* (<https://github.com/r1t3/php-barcode>) que encaja con lo que se busca.

En la integración con el proyecto, como es de esperar en cualquier integración, se ha tenido un problema con el renderizado de los caracteres ya que utiliza una fuente que no estaba instalada en el sistema, como solución se ha añadido la fuente *FreeSansBold.ttf* a la librería y se carga indicando la ruta a la fuente, así no se depende de la configuración del servidor. Además, se ha realizado un refactor de algunas de las funciones de la librería.

Una vez resueltos los problemas se ha realizado un *Fork* de la librería, dejando los cambios refactorizados accesibles para futuros usos de la librería ó para cualquier otro usuario que realice una búsqueda de la misma.

A continuación se muestra el código para imprimir por pantalla un código de barras:

```
1 public function generar_barcode($number) {  
2     $barCode = new Barcode($number);  
3     $barCode->display();  
4 }
```

### 5.3. COMPONENTE CLIENTE

El componente cliente se ha implementado utilizando varias tecnologías web:

- PHP

El cliente se encuentra desarrollado en la misma aplicación servidora. De esta manera, las sesiones de usuario son compartidas por lo que hay una mayor facilidad de integración, y por otro lado se facilita la instalación en un único servidor.

La única funcionalidad que se utiliza es la comprobación de la sesión de un usuario, pintando en un caso la página pública o en otro la zona privada. Estas son las dos únicas páginas que renderiza el servidor, a partir de ahí el cliente es quien pide la información vía peticiones *RESTful* en formato *JSON* y renderiza la web a través de las plantillas con *handlebars*.

- HTML

Es un lenguaje de marcas de hipertexto (*HyperText Markup Language*) que permite definir la estructura y el contenido de una página web (texto, imágenes, hipervínculos...). Es un estándar que regula la W3C (*World Wide Web Consortium*) que todos los navegadores intentan cumplir, de manera que el contenido HTML de la página se visualice de la manera más parecida en los distintos navegadores.

- CSS

Es un lenguaje de hojas de estilo (*Cascading Style Sheets*) que permite definir el aspecto y el formato de un documento escrito en un lenguaje de marcas, como HTML, y al igual que este las especificaciones están reguladas por el consorcio de la *W3C*.

- Bootstrap

Es un framework CSS que define unas reglas de estilos para estructurar el diseño de la página web. Está desarrollado por *Twitter* el cual ha publicado el código bajo la licencia *MIT*.

- JavaScript

Es un lenguaje de programación interpretado, dinámico y normalmente usado en páginas web que corren bajo un navegador web. Aunque, también se puede utilizar desde el lado de servidor a través de tecnologías como *Node.js*.

Desde el lado del cliente a través del navegador permite:

- Interactuar con el usuario.
- Controlar el navegador.
- Realizar comunicaciones asíncronas con el servidor (*AJAX*).
- Alterar el contenido del documento que se está visualizando.

■ Handlebars

Es una librería JavaScript para la creación de plantillas, las cuales son compiladas y reciben un objeto JSON con los datos, siendo procesado por *Handlebars* para obtener el documento HTML.

■ JQuery

Es una librería JavaScript que simplifica el manejo del documento HTML (manipulación, eventos, animaciones, Ajax...). Además, ofrece compatibilidad con la mayoría de los navegadores del mercado.

■ Director

Es una librería JavaScript que simplifica la creación de rutas enfocada a webs de una única página, donde toda la información se carga vía Ajax y plantillas. Utiliza la HTML5 History API para manipular la URL del navegador sin recargar la página entera, dando la sensación de navegación entre distintas páginas web.

Se han creado dos zonas web diferenciadas; una parte pública donde el usuario puede loguearse o registrarse en el sistema; y una parte privada donde el usuario puede acceder a su información.

La zona privada de la plataforma web se ha creado utilizando un diseño conocido como Single Page App, es decir, se carga la página una sola vez y toda la navegación posterior se realiza mediante *Ajax* y un sistema de plantillas.

Las ventajas de este diseño es que no es necesario recargar la página entera cada vez que se quiere acceder a una nueva sección, solo se necesitan pintar pequeñas zonas de la página web. Una de las desventajas es que las páginas que se crean con este diseño no son fácilmente indexadas por los buscadores, por ello, es un diseño que se suele utilizar en zonas privadas donde no tienen acceso a los buscadores y no importa que el contenido no sea indexado.

### 5.3.1. Flujo de la aplicación

A continuación se muestra un diagrama con el flujo por las distintas páginas de la aplicación web y las acciones que se pueden realizar en cada una de ellas.

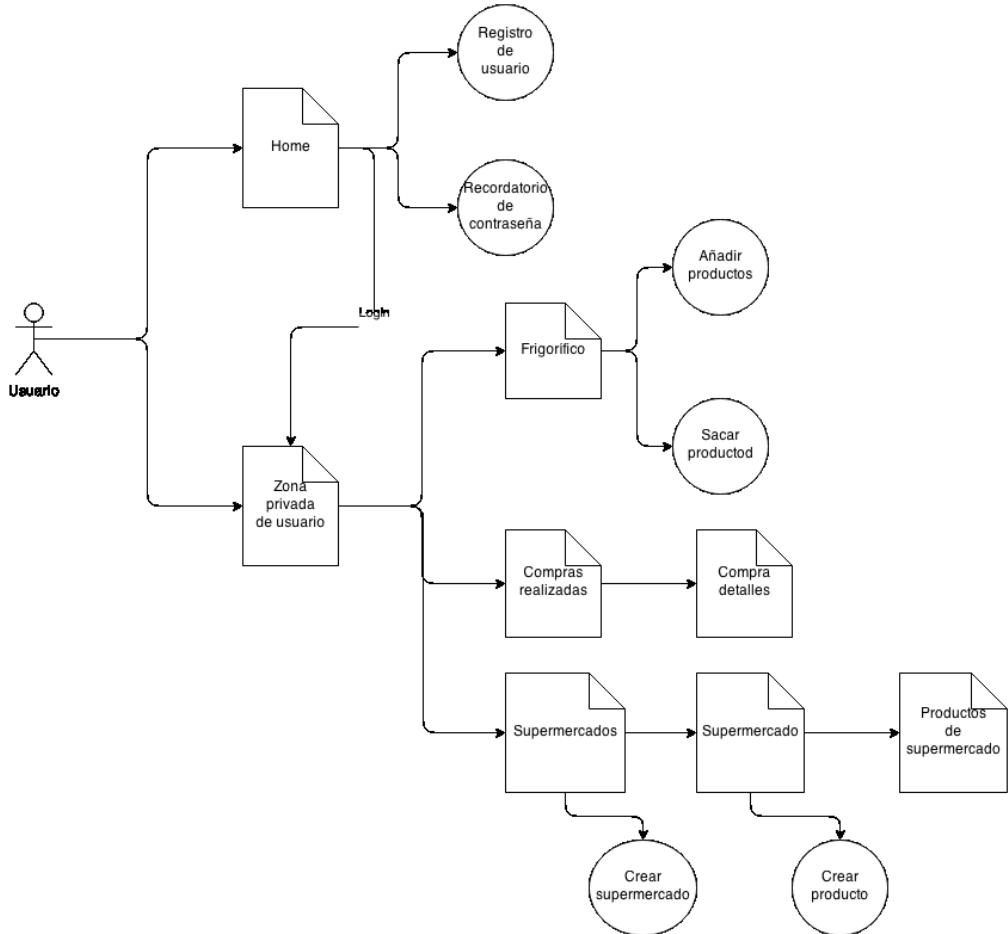


Figura 5.19: Flujo de la aplicación web

### 5.3.2. Secciones de la página web

En esta subsección se dará una breve explicación de cada sección de la plataforma web y se acompañará de una imagen.

- Home

Es la página web que visualiza un usuario no logueado en el sistema donde puede registrarse en ella, recordar la contraseña en caso de que la haya olvidado o registrarse.

Además, también se muestra información pública sobre los objetivos, la misión y la visión del proyecto.

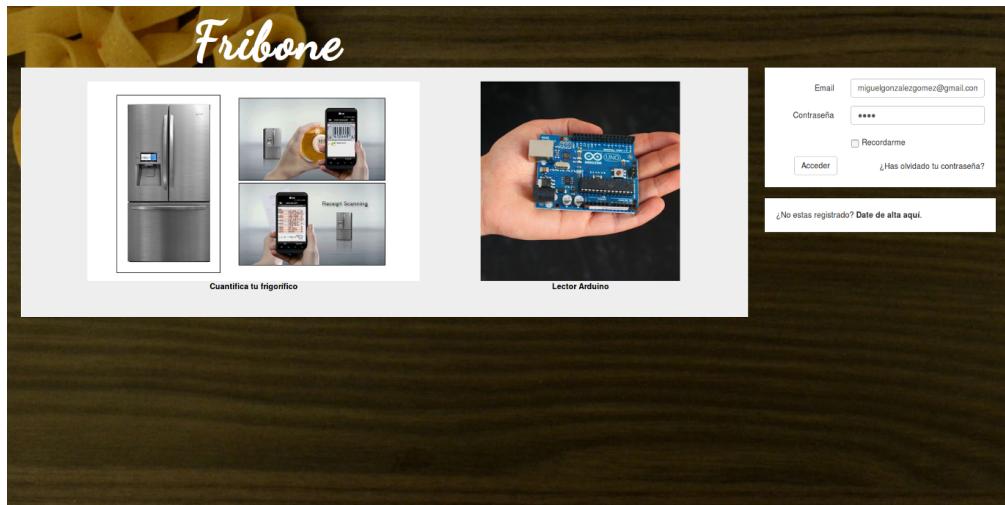


Figura 5.20: Web - Home pública

- Zona privada de usuario

Esta zona solo es accesible cuando un usuario registrado en la plataforma web se encuentra logueado en el sistema. En caso de no estarlo, se redirige a la Home pública.

La zona privada se encuentra formada de las siguientes secciones:

- Frigorífico

Es una de las secciones más importantes de toda la web ya que permite visualizar rápidamente los productos que se tienen en el frigorífico.

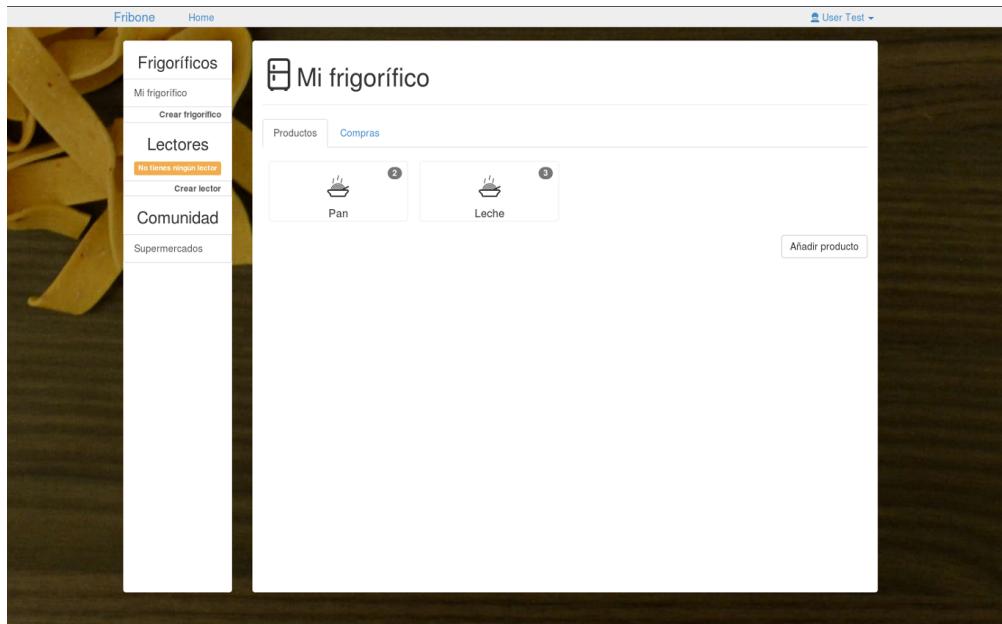


Figura 5.21: Web - Sección frigorífico

Se puede clickar en cada uno de los productos que se dispongan en el frigorífico para consultar más información acerca de cada uno de los productos (Precio, unidades, código de barras y código rfid).

- Compras realizadas

Las compras se crean automáticamente al añadir productos al frigorífico, sea desde la web o desde el componente hardware realizado con arduino.

Al introducir un producto se busca si hay una compra creada en la última media hora, si es así, el producto se mete a dicha compra, si no se crea una nueva compra. En resumen, una compra engloba todos los productos introducidos en un rango de tiempo de media hora.



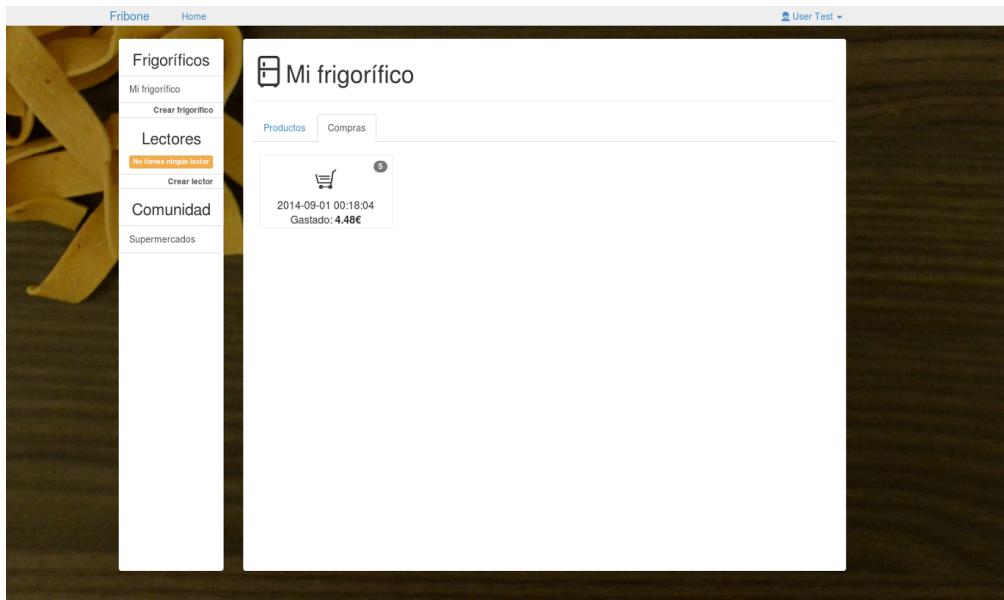


Figura 5.22: Web - Sección compras

- Detalles de una compra

Además de poder visualizar las compras realizadas, al clickar sobre una de ellas, se nos despliega toda la información de dicha compra; productos introducidos en el frigorífico y el total de la compra.

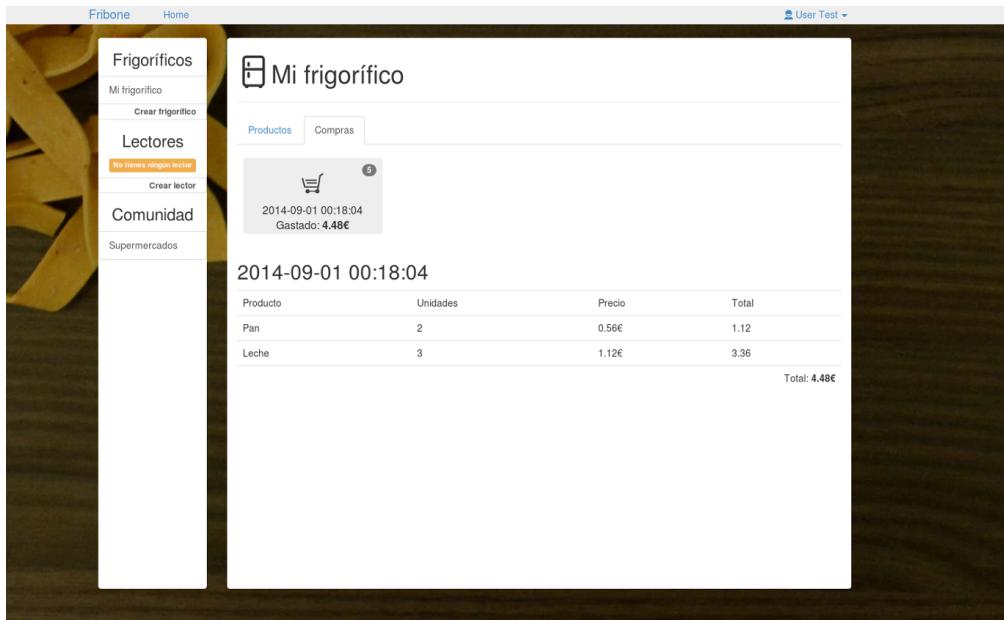


Figura 5.23: Web - Sección compra

- Supermercados

La sección de supermercados es común a todos los usuarios, es una zona colaborativa, donde se crean los supermercados existentes para poder añadirles productos.

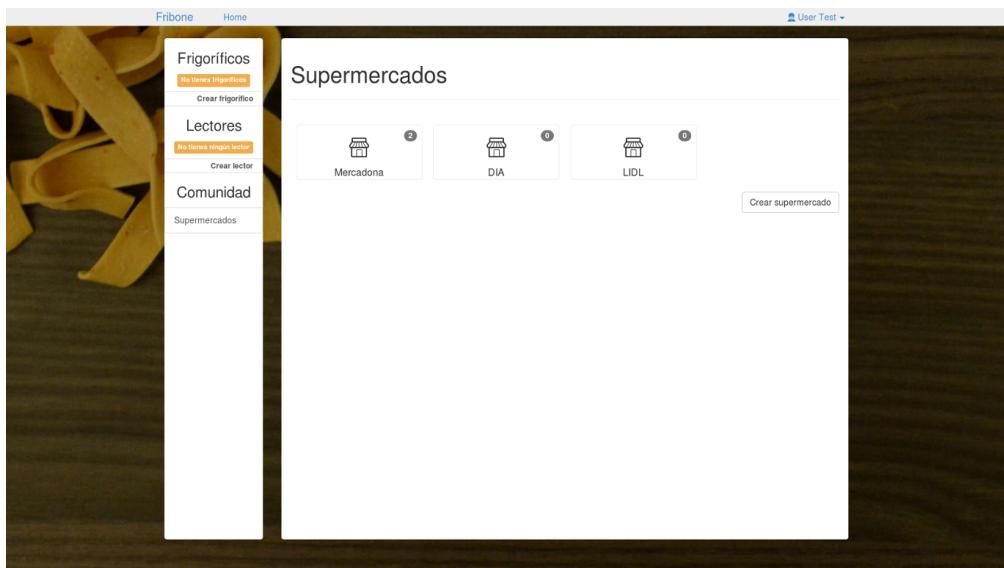


Figura 5.24: Web - Sección supermercados

- Vista de un supermercado

En esta sección se pueden visualizar los productos que dispone un supermercado, mostrar la información de cada producto y añadirle nuevos productos.

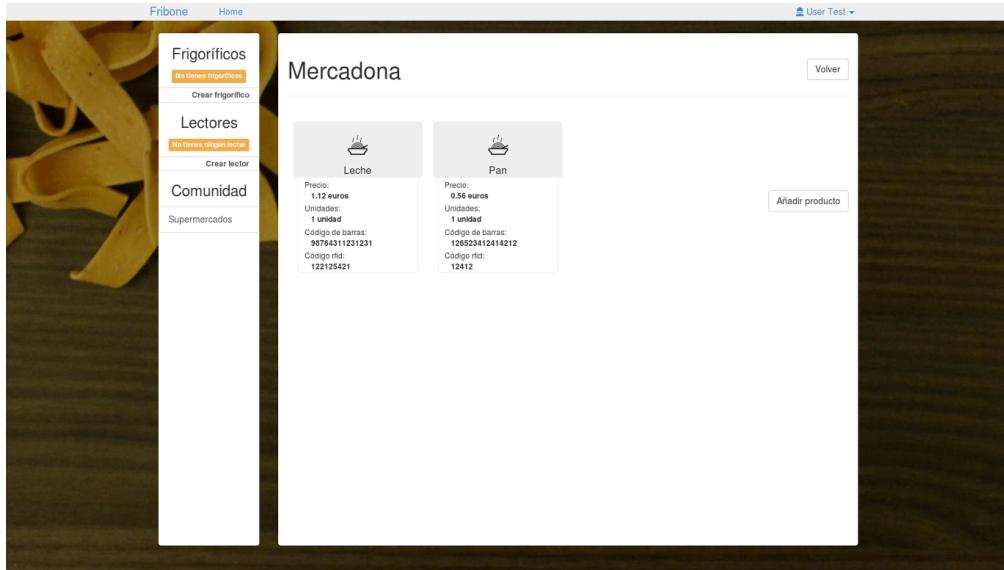


Figura 5.25: Web - Sección de un supermercado

Al añadir un producto hay que rellenar el siguiente formulario:

A modal dialog titled "Añadir producto" with a close button "x". It contains six input fields: "Nombre del producto", "Precio en euros", "Unidades del paquete", "Descripción del producto" (with a text area and scroll bar), "Código de barras", and "Código rfid". At the bottom are "Cancelar" and "Aceptar" buttons.

Figura 5.26: Web - Añadir supermercado

Estando disponible para todos los usuarios, así, al agregar dicho producto ya se dispone de toda su información.

- Lector

Esta parte de la web permite enlazar un lector con la cuenta de usuario. Para ello, explicado brevemente, se generan dos códigos de barras que deben ser escaneados por el lector en menos de una hora para que se efectúe la sincronización.

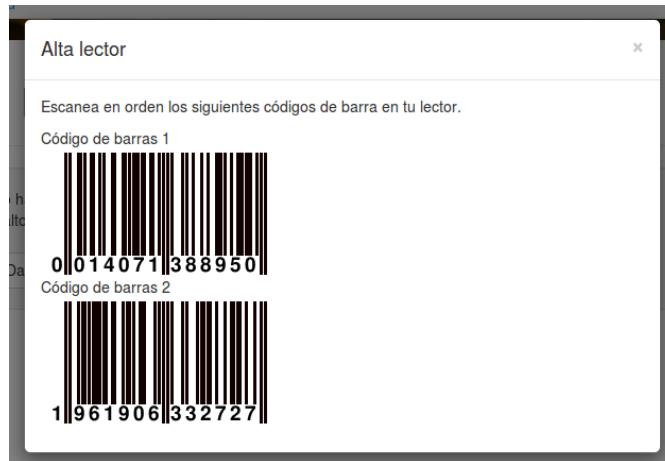


Figura 5.27: Web - Alta de un lector

### 5.3.3. Sistema de rutas

La navegación en la aplicación web se realiza a través de un sistema de rutas utilizando para ello la librería JavaScript *Director*. Esta librería funciona tanto desde lado cliente como desde el lado servidor con *Node.js*, de hecho, forma parte del framework *flatiron* para desarrollo web utilizando como tecnología *Node.js*.

Con *Director* se puede definir un sistema de rutas, y de rutas anidadas, que permitan ejecutar una (o varias) funciones al encontrarse sobre ella.

Una vez se carga la página web lo primero que se debe realizar es la creación de un objeto del tipo *Router*, el cual ofrece la librería. El objeto *Router* recibe en su constructor el sistema de rutas, de esta manera, se encarga automáticamente de llamar a las funciones que concuerden con la ruta que coincide con la dirección web del navegador. También puede recibir otros parámetros opcionales de configuración, en nuestro caso le hemos

indicado que utilice la *HTML5 History API* en lugar del sistema de Hashes que emplea por defecto.

El objeto *Router* también tiene un método para indicarle a que zona se quiere navegar, para ello se llama a *setRoute* pasando como parámetro la *URI*. El objeto *Router* se encargará de modificar la *URI* y de llamar a las funciones que se hayan definido para dicha ruta.

A continuación se muestra el sistema de rutas que se ha definido para el portal privado del usuario:

```

1 var routes = {
2     '/tablon': tablon.draw,
3     '/fridge/:id': {
4         '/productos': fridge.draw_productos,
5         '/compras': {
6             '/:id': fridge.draw_compra,
7             on: fridge.draw_compras
8         },
9         on: fridge.draw
10    },
11    '/supermercados' : {
12        '/:id': supermercado.draw,
13        on: supermercados.draw
14    }
15};
```

Al construir el objeto *Router* se le pasan las rutas que hemos definido y en la configuración se activa el uso del *HTML5 History API*.

```

1 var router = new Router(routes).configure({
2     html5history: true
3});
```

Existe otro parámetro de configuración muy interesante, el parámetro *recursión*, por defecto está deshabilitado en el lado cliente, si se activa se puede indicar el orden de recursión; *backward* ó *forward*. La recursión se utiliza cuando se crean subrutas, si se habilita, se puede definir si es llamado primero el hijo y luego el padre (o padres); o

viceversa.

Se ha dejado deshabilitado esta funcionalidad debido a que se controlan las llamadas entre rutas, subrutas... Desde los controladores, lo cual permite controlar si hay que renderizar el padre o si ya está renderizado.

Por último, para que *Router* empiece a funcionar, hay que llamar a su método *init*.

```
1 | router.init();
```

## 6. CONCLUSIONES

---

Este proyecto ha abordado el diseño y la construcción de un componente hardware apoyado por una plataforma en la nube para el seguimiento del consumo en el hogar, en concreto, las compras realizadas en los supermercados. Para ello, se ha utilizado *Arduino Yún* y una serie de componentes electrónicos con los que se ha creado un prototipo y además, a través de distintas tecnologías web, se ha diseñado una aplicación web.

La elección de este proyecto surgió de la curiosidad acerca de *Arduino*, del creciente entusiasmo en cuantificar todo lo que se pueda y de la multitud de aplicaciones que se utilizan para llevar el control de los gastos en el hogar.

Este proyecto se comenzó en Marzo del 2014 y se ha desarrollado a la par que la finalización del cuarto curso de *Grado en Ing. de Sistemas de Información* y un trabajo a jornada completa en *Informática ECI*. Se ha finalizado en Septiembre del 2014.

A continuación se muestran los distintos pasos o fases que se han realizado:

### 6.1. Primer paso

El primer paso en el proyecto fue hacerse con un *Arduino* para empezar a jugar con él y familiarizarse con su entorno de programación. Simultáneamente se adquirió un lector de código de barras con conector *USB HID*, un lector de códigos *RFID* y una pantalla *TFT*.

La versión de *Arduino* elegida fue la *UNO* debido a que su precio económico y a la amplia documentación que hay sobre la placa. El primer problema que se tuvo fue la falta de conexión *USB* de la placa, se resolvió adquiriendo una *Shield USB*. El segundo problema vino por la falta de conexión *WiFi*, se intentó resolver adquiriendo una *Shield WiFi* no oficial, pero no se logró hacer que funcionara.

Con estos problemas se optó por investigar *Arduino Yún*, una placa *Arduino* que resuelve estos problemas ya que dispone de un puerto *USB* y de un microprocesador con *Linux* y conexión *WiFi*, por lo que se optó por adquirir este componente.

Con todos los componentes adecuados se diseñó el primer software para el escaneo

de códigos de barras y de códigos RFID, así como una interfaz para mostrar los códigos en la pantalla *TFT*.

## 6.2. Segundo paso

El segundo paso consistió en desarrollar la aplicación servidor utilizando para ello algunas de las piezas que componen las metodologías ágiles; control de versiones, tests unitarios e integración continua.

- Para el control de versiones se ha utilizado *GitHub*, una plataforma que permite la creación de repositorios públicos de manera gratuita y que actualmente es la que más auge tiene en los proyectos de código abierto.
- Para los tests unitarios se ha utilizado *PHPUnit* debido a que es uno de los frameworks más conocidos. Se tuvieron que resolver problemas con *CodeIgniter* ya que no está pensado para realizar tests sobre él.
- Para la integración continua se ha utilizado *TravisCI*, una herramienta web que se sincroniza con tu cuenta de *GitHub* para construir y pasar los tests unitarios de todas las modificaciones que realices sobre el repositorio.

## 6.3. Tercer paso

A continuación, con las primeras versiones de la aplicación servidor, se fue realizando el diseño de la página web utilizando para ello *Bootstrap* y algunas librerías *JavaScript* como; *JQuery*, *Director*, *Handlebars*, etc.

Se ha realizado un diseño conocido como Single Page, es decir, la página web solamente se carga la primera vez que se accede y para las demás acciones se utiliza *AJAX* para traer la información y un sistema de plantillas para repintar partes de la página con dicho contenido.

## 6.4. Cuarto paso

En el cuarto paso se ha procedido a realizar la integración entre el componente hardware y la aplicación servidor. Para ello se han utilizado dos puntos de vista:

- Componente Hardware: A través de la librería *HttpClient* realiza una serie de peticiones a la *API* expuesta por la aplicación servidor.
- Aplicación Servidor: Expone una serie de métodos públicos (*API*) para poder realizar peticiones desde el exterior.

Para facilitar el manejo del componente hardware al usuario se ha procedido a realizar un menú sencillo a través de cuatro pulsadores (*botones*) y de la pantalla *TFT*. A través de este menú se pretende mostrar las distintas opciones que se pueden realizar con el aparato: introducir productos, sacar productos, etc.

## 7. LÍNEAS FUTURAS

---

La principal motivación de este proyecto es acercar la tecnología a una de las actividades cotidianas que menos ha cambiado a lo largo de la historia; realizar la compra.

Para ello se ha diseñado un dispositivo que permita cuantificar y ofrecer información al usuario acerca de sus compras. Por ejemplo, permitir consultar los productos que dispone en el frigorífico, las compras que ha realizado a lo largo del tiempo, análisis de gastos, etc.

Las grandes compañías a nivel mundial aún no han dado el salto a utilizar este tipo de tecnologías en sus productos convencionales. Sí disponen de algún frigorífico de gama alta con tecnología futurista, los cuales, no son accesibles para el consumidor medio.

Por ello, con este proyecto se pretende acercar al consumidor a parte de esta tecnología futurista. A través de un componente sencillo, barato y elegante. Así, con el objetivo de poder comercializar realmente este proyecto se proponen las siguientes líneas futuras.

### 7.1. COMPONENTE HARDWARE

El componente hardware que se ha diseñado es un prototipo funcional del producto final. Pero, para una comercialización del producto es recomendable realizar las siguientes innovaciones:

- Miniaturización

El componente que más ocupa del prototipo es el lector de código de barras. Es necesario encontrar un proveedor que proporcione un lector pequeño y discreto, de esta manera se puede integrar en la parte inferior del producto sin ocupar un gran espacio.

- Diseño

Fribone es un producto pensado para el hogar, en concreto, la cocina. Por lo que tiene que ser un producto elegante e integrador, nadie quiere un producto que rompa la armonía de la cocina.

- Ergonomía

La mayoría de las personas experimentan algún grado de limitación física en algún momento de su vida. Por lo tanto, el diseño final del producto, además de ser elegante, debe estar pensado para que pueda ser manejado por personas con ciertas discapacidades físicas.

Una manera de afrontar esta tarea es pensar en la eliminación de los botones del aparato, que pueden ser difíciles de pulsar, y añadir una pantalla más grande y con respuesta táctil. Así, se puede realizar una interfaz con unos botones en pantalla grandes y visibles.

## 7.2. COMPONENTE SERVIDOR Y SOFTWARE

La arquitectura y diseño del software ha seguido una metodología ágil. Así, se ha conseguido levantar una infraestructura usable en un tiempo breve. Gracias a los tests unitarios y a la plataforma de integración continua, *TravisCI*, se pueden seguir desarrollando pequeños cambios incrementales para dotar de más potencia a la aplicación:

- Universalidad

Integrar soluciones de acceso a la plataforma a partir de otros servicios (*Facebook*, *Google*, etc.), y ofrecer una *API RESTful* basada en *OAuth 2* para la autorización del acceso a la información.

- Data Mining

Explorar y buscar nuevas técnicas de análisis de la información que ingresan los usuarios para poder ofrecerles sugerencias en base a sus gustos, consejos a la hora de comprar, etc.

- Servicios a terceros

Animar a las compañías del sector comercial a ofrecer sus productos a través de la plataforma. De esta manera, se le pueden ofrecer nuevas experiencias al usuario como la planificación de compras automáticas en base a los productos que dispone en su frigorífico.

## 8. BIBLIOGRAFÍA

---

- [1] Apache. Apache - HTTP Server Project). <https://httpd.apache.org>.
- [2] Arduino. Forum Arduino). <http://forum.arduino.cc/>.
- [3] Arduino. Página Oficial de Arduino. <http://www.arduino.cc/>.
- [4] dompdf. dompdf - Is an HTML to PDF converter). <https://github.com/dompdf/dompdf>.
- [5] EllisLab. CodeIgniter - A Fully Baked PHP Framework). <http://ellislab.com/codeigniter>.
- [6] Flatiron. Librería JavaScript Director). <https://github.com/flatiron/director>.
- [7] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. <http://www.buyya.com/papers/Internet-of-Things-Vision-Future2013.pdf>, 2013. [ELSEVIER].
- [8] Dante Israel Tapia Martínez. Desarrollo e implementación de un sistema domótico en un hogar del estado de Colima. <http://hdl.handle.net/10317/1787>. [Tesis - Universidad de Colima].
- [9] David Montoro Mouzo. Generación automática de código para la plataforma domótica KNX/EIB en un marco dirigido por modelos. <http://hdl.handle.net/10317/1787>. [PFM/TFM-Máster en Tecnologías de la Información y las Comunicaciones].
- [10] MySQL. MySQL - A Open Souce database). <http://www.mysql.com>.
- [11] Renato Jorge Caleira Nunes. Home Automation - A Step Towards Better Energy Management. <http://www.icrepq.com/pdfs/CALEIRA416.PDF>. [IST – Technical University of Lisbon].
- [12] PHP. PHP (Hypertext Preprocessor). <http://www.php.net>.
- [13] PHPUnit. PHPUnit – The PHP Testing Framework). <http://phpunit.de/>.
- [14] Wikipedia. Arduino. <http://es.wikipedia.org/wiki/Arduino>.

- [15] Wikipedia. Representational state transfer. [http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer).
- [16] Wikipedia. X10. <http://es.wikipedia.org/wiki/X10>.
- [17] Zeya Zheng. Home Automation - Smart home technology and template house design. [http://www.theseus.fi/bitstream/handle/10024/62819/Zheng\\_Zeya.pdf](http://www.theseus.fi/bitstream/handle/10024/62819/Zheng_Zeya.pdf). [Bachelor's degree (UAS)].