Gestión de alimentos en el hogar mediante RFID

Alfonso Palazón González
Máster Universitario en Ingeniería de Telecomunicación
Sistemas de comunicación

Raúl Parada Medina
Carlos Monzo Sánchez

09/06/2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Gestión de alimentos en el hogar mediante RFID*
Nombre del autor:	*Alfonso Palazón González*
Nombre del consultor/a:	*Raúl Parada Medina*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	*09/06/2016*
Titulación:	*Máster Universitario en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Sistemas de Comunicación*
Idioma del trabajo:	*Castellano*
Palabras clave	*RFID, Gestión de alimentos, Aplicación web, Plataforma de prototipado electrónico.* *RFID, Food Management, Web Application, Electronic prototyping platform.*

Resumen del Trabajo:

Un tercio de los alimentos que se producen en el planeta termina en la basura, estos alimentos serían suficientes para acabar con el hambre en el mundo. El impacto de este desperdicio no es solo económico, para producir alimentos se emplean fertilizantes y pesticidas, así como combustible para el transporte de los mismos, lo que contribuye al calentamiento global.

Por otra parte, la sociedad está cada vez más preocupada por llevar unos hábitos alimentarios saludables, el sobrepeso y la obesidad afectan a gran parte de la población.

Tratando de buscar una solución a los problemas anteriormente descritos, se plantea un sistema de gestión de alimentos etiquetados con RFID (*Radio Frequency IDentification*), consistente en lectores RFID ubicados en la despensa y el frigorífico del usuario, que leen la información de los alimentos, de forma automática, y sin necesidad de visión directa entre lector y etiqueta, como sí ocurre con el código de barras.

Como complemento, se desarrolla una aplicación web multi-plataforma, desde la que consultar fecha de caducidad e información detallada de los alimentos. La aplicación avisa de cuándo va a caducar un determinado producto, y ofrece recetas que se pueden preparar con los alimentos de los que se dispone, evitando que se desperdicien. Dichas recetas se acompañan de su información nutricional, para que el usuario pueda llevar un control exhaustivo de lo que come.

Este sistema aporta beneficios económicos al fabricante, ya que se ofrece a los supermercados pagar por mostrar publicidad de sus productos a los usuarios de la aplicación.

Tras la implementación del sistema se muestran las conclusiones del diseño, y se indican futuros puntos de mejora.

Abstract:

A third of the food produced in the world ends up in the rubbish, this food would be enough to put an end to world hunger. The impact of this waste is not only economic, but also environmental, the fertilizers and pesticides employed during food production, and the fuel used in its transportation, contribute to global warming.

On the other hand, society is increasingly concerned to bring healthy eating habits. Overweight and obesity affect much of the population.

A RFID (Radio Frequency IDentification) food management system, is designed in order to find a solution to previously described issues. It consists of RFID readers placed on a user's fridge and pantry, which automatically read food information. There is no need of direct sight between reader and tag, as it occurs through barcode technology.

As a complement, a multi-platform web application is developed, allowing its users to check the date of food expiration and other detailed information. The application notifies the user when a product is about to expire. It also offers recipes that might be prepared with available foods, preventing them from being wasted. The recipes are accompanied by their nutritional information, so that the user can exhaustively monitor what he eats.

This system provides economic benefits to the manufacturer, since it allows supermarkets to pay for displaying their products advertised through the application.

After system deployment, design conclusions are shown, and future improvement points are indicated.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Un tercio de los alimentos que se producen para consumo humano acaban en la basura. Esto significa que un tercio de los recursos económicos y humanos empleados en su producción son desperdiciados. Por otra parte, un tercio de las emisiones de gas generadas por la industria alimentaria se producen en vano [1].

En la figura 1 se observa cómo en los países más industrializados es donde más alimentos se desperdician. Este desperdicio se produce bien por parte del productor, desde la producción hasta la distribución, o bien por parte del consumidor final, en el hogar.

Hay una relación directamente proporcional entre el nivel de industrialización de un país y el volumen de alimentos desperdiciados. Además, en los países más industrializados, el porcentaje de comida desperdiciada por el consumidor, es mayor que en el resto.

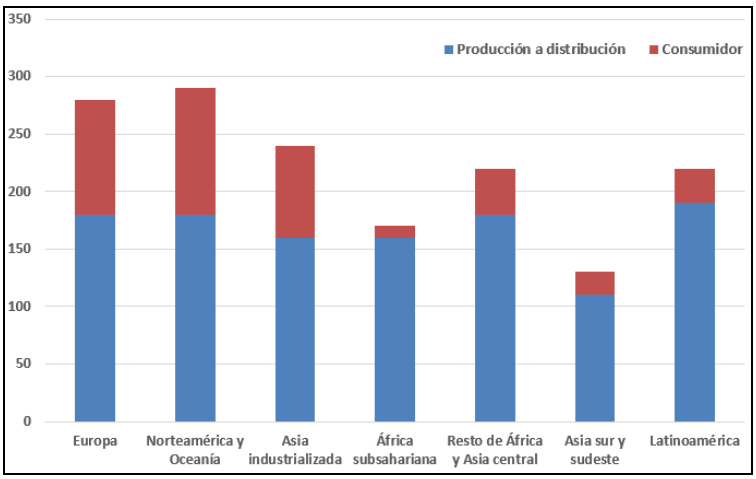

Figura 1 Pérdida y desperdicio de alimentos por persona (kg/año) [2]

No es este el único problema relacionado con los alimentos, el sobrepeso y la obesidad se han convertido en auténticas epidemias. En la figura 2 se muestra el porcentaje de población de cada región que sufre estos problemas:

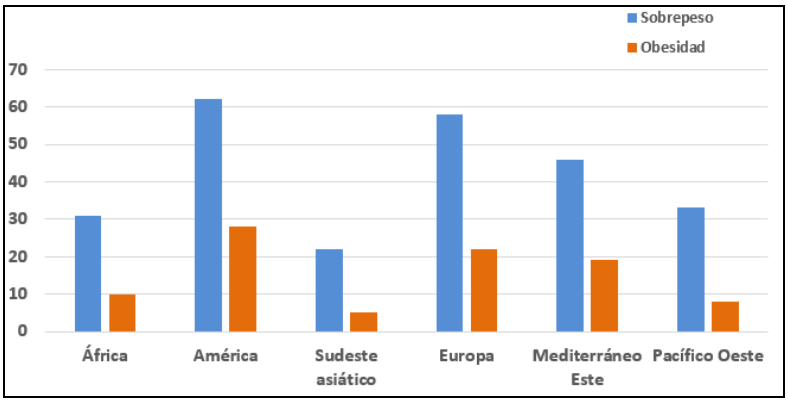

Figura 2 Porcentaje de la población con sobrepeso u obesidad por región [3]

El sobrepeso y la obesidad generan efectos adversos en el metabolismo, produciendo un aumento en la presión sanguínea, colesterol, triglicéridos, resistencia insulínica, y el riesgo de padecer enfermedades coronarias y diabetes [4].

Además de los efectos en la salud de los ciudadanos, la economía de sus gobiernos se ve enormemente impactada por el coste sanitario asociado al tratamiento de las enfermedades derivadas del sobrepeso y la obesidad [5].

A pesar del esfuerzo de los gobiernos, tanto en el intento de reducir el volumen de alimentos desperdiciados, como en el de promover hábitos saludables entre sus ciudadanos, el sobrepeso y la obesidad son problemas que están aún lejos de ser erradicados, por lo que se deben buscar nuevas formas de combatirlos [6].

Este trabajo pretende aportar una solución que reduzca el desperdicio de alimentos por parte del consumidor final, y que al mismo tiempo le permita llevar un control de su alimentación, facilitándole llevar una dieta saludable.

La solución desarrollada se basa en un nuevo paradigma de computación, el IoT (*Internet of Things),* en el cual, el uso de internet se extiende a los objetos físicos, permitiendo su interacción con el mundo digital. En este caso, los alimentos incorporan una etiqueta RFID (*Radio Frequency IDentification*).

La tecnología RFID permite que el usuario pueda saber de manera sencilla qué productos tiene, y cuándo van a caducar, evitando que se malgasten. Esto sería muy complicado de conseguir con el etiquetado tradicional mediante código de barras, ya que sería necesaria visión directa entre el lector y los alimentos, y no se podrían hacer lecturas simultáneas y automáticas, algo que sí permite el RFID [7].

El usuario dispone de lectores RFID tanto en su frigorífico como en su despensa, estos lectores envían la información obtenida a un servidor. El usuario puede consultar mediante una aplicación web los alimentos de que dispone, además de su fecha de caducidad o envasado, puede consultar su información nutricional, y qué recetas puede preparar con ellos.

Por último, se plantea una manera de monetizar el uso de la aplicación, se abre la posibilidad de que los supermercados paguen al fabricante del sistema por hacer ofertas de sus productos a los usuarios.

1.2 Objetivos del Trabajo

Los objetivos del trabajo son los siguientes:

- Desarrollar una aplicación *web* multiplataforma que permita: gestionar los alimentos del usuario, consulta y recomendación de recetas, acceso a ofertas de productos.
    - Implementar la capa de presentación e interactuación con el usuario, o *front-end*.

    - Implementar la capa de acceso a datos, o *back-end*.
    - Implementar la base de datos.
- Estudiar varios algoritmos de recomendación de recetas, compararlos, e implementar el mejor de ellos. La recomendación funcionará en base a diferentes criterios: alimentos disponibles, tipo de receta, filtrado por palabras clave, filtrado en base a las intolerancias alimentarias del usuario.
- Crear un prototipo del sistema de gestión de alimentos mediante RFID.
    - Crear un prototipo con una placa de prototipado *Arduino*, un lector RFID, y un módulo *Ethernet*.
    - Crear un programa para grabación de tarjetas con información de productos.
    - Crear un programa para lectura de tarjetas, y envío de su contenido al servidor de la aplicación *web*.

1.3 Enfoque y método seguido

La estrategia empleada para la consecución de los objetivos consiste en desarrollar un nuevo sistema que permita leer la información de los alimentos que el usuario tiene tanto en su frigorífico como en su despensa, mediante la tecnología RFID.

Se desarrolla una aplicación web que permite llevar un control de los alimentos de que se dispone. Entre la información de los alimentos, se tiene su fecha de caducidad o envasado, información nutricional y de alérgenos. Además, la aplicación cuenta con un algoritmo que hace recomendaciones de recetas en base a diferentes criterios.

Para simular el funcionamiento del frigorífico y despensa se emplea un *Arduino*, al que se le conecta un módulo RFID. Este módulo puede grabar y leer tarjetas, en las que se guardará la información de diferentes productos.

Este enfoque eminentemente práctico se ha considerado el más apropiado para el trabajo, pues el producto final debe ser totalmente operativo, no se trata de obtener un resultado teórico.

1.4 Planificación del Trabajo

Se divide la planificación del trabajo en cinco grandes tareas, que dan como resultado las cinco PECs (Pruebas de Evaluación Continua) de la asignatura:

<u>Introducción y planificación</u>
En esta primera tarea se redacta la introducción del proyecto, que consiste en su resumen, contexto, motivación, objetivos, planificación, enfoque, y método seguido.

A continuación, se realizan unas pruebas iniciales de lectura/escritura con un módulo RFID conectado a un *Arduino*, con el objetivo de conocer el detalle de su funcionamiento.

Estado del arte

Se hace una labor de investigación para conocer el estado del arte en materia de RFID e Internet de las cosas, se redactarán los aspectos más importantes, incluyéndolos en la memoria. Por último, se enumeran las soluciones existentes, que distintos autores y fabricantes han dado al problema descrito en este trabajo, y se comparan con el mismo.

Implementación del sistema y simulaciones

En primer lugar se debe hacer un diseño de la arquitectura a nivel de *hardware* y *software*, y decidir qué tecnologías se van a emplear.

Hecho esto, se modela la base de datos, y se hace una carga inicial de datos, con los que se trabajará durante toda la fase de desarrollo de la aplicación.

Tras tener lista la base de datos, se trabajará en paralelo en el *front-end* y *back-end* de la herramienta, diseñando su interfaz, configurando la interacción con el usuario, la relación entre *front-end* y *back-end*, las operaciones en la base de datos, y algoritmos de recomendación de recetas.

A continuación, se hacen varias simulaciones, para probar el funcionamiento, tanto del sistema RFID, como de la aplicación, y de todo el conjunto.

Redacción de la memoria

Se redacta la versión final de la memoria del trabajo, incluyendo las conclusiones, glosario, y anexos, dejándola lista para su evaluación.

Defensa trabajo fin de máster

Esta última tarea consiste en preparar la defensa del trabajo, se hace una labor de síntesis para extraer la información más importante, que se incluirá en la presentación. Además de realizar la presentación, se grabarán unos vídeos que demuestren el funcionamiento del sistema, de principio a fin.

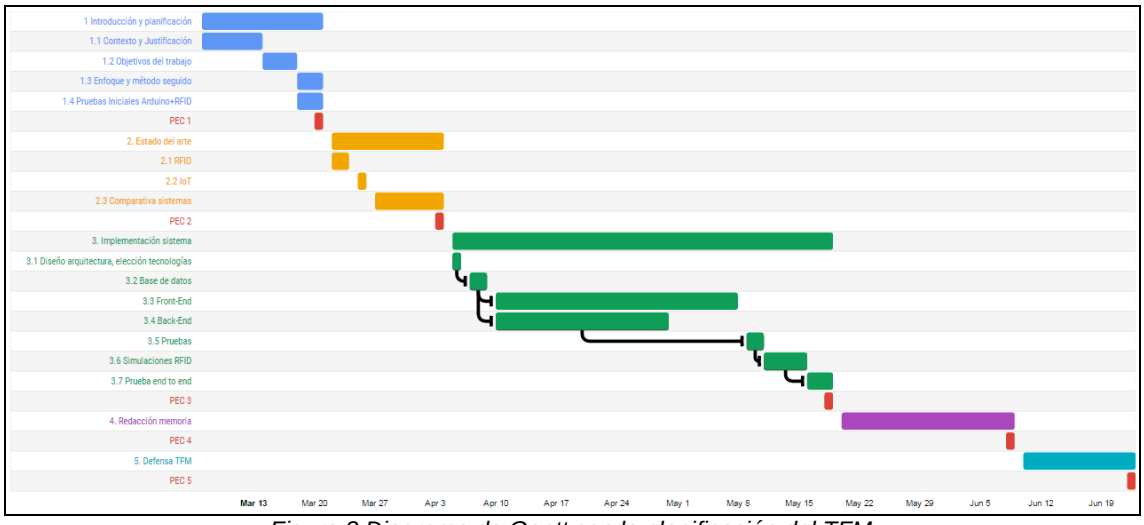

Figura 3 Diagrama de Gantt con la planificación del TFM

1.5 Breve sumario de productos obtenidos

Se ha implementado un sistema para la gestión de alimentos en el hogar, denominado RFM (*RFID Food Management*), con el objetivo de reducir el desperdicio de alimentos, y que sus usuarios lleven una dieta más saludable.

Los alimentos cuentan con etiquetado RFID, estas etiquetas almacenan información detallada de los mismos. Dicha información se lee desde lectores RFID situados en el frigorífico y despensa del usuario.

En primer lugar, se ha desarrollado una aplicación *web* multiplataforma, llamada RFM App, tanto su *front-end*, como su *back-end* y base de datos. Desde esta aplicación se puede consultar información detallada de los productos del usuario. Además, cuenta con una base de datos de recetas, y un algoritmo que las recomienda en base a los productos disponibles, y a varios criterios más. La aplicación es una interesante fuente de ingresos para el fabricante del sistema, ya que los supermercados pueden contratar publicidad para mostrar ofertas de sus productos en ella.

En segundo lugar, se implementa un sistema RFID, encargado de leer la información de los productos. Se ha diseñado un prototipo de dicho sistema RFID empleando una placa *Arduino*. El prototipo permite hacer lecturas de tarjetas RFID que contienen información de productos, y enviar el resultado de las mismas al servidor de la aplicación.

1.6 Breve descripción de los otros capítulos de la memoria

- En el capítulo de **introducción** se explica cuál es la motivación del trabajo, se plantea una primera aproximación del sistema propuesto, y se realiza una planificación de las distintas actividades que se llevarán a cabo.
- El capítulo de **estado del arte**, presenta los aspectos más importantes de la tecnología RFID, y la compara con el código de barras. Por último, se analizan sistemas similares al desarrollado en este trabajo, planteados en artículos científicos, y por diferentes fabricantes, que son comparados con el sistema RFM.
- En el capítulo de **arquitectura del sistema y aplicación**, se hace un diseño de la arquitectura del sistema, y se eligen las tecnologías que se van a emplear en el desarrollo de la aplicación *web*. A continuación, se detalla cómo se realiza la implementación de la aplicación, tanto su *front-end*, como el *back-end*, y la base de datos.
- El capítulo **sistema RFID**, describe, en primer lugar, qué *hardware* se ha elegido para la implantación del prototipo del sistema RFID, para lo que se hace uso de una placa *Arduino*. En la segunda parte de este capítulo, se explica cómo se programa el *Arduino*, tanto para grabar tarjetas que contendrán información de diferentes productos, cómo para la posterior lectura y envío de dicha información al servidor de la aplicación.
- El capítulo **diseño final y simulaciones**, muestra cómo queda el sistema en su conjunto, tanto a nivel de aplicación *web*, como de sistema RFID. Más tarde, se realizan diversas pruebas: una prueba de

inicio a fin, que demuestra el correcto funcionamiento del sistema, y la interconexión entre la aplicación y el sistema RFID. Así como pruebas para evaluar el rango y velocidad de lectura y escritura en diferentes escenarios, o pruebas de lectura simultánea de tarjetas. Por último, se prueban los algoritmos de recomendación de recetas.

- El capítulo de **conclusiones**, expone las conclusiones del trabajo, indicando si se han conseguido los objetivos del mismo, qué se ha aprendido durante su realización, y qué puntos de mejora se proponen.

2. Estado del arte

2.1 La tecnología RFID

2.1.1 Introducción

La tecnología RFID (siglas de *Radio Frequency IDentification*, en castellano identificación por radiofrecuencia), es un sistema para almacenamiento y recuperación de datos de forma remota, para ello hace uso de dispositivos llamados etiquetas, o *tags* RFID. Estas etiquetas logran almacenar una gran cantidad de información. Gracias a su pequeño tamaño y diversas formas, pueden ser adheridas a un producto, animal, o incluso personas. Esta información, es leída por una antena RFID, y procesada con posterioridad [8].

Los sistemas RFID se suelen clasificar según el tipo de etiquetas que emplean:

- **Etiquetas activas**, poseen su propia fuente de energía, que utilizan para alimentar sus circuitos integrados y propagar su señal al lector. Son las etiquetas más fiables, tienen menos errores, y pueden transmitir información a más distancia que el resto.
- **Etiquetas pasivas**, no poseen ningún tipo de alimentación. La señal proveniente del lector induce una corriente eléctrica para operar el circuito integrado de la etiqueta, y para generar y transmitir una respuesta. Cuentan con la ventaja de que pueden ser de un tamaño muy reducido, son baratas, y se pueden adherir a casi cualquier objeto. La mayoría de sistemas RFID emplean este tipo de etiquetas.
- **Etiquetas semipasivas**, poseen una fuente de alimentación propia, aunque en este caso se utiliza principalmente para alimentar el circuito integrado y no para transmitir una señal. La transmisión de señal al lector se realiza como en las etiquetas pasivas.

2.1.2 Historia

La tecnología RFID aparece durante la 2ª Guerra Mundial. Los distintos ejércitos utilizaban el radar, gracias a él, se podía avisar del acercamiento de aviones cuando aún estaban a kilómetros de distancia. Sin embargo, no había manera de saber si el avión que se aproximaba era aliado o enemigo. Los alemanes descubrieron que si balanceaban sus aviones al volver a la base, la señal de radio reflejada en ellos variaba, este sistema avisaba al equipo encargado del radar de que el avión que se aproximaba era aliado, este es, de manera esencial, el primer sistema RFID pasivo. En 1942, el ejército británico, desarrolló el primer sistema IFF activo (siglas de *Identify Friend or Foe*, sistema de identificación de amigo o enemigo). Instalaron un transmisor en cada avión británico, cuando estos transmisores recibían la señal del radar, emitían una señal que identificaba al avión como aliado. Este fue el primer sistema RFID activo. [9]

En los últimos años se está potenciando la comercialización de la tecnología RFID, introduciendo nuevas aplicaciones, mejoras en la lectura, reducción de

costes del equipamiento etc. Se prevé que se implante de forma masiva en la cadena de suministro, siendo el sustituto del código de barras.

2.1.3 Comparativa entre RFID y el código de barras

La tecnología RFID y el código de barras nacen con un mismo objetivo: la identificación automática. Sin embargo, existen muchas diferencias entre ellas, como se puede ver en la siguiente tabla:

Código de barras	Etiqueta RFID
Requiere línea directa de visión con el lector.	Se puede leer sin línea directa de visión con el lector.
Solo se puede leer de manera individual.	Se pueden leer múltiples etiquetas al mismo tiempo.
No se puede leer si está dañado o sucio.	Se pueden hacer lecturas en entornos con suciedad.
Solo se puede identificar el tipo de artículo leído.	Pude identificar un artículo específico.
No se puede actualizar su información.	Se puede reescribir nueva información.
Requiere un seguimiento manual, susceptible a error humano.	No requiere seguimiento manual.

Tabla 1 Comparativa entre el código de barras y RFID [10]

Las características anteriormente expuestas hacen que la tecnología RFID sea preferible sobre el código de barras en la mayoría de situaciones, particularmente en las que se desea leer productos de manera rápida. No obstante, en algunas situaciones, como en aquellas en las que se produce contacto con metales o líquidos, o en presencia de interferencias, la tecnología RFID podría no ser adecuada [11].

2.2 Internet de las cosas

El IoT *(Internet of Things*), o Internet de las cosas, es un nuevo paradigma de computación en el que los objetos físicos de la vida cotidiana se integran con las redes de información, y se convierten en participantes activos en los procesos de negocio. Existe la posibilidad de interactuar con estos “objetos inteligentes” a través de *Internet*, pudiendo consultar su estado e información relacionada con los mismos [12].

Desde un punto de vista técnico, la arquitectura del IoT, se basa en herramientas de comunicación de datos, principalmente objetos con etiquetado RFID. Gracias al RFID, se pueden identificar, rastrear y localizar objetos.

2.3 Antecedentes

En este apartado se enumeran algunos ejemplos de proyectos relacionados con el objeto de este trabajo: la gestión de alimentos en el hogar.

2.3.1 Samsung RF28K95800SR smartfridge

En el CES (*Consumer Electronics Show*) del año 2016, *Samsung* presentó su *smartfridge*, o frigorífico inteligente, modelo RF28K95800SR. Se trata de un frigorífico equipado con cámaras. Estas cámaras hacen fotografías de lo que hay en su interior cada vez que se abre la puerta. Las fotografías se pueden visualizar desde un dispositivo móvil, gracias a una aplicación desarrollada para este este fin, por lo que los usuarios pueden saber en todo momento los productos de los que disponen. Además, el frigorífico tiene una pantalla táctil en su exterior, desde la que se puede hacer la compra, tomar notas, incluso ver la televisión o escuchar música [13].

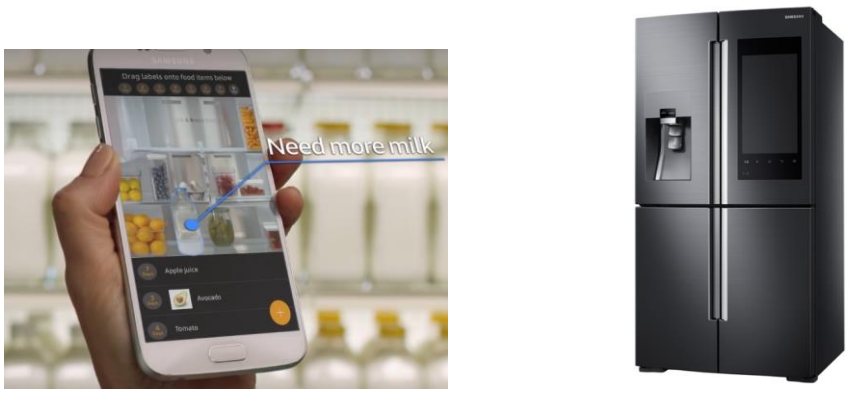

Figura 4 Izquierda: App móvil Samsung RF28K95800SR [13]. Derecha: Samsung RF28K95800SR [13]

En comparación con el sistema que se propone en este trabajo, su funcionalidad se ve muy limitada porque no dispone de lectores RFID, ya que actualmente, los productos siguen llevando código de barras, por tanto, solamente se tiene una foto de los productos que hay dentro del frigorífico, no se dispone ni de información nutricional de los alimentos, ni de su fecha de caducidad, ni de qué recetas se pueden preparar con ellos.

Además, solo se tiene información de los alimentos que hay en el frigorífico, el sistema que se va a desarrollar en el presente trabajo, será capaz de gestionar los alimentos que se tengan en el frigorífico y en la despensa.

2.3.2 LG Smart ThinQ LFX31995ST smartfridge

LG lanzó al mercado, a finales de 2012, su *smartfridge*. Se trata de un frigorífico tradicional que dispone de una pantalla táctil con varias aplicaciones. Estas aplicaciones permiten realizar un inventariado de los productos, por lo que cada vez que se introduce un nuevo alimento, o que se consume por completo, hay que actualizar esta información manualmente a través de su correspondiente aplicación. También se puede indicar la fecha de caducidad de los alimentos, y que, bien desde la pantalla del frigorífico, o desde el teléfono móvil, se avise al usuario cuando un producto está a punto de caducar. Existen otras aplicaciones desde las que consultar recetas, o llevar un control de los alimentos que se han consumido [14].

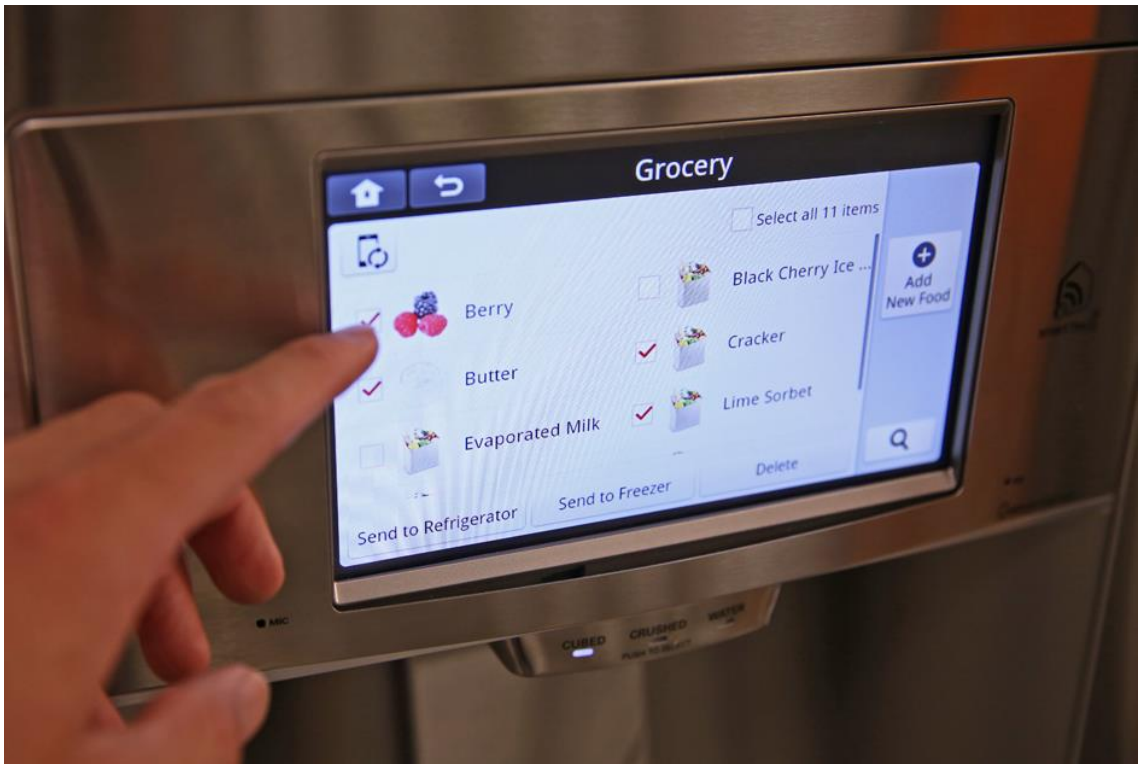

Figura 5 LG Smart ThinQ LFX31995ST, usuario haciendo el inventariado de sus productos [14]

Tal y como ocurre con el frigorífico de *Samsung*, este frigorífico no dispone de lectores RFID, por lo que todo el inventariado se debe hacer de forma manual, lo que presenta una gran desventaja respecto al sistema ideado en este trabajo.

2.3.3 Prototipo de frigorífico inteligente con capacidad para mejorar la salud y hábitos alimentarios

En 2009 se publica un artículo en el que se expone un prototipo de frigorífico inteligente [15]. Las aplicaciones de este frigorífico se centran en ofrecer información nutricional y dietética de los alimentos, y controlar los hábitos alimentarios de sus usuarios. Esta nevera dispone de una pantalla táctil en la puerta, desde la que introducir qué alimentos hay en ella, indicando su fecha de caducidad. Gracias a una base de datos, se puede consultar la información nutricional de los alimentos y recetas que se pueden realizar con los productos disponibles.

Este prototipo permite también que cada usuario introduzca su información médica, esto es, si padece diabetes, o si tiene alto el colesterol, así como su peso y altura, para llevar un seguimiento de su índice de masa corporal.

Por último, el frigorífico avisa cuando un producto va a caducar.

Este prototipo tiene opciones muy interesantes, que no están dentro del alcance de este proyecto, como es la monitorización de la salud de sus usuarios. Tiene incluso más funcionalidades que algunos productos que se comercializan en la actualidad. Como puntos negativos destacar, una vez más, que el inventariado de productos se hace de forma manual, conscientes de este inconveniente, los autores de este prototipo, indican como punto de

mejora emplear la tecnología RFID para leer los productos de manera automática. Además, debido a que fue ideado en el año 2009, no se pensó en la integración del frigorífico con dispositivos móviles, por lo que es necesario estar junto al frigorífico para hacer uso de él.

Figura 6 Lista de la compra en la aplicación del prototipo de frigorífico inteligente [15]

2.3.4 Prototipo de "*Pervasive Fridge*"

Este prototipo es presentado en un artículo del año 2012 [16]. Su nombre, *"Pervasive Fridge",* hace referencia al *Pervasive Computing* o computación ubicua, paradigma de computación en el que se integra la informática en el entorno de las personas.

Este prototipo consiste en una aplicación móvil con capacidad para registrar las entradas y salidas de productos del frigorífico, mediante lectura de código de barras, etiquetas RFID o reconocimiento de voz. Una vez se han registrado los alimentos, se avisa al usuario de cuándo van a caducar, evitando que se malgasten. También se puede consultar la información nutricional de los alimentos, recetas, y recomendaciones dietéticas.

Este sistema es compatible con cualquier frigorífico, este no necesita disponer ni de lectores de código de barras o RFID, ni de conexión a Internet, ni de pantalla táctil. Todo el valor añadido que ofrece este prototipo lo proporciona una aplicación para *smartphone*.

La principal ventaja de este prototipo es que es un sistema muy económico, pues no hay que hacer una inversión en un nuevo frigorífico, basta con tener un *smartphone*, dispone de muchas de las opciones que se plantean en este trabajo. Sin embargo, el proceso de inventariado es muy tedioso, pues hay que leer los productos uno a uno, manualmente.

Figura 7 Pervasive Fridge, escaneado de un producto mediante código de barras [16]

2.3.5 Prototipo de frigorífico basado en RFID en un hogar inteligente

En el año 2009 se publica un artículo [17] en el que se idea un frigorífico equipado con lectores RFID. Este frigorífico ofrece diferentes servicios para ajustar los hábitos alimentarios de sus usuarios. Si alguno de los usuarios padece alguna enfermedad, el sistema se comunica con el hospital para obtener su información médica, e informar al paciente de qué alimentos debe consumir. Por otra parte, se hacen recomendaciones de recetas para que todos los miembros de la familia lleven una dieta equilibrada.

El sistema trabaja en tres ámbitos:

- Hogar, en el hogar hay un frigorífico con lectores RFID, un servidor del que se obtienen los datos de los alimentos, y en el que se almacenan los productos disponibles, y un cliente móvil desde el que gestionar todo.
- Centro médico, si alguno de los usuarios tiene una enfermedad relacionada con su alimentación, se descarga su historial médico con el objetivo de realizar un seguimiento más exhaustivo que al resto de usuarios.
- Supermercados, en función de los hábitos alimentarios de la familia, y de los productos que se tienen, se realiza una lista de la compra en base a los productos de un determinado establecimiento, que se manda al usuario mediante SMS.

Este prototipo está muy centrado en el control de la dieta de personas que tengan alguna enfermedad. Esta integración con los diferentes centros médicos no parece fácil de llevar a cabo. El presente trabajo pide información a cada usuario sobre posibles intolerancias alimentarias cuando se registra, por lo que no es necesario disponer de un historial clínico, ni integrar el sistema con las bases de datos de los hospitales.

En cuanto a la relación con los supermercados, aquí se trata de realizar la lista de la compra de manera automática, esta integración es muy interesante, aunque también parece complicada, debido al gran número de establecimientos que existen. En el sistema propuesto en este trabajo, la

integración va dirigida únicamente a mostrar ofertas de diferentes establecimientos.

Por último, en el sistema propuesto en este trabajo, el *back-end* está centralizado, mientras que en este prototipo en cada casa se debe disponer de un servidor, lo que es una desventaja, ya que incrementa los costes, y el fabricante tiene que mantener actualizadas las bases de datos de cada usuario.

2.3.6 Comparativa de los sistemas

En la siguiente tabla se puede ver una comparativa de las principales características de los sistemas que se acaban de evaluar, así como del sistema descrito en el presente trabajo, al que se ha llamado RFM (*RFID Food Management*):

Sistema	RFID	Auto inventario	Multiplataforma	Información nutricional	Recomendación recetas	Arquitectura Centralizada	Despensa
2.3.1	No	No	Sí	No	No	Sí	No
2.3.2	No	No	Sí	Sí	Sí	Sí	No
2.3.3	No	No	No	Sí	No	No	No
2.3.4	Sí	No	Sí	Sí	Sí	Sí	No
2.3.5	Sí	Sí	Sí	Sí	Sí	No	No
RFM	Sí	Sí	Sí	Sí	Sí	Sí	Sí

Tabla 2 Comparativa sistemas de gestión de alimentos

Se observa cómo, para que se pueda realizar un inventariado automático de los alimentos, es necesario emplear la tecnología RFID. Los productos aún utilizan el código de barras, por lo que las soluciones comerciales no incorporan RFID, mientras que los prototipos estudiados sí suelen emplearla.

En cuanto a las opciones que tienen los diferentes sistemas, casi todas dan a sus usuarios información nutricional de los alimentos, y recomiendan recetas a preparar con ellos, aunque todas se orientan a controlar únicamente los alimentos que hay en la nevera, lo que dificulta bastante el llevar un control de todos los productos.

Como ventajas principales del sistema RFM, destaca que gestiona el inventariado de productos de manera automática, tanto de la nevera como de la despensa, algo que no ocurre en el resto de sistemas, que es accesible desde cualquier dispositivo, que tiene un algoritmo de recomendación de recetas muy versátil, ya que puede recomendar recetas en base a muchos condicionantes: “alimentos disponibles”, “tipo de receta” o “intolerancias alimentarias”. Además, no necesita un servidor en el hogar, ya que el *back-end* está alojado en un servidor *web*, lo que abarata costes, y facilita la configuración del sistema por parte del usuario.

La principal desventaja es, que por el momento, solo puede ser un prototipo, ya que depende de que los productos tengan etiquetado RFID.

3. Arquitectura del sistema y aplicación

3.1 Diseño de arquitectura y elección de tecnologías

El sistema de gestión de alimentos que se va a desarrollar, tiene una serie de requisitos, en base a los cuáles se deberá diseñar la arquitectura del sistema, y realizar la elección de tecnologías a emplear:

- Lectura automática de los alimentos existentes en el frigorífico y despensa, envío de esta información a la base de datos del sistema.
- Aplicación multiplataforma desde la que:
    - Consultar los productos disponibles, su fecha de caducidad o de envasado, información nutricional y de alérgenos.
    - Informar sobre qué productos están caducados o próximos a caducar.
    - Consultar una base de datos de recetas, sus instrucciones de preparación, información nutricional y de alérgenos.
    - Consultar qué recetas se pueden preparar en base a los productos disponibles, y otros criterios como tipo de receta.
    - Consultar ofertas de productos.

Para hacer el diseño de la arquitectura habrá que estudiar por una parte el sistema de lectura de alimentos mediante RFID, por otra, la aplicación, a la que se llama “RFM App”. La integración de ambas se estudia en el apartado 5.1.

3.1.1 Arquitectura del sistema RFID

La arquitectura del sistema RFID tiene cuatro componentes:

Lector

Transporta la corriente que viaja por cable a una antena, que radia una señal en el espacio, a una frecuencia determinada, para que otros elementos lo escuchen.

Por otra parte, procesa las respuestas de los *tags*, y transmite y recibe ondas analógicas que transforma en cadenas de bits.

Cada lector es conectado a una o más antenas. Mientras que el lector crea la señal electromagnética, la antena realiza la difusión en su zona de interrogación (campo de radiación).

Además, el lector también se conecta a la red o a una máquina mediante varios tipos de interfaz, como pueden ser RS-232 o *Ethernet*.

En el sistema desarrollado en este trabajo, el frigorífico irá equipado con un lector RFID, encargado de leer la información de los productos que hay en su interior. Adicionalmente, se colocarán una o varias antenas RFID en la despensa, para poder obtener el inventariado completo de los alimentos del usuario.

Etiquetas

Cuando el lector transmite en el espacio, espera una respuesta de otro elemento para mantener la comunicación, en los sistemas RFID es un *tag* o etiqueta quien responde.

Un *tag* RFID está compuesto por tres partes [18]:

- ***Chip* o circuito integrado**, es un minúsculo ordenador que almacena una serie de información. También contienen la lógica de lo que hay que hacer para responder a un lector.
- **Antena**, permite al *chip* recibir la energía y comunicación procedente del lector, para emitir la suya y poder intercambiar flujos de datos entre ellos.
- **Sustrato**, es comúnmente de plástico. Tanto la antena como el *chip* están unidos al sustrato, encargado de mantener todas las partes de la etiqueta juntas.

Cada alimento llevará un *tag* RFID pasivo, por lo que no necesitan alimentación externa. Este *tag* sustituye al código de barras, en nuestro caso particular dispone de 768 bytes que contienen información detallada de cada producto: nombre, fecha de caducidad/envasado, identificador del alimento, información nutricional, e información de alérgenos.

Ordenador/Middleware

Dispositivo situado entre el *hardware* RFID y las aplicaciones *software* del cliente, tales como un sistema de gestión de inventario. Su función es la de gestionar todo el sistema RFID a nivel de *hardware*, recibir la totalidad de la señales de los *tags* y filtrar la información, para transmitir solamente información útil a los sistemas empresariales.

También se puede tener un *software* diseñado expresamente para una aplicación concreta, que lo único que haga es transmitir la información recogida por los lectores a la aplicación correspondiente.

La información obtenida por los lectores RFID del sistema RFM, se envía a un servidor, gracias a que incorporan un módulo *WiFi*, (módulo *Ethernet* en el prototipo). Este módulo va integrado tanto en el frigorífico, como en los lectores situados en la despensa.

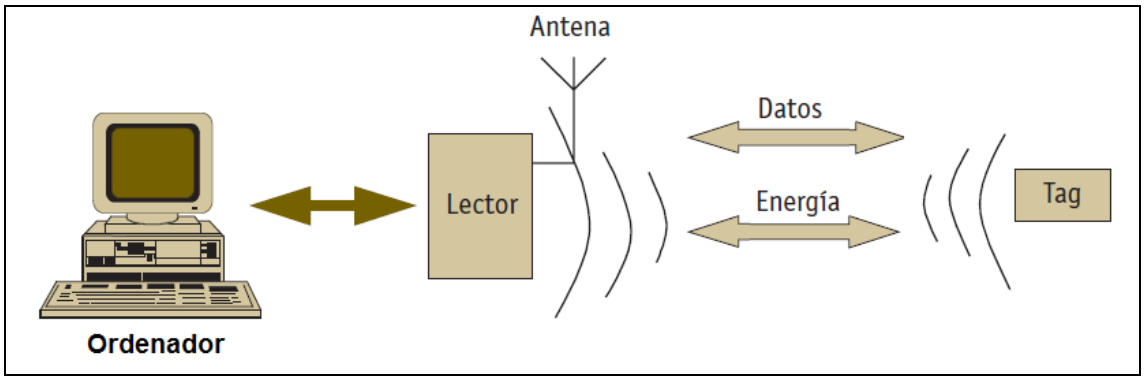

Figura 8 Funcionamiento de un sistema RFID

Servidor y base de datos

De manera adicional a los componentes básicos de cualquier sistema RFID enumerados anteriormente, nuestro sistema cuenta con un servidor y una base de datos.

El fabricante del sistema RFM dispondrá de un servidor al que llega la información de los productos leídos por los lectores RFID, que será almacenada posteriormente en una base de datos.

La aplicación RFM App necesitará acceder a esta base de datos para su funcionamiento. Tanto los servidores como la base de datos están centralizados en un centro de procesado de datos del fabricante, por lo que el usuario no debe disponer de esta infraestructura en su hogar, como si ocurre en otros sistemas.

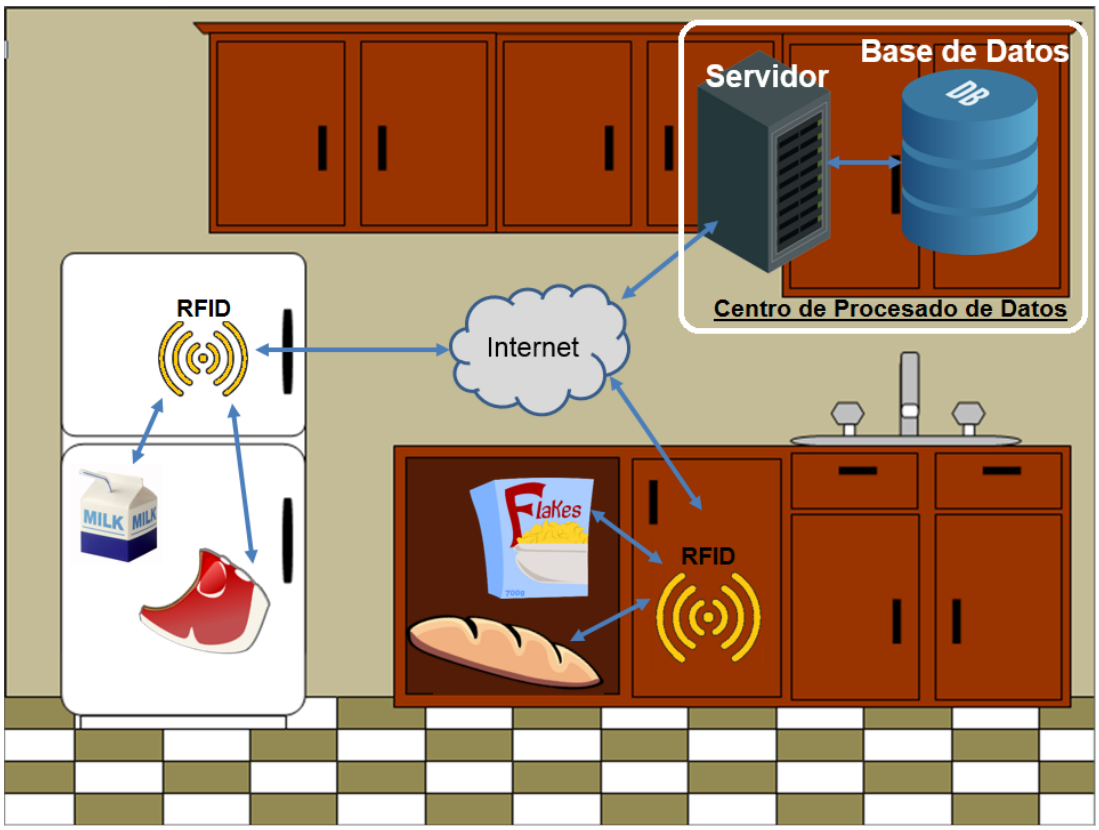

Figura 9 Arquitectura del sistema RFM

En el apartado 4.1 se explica en detalle qué *hardware* RFID se ha elegido para hacer las pruebas de funcionamiento del sistema.

3.1.2 Arquitectura y elección de tecnologías para RFM App

Para desarrollar la aplicación RFM App, que permita al usuario gestionar el sistema RFM, se parte de la premisa de que esta debe ser multiplataforma, esto es, que la aplicación funcione tanto en ordenadores personales como en dispositivos móviles, y en los diferentes sistemas operativos de los mismos.

Una alternativa para conseguirlo, es crear una aplicación para cada sistema operativo, o al menos para los principales, por lo que se habrían de desarrollar aplicaciones para los sistemas operativos: *Windows*, *Mac OS*, *Linux, Android*, *iOS* y *Windows Phone*.

Este enfoque implica un gran esfuerzo, pues, en definitiva, habría que desarrollar y mantener seis aplicaciones, empleando diferentes lenguajes de programación y entornos de desarrollo, lo que no es nada eficiente.

La alternativa elegida para este trabajo es la de desarrollar una aplicación *web*, es decir, una aplicación que se ejecuta en un navegador *web*. Las ventajas que aporta una aplicación *web* son numerosas [19]:

- **Compatibilidad**, es suficiente con tener un navegador actualizado para hacer uso de ella.
- **Almacenamiento**, no es necesario instalar la aplicación, por lo que se ahorra espacio en el disco duro del usuario.
- **Multiplataforma**, funciona en cualquier sistema operativo.
- **Actualización**, no es necesario instalar actualizaciones de la aplicación, siempre se accede a la última versión de la misma.
- **Mantenimiento**, no hay que mantener varias aplicaciones, solo una.

También existen inconvenientes, como que hace falta tener acceso a *Internet* para que funcionen, o que tienen menos funcionalidades que las aplicaciones de escritorio.

Las aplicaciones *web* tienen una arquitectura cliente-servidor, que usa el navegador *web* como cliente.

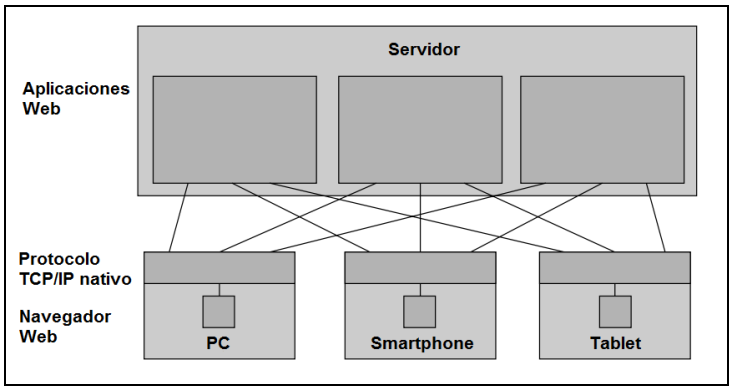

Figura 10 Modelo cliente-servidor en una aplicación web [20]

El navegador envía solicitudes a un servidor, y el servidor genera una respuesta, que es devuelta al navegador. La principal diferencia con el modelo cliente-servidor tradicional, es que emplea un programa cliente común, el navegador *web* [20].

Hasta hace unos años, las aplicaciones *web* se ejecutaban en el servidor, que contenía toda la lógica de la aplicación, escrita en algún lenguaje de servidor como *php* o *ruby*. El servidor respondía a las peticiones de los clientes, y les devolvía una página *web* que el navegador se encargaba de representar.

Las aplicaciones *web* que siguen este enfoque, se conocen como MPA (*multi page web applications*), en ellas, cada vez que la aplicación necesita mostrar información, o mandarla al servidor, tiene que solicitar una nueva página al servidor, para renderizarla posteriormente en el navegador.

En los últimos años, los servidores tienen cada vez una lógica más sencilla, y delegan la mayor parte de la aplicación al cliente.

De este nuevo enfoque surgen las SPA *(Single Page Application)*, aplicaciones *web* contenidas en una única página, que incluye todo el código necesario para ejecutar la aplicación. Una vez es descargada esta página, el navegador dispone de todo los necesario para pedir recursos al servidor, cargarlos, y mostrarlos en el navegador.

Las ventajas de una *SPA* frente a las aplicaciones *MPA* son numerosas [21]:

- Carga de páginas más rápida.
- Mejor experiencia de usuario.
- No hay necesidad de escribir código para renderizar las páginas en el servidor.
- Separación del desarrollo de la capa de presentación y de la de datos.

Existen algunas desventajas, como que es más difícil depurar el código, o que el posicionamiento *web* se complica.

En este trabajo, se va a realizar una aplicación *web* y no una página *web,* por lo que el posicionamiento *web* es indiferente, además, la complejidad no va a ser excesiva, por lo que no habrá que depurar demasiado código.

En vista de las ventajas que aportan las SPA, la aplicación RFM App, será una aplicación *web* de este tipo.

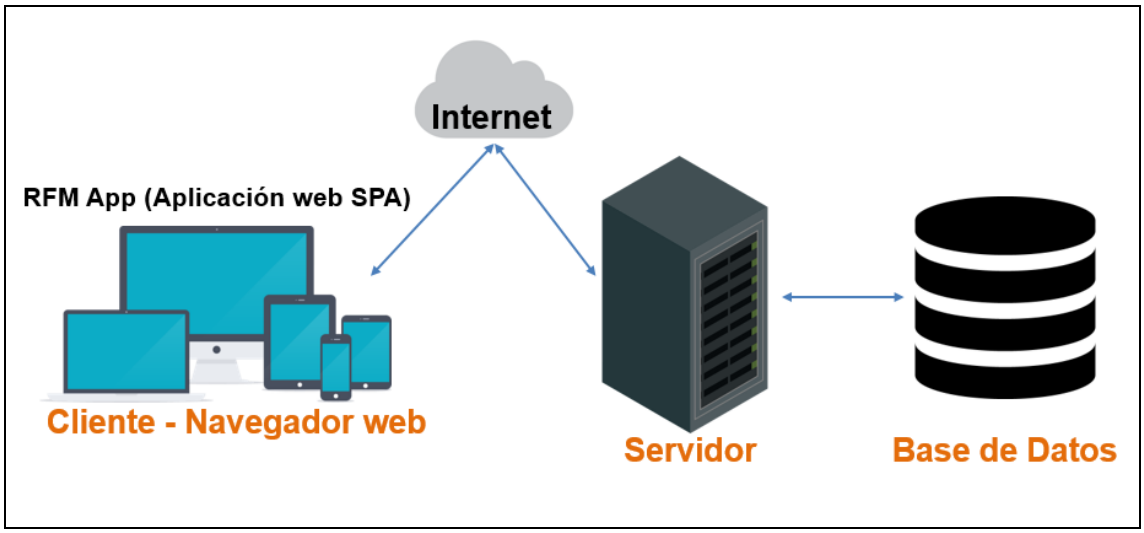

Figura 11 Arquitectura de la aplicación RFM App

Una vez se tiene clara la arquitectura de la aplicación RFM App, hay que elegir qué tecnologías se van a emplear para el desarrollo de la misma. En toda aplicación *web*, se tienen dos partes [22]:

Front-end

Es la capa de presentación con la que interactúa el usuario, lo que ve cuando accede a una aplicación *web*, su estructura, colores, animaciones y efectos. Es responsable de recolectar los datos de entrada del usuario, para almacenarlos posteriormente.

Para desarrollar el *front*-end de una página o aplicación *web*, hay que determinar su estructura, su estilo, y dotarla de lógica. Haciendo un símil con el cuerpo humano, la estructura sería el esqueleto, el estilo correspondería a nuestra vestimenta o color de ojos, y la lógica a nuestro cerebro.

Las tecnologías empleadas para llevar a cabo estas tareas están muy estandarizadas, y son compatibles con cualquier navegador moderno:

- **HTML** (*Hypertext Markup Language*), se trata de un lenguaje de marcado empleado en la elaboración de páginas *web*. Define la estructura, el esqueleto de la página *web*, y su contenido: texto, imágenes, vídeos, gráficos etc. Es el estándar adoptado por todos los navegadores *web*. Un documento HTML consiste en un árbol de elementos. Cada elemento tiene una etiqueta de apertura y cierre (en algunos casos se puede omitir una de ellas). A los elementos se les pueden dar atributos, que se colocan dentro de la etiqueta de apertura. Una vez se ha definido un documento HTML, el navegador lo interpreta y lo convierte en un árbol DOM (*Document Object Model*), que es una representación en memoria del documento HTML. La versión actual del estándar es **HTML5**, que será la empleada en este trabajo. Tiene como principales novedades respecto a sus predecesoras, la posibilidad de emplear etiquetas para gestionar elementos multimedia, y la capacidad para trabajar con gestos propios de dispositivos móviles [23].

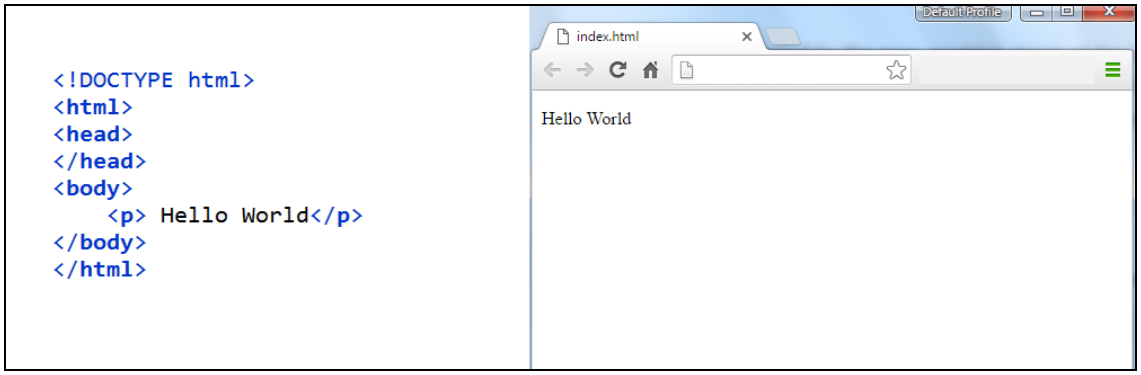

Figura 12 Documento HTML y su visualización en un navegador

- **CSS** (*Cascading Style Sheets*), es un lenguaje empleado para definir el estilo de un documento escrito en HTML. Pretende separar la estructura de un documento de su presentación, creando hojas de estilo. El estilo se puede definir tanto en el propio documento HTML como en un documento independiente. La versión actual de este lenguaje es **CSS3**, y es la que se va a emplear. CSS3 está dividido en módulos: selectores, fondos y bordes, efectos de texto, transformaciones 2D/3D, animaciones etc. Funciona en base a reglas, que se componen de un selector y de un bloque de declaración. El selector apunta al elemento del documento HTML al que se quiere dar un estilo. El bloque de declaración indica un conjunto de parejas propiedad-valor separadas por punto y coma. En la figura 13 se puede ver cómo se cambia el color y el tamaño de la fuente de un elemento h1 dentro de un documento HTML [24].

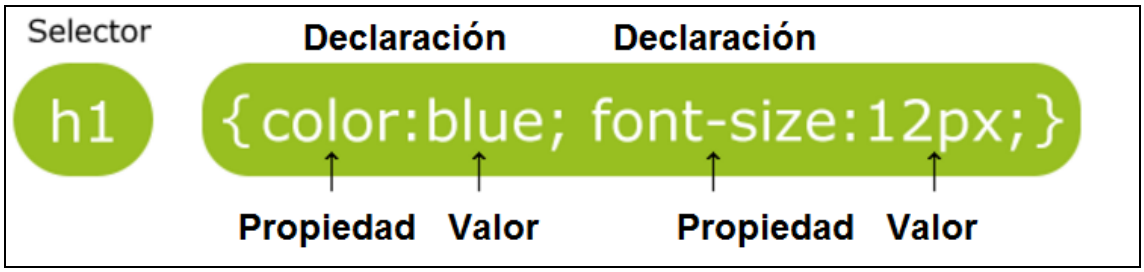

Figura 13 Ejemplo de regla CSS para definir el color y tamaño de fuente [24]

- ***JavaScript***, nace en 1997, es el lenguaje de programación empleado en la *web*. Es un lenguaje interpretado, orientado a objetos, y débilmente tipado. Con este lenguaje se programa toda la lógica de una página o aplicación *web*. Permite mejorar la interfaz de usuario y realizar páginas *web* dinámicas. Cualquier navegador moderno es capaz de interpretar código *JavaScript.* Suele funcionar en el lado del cliente, aunque también existen versiones que corren en el lado del servidor, sin necesidad de navegador *web.* Su versión actual es la 6, aunque por no ser compatibles todas sus nuevas funcionalidades con navegadores antiguos, se empleará para este trabajo la versión 5. [25]

Con el objetivo de estructurar de manera modular el código de una aplicación *web* y que este sea mantenible y reutilizable, han aparecido en los últimos años multitud de librerías y *frameworks* que simplifican esta labor [26].

En este trabajo se emplea ***AngularJS*** [27], se trata de un proyecto de código abierto mantenido por *Google*. Este *framework* permite escribir aplicaciones *web* en el lado del cliente de manera estructurada, y facilita la comunicación con el servidor.

AngularJS sigue el patrón de diseño de *software* MVC (*Model View Controller*). Este patrón permite aislar la capa de interfaz de usuario de la lógica de la aplicación [28].

Por último, y con el objetivo de que la aplicación sea visualmente atractiva, y que su interfaz se adapte a cualquier dispositivo, lo que se conoce como diseño "*responsive*", se hace uso de ***Bootstrap*** [29]**.**

Bootstrap es un *framework* de código abierto creado por *Twitter*. Se emplea para el diseño de sitios y aplicaciones *web*. Consiste en una serie de plantillas de diseño con tipografías, iconos, formularios, botones, menús, y otros muchos elementos basados en HTML y CSS.

Back-end

Es la capa de acceso a datos, consiste en un servidor, una aplicación y una base de datos. No es visible para el usuario, procesa la entrada de datos que se produce desde el *front-end*, resuelve las peticiones de los usuarios, y gestiona las operaciones que se producen sobre la base de datos.

En la aplicación RFM App es necesario un *back-end* que permita gestionar usuarios, productos, recetas, y ofertas. Debe proporcionar un sistema de acceso a estos datos, a través de una *API REST* o *RESTful.*

Cuando hablamos de *RESTful* nos referimos a un servicio *web* que implementa una arquitectura REST *(Representational State Transfer*). Esta arquitectura aglomera unas reglas de diseño para ofrecer recursos a través de una URI *(Uniform Resource Identifier)*, que es una cadena de caracteres con la información necesaria para realizar una determinada operación sobre los recursos [30].

Las operaciones que debe proporcionar el servicio *RESTful* son aquellas usadas más comúnmente en el protocolo HTTP *(Hypertext Transfer Protocol)*, que permite las transferencias de información en la *web* [31]:

- ***GET***, obtiene la información indicada por una *URI*. Se emplea para que el *front-end* solicite un recurso a una base de datos.
- ***PUT***, almacena un recurso en una determinada ruta, tanto el recurso como la ruta se indican en una *URI*. Si ya existe un recurso en la ruta seleccionada, se sobrescribe, si no existe, se crea.
- ***POST****,* almacena un recurso en una ruta, ambos indicados en una URI, y devuelve la url que identifica al recurso.
- ***DELETE****,* borra un recurso, identificado mediante una URI.

Además de estas operaciones, nuestro *back-end* debe controlar qué operaciones se pueden realizar en cada momento, y quién puede hacerlas.

Para construir el servicio *REST* de manera sencilla, se ha empleado la herramienta ***deployd*** [32]. Esta herramienta de código abierto permite "diseñar, construir, y escalar servicios *REST* para aplicaciones *web* en minutos, en lugar de días".

Deployd proporciona un modelo usuario con métodos para inicio y cierre de sesión. Cuenta con un panel de control desde el que se configura tanto la base de datos, como la lógica del *back-end*.

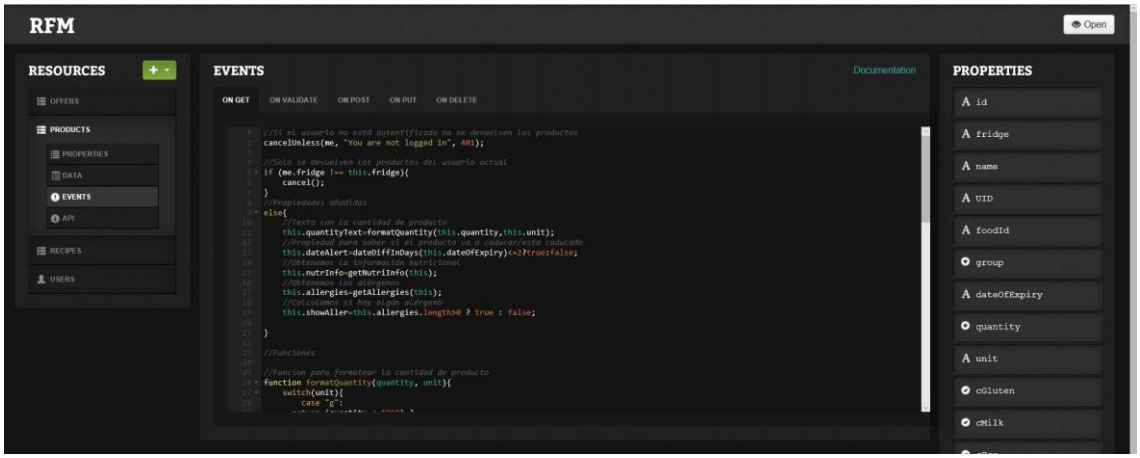

Figura 14 Panel de control de deployd

La base de datos utilizada en *deployd* es ***Mongodb*** [33], se trata de una base de datos NoSQL, o no relacional, de código abierto, que en lugar de usar el modelo tradicional de tablas relacionales, tiene una arquitectura de colecciones y documentos.

Se ha elegido una base de datos no relacional, ya que estas presentan una serie de ventajas sobre las bases de datos relacionales, que son de gran interés para este trabajo [34]. Entre otras ventajas, cabe destacar, que optimizan las consultas en bases de datos con una gran cantidad de registros, incluso si se emplean máquinas con pocos recursos.

Nuestra base de datos debe almacenar los productos de los usuarios de la aplicación, esto supone manejar millones de registros, lo que tendría un coste enorme si se emplearan las tradicionales bases de datos relacionales.

Por último, *deployd* está construido empleando ***NodeJS*** [35], que es un *framework* de código abierto, que permite trabajar con *JavaScript* en el lado del servidor. Es el encargado de realizar todas las operaciones de entrada y salida de datos en nuestra base de datos, y de dotar de lógica al *back-end*.

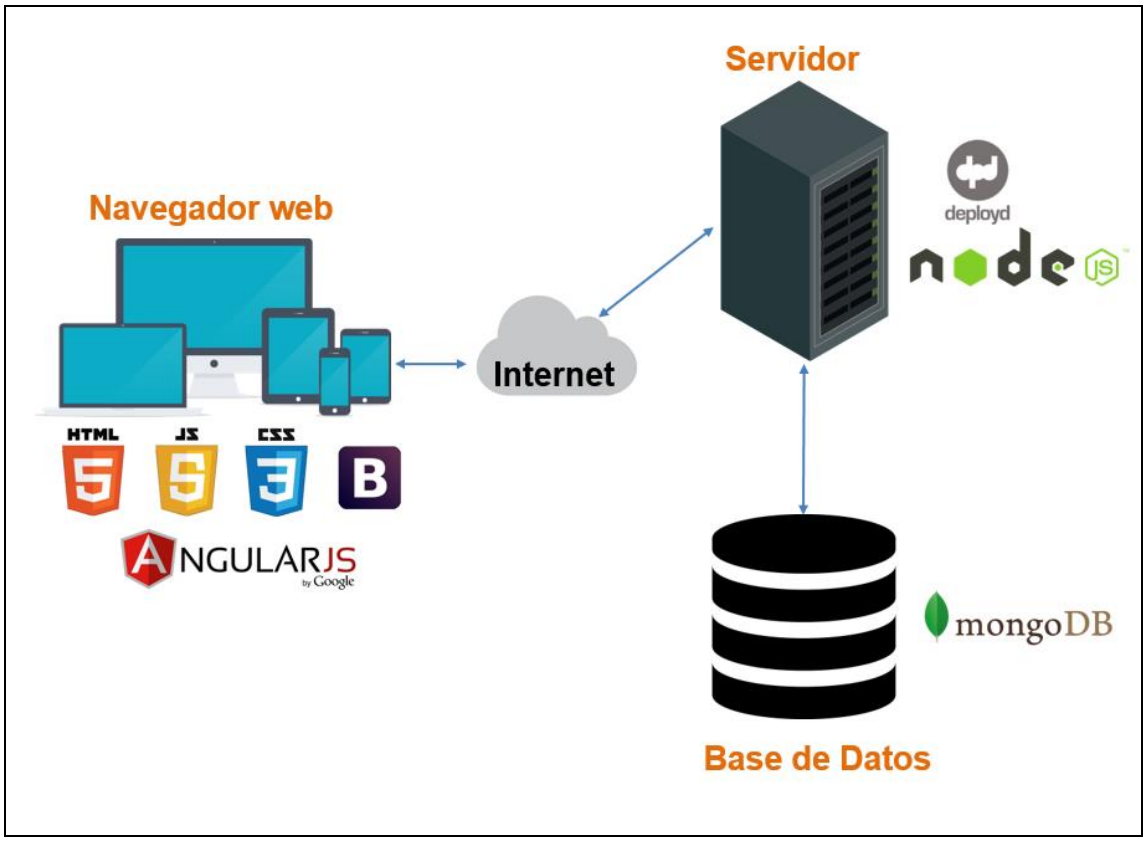

Figura 15 RFM App, arquitectura y tecnologías empleadas

3.2 Implementación de RFM App

3.2.1 Implementación de la base de datos

Para empezar a trabajar con *deployd*, se descarga el archivo de instalación desde su *web* [36], se instala, y desde la línea de comandos de nuestro sistema operativo, nos situamos en el directorio donde vamos a crear nuestra aplicación y ejecutamos el comando:

```
dpd create RFM
```

Con esto habremos creado una aplicación llamada RFM. A continuación, accedemos al *dashboard* o panel de control, de la aplicación, ejecutando el comando:

```
dpd dashboard
```

Deployd crea de forma automática una base de datos de *Mongodb*, que configuramos desde el panel de control.

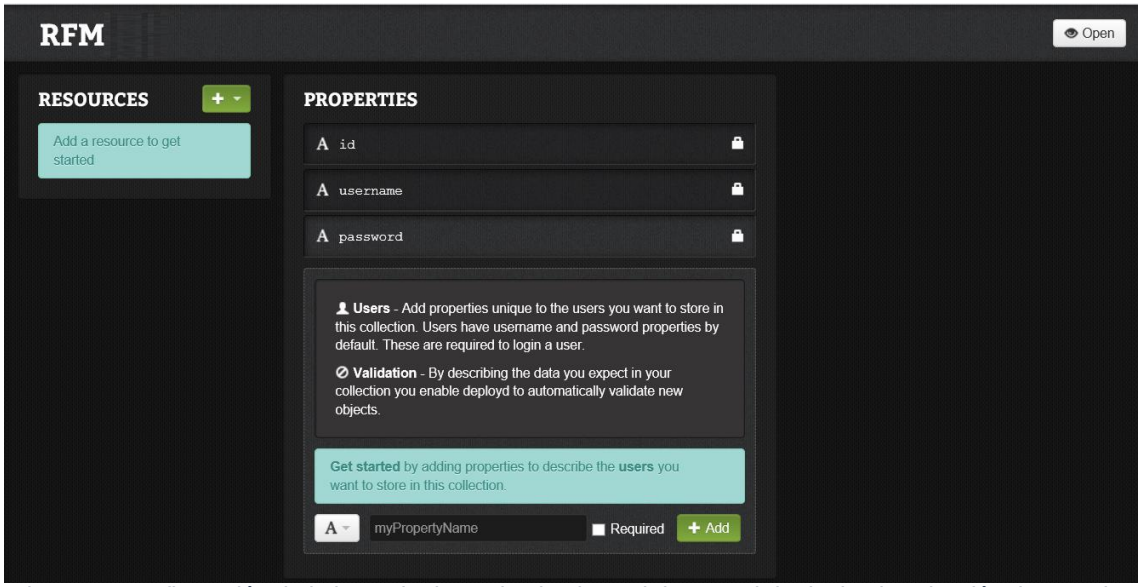

Figura 16 Configuración de la base de datos desde el panel de control de deployd. Colección de usuarios creada por defecto

Mongodb es una base de datos no relacional, trabaja con **colecciones** y **documentos BSON** (*Binary* JSON), el equivalente a las tablas y registros de las bases de datos relacionales.

Un documento *BSON* es una representación binaria de un documento JSON (*JavaScript Object Notation*), un formato de texto plano para definir datos de forma estructurada. Hay una gran cantidad de tipos de datos admitidos para los valores almacenados en un documento [37], básicamente: números, cadenas, objetos, *booleanos* y *arrays*.

```
{
    "Título":"Pulp Fiction",
    "Director":"Quentin Tarantino",
    "Año":1984
}
```

Figura 17 Documento JSON con la información sobre una película

La decisión clave en el modelado de una base de datos *Mongodb*, es decidir la estructura de los documentos, y cómo se van a relacionar. Existen dos enfoques para representar las relaciones en nuestra aplicación [38]:

- **Modelado de datos normalizado mediante referencias**, describe relaciones que usan referencias entre documentos. Es el modelo que se emplea en las bases de datos tradicionales. Es adecuado cuando existen relaciones complejas entre diferentes colecciones, con estructuras muy jerarquizadas.
- **Modelado de datos mediante documentos incrustados**, este modelo de datos permite a una aplicación almacenar distintos fragmentos de información que guardan relación entre sí en un mismo documento de la base de datos. Cada uno de estos fragmentos, es un documento incrustado en el documento principal. Es adecuado cuando las

relaciones son de uno a muchos, donde los elementos hijo siempre se consultan en el contexto del elemento padre, permite hacer lecturas de manera más rápida, ya que no se realizan operaciones *join* (uniones) entre las diferentes bases de datos.

Figura 18 A la izquierda relación mediante referencia, a la derecha mediante documentos incrustados [39]

En esta aplicación se va a modelar la base de datos empleando la segunda opción, se considera la más apropiada, pues las colecciones no tienen relaciones complejas entre ellas, sino relaciones de uno a muchos, en las que los elementos hijo no se van a consultar individualmente.

La base de datos va a tener cuatro colecciones:

Users

Colección para almacenar los datos de los usuarios, un usuario se define mediante las siguientes propiedades:

Nombre	**Tipo**	**Descripción.**
Id	*string*	Identificador único generado por *Mongodb.*
username	*string*	Nombre de usuario para acceder a la aplicación.
password	*string*	Contraseña del usuario.
firstName	*string*	Nombre del usuario.
secondName	*string*	Apellidos del usuario.
rfm	*string*	Identificador del sistema RFM del usuario.
cGluten	*boolean*	Indica si el usuario es alérgico al gluten.
cEgg	*boolean*	Indica si el usuario es alérgico al huevo.
cMilk	*boolean*	Indica si el usuario es alérgico a los lácteos.
cSoy	*boolean*	Indica si el usuario es alérgico a la soja.
cPeanuts	*boolean*	Indica si el usuario es alérgico a los cacahuetes.
cNut	*boolean*	Indica si el usuario es alérgico a los frutos con cáscara.
cFish	*boolean*	Indica si el usuario es alérgico al pescado.
cShellFish	*boolean*	Indica si el usuario es alérgico al marisco.
cCelery	*boolean*	Indica si el usuario es alérgico al apio.
cSesame	*boolean*	Indica si el usuario es alérgico al sésamo.
cMustard	*boolean*	Indica si el usuario es alérgico a la mostaza.

Tabla 3 Propiedades de la colección Users

Products

Colección para almacenar los datos de los productos de los usuarios, un producto queda definido por las propiedades:

Nombre	**Tipo**	**Descripción**
id	*string*	Identificador único generado por *Mongodb.*
rfm	*string*	Identificador del sistema RFM que ha leído el producto.
antenna	*string*	Indica qué antena ha leído el producto.
name	*string*	Nombre del producto.
UID	*string*	Identificador único del producto grabado en el *tag* RFID.
foodId	*string*	Identificador de alimento.
group	*string*	Grupo al que pertenece el alimento.
dateOfExpiry	*string*	Fecha de caducidad o de envasado del producto.
fresh	*boolean*	Indica si el producto es fresco (envasado en el supermercado)
quantity	*string*	Cantidad de producto.
unit	*string*	Unidades en las que se indica la cantidad de producto.
cGluten	*boolean*	Indica si el producto contiene gluten.
cEgg	*boolean*	Indica si el producto contiene huevo.
cMilk	*boolean*	Indica si el producto contiene lácteos.
cSoy	*boolean*	Indica si el producto contiene soja.
cPeanuts	*boolean*	Indica si el producto contiene cacahuetes.
cNut	*boolean*	Indica si el producto contiene frutos con cáscara.
cFish	*boolean*	Indica si el producto contiene pescado.
cShellFish	*boolean*	Indica si el producto contiene marisco.
cCelery	*boolean*	Indica si el producto contiene apio
cSesame	*boolean*	Indica si el producto contiene sésamo
cMustard	*boolean*	Indica si el producto contiene mostaza
energy	*number*	Kcals por cada 100g de producto.
fat	*number*	Grasa en 100g de producto.
carbo	*number*	Carbohidratos en 100g de producto.
protein	*number*	Proteínas en 100g de producto.

Tabla 4 Propiedades de la colección Products

Se hace una diferenciación entre producto y alimento. Un producto es un alimento de una determinada marca, por ejemplo, un litro de leche entera de la marca “Pascual”. En este caso, el alimento sería leche, que tiene un identificador (*foodId)*, un grupo (*group*), e información nutricional y de alérgenos, comunes a todos los cartones de leche de las distintas marcas. Un producto que tenga su propiedad *fresh* como verdadera, será un producto que ha sido envasado en el propio supermercado, y su fecha de caducidad no será interpretada como tal, sino como la fecha de dicho envasado.

La información de los alimentos que se van a utilizar para probar el sistema, se ha extraído de la base de datos de alimentos del USDA *(United States Department of Agriculture*), departamento de agricultura de los Estados Unidos

[40]. Esta base de datos tiene cerca de 9000 referencias de alimentos, se ha utilizado la misma nomenclatura que se emplea en ella para el identificador (*foodId*) y grupo al que pertenecen los alimentos, además se ha extraído su información nutricional.

En la siguiente figura se puede ver cómo se almacena un producto:


```
{
    "id":"231880f5d2d308f4",
    "rfm":"0001",
    "antenna":"01",
    "name":"Parmesano",
    "UID":"FFFFFF",
    "foodId":"01032",
    "group":"01",
    "fresh":false,
    "dateOfExpiry":"25/02/2016",
    "quantity":250,
    "unit":"g",
    "cGluten":false,
    "cMilk":true,
    "cEgg":false,
    "cShellFish":false,
    "cFish":false,
    "cPeanuts":false,
    "fresh":false,
    "cSoy":false,
    "cNut":false,
    "cCelery":false,
    "cSesame":false,
    "cMustard":false,
    "energy":420,
    "fat":27.84,
    "carbo":13.91,
    "protein":28.42
}
```


Figura 19 Documento de la colección Products

Recipes

Colección para almacenar los datos de las recetas. El listado completo de propiedades que definen una receta es el siguiente:

Nombre	**Tipo**	**Descripción**
id	*string*	Identificador único generado por *Mongodb.*
name	*string*	Nombre de la receta.
type	*string*	Tipo de receta.
instructions	*string*	Instrucciones para preparar la receta.
photo	*string*	Nombre de la imagen asociada a la receta.
cGluten	*boolean*	Indica si la receta contiene gluten.
cEgg	*boolean*	Indica si la receta contiene huevo.
cMilk	*boolean*	Indica si la receta contiene lácteos.
cSoy	*boolean*	Indica si la receta contiene soja.
cPeanuts	*boolean*	Indica si la receta contiene cacahuetes.

cNut	*boolean*	Indica si la receta contiene frutos con cáscara.
cFish	*boolean*	Indica si la receta contiene pescado.
cShellFish	*boolean*	Indica si la receta contiene marisco.
cCelery	*boolean*	Indica si la receta contiene apio.
cSesame	*boolean*	Indica si la receta contiene sésamo.
cMustard	*boolean*	Indica si la receta contiene mostaza.
energy	*number*	Kcals por cada ración.
fat	*number*	Grasa en cada ración.
carbo	*number*	Carbohidratos cada ración.
protein	*number*	Proteínas en cada ración.
people	*number*	Número raciones de la receta.
ingredients	*array*	Array de objetos que contiene los ingredientes.

Tabla 5 Propiedades de la colección Recipes

En la figura 20 se puede ver un ejemplo de cómo se almacena una receta en nuestra base de datos:


```
{
    "name": "Papillote de merluza",
    "type": "Pescados",
    "instructions": "Limpia el calabacín[...]",
    "photo": "Papillote merluza",
    "cGluten": false,
    "cMilk": true,
    "cSoy": false,
    "cEgg": false,
    "cPeanuts": false,
    "cNut": false,
    "cFish": true,
    "cShellFish": false,
    "cCelery": false,
    "cSesame": false,
    "cMustard": false,
    "people": 4,
    "ingredients": [
        {"foodId": "15130","name": "Merluza","unit": "g","quantity": "400","weight":8},
        {"foodId": "11246","name": "Puerros","unit": "g","quantity": "80","weight":4},
        {"foodId": "11124","name": "Zanahorias","unit": "g","quantity": "120","weight":4},
        {"foodId": "11477","name": "Calabacín","unit": "g","quantity": "100","weight":4},
        {"foodId": "14106","name": "Vino blanco","unit": "ml","quantity": "150","weight":1},
        {"foodId": "02030","name": "Pimienta","unit": "","quantity": "","weight":1},
        {"foodId": "04053","name": "Aceite de Oliva","unit": "","quantity": "","weight":1},
        {"foodId": "02047","name": "Sal","unit": "","quantity": "","weight":1}
    ],
    "energy": 180,
    "fat": 5,
    "carbo": 20,
    "protein": 35
}
```


Figura 20 Documento de la colección Recipes

Para relacionar una receta con los alimentos que la componen, se incrusta un *array* de objetos, cada objeto representa un alimento.

Cada ingrediente tiene un campo *foodId* que especifica qué alimento es, además se indica su nombre, cantidad, y unidades en las que se expresa esta cantidad. Por último, tiene una propiedad para determinar su importancia en la preparación de la receta, esto se verá en el apartado 5.3.2.

Offers

Colección para almacenar los datos de las ofertas que los supermercados ofrecen a los usuarios de la herramienta. Una oferta tiene las siguientes propiedades:

Nombre	Tipo	Descripción
id	*string*	Identificador único generado por *Mongodb.*
foodId	*string*	Identificador de alimento.
url	*string*	Enlace a la oferta.
text	*string*	Texto descriptivo de la oferta.
price	*number*	Precio del producto ofertado.
unit	*string*	Unidad en la que se oferta el producto.

Tabla 6 Propiedades de la colección Offers

Una vez que se han definido estas cuatro colecciones, se crean desde el *dashboard* de *deployd* y se hace una carga inicial de datos para poder realizar pruebas.

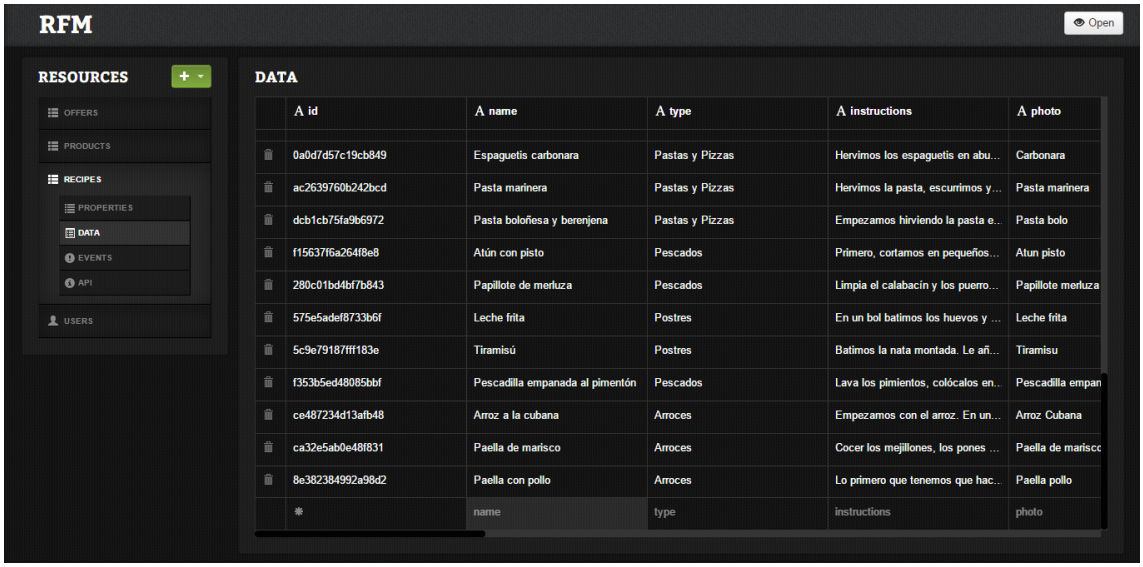

Figura 21 Dashboard de deployd, se pueden ver las cuatro colecciones y una carga inicial de datos.

3.2.2 Implementación del *front-end*

En este apartado se va a explicar cómo se ha implementado el *front-end* de la aplicación. Se ha construido una *Single Page Application* con *AngularJS*, el código se ha estructurado de la siguiente manera:

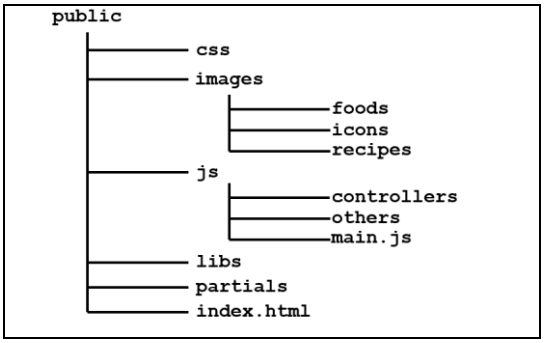

Figura 22 Estructuración del código de RFM App

El código completo de la aplicación está disponible en un repositorio de *Github* [41] accesible desde aquí [42]. En este capítulo solo se mostrará parte del código, a modo de ejemplo.

En la figura 22 podemos observar que todo el código está dentro de una carpeta llamada *public*, dentro hay otras cinco carpetas: *css, images*, *js*, *libs,* y *partials*. Además, hay un archivo HTML, *index.html*. Se va a detallar el cometido de cada una de estas carpetas y archivos:

Index.html

Este archivo HTML define la página principal de nuestra aplicación *web*, es lo primero que se cargará cuando accedamos a ella. Desde aquí se inicializa nuestra aplicación. Esto se consigue con la directiva *ng-app* de *AngularJS*, una directiva añade funcionalidades adicionales a las propiedades y atributos de un HTML [43].

```
1  <!DOCTYPE html>
2  <html ng-app="RFM">
```

Figura 23 Primeras dos líneas de index.html, inicialización de la aplicación mediante una directiva

Además, desde aquí referenciamos a las diferentes librerías, tipografías, y archivos *css* y *js* que vamos a necesitar para el funcionamiento de la aplicación.

```
3  <head>
7      <link rel="stylesheet" type="text/css" href="/css/nav.css">
8      <link rel="stylesheet" type="text/css" href="/css/main.css">
11 </head>
12 <body>
13     <div class="main">
14         <div ng-view>
15         </div>
16     </div>
17     <script type="text/javascript" src="/dpd.js"></script>
18     <script type="text/javascript" src="/libs/angular.min.js"></script>
```

Figura 24 Algunas líneas de index.html, se referencian archivos css y js

Partials

Esta carpeta contiene todos los archivos HTML que se van a ir cargando a medida que navegamos por las distintas vistas de la aplicación: registro, entrada, productos, recetas, ofertas y ayuda. Por último, se define la barra de navegación, que nos ayudará a desplazarnos de una sección a otra.

```
<div ng-include="'partials/navbar.html'"></div>
<div>
  <ul class="list-group">
    <li class="list-group-item recipeList" ng-repeat="offer in offersList |
    orderBy:'+foodId'">
      <div class="container container-fluid">
        <div class="row">
          <div class="col-lg-2 col-md-2 col-sm-3 col-xs-5">
            <img ng-src="images/foods/{{offer.foodId}}.PNG" class="img-responsive
            centerPic noPadTop" alt="" height="auto" width="60"> </img>
          </div>
          <div class="col-lg-2 col-md-2 col-sm-2 col-xs-7">
            <h4>{{offer.offerText}}</h4>
          </div>
          <div class="col-lg-8 col-md-8 col-sm-7 col-xs-12">
            <h4><a ng-href="{{offer.url}}" target="_blank">{{offer.text}}</a></h4>
          </div>
        </div>
    </li>
  </ul>
</div>
```

Figura 25 Archivo offers.html, define la vista de ofertas.

Libs

En esta carpeta se almacena la librería de *AngularJS*, y la de *jquery* [44], esta última nos permite entre otras cosas, manipular con facilidad el árbol *DOM* de un documento HTML. Al resto de librerías se accede desde las referencias que se han escrito en el archivo *index.html*.

Js

Aquí se almacenan todos los archivos *javascript* con los que se va a trabajar. En la carpeta *controllers* se almacenan los controladores de la aplicación. Un controlador maneja la lógica de una determinada vista [45], así pues, tendremos controladores para todos y cada uno de los documentos HTML alojados en la carpeta *partials* (a excepción del de ayuda), y otro para gestionar el panel de detalles de las recetas.

```
//Controlador de ofertas
RFM.controller('OffersController',['$scope', '$http', function($scope, $http){

  $scope.offersList=[];

  //Obtenemos el listado de ofertas
  $http.get('/offers').success(function(data){
    $scope.offersList=data;
  });

}]);
```

Figura 26 Archivo offersController.js, se configuran las consultas a la base de datos para obtener el listado de ofertas.

El archivo *main.js* es desde el que se crea la aplicación *AngularJS*, inicializada en *index.html*. Desde aquí se configura un mecanismo de *routing* [46], para que la aplicación sepa qué documento HTML se tiene que mostrar en cada momento, y qué controlador se va a hacer cargo de su comportamiento.

```
//Creación del módulo
var RFM=angular.module('RFM', ['ngRoute'])

//Configuración de las rutas
.config(function($routeProvider){
  $routeProvider
  .when('/recipes', {
    templateUrl: 'partials/recipes.html',
    controller: 'RecipesController'
  })
  .when('/products',{
    templateUrl: 'partials/products.html',
    controller: 'ProductsController'
  })
  .when('/offers',{
    templateUrl: 'partials/offers.html',
    controller: 'OffersController'
  })
  .when('/login',{
    templateUrl: 'partials/login.html',
    controller: 'LoginController'
  })
  .when('/register',{
    templateUrl: 'partials/register.html',
    controller: 'RegisterController'
  })
  .when('/help',{
    templateUrl: 'partials/help.html'
  })
  .otherwise({
    redirectTo: '/login'
  });
});
```

Figura 27 Archivo main.js, se crea la aplicación y a continuación se configura el routing

Por último, se tiene una carpeta *others* donde se almacenan los archivos *js* que no son controladores, en este caso existe un archivo para gestionar algunos aspectos de la barra de navegación, y otro para realizar las pruebas del apartado 5.3.

Css

La aplicación hace uso de dos archivos *css*, uno de ellos para la configuración de estilo de la barra de navegación, y otro para el resto de estilos de la aplicación.

Images

Aquí se almacenan todas las imágenes que utiliza la aplicación

Tras explicar cómo está diseñado el *front-end*, se van a mostrar algunas capturas de la aplicación a modo de ejemplo, el objetivo es demostrar que la aplicación se adapta perfectamente a diferentes dispositivos, es decir, tiene un diseño *responsive*, que se ha conseguido gracias al uso de *Bootstrap*. En el anexo 9.1 se adjunta un completo manual desde el que se detalla todo el funcionamiento de la aplicación y sus diferentes opciones, y en el apartado 5.4 se prueba la aplicación.

En primer lugar se muestra cómo se ve el menú de acceso en diferentes dispositivos:

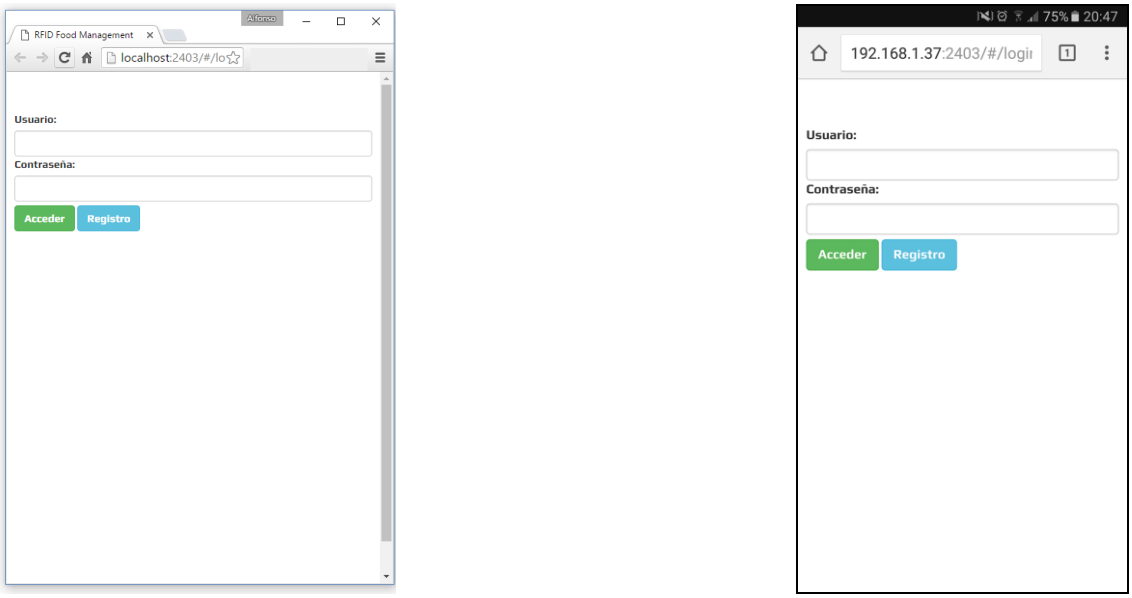

Figura 28 A la izquierda menú de acceso desde un ordenador, a la derecha desde un Smartphone Samsung Galaxy S6

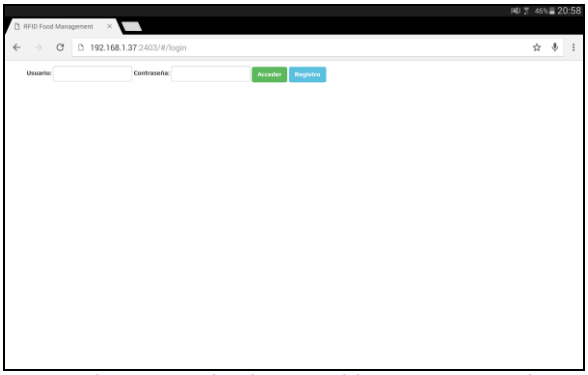

Figura 29 Menu de acceso desde una tablet Samsung Galaxy Note 10.1

Tras acceder a la aplicación, el usuario pasa a ver los productos de los que dispone:

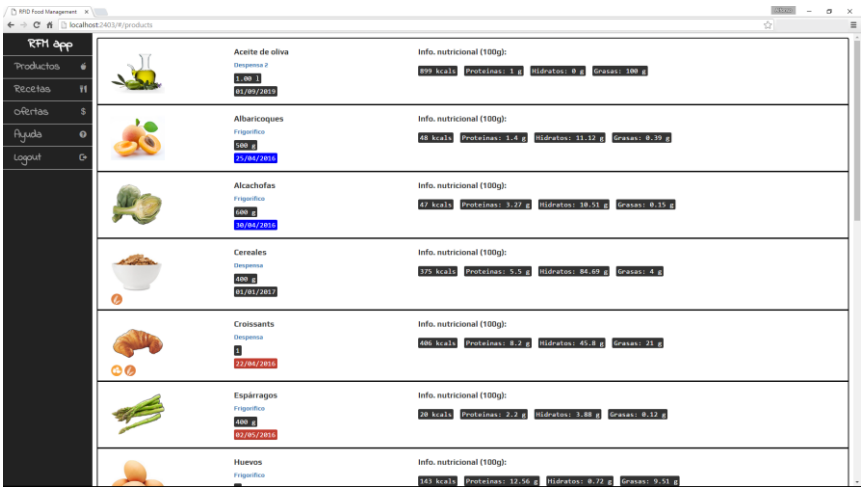

Figura 30 Menú de productos desde un ordenador

Figura 31 A la izquierda, menú de productos en una tablet Samsung Galaxy Note 10.1, a la derecha en un smartphone Samsung Galaxy S6

En las figuras 31 y 32 se puede ver cómo la aplicación se adapta al dispositivo. En la figura 31 el menú lateral muestra texto y un icono asociado a cada elemento del menú. En la *tablet* se muestran solo los iconos del menú, mientras que en el *smartphone* el menú pasa a estar en la esquina superior derecha. Por otra parte, en el *smartphone* la información de cada producto, no cabe en una única fila, por lo que se va apilando verticalmente.

El aspecto del menú de recetas en diferentes dispositivos se puede ver a continuación:

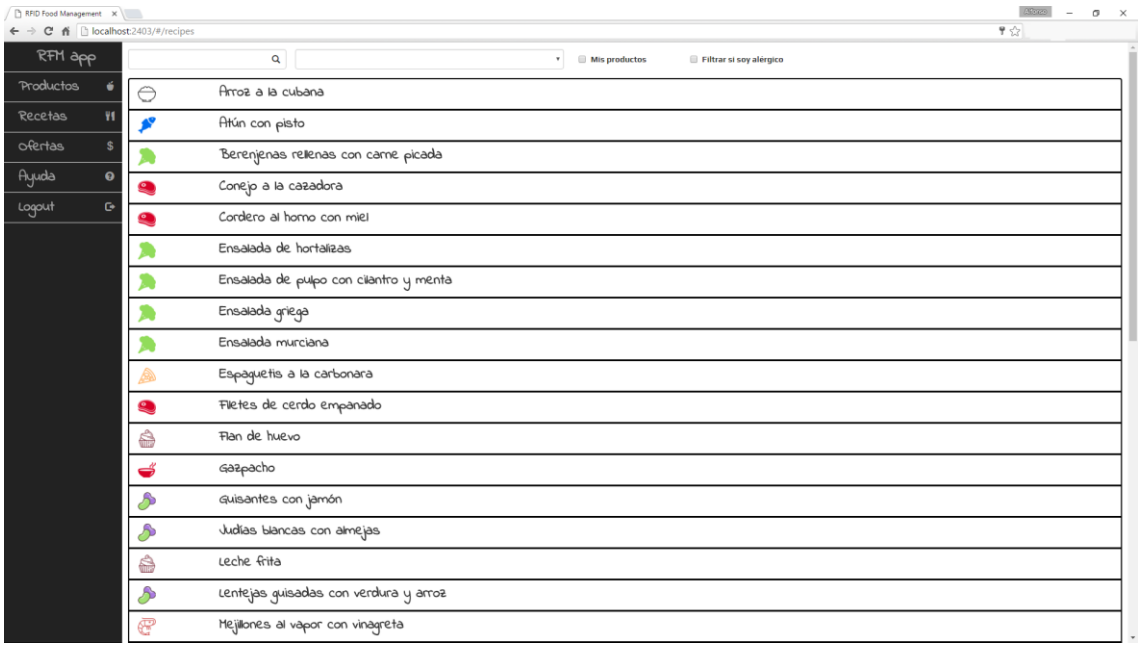

Figura 32 Menú de recetas desde un ordenador

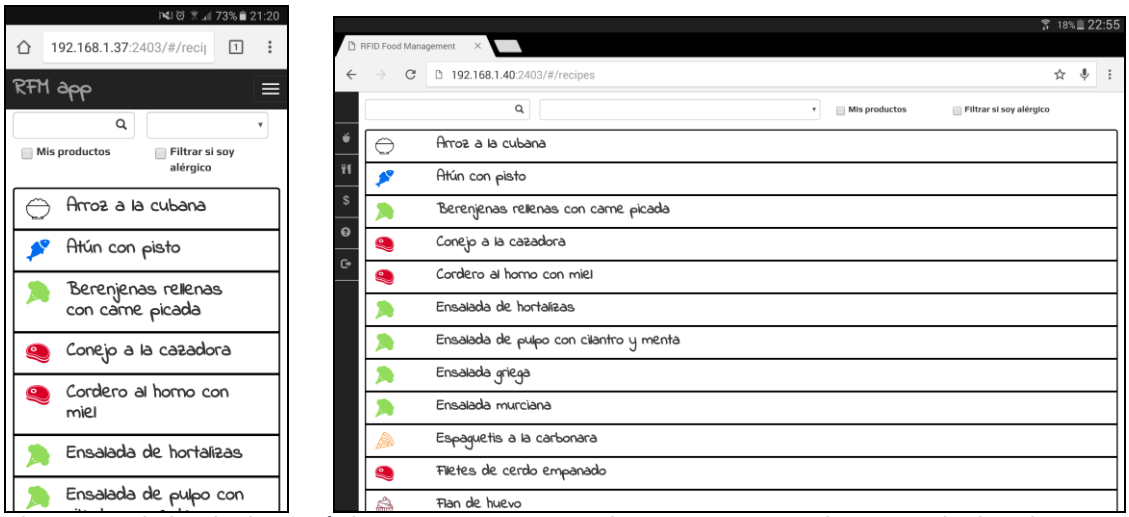

Figura 33 A la izquierda menú de recetas en un smartphone Sasmsung Galaxy S6, a la derecha en una tablet Samsung Galaxy Note 10.1

El menú de recetas es bastante similar en un ordenador y en una *tablet*, salvo por el menú lateral.

La diferencia entre el *smartphone* y los otros dispositivos es más notable, por ejemplo, al estar la pantalla orientada verticalmente, los campos de búsqueda y filtrado de la parte superior no caben en una única fila, por lo que automáticamente pasan a ocupar dos filas.

Se puede concluir que la aplicación adapta perfectamente su interfaz a los diferentes dispositivos.

3.2.3 Implementación del *back-end*

El *back-end* debe ser capaz de gestionar las colecciones de usuarios, productos, ofertas y recetas. Para ello se debe implementar un sistema de acceso a estos datos mediante un servicio *RESTful*. Este servicio, y los métodos necesarios para el acceso de usuarios, se crean de manera sencilla con *deployd*.

Se han creado una serie de operaciones CRUD *(Create Read Update and Delete*) que permitirán crear, leer, actualizar, y eliminar documentos.

API

HTTP | JAVASCRIPT

Access the `/products` collection over http.

Task	Method	Route	Accepts	Returns
Listing data	GET	/products	*Nothing*	An array of objects
Creating an object	POST	/products	A single object	The saved object (or errors)
Getting an object	GET	/products/<id>	*Nothing*	A single object
Updating an object	PUT	/products/<id>	A single object	The saved object (or errors)
Deleting an object	DELETE	/products/<id>	A single object	*Nothing*

Figura 34 Operaciones que se pueden realizar sobre la colección Products

Estas operaciones son invocadas desde el *front-end*, en la Figura 35 se muestra cómo obtenemos el listado completo de productos del usuario.

```
12    //Obtenemos el listado de productos
13    $http.get('/products').success(function(data){
```

Figura 35 llamada a la operación GET de la colección products desde el archivo productsController.js del front-end

Algunas de las operaciones descritas en la tabla de la Figura 34 solo se deben ejecutar bajo determinadas condiciones. Estas validaciones se configuran desde el *dashboard* de *deployd*. Cada colección tiene una serie de eventos que deberemos programar en base a nuestras necesidades: *"ON GET"*, "*ON VALIDATE*", "*ON PUT*", y "*ON DELETE*". Estos eventos se activan al consultar un documento, al validarlo, al añadirlo, y al borrarlo respectivamente.

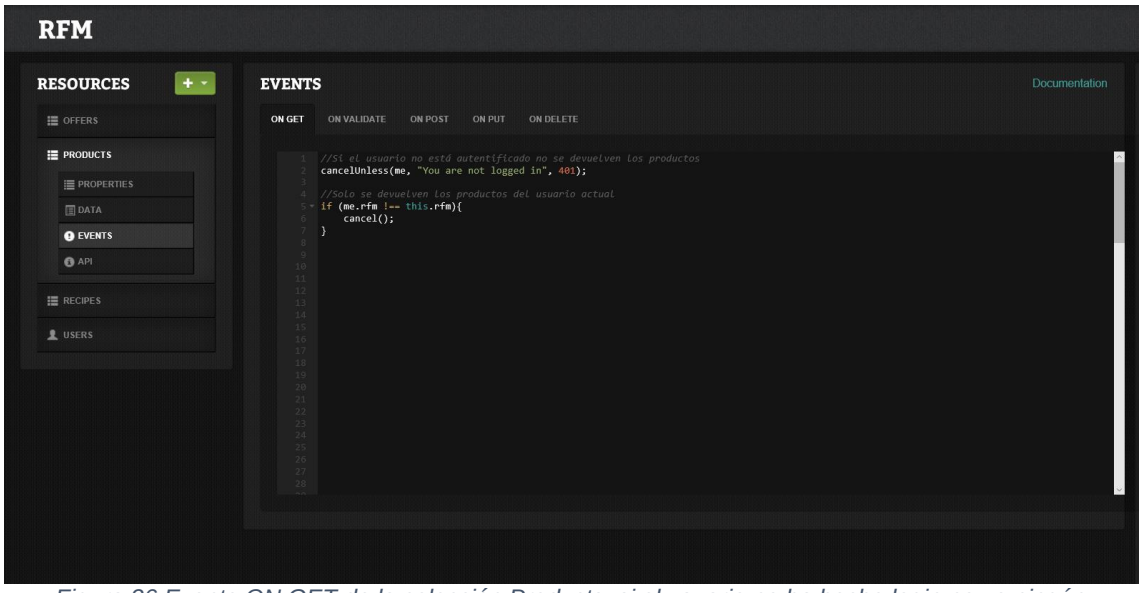

Figura 36 Evento ON GET de la colección Products, si el usuario no ha hecho login no ve ningún producto, si lo ha hecho ve solo los suyos

Además de realizar validaciones, se emplean los eventos para añadir propiedades a las colecciones.

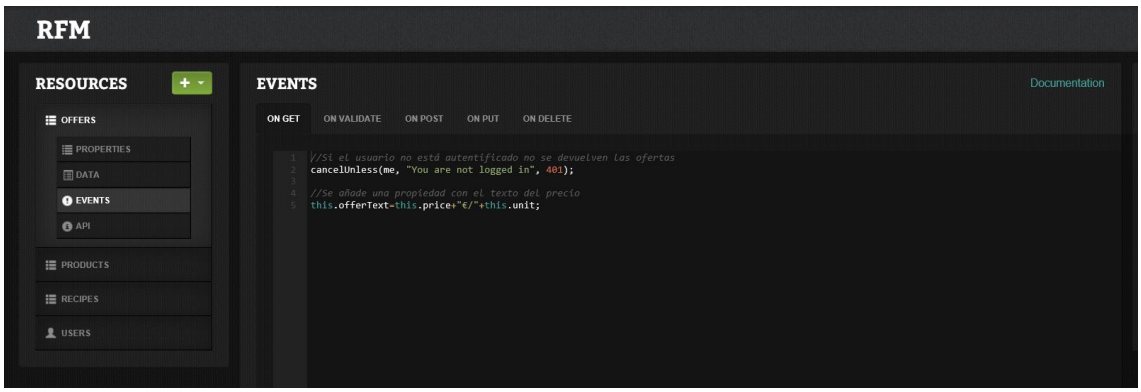

Figura 37 Cálculo del texto de una oferta desde el evento ON GET de la colección Offers

El código completo del *back-end* está disponible desde este repositorio [42].

4. Sistema RFID

De manera ideal, el sistema RFM dispondrá de un *hardware* RFID capaz de leer de manera simultánea todos los productos ubicados en el frigorífico y en la despensa del usuario.

Para el prototipo que se implementa en este trabajo, no se dispone de este *hardware*, y se va a emplear otro más asequible. Los detalles de dicho hardware se definen en los diferentes apartados del presente capítulo. Por último, se habla sobre el *software* empleado para hacer que el *hardware* se comporte como deseamos.

4.1 Hardware del sistema RFID

4.1.1 Arduino Mega

Arduino es una plataforma de código abierto para prototipos electrónicos, fácil de utilizar, y que cuenta con una gran comunidad de usuarios.

El *hardware* de *Arduino* consiste en una placa que hace uso de un microcontrolador *Atmel*. Este microcontrolador, una vez programado, es capaz de leer entradas. Las entradas pueden ser lecturas de sensores, pulsaciones de un botón o incluso mensajes de *Twitter*. A partir de las entradas se pueden activar salidas, como motores o LEDs [47].

Para programar un *Arduino* se emplea un IDE (*Integrated Development Environment*) de desarrollo [48] y una versión simplificada del lenguaje de programación C++.

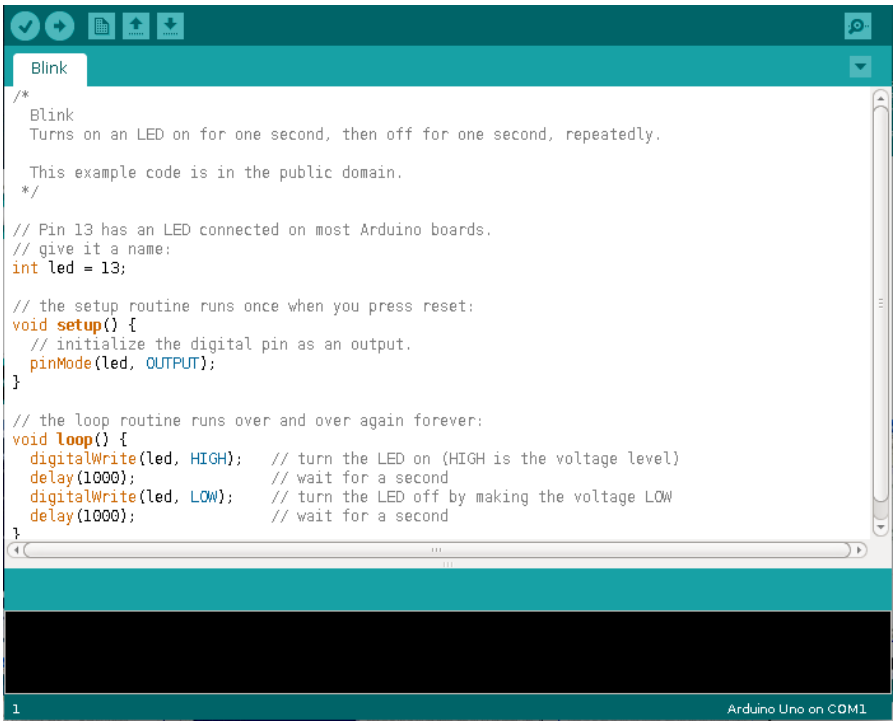

Figura 38 IDE de desarrollo de Arduino, código para apagar y encender un LED

Tras realizarse la programación desde el IDE de desarrollo, se compila el código y se vuelca al microcontrolador.

Existen decenas de modelos diferentes de placas *Arduino*, uno de los más utilizados es el modelo *Arduino Mega* [49], que será el que se emplee en el prototipo del sistema RFM.

Se emplea el *Arduino Mega* ya que tiene el factor de forma empleado por los módulos acoplables a una placa Arduino, como por ejemplo el módulo *Ethernet*, lo que facilita el conexionado. Además, dispone de más memoria que otros modelos más sencillos, lo que nos permitirá hacer programas más complejos. Mediante la placa *Arduino* se gestionarán las lecturas de un módulo RFID, y se transmitirán al servidor vía *Ethernet*.

A continuación se enumeran las principales características del *Arduino Mega*:

Parámetro	**Valor**
Microcontrolador	ATmega2560
Voltaje de operación	5V
Voltaje de entrada recomendado	7-12V
Voltaje de entrada máximo	6-20V
Pines digitales de entrada/salida	54 , 15 de ellos PWM
Pines analógicos de entrada	16
Corriente en los pines de entrada/salida	20mA
Memoria Flash	256KB
SRAM	8KB
EEPROM	4KB
Frecuencia de reloj	16MHz

Tabla 7 Características principales del Arduino Mega

El listado completo de características se puede consultar en [49].

Figura 39 Arduino Mega (version USA), y Genuino Mega (version fuera de USA) [49]

4.1.2 Arduino Ethernet Shield W5100

Existen distintos módulos o *shields* para *Arduino*. Estos módulos se acoplan sobre la placa *Arduino* ampliando sus capacidades. Para hacer uso de estas capacidades, cuentan con una serie de librerías que deben ser importadas en el código de nuestro programa.

En el prototipo del sistema RFM se utiliza una de estas placas para dotar a nuestro *Arduino* de conectividad a *Internet* a través de *Ethernet*.

Se emplea un modelo basado en el *chip Ethernet Wiznet W5100* [50]. Una vez acoplado al *Arduino*, se comunica con él mediante el protocolo *SPI* (*Serial Peripheral Interface*) [51]. Esta placa trabaja con una alimentación de 5V y se pueden alcanzar velocidades de transmisión de hasta 100Mbps de bajada y 10Mbps de subida.

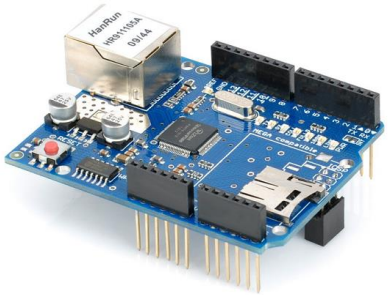

Figura 40 Arduino Ethernet Shield W5100 [50], módulo empleado para dotar a Arduino de conexión a Internet

4.1.3 Lector RFID MIFARE RC522

El lector RFID que se ha empleado en el prototipo del sistema RFM es el MIFARE RC522 [52]. Este módulo se alimenta a 3.3V, e incorpora una antena que trabaja en la frecuencia de 13.56MHz. Se controla a través del protocolo SPI, lo que lo hace compatible con casi cualquier microcontrolador o tarjeta de desarrollo. En la siguiente tabla se indican sus principales características:

Parámetro	**Valor**
Corriente de operación	60-26mA a 3.3V
Corriente de stand by	10-13mA a 3.3V
Corriente de sleep-mode	<80uA
Corriente máxima	30mA
Frecuencia de operación	13.56MHz
Distancia de lectura	0-60mm
Protocolo de comunicación	SPI
Velocidad de transmisión máxima	10Mb/s
Dimensiones	40x60mm
Temperatura de operación	-20 a 80ºC
Humedad de operación	5-95%

Tabla 8 Características principales del módulo RFID MIFARE RC522

Su hoja de características completa se encuentra disponible en [52].

Figura 41 Lector RFID MIFARE RC522 [52] empleado para leer tarjetas con información de productos

El conexionado entre el *Arduino Mega* y el módulo de lectura RFID se hace de la siguiente manera:

Arduino Mega	**RC522**
Pin 5 (digital)	RST
Pin 53 (digital)	SDA
Pin 51 (digital)	MOSI
Pin 50 (digital)	MISO
Pin 52 (digital)	SCK
3.3V	3.3V
GND	GND

Tabla 9 Conexionado de Arduino Mega y RC522

En la figura 42 tenemos la conexión entre el *Arduino* y el lector en una *protoboard.* Se ha acoplado un módulo *Ethernet* sobre el *Arduino Mega*, todo el conjunto se alimenta mediante USB.

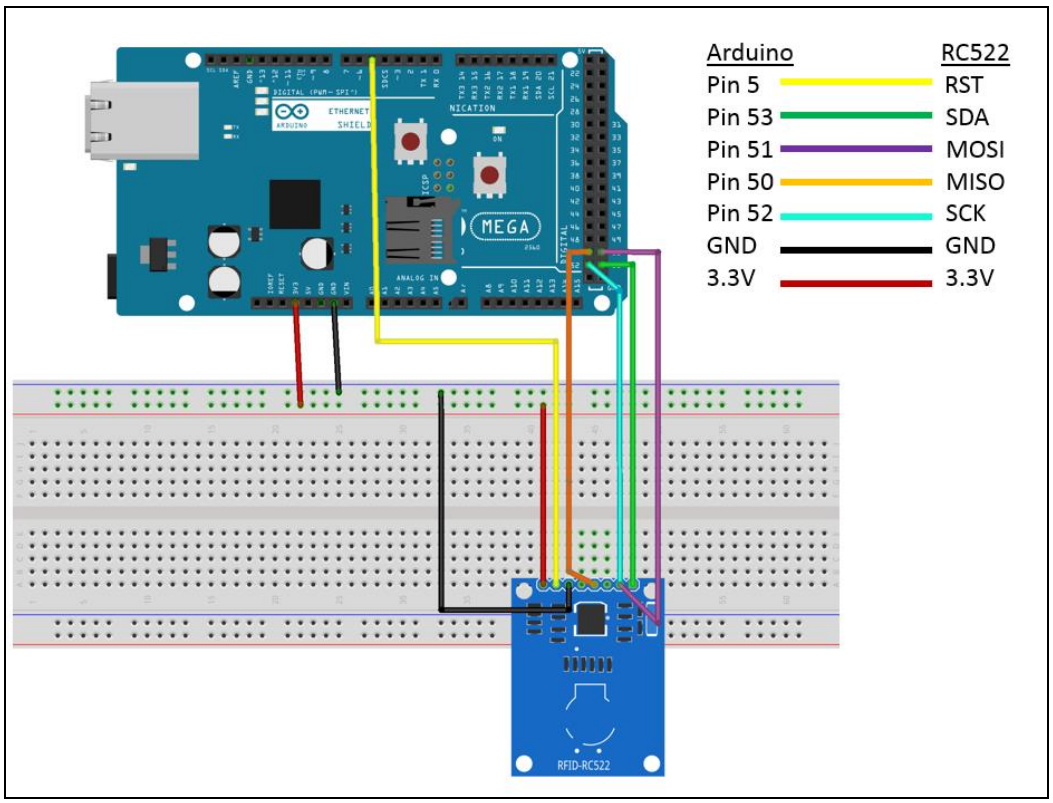

Figura 42 Conexión entre Arduino Mega con Ethernet Shield y RC522 en una protoboard

4.1.4 Tarjetas RFID MIFARE Classic 1K

Para el prototipo del sistema RFM se emplean las etiquetas RFID *MIFARE Classic 1K* [54]. Estas etiquetas se presentan en varios formatos, siendo los más habituales el de llavero y el de tarjeta. Se emplean habitualmente en aplicaciones como el control de accesos a recintos, o en el transporte público.

En este caso se emplearán para grabar información de diferentes productos, leídos posteriormente por el lector *MIFARE RC522*.

Se va a emplear el formato tarjeta, trabajan en la frecuencia de 13.56 MHz, pueden ser leídas a distancias de hasta 10cm, transmiten datos a una velocidad de 106 kbits/s, poseen un mecanismo de anti-colisión, y disponen de memoria EEPROM (*Electrically Erasable Programmable Read-Only Memory*) de 1KB, organizada en 16 sectores con 4 bloques de 16 bytes cada uno.

En cada uno de los sectores de memoria, tres de los bloques son de datos, y el cuarto se llama sector *trailer*. En él se almacenan las claves secretas A y B, que necesitamos conocer para poder leer el bloque, y los bits de acceso. Por tanto, quedan libres para guardar los datos, en este caso de un producto, tres de cada cuatro bloques, lo que hace un total de 768 bytes. La hoja de características completa de las tarjetas es accesible desde aquí [53].

Figura 43 Tarjetas RFID MIFARE Classic 1k [53] que contendrán la información de los productos

En la siguiente tabla se indica cómo se va a almacenar la información de los productos, esta información se almacena en formato hexadecimal [54]:

Sector	**Bloque**	**Bytes**	**Información**
15	0-2	0-15	Nombre del producto.
14	2	0-4	Identificador de alimento (*foodId*).
14	2	5-6	Grupo al que pertenece el alimento.
14	2	7-10	Energía que tiene 100g el producto, en kcals.
14	2	11-15	Cantidad de producto.
14	1	0-2	Gramos de grasa en 100g de producto.
14	1	3-5	Gramos de carbohidratos en 100g de producto.
14	1	6-8	Gramos de proteína en 100g de producto.
14	1	9	Indica si el producto contiene gluten.
14	1	10	Indica si el producto contiene huevo.
14	1	11	Indica si el producto contiene lácteos.
14	1	12	Indica si el producto contiene soja.
14	1	13	Indica si el producto contiene cacahuetes.
14	1	14	Indica si el producto contiene frutos con cáscara.
14	1	15	Indica si el producto contiene pescado.
14	0	0	Indica si el producto contiene marisco.
14	0	1	Indica si el producto contiene apio.
14	0	2	Indica si el producto contiene sésamo.
14	0	3	Indica si el producto contiene mostaza.
14	0	4	Indica si el producto es fresco.
14	0	5-6	Indica el día de la fecha de caducidad/envasado.
14	0	7-8	Indica el mes de la fecha de caducidad/envasado.
14	0	9-12	Indica el año de la fecha de caducidad/envasado.
14	0	13-14	Unidades en las que se mide la cantidad de producto.

Tabla 10 Disposición de la información de producto en una tarjeta MIFARE Classic 1k

Además de la información de la tabla 10, todas las tarjetas disponen de un código llamado *NUID* (*Non-Unique Identifier*), de 4 bytes, viene grabado de fábrica, es inalterable, y puede haber varias tarjetas que lo compartan.

Otras etiquetas RFID disponen de un código *UID* (*Unique Identifier*), en este caso, el fabricante garantiza que no habrá dos etiquetas con el mismo identificador. Este tipo de código es el deseable para el sistema RFM real, de modo que cada producto tenga un número de serie único, y pueda ser identificado de manera unívoca.

Para el prototipo implementado en el presente trabajo, se supondrá que el código *NUID* es distinto en cada tarjeta, es decir, un *UID*, pues se pueden tener 16^4 códigos diferentes, y se va a trabajar con 10 o 12 tarjetas, la probabilidad de que este código se repita es ínfima.

4.2 Software del sistema RFID

En este apartado se va a hablar del *software* que hace posible el funcionamiento del sistema RFID.

Para trabajar con el lector RC522, hay que descargar una librería e incluirla en nuestro código. Existen varias librerías para este propósito, en este caso se va a emplear una librería llamada MFRC522, disponible en este repositorio de *Github* [55]. Se descarga, e instala, tal y como se explica desde la *web* de *Arduino* [56].

```
#include <MFRC522.h>
```

Figura 44 Inclusión de la librería MFRC522 en el código para Arduino

4.2.1 Software de grabación de tarjetas

Para realizar las pruebas del sistema RFID, lo primero que se hace es grabar información de diferentes productos en las tarjetas RFID. Para ello se escribe un programa, en el que se va solicitando la información del producto, una vez se ha recogido esta información, se aproxima una tarjeta al módulo RC522, y se graba.

Se va a grabar una tarjeta a modo de ejemplo, en este caso, la información de un paquete de galletas, con el siguiente detalle:

Sector	Bloque	Bytes	Información	Valor
15	0-2	0-15	Nombre	Galletas María
14	2	0-4	Identificador de alimento (*foodId*)	18191
14	2	5-6	Grupo	18
14	2	7-10	Energía	401
14	2	11-15	Cantidad	300
14	1	0-2	Grasa	14
14	1	3-5	Carbohidratos	68
14	1	6-8	Proteína	4
14	1	9	¿Contiene gluten?	Y

14	1	10	¿Contiene huevo?	Y
14	1	11	¿Contiene lácteos?	N
14	1	12	¿Contiene soja?	N
14	1	13	¿Contiene cacahuetes?	N
14	1	14	¿Contiene frutos con cáscara?	N
14	1	15	¿Contiene pescado?	N
14	0	0	¿Contiene marisco?	N
14	0	1	¿Contiene apio?	N
14	0	2	¿Contiene sésamo?	N
14	0	3	¿Contiene mostaza?	N
14	0	4	¿Producto fresco?	N
14	0	5-6	Día fecha caducidad/envasado	15
14	0	7-8	Mes fecha caducidad/envasado	11
14	0	9-12	Año fecha caducidad/envasado	2018
14	0	13-15	Unidades	g

Tabla 11 Información de un paquete de galletas para grabarla en una tarjeta RFID

Una vez se sabe qué información se va a grabar, hay que guardar en el *Arduino* el programa que se ha realizado para la grabación de tarjetas, cuyo código está disponible desde un repositorio de *Github* [57], en la carpeta *rfid*_write.

Hecho esto, se abre el monitor de puerto serie, que se empleará para la comunicación con el *Arduino*. Se puede usar el monitor de puerto serie que viene con el IDE de desarrollo de *Arduino*, o cualquier otro, en este caso se va a emplear uno disponible en la *web* [58].

Tras arrancar el monitor serie, el programa irá pidiendo todos los detalles del producto, tal y como se puede ver en la figura:

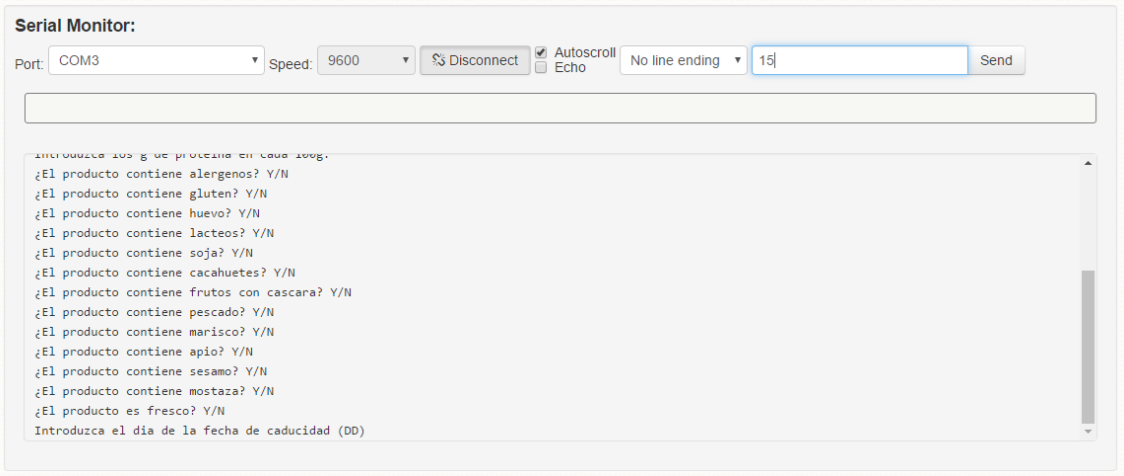

Figura 45 Introducción de detalles de un paquete de galletas para su grabación en una tarjeta RFID

A continuación, el programa pide al usuario que aproxime una tarjeta, en la que se graba toda la información:

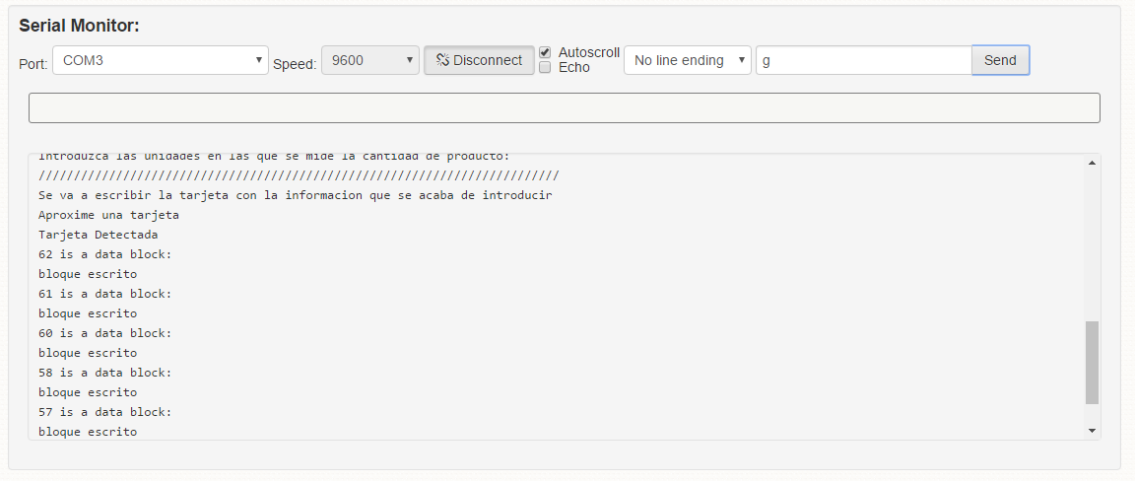

Figura 46 Grabación de una tarjeta RFID tras introducir el detalle de un producto

Con la tarjeta ya grabada, se va a comprobar cómo se ha guardado todo el detalle del producto en cada uno de los bloques de memoria, para ello se lee el contenido de la tarjeta mediante el programa *rfid_dump_info*, incluido en la librería [55].

Tras grabar el programa en el *Arduino*, se arranca de nuevo el monitor serie y se aproxima la tarjeta, tal y como se puede observar en la figura 47, podemos leer todo el contenido de la memoria EEPROM de la tarjeta:

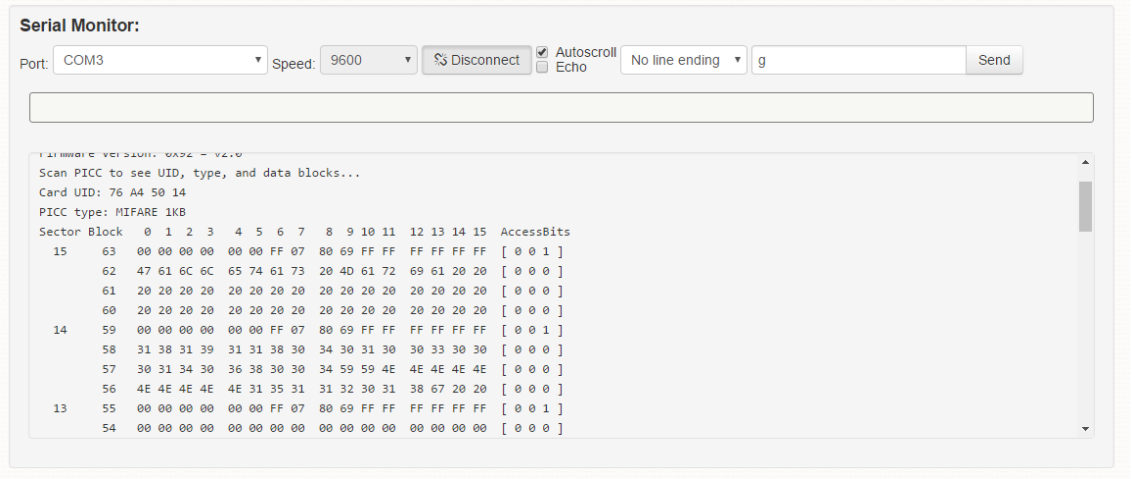

Figura 47 Contenido de la memoria EEPROM de la tarjeta RFID

En la figura 48 podemos ver toda la información del paquete de galletas, en formato hexadecimal. Se pueden traducir estos datos a formato ASCII de manera sencilla desde una *web* como esta [59].

En la siguiente figura se muestra la traducción de la información de los bloques 62, 61, 60, 58, 57 y 56, se comprueba cómo la información leída coincide con la de la tabla 11:

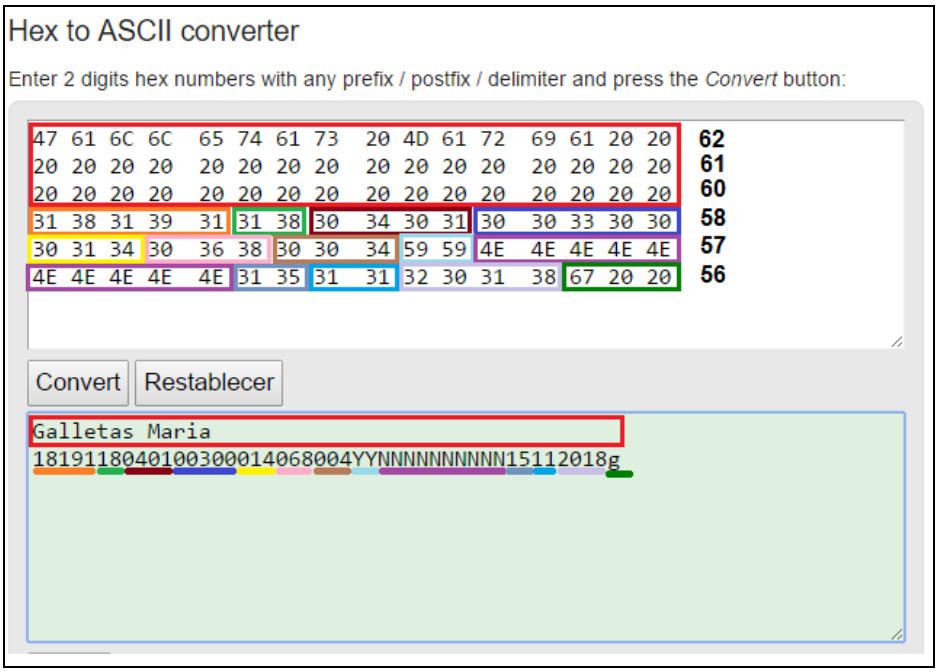

Figura 48 Información leída de tarjeta RFID en formato ASCII

4.2.2 Software de lectura de tarjetas y envío de información al servidor

Ya se ha explicado cómo se graba la información de un producto en una tarjeta, se va a pasar a programar el *Arduino* para leer el contenido de una tarjeta, y enviarlo al servidor de la aplicación RFM App.

El proceso de lectura y envío de información al servidor se muestra en el siguiente diagrama de flujo:

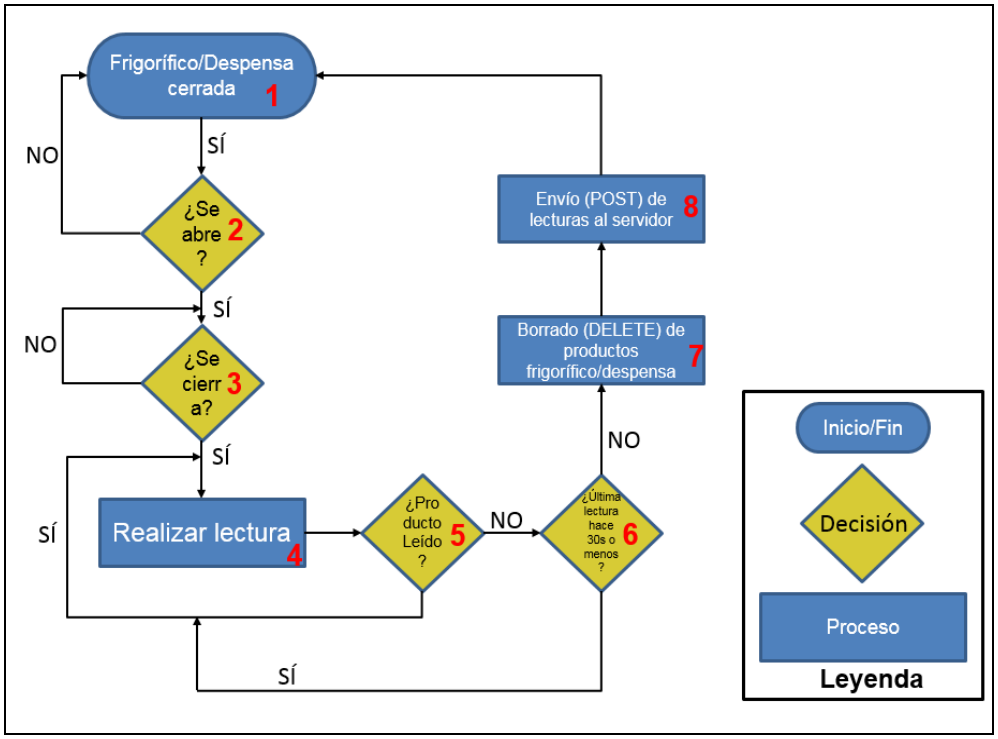

Figura 49 Diagrama de flujo del proceso de lectura y envío al servidor.

Se parte de una situación en la que el frigorífico o despensa está cerrado (1). A continuación, se comprueba si el frigorífico o despensa se ha abierto (2). Esta comprobación se puede implementar mediante un sensor de luminosidad, en el diagrama, por simplicidad, se realiza esta comprobación de forma continua.

Si el frigorífico o despensa se abre, hay que esperar hasta que se vuelva a cerrar (3), cuando esto ocurra, habrá que hacer una lectura de todos los productos presentes en el frigorífico o despensa, ya que pueden haber variado, se pueden haber retirado algunos alimentos y/o guardado otros.

El lector RFID se activa y empieza a leer (4), siempre que se haya leído un producto, se volverá a activar la lectura, para seguir leyendo el resto de productos (5), gracias al algoritmo anticolisión explicado en el apartado 5.2.3, se evitará leer el mismo producto en varias ocasiones.

Si el lector permanece treinta segundos o más sin leer un nuevo producto (6), se dará por hecho que ya se han leído todos los productos, se dará la lectura por finalizada, y se mandará al servidor la orden *DELETE* para borrar todos los productos del frigorífico o despensa en el que se han realizado las lecturas (7).

Por último, se manda al servidor una orden *POST* para almacenar todos los productos que se acaban de leer (8), quedando así actualizado el listado de productos para el frigorífico o despensa actual.

Para el prototipo no se ha implementado el algoritmo completo, sino una versión simplificada, en la que cada producto leído es mandado directamente al servidor, es decir, se realizan los pasos 4 y 8.

El código del programa empleado es accesible desde el repositorio de *Github* [57], en la carpeta *rfid_read*.

A continuación, se va a mostrar cómo se lee el producto grabado en el apartado 4.2.1, su información se enviará al servidor, será grabada en la base de datos, y podrá ser consultada desde la aplicación RFM App.

En primer lugar, hay que grabar el *Arduino* con el programa *rfid_read*, este se alimenta por *USB*, y se conecta a *Internet* mediante un cable de red, a través del puerto *Ethernet* del módulo *Ethernet* acoplado al dispositivo. Se abre el monitor serie, que permitirá saber qué está ocurriendo en todo momento:

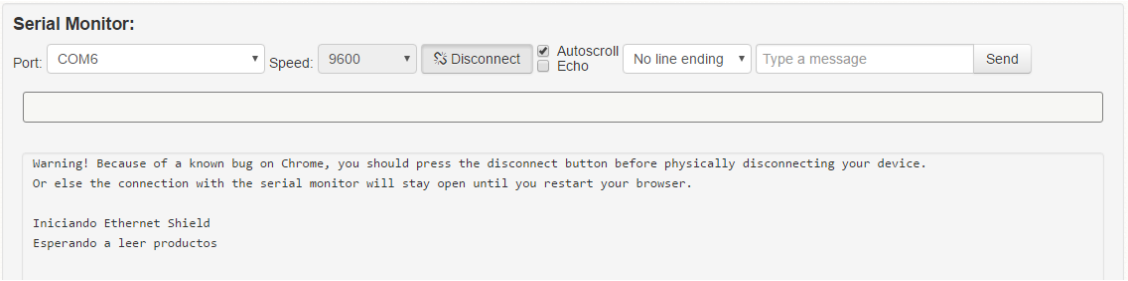

Figura 50 Monitor serie, el Arduino se encuentra esperando a la lectura de un producto

El *Arduino* se conecta a *Internet* y queda a la espera de leer un producto. Tras leer una tarjeta con la información de un producto, conecta con el servidor donde está alojada la aplicación *web*, y hace un *POST* de la información que se ha leído en la tarjeta, junto a su identificador *NUID*.

Además de la información que viene guardada en la tarjeta, se envía, de manera simultánea, otra información que viene grabada en el programa. Esta información es:

- **Campo *antenna***, indica si el lector está situado en el frigorífico, en la despensa 1, o en la despensa 2, según su valor sea, 01, 02, o 03, respectivamente.

- **Campo *rfm***, identifica qué sistema RFM ha leído el producto. Cuando un usuario se registra en la aplicación *web*, indicará éste valor, quedando relacionado el usuario, con los productos que tiene en su sistema RFM.

Con la información del producto leída de la tarjeta, que se almacena en formato hexadecimal, en una variable llamada *strBlocks*, el *NUID*, y los campos *antenna* y *rfm*, se construye un JSON que será enviado al servidor:

```
data="{\"rfm\":\""+rfm+"\",\"antenna\":\""+antenna+"\",\"UID\":\""+nuid+"\",\"arduino\":\""+strBlocks+"\"}";
```

```
{
    "rfm":"0001",
    "antenna":"01",
    "UID":"d145f5f9",
    "arduino":"47616c6c65746173204d617269612020202020202020202020202020202020202020202020202020202020202020
    2020313831393131383034303130303330303031343036383030303459594e4e4e4e4e4e4e4e4e4e3135313132303138472020"
}
```

Figura 51 Código extraído del programa rfid_read. Arriba, variable data en la que se almacena el JSON que se enviará al servidor. Abajo, contenido de la variable data tras la ejecución del programa.

El *back-end* se encargará de procesar el contenido del campo *arduino*, para extraer toda la información del producto, y transformarla en caracteres *ASCII*.

Con el JSON que se quiere enviar ya construido, se realiza un *POST* al servidor, con el objetivo de crear un nuevo documento en la colección *products* de la base de datos:

```
//Si se consigue conectar con el servidor
if (client.connect(server, 80)) {
  Serial.println("conectado");
  client.println("POST /products HTTP/1.1");
  client.println("Host: rfmapp.cloudno.de");
  client.println("Content-Type: application/json");
  client.print("Content-Length: ");
  client.println(data.length());
  client.println();
  client.print(data);
  Serial.println("post realizado");
}
else {
  //En caso contrario se informa de que la conexion ha fallado
  Serial.println("conexion fallida");
}
```

Figura 52 Código extraído del programa rfid_read. Se envía la información de un producto al servidor mediante un POST.

En la siguiente figura se comprueba cómo la lectura se ha realizado con éxito, y la información del producto se ha mandado al servidor:

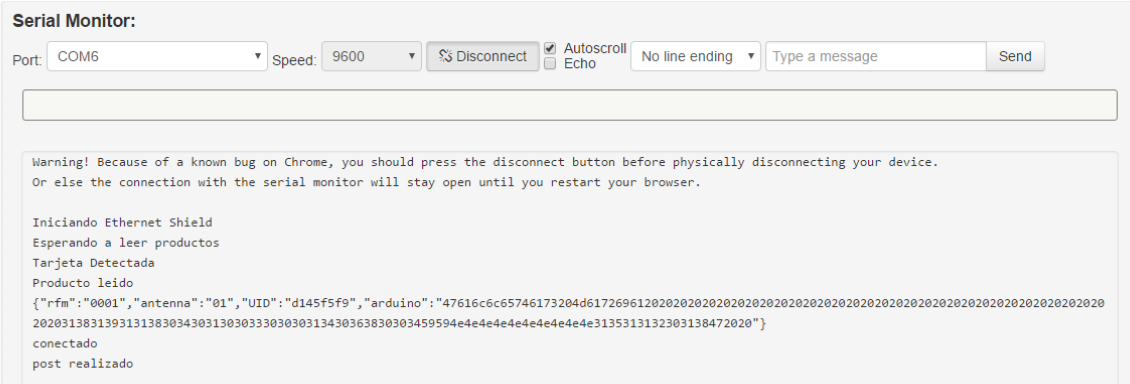

Figura 53 Monitor serie, el producto ha sido leído y mandado al servidor

El documento se ha guardado en la base de datos, tal y como se puede apreciar en la figura:

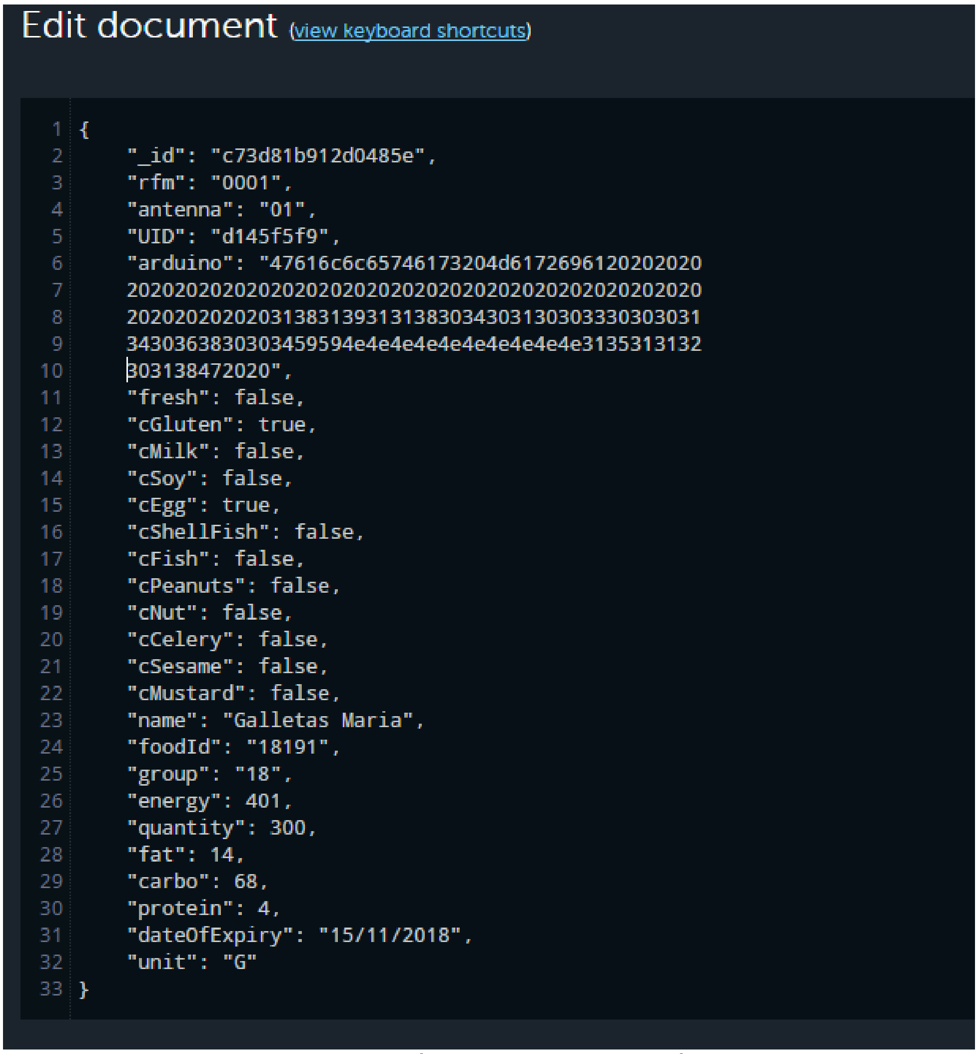

Figura 54 Documento recién grabado en la colección de productos

5. Diseño final y simulaciones

5.1 Diseño final

Una vez probada la aplicación RFM App, y el *hardware* RFID por separado, es el momento de concretar cómo queda el sistema RFM completo, el resultado se puede ver en la siguiente figura:

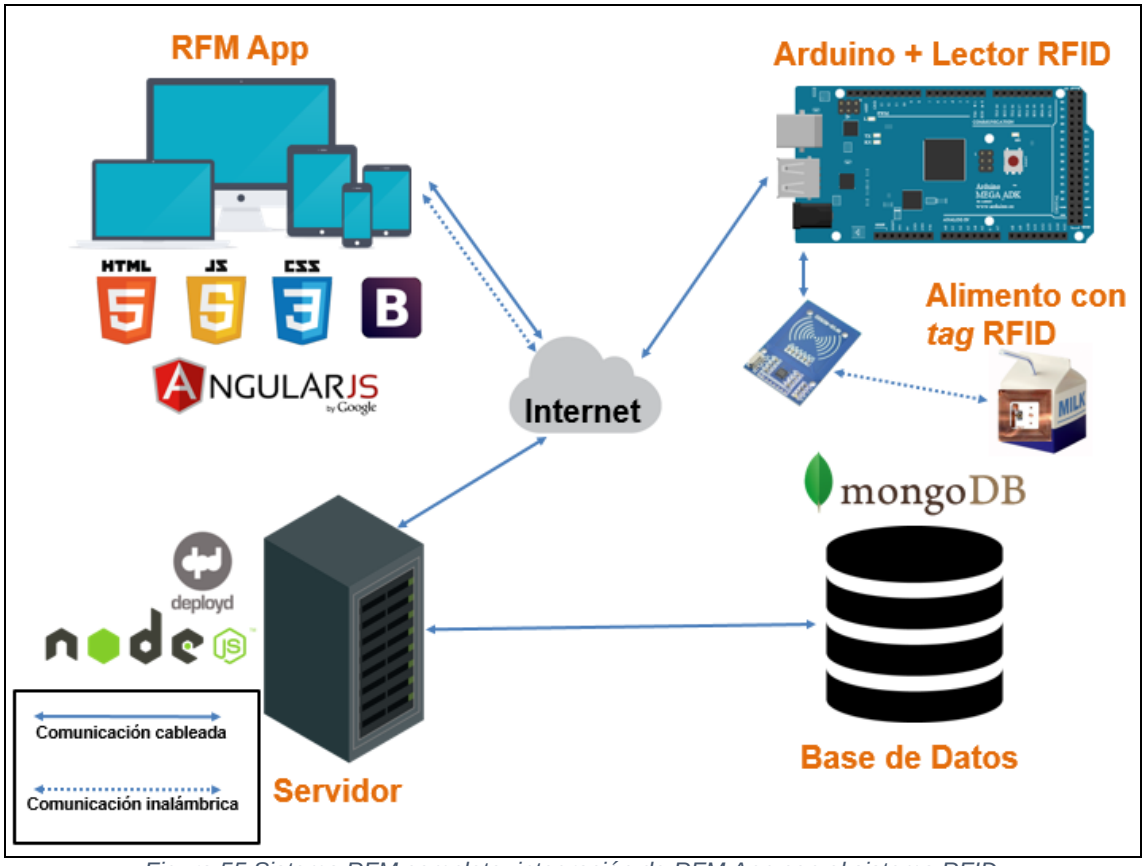

Figura 55 Sistema RFM completo, integración de RFM App con el sistema RFID

Aunque en la figura se ha representado un único *Arduino*, la realidad es que habría más de uno, ya que en el sistema final, se harían lecturas desde el frigorífico, y una o varias despensas.

5.2 Pruebas del sistema RFID

5.2.1 Prueba de rango de lectura y escritura

En este apartado se va a medir el rango de lectura y escritura del sistema RFM. En primer lugar, se aumenta la ganancia de la antena al máximo, para ello hay que modificar la librería MFRC522. Esta modificación se realiza a partir de lo indicado en el apartado 9.3.3.6 de la hoja de características del lector MFRC522 [52].

Existe un registro llamado *RFCfgReg*, de un *byte*, cuyos *bits* 4, 5 y 6, permiten configurar la ganancia del lector.

Bit	7	6	5	4	3	2	1	0
Símbolo	Reservado	RxGain[2:0]			Reservado			

Tabla 12 Registro RFCfgReg [52]

RxGain	Ganancia (dB)
000	18
001	23
010	18
011	23
100	33
101	38
110	43
111	48

Tabla 13 Posibles valores de RxGain [52]

El valor más bajo de la ganancia, 18 dB se consigue estableciendo el valor de los tres *bits* a cero, el valor máximo, 48dB, se obtiene cuando los tres *bits* valen uno.

Se edita la librería MFRC522, concretamente el archivo MFRC522.ccp, incluyendo las siguientes líneas:

```
282 void MFRC522::PCD_SetAntennaGain(byte mask) {
283     if (PCD_GetAntennaGain() != mask) {                    // only bother if there is a change
284         PCD_ClearRegisterBitMask(RFCfgReg, (0x07<<4));     // clear needed to allow 000 pattern
285         PCD_SetRegisterBitMask(RFCfgReg, mask & (0x07<<4)); // only set RxGain[2:0] bits
286     }
```

Figura 56 MFRC522.ccp, configuración de la ganancia de la antena a 48dB

Se coloca el lector tal y como se muestra en la siguiente figura. Se realizarán distintas pruebas, con la tarjeta en dos posiciones diferentes: la posición A, en la que la tarjeta se pone en horizontal, y la posición B, en la que se pone en vertical.

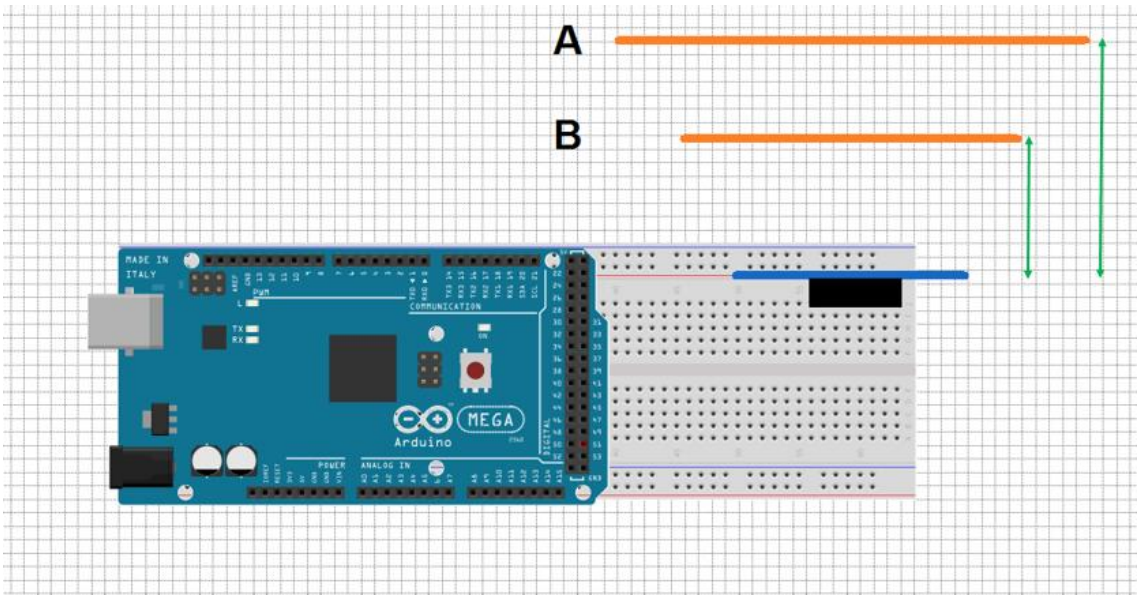

Figura 57 Colocación del lector RC522 para las pruebas de rango de lectura/escritura. Las dos posiciones de las tarjetas en color naranja. En color verde las distancias que se medirán

Además de colocar las tarjetas en dos posiciones diferentes, se harán pruebas de lectura y escritura en tres escenarios:

1. Lectura de tarjeta.
2. Lectura de tarjeta adherida a un recipiente que contiene líquido.
3. Lectura de tarjeta adherida a un recipiente metálico.

Se realizan cinco mediciones por cada combinación de posición de la tarjeta, A o B, y de cada uno de los tres escenarios. Por tanto, se realizan un total de sesenta mediciones, treinta de lectura, y treinta de escritura. Cada medición indica a qué distancia máxima ha podido ser realizada una lectura o escritura.

Figura 58 Izquierda: tarjeta en posición vertical adherida a un envase con líquido. Centro:Tarjeta adherida a un recipiente metálico. Derecha: tarjeta en posición horizontal adherida a un envase con líquido

El resultado de las mediciones, en centímetros, se muestra en la siguiente tabla:

	Lectura						Escritura					
Escenario	**1**		**2**		**3**		**1**		**2**		**3**	
Posición	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
Medida 1	5.75	6.25	2.25	2.50	2.00	2.75	5.50	6.25	2.75	2.25	0.50	1.00
Medida 2	5.50	6.00	2.00	2.25	2.25	2.50	5.75	6.25	2.50	2.00	0.75	1.00
Medida 3	5.50	6.00	2.00	2.25	2.25	2.75	5.75	6.25	2.75	2.00	0.75	0.75
Medida 4	5.75	6.25	2.25	2.50	2.25	2.75	5.75	6.50	2.75	2.25	0.75	1.00
Medida 5	5.75	6.25	2.25	2.25	2.00	2.75	5.75	6.50	2.75	2.25	0.75	1.00
Media	5.65	6.15	2.15	2.35	2.15	2.70	5.70	6.35	2.70	2.15	0.70	0.95

Tabla 14 Mediciones de rango de lectura y escritura en diferentes escenarios

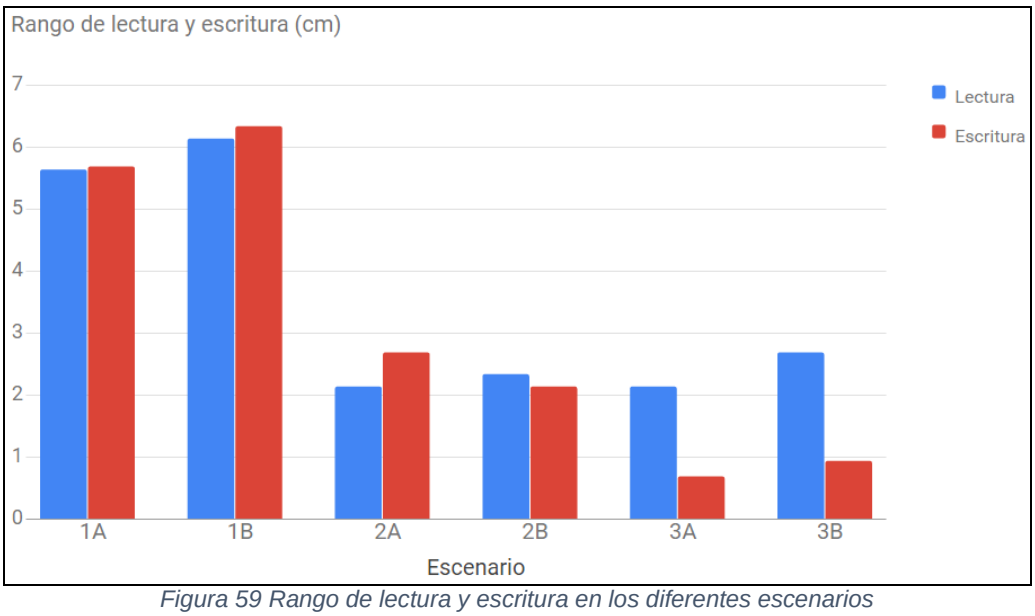

Figura 59 Rango de lectura y escritura en los diferentes escenarios

De los resultados mostrados en la figura anterior se pueden sacar varias conclusiones:

- El rango de lectura y escritura, salvo la escritura en presencia de metal, es mayor cuando la tarjeta se coloca en la posición B, es decir, en vertical. Esto tiene su explicación en que la superficie de tarjeta radiada por la antena del lector es mayor en este caso.
- En ausencia de líquidos y metales, en el escenario 1, se obtienen rangos de lectura y escritura muy similares a los teóricos, que tal y como se mostró en el apartado 4.1.3 son de 6 cm.
- El rango de escritura es mayor que el de lectura para el escenario 1, en el resto de escenarios es menor. Es preferible que ocurra esto, ya que las etiquetas se pueden grabar antes de ser incorporadas a un producto que contenga líquido o metal, y en cualquier caso, la lectura se hará una única vez, mientras que la lectura se hará decenas de veces.
- La presencia de líquidos y de metales afecta muy negativamente, en todos los casos, el rango de lectura y escritura disminuye al menos un 50% en su presencia. Esto era predecible, y se advirtió en el apartado 2.1.3, es uno de los principales retos de la tecnología RFID.
- Es particularmente reseñable la dificultad que existe para grabar una tarjeta en presencia de un recipiente metálico, en este caso, el rango de lectura es unas seis veces menor que en el escenario 1.

5.2.2 Prueba de velocidad de lectura y escritura

Empleando los mismos escenarios que en el apartado anterior, se van a realizar pruebas de velocidad de lectura y escritura, para determinar cómo de rápido se leen y se graban las tarjetas. Para ello se añadirán unas líneas de código a los programas *rfid_read* y *rfid_write* para saber en qué momento exacto empieza la lectura/escritura, y cuando acaba, para finalmente obtener la diferencia entre ambos instantes. Por ejemplo, en el programa *rfid_read* se hacen los siguientes cambios:

```
51 //Variables para calcular el tiempo que tarda en completarse una lectura
52 unsigned long StartTime, CurrentTime, ElapsedTime;
```

```
97   //Se almacena el instante en el que se empieza a leer la tarjeta
98   StartTime=millis();
```

```
122     //Se mide cuanto ha durado la lectura
123     CurrentTime=millis();
124     ElapsedTime=CurrentTime - StartTime;
125     Serial.println(ElapsedTime);
```

Figura 60 Código de rfid_read, modificaciones para calcular cuánto tarda en hacerse una lectura

Tras realizar las modificaciones oportunas en los programas de lectura y escritura, se hacen un total de sesenta mediciones, de manera similar a lo realizado en el apartado 5.2.1, los resultados se expresan en milisegundos:

	Lectura						**Escritura**					
Escenario	**1**		**2**		**3**		**1**		**2**		**3**	
Posición	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
Medida 1	108	110	109	108	109	108	185	184	187	186	185	185
Medida 2	109	109	110	108	109	110	185	185	184	185	185	184

Medida 3	109	108	109	109	109	110	184	184	185	184	185	185
Medida 4	108	109	108	109	110	108	185	186	186	186	186	186
Medida 5	109	108	110	109	109	109	185	185	185	185	185	185
Media	108.6	108.8	109.2	108.6	109.2	109	184.8	184.8	185.4	185.2	185.2	185

Tabla 15 Mediciones de velocidad de lectura y escritura en diferentes escenarios

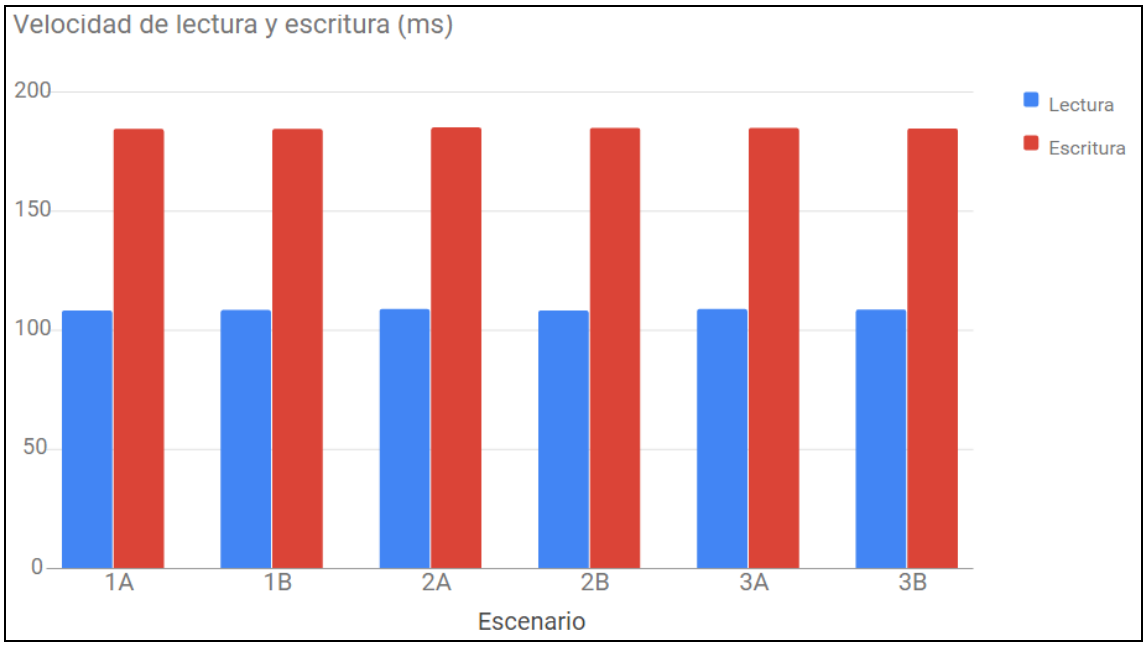

Figura 61 Velocidad de lectura y escritura en los diferentes escenarios

En vista de los resultados anteriores, se puede concluir que:

- La velocidad de lectura es mayor que la de escritura, unos 75ms más rápida.
- Las velocidades de lectura y escritura son muy similares en todos los escenarios, independientemente de la posición de la tarjeta y de la presencia de líquidos y metales.

5.2.3 Prueba de lectura simultánea de tarjetas

Se va a estudiar cómo gestiona las lecturas simultáneas de tarjetas el lector RC522. Según se indica en la hoja de características [52], en el capítulo 2, "*General Description*", el lector se comunica en base a lo descrito en el estándar internacional de tarjetas de identificación ISO/IEC 14443 [60].

El estándar ISO/IEC 14443, define un algoritmo anticolisión, este algoritmo gestiona la lectura de etiquetas RFID, concretamente en el caso de que varias de ellas se encuentren simultáneamente en la zona de interrogación del lector.

El lector solo establece comunicación con una etiqueta al mismo tiempo. Lo que consigue el algoritmo, es, que si hay varias etiquetas en el campo de radiación de lector, se comunique con una de ellas, y no lo vuelve a hacer mientras esta no abandone la zona de interrogación.

Gracias a esto, se ignoran a las tarjetas que ya han sido leídas, y se logra una lectura de tarjetas rápida y eficiente, dando la impresión de que las lecturas se producen al mismo tiempo.

Con el objetivo de probar el algoritmo anticolisión, se ha realizado un programa llamado *rfid_multiple_read*, disponible en el repositorio [57].

Se graba el programa en el *Arduino*, se abre el monitor serie, y se colocan tres tarjetas juntas, próximas al lector, que permanecerán en su zona de interrogación durante unos segundos. El resultado es el siguiente:

Figura 62 Monitor serie. Lectura de tres tarjetas situadas de manera simultánea en la zona de interrogación del lector.

En la figura anterior se observa que se realiza la lectura de las tres tarjetas de manera secuencial, además, se leen una única vez. Hasta que las tarjetas no salgan de la zona de interrogación, gracias al algoritmo anticolisión, no se volverán a leer.

Para terminar de comprobar el correcto funcionamiento del algoritmo, se sacan las tarjetas de la zona de interrogación, y se vuelven a acercar al lector:

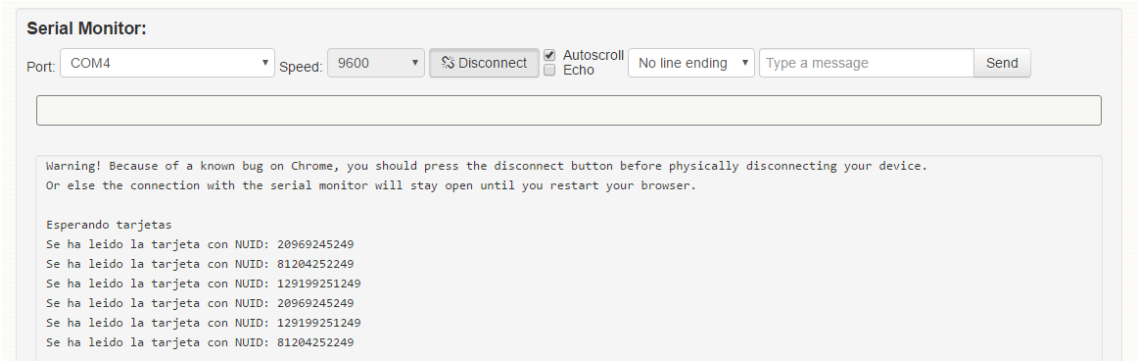

Figura 63 Monitor serie. Lectura de tres tarjetas situadas de manera simultánea en la zona de interrogación del lector, tras salir de la zona de interrogación y volver a entrar.

A continuación, se va a probar si es posible leer un número elevado de tarjetas de manera simultánea. Para ello se modifica el programa *rfid_multiple_read* y se obtiene el programa *rfid_multiple_read2*, disponible en el repositorio [57].

Con estas modificaciones, se muestra por el monitor serie cuántas tarjetas diferentes se han leído y el tiempo total que se tarda en realizar las lecturas. Solo se muestra por el monitor serie la lectura de aquellas tarjetas que son leídas por primera vez.

Se graba el programa *rfid_multiple_read2* en el *Arduino*. Posteriormente, se apilan diez tarjetas en dos bloques de cinco tarjetas, se arranca el monitor serie, y se aproximan a la misma vez los dos bloques de tarjetas al lector.

De este modo se comprueba si es posible leer todas las tarjetas a la vez, y cuánto tiempo se tarda en ello. Esto se puede ver en la siguiente figura:

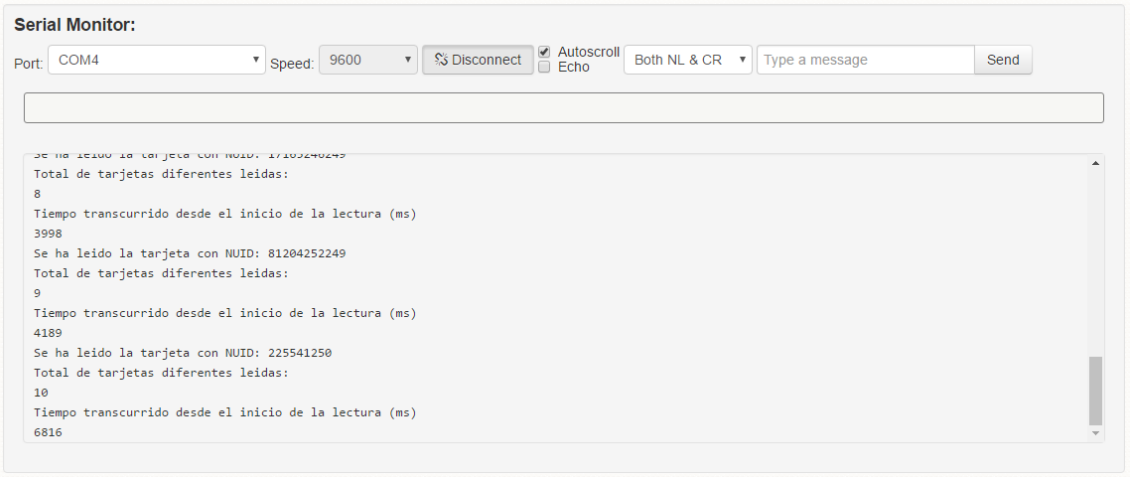

Figura 64 Monitor serie. Lectura de diez tarjetas situadas de manera simultánea en la zona de interrogación del lector.

En el monitor serie comprobamos cómo se han leído todas las tarjetas en algo menos de siete segundos. Se realizan un total de diez lecturas, los resultados se muestran en la siguiente tabla:

	Tiempo que tarda cada tarjeta en ser leída, en milisegundos										
Tarjeta / Prueba	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**Total**
#1	236	152	204	245	154	1960	893	154	191	2627	6816
#2	1375	156	1868	157	153	2164	938	343	404	1178	8736
#3	474	3514	156	332	464	155	1429	156	191	2079	8950
#4	929	162	155	154	167	1286	1192	154	335	1019	5553
#5	926	688	2040	151	180	80	81	49	50	1061	5306
#6	97	43	44	45	1061	1830	48	49	49	50	3316
#7	1018	221	267	348	607	2176	78	192	50	221	5178
#8	412	243	830	45	948	1290	48	974	206	1121	6117
#9	124	751	518	799	884	47	47	844	50	6006	10070
#10	1063	79	1994	115	107	160	93	753	445	938	5747
Media	665	601	808	239	473	1115	485	367	197	1630	6579

Tabla 16 Prueba de lectura de 10 tarjetas de manera simultánea

De los resultados mostrados en la tabla anterior se concluye que el tiempo medio de lectura de todas las tarjetas es de unos 6.5 segundos, un tiempo bastante bajo, aunque es de esperar que en un caso real fuera inferior, ya que el *hardware* empleado es de bajo coste, su corto alcance dificulta introducir en su zona de interrogación todas las tarjetas a la vez, y los tiempos de lectura son mejorables.

Cabe destacar que la última tarjeta es la que más tiempo tarda en ser leída, esto tiene lógica, pues al haber tantas tarjetas juntas, algunas pueden salir de

la zona de interrogación y volver a entrar, por lo que se leerían tarjetas ya leídas previamente, en lugar de leer la décima y última tarjeta.

5.2.4 Cálculo de caída de tensión entre el lector RC522 y el *Arduino*

Con el objetivo de saber si sería posible tener los lectores RC522 dentro del frigorífico/despensa, y el *Arduino* fuera, se va a calcular cuánto cae la tensión al separar el lector del *Arduino*. Vamos a suponer que se alejan diez metros entre sí. Se emplea un cable de cobre de diez metros de longitud, y una sección de 0.2mm^2. La resistencia de este cable se calcula como:

$$R = \frac{\rho\, l}{s} = \frac{1.78 * 10^{-8}\Omega m * 10m}{0.2 * 10^{-6}mm^2} = 0.89\Omega$$

Figura 65 Cálculo de la resistencia de un cable de cobre

Siendo ρ la resistividad del cobre a temperatura ambiente, l la longitud, y s la sección.

Sabiendo que el *Arduino Mega*, alimentado mediante USB proporciona una tensión de 3.3V y 100mA, la caída de tensión se calcula como:

$$V = I * R = 100mA * 0.89\Omega = 89\text{mV}$$

Figura 66 Caída de tensión entre lector RC522 y Arduino

La caída de tensión es prácticamente despreciable, por lo que se podría instalar el *Arduino* fuera del frigorífico o despensa.

5.3 Pruebas de RFM App, algoritmo de recomendación de recetas

En este apartado se comparan los dos algoritmos de recomendación de recetas ideados para la aplicación. Se prueban bajo las mismas condiciones, se comparan sus resultados, y se determina cuál es mejor, para implementarlo en RFM App. El código empleado para realizar estas pruebas se puede encontrar en el repositorio [42], dentro de la carpeta “js/others”, con el nombre “5_6_test.js”.

5.3.1 Algoritmo de recomendación basado en el porcentaje de ingredientes disponibles

Este primer algoritmo, recomienda una receta siempre y cuando se disponga en nuestro frigorífico y/o despensa de al menos la mitad de ingredientes necesarios. Recibe como parámetros un *array* con todas las recetas, y otro con los productos disponibles en el sistema RFM. Devuelve un *array* con las recetas recomendadas. En la siguiente figura se muestra su implementación:

```
function getRecRecipes(recipes,products){
  ///Si no hay recetas o no hay productos se devuelve un array vacio
  if(!recipes.length || !products.length) return [];
  //Se recorre cada receta, almacenando qué ingredientes se tienen y cuáles no
  recipes.forEach(function(recipe){
    ////Array para almacenar los productos que se necesitan
    var neededProds=[];
    //Se recorren los ingredientes de la receta
    recipe.ingredients.forEach(function(ingredient){
      //Se comprueba si se tiene el ingrediente actual
      var prod=products.filter(function(product){
        return ingredient.foodId === product.foodId;
      });
      //Si no se tiene el ingrediente se añade al array de productos que se necesitan
      if (!prod.length) neededProds.push(ingredient.name);
    });
    //Contador de productos disponibles
    var availableProducts = recipe.ingredients.length - neededProds.length;
    //Se añade al objeto recipe el % de productos que se tienen
    recipe.cProds=Number(((availableProducts*100)/recipe.ingredients.length).toFixed(2));
  });
  //Se devuelven aquellas recetas para las que se tenga la mitad o más de ingredientes
  return recipes.filter(function(recipe){
    return recipe.cProds >= 50
  });
};
```

Figura 67 Algoritmo de recomendación basado en el porcentaje de ingredientes disponibles

5.3.2 Algoritmo de recomendación basado en la importancia de los ingredientes disponibles

En este segundo algoritmo de recomendación, se tiene en cuenta cómo de importante es cada uno de los ingredientes de una receta para la preparación de la misma.

```
function getRecRecipes2(recipes,products){
  //Si no hay recetas o no hay productos se devuelve un array vacio
  if(!recipes.length || !products.length) return [];
  //Se recorre cada receta
  recipes.forEach(function(recipe){
    //Variable para almacenar el peso total de la receta
    //y el de los productos disponibles
    var recipeWeight=0;
    var productsWeight=0;
    //Se recorren los ingredientes de la receta
    recipe.ingredients.forEach(function(ingredient){
      //Se suma el peso del ingrediente actual al peso de la receta
      recipeWeight+=ingredient.weight;
      //Se comprueba si se tiene el ingrediente actual
      var prod=products.filter(function(product){
        return ingredient.foodId === product.foodId;
      });
      //Si se tiene el ingrediente se suma su peso
      if (prod.length){
        productsWeight+=ingredient.weight;
      }
    });
    //La receta se recomienda si el peso de los ingredientes disponibles
    //es la mitad o mas del peso total
    recipe.recommended = productsWeight>=(recipeWeight/2) ? true : false;
  });
  //Se devuelven las recetas recomendadas
  return recipes.filter(function(recipe){
    return recipe.recommended;
  });
};
```

Figura 68 Algoritmo de recomendación basado en la importancia de los ingredientes disponibles

Se asigna un peso a cada ingrediente, los ingredientes principales tienen más peso que los ingredientes empleados para realizar la guarnición, que a su vez tienen más peso que ingredientes como las especias, el aceite o la sal.
Si el peso de los ingredientes disponibles es mayor o igual al de los no disponibles, la receta es recomendada.

5.3.3 Caso de prueba y comparativa de los algoritmos

Se van a probar ambos algoritmos con un mismo caso de prueba. Se parte de una situación en la que el frigorífico y la despensa están vacíos. Se leen una serie de productos de manera progresiva, y se comprueba como varían las recomendaciones de recetas.

La prueba se compone de diez pasos, definidos en el *script* "5_6_test.js", anteriormente mencionado. En cada uno de los pasos, se añadirá uno o dos productos. Tras la ejecución del *script*, se obtendrán las recetas recomendadas por cada algoritmo en cada paso.

En la siguiente tabla se muestran los productos que se añaden en cada paso, y las nuevas recetas recomendadas respecto al paso anterior. De este modo se puede realizar una comparativa del funcionamiento de ambos algoritmos:

		Nuevas recetas recomendadas	
Paso	**Nuevos productos**	**Algoritmo basado en % de ingredientes disponibles**	**Algoritmo basado en la importancia de los ingredientes disponibles**
1			
2	Patatas, Leche	Puré de zanahoria	Puré de patatas
3	Sal, Pimienta negra	Puré de patatas	
4	Huevos	Filetes de cerdo empanado, Flan de huevo	Flan de huevo
5	Aceite de oliva, Cebolla	Berenjenas rellenas con carne picada, Filetes de cerdo empanado, Ensalada murciana, Espaguetis a la carbonara, Leche frita	Leche frita
6	Arroz	Lentejas guisadas con verdura y arroz	
7	Berenjenas, Tomate	Ensalada de hortalizas, Parrillada de verduras, Mejillones al vapor con vinagreta, Ensalada griega, Pasta boloñesa y berenjena	Ensalada de hortalizas, Berenjenas rellenas con carne picada
8	Muslos de pollo		Paella con pollo
9	Atún	Atún con pisto	Atún con pisto
10	Pulpo, queso feta		Ensalada de pulpo con cilantro y menta, Ensalada griega

Tabla 17 Resultados del caso de prueba

En el primer paso, el primer algoritmo recomienda puré de zanahoria, sin disponer de zanahoria, y no recomienda el de patatas teniéndolas. Aquí se ve claramente cuáles son las deficiencias de este algoritmo. La receta del puré de zanahoria tiene zanahoria, patatas y leche, por lo que al tener dos de los tres ingredientes, se recomienda.

No es muy lógico recomendar un puré de zanahoria si no se tienen zanahorias, aunque resulta más ilógico, que no se recomiende puré de patatas, si se dispone de patatas. Esto ocurre porque la receta del puré de patatas tiene cinco ingredientes: patatas, mantequilla, sal, pimienta, y ajo. Como solo se tiene uno de los cinco, no se recomienda.

Por el contrario, el algoritmo basado en la importancia de los ingredientes, sí recomienda el puré de patatas, ya que da más peso a las patatas que a los otros cuatro ingredientes juntos. Siguiendo este mismo razonamiento, no recomienda el puré de zanahoria, pues da más peso a la zanahoria, que a la suma del resto de ingredientes. En el paso 4, el primer algoritmo recomienda filetes empanados, sin que se disponga de filetes, esto se repite en los siguientes pasos, se recomiendan recetas en las que no se dispone de los ingredientes principales.

En el último paso, se añade pulpo. En el primer algoritmo, aunque el pulpo es un ingrediente principal de la ensalada de pulpo, la receta no es recomendada, algo que sí ocurre con el segundo algoritmo.

En definitiva, se puede concluir que el segundo algoritmo funciona mejor, gracias a que cada ingrediente tiene un peso en la receta, hace recomendaciones más acertadas, pues recomienda aquellas recetas para las que se dispone de los ingredientes principales.

Es por esto que en la aplicación *web* se ha implementado el algoritmo de recomendación basado en la importancia de los ingredientes disponibles.

Por último, se muestran algunas capturas, donde se ve la evolución del caso de prueba en la aplicación, que implementa el segundo algoritmo:

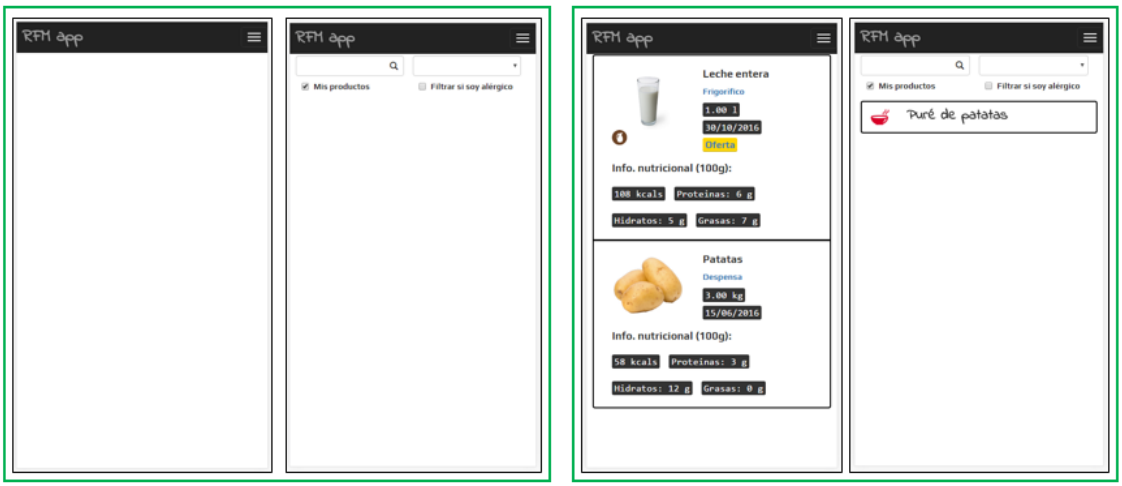

Figura 69 RFM App en iPhone 6 Plus. Pasos 1 y 2 del caso de prueba

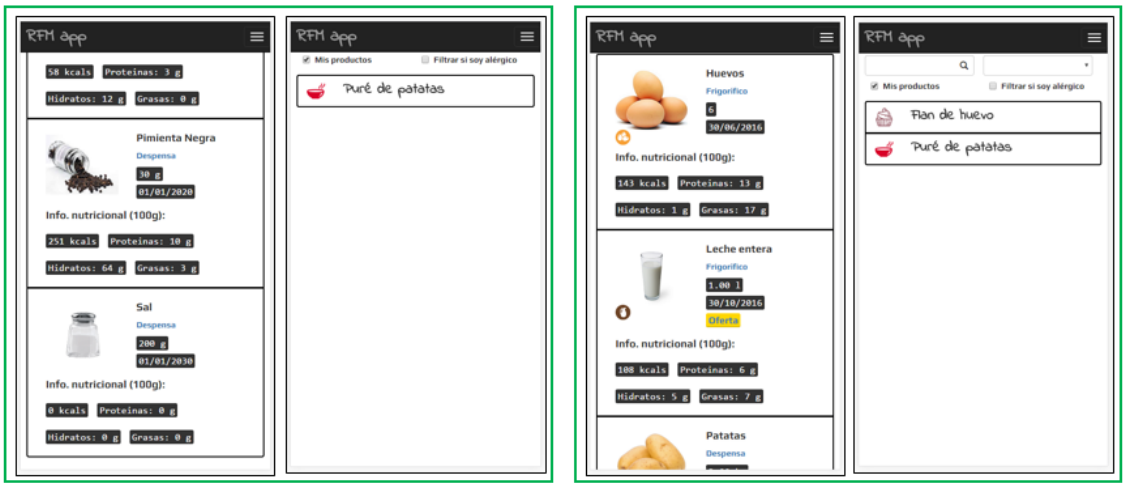

Figura 70 RFM App en iPhone 6 Plus. Pasos 3 y 4 del caso de prueba

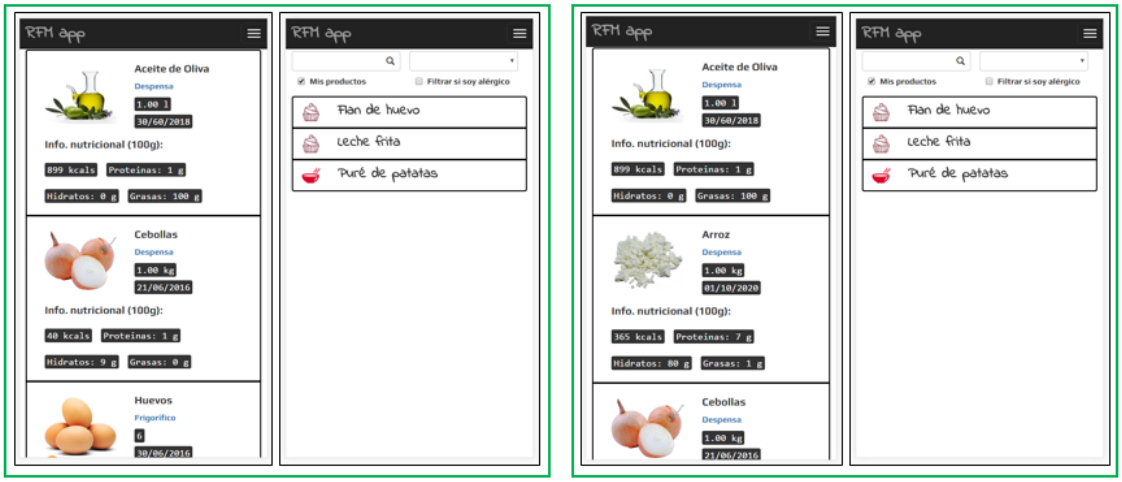

Figura 71 RFM App en iPhone 6 Plus. Pasos 5 y 6 del caso de prueba

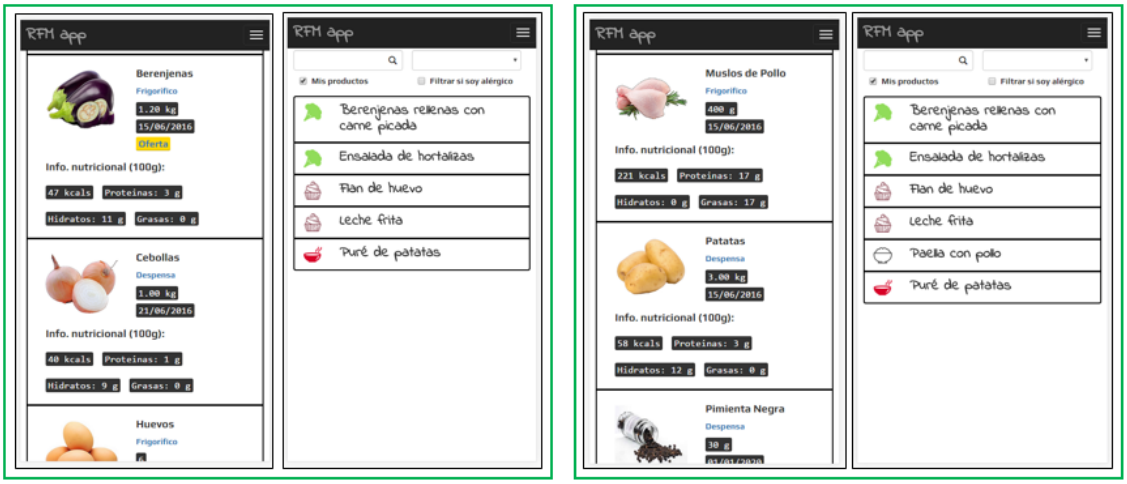

Figura 72 RFM App en iPhone 6 Plus. Pasos 7 y 8 del caso de prueba

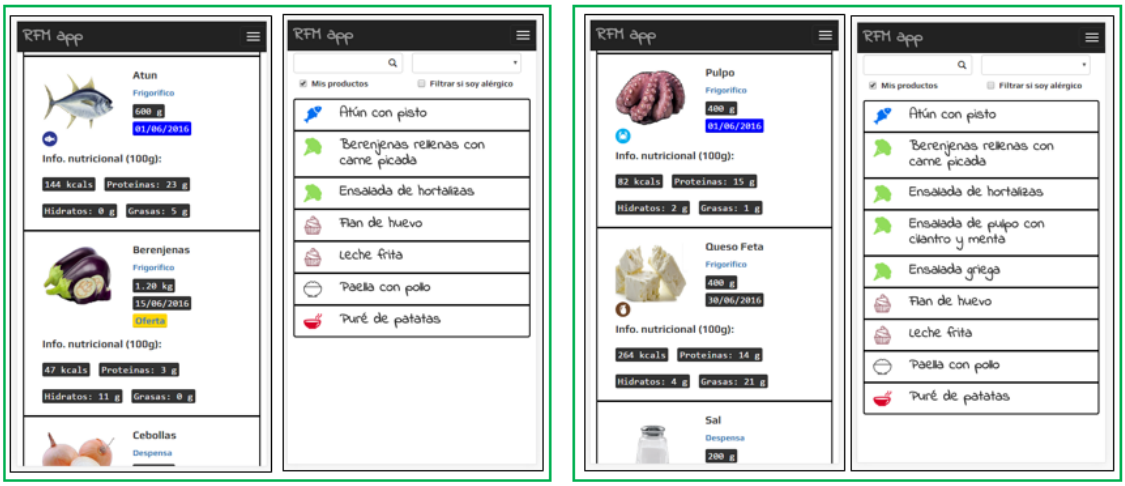

Figura 73 RFM App en iPhone 6 Plus. Pasos 9 y 10 del caso de prueba

Tras realizar todos los pasos de la prueba, y comprobar que los resultados coinciden con los esperados, se puede concluir que el algoritmo de recomendación de recetas funciona a la perfección.

5.4 Prueba *end to end* conjunta del sistema RFID y RFM App

Se va a realizar una prueba de inicio a fin. En dicha prueba, un usuario se registrará en la aplicación RFM App, se leerán sus productos, y se consultarán posteriormente desde la aplicación. Por último, se consultarán recetas. Para ver de manera más detalladas todas las funcionalidades de la aplicación, en el anexo 9.1 se puede consultar un manual completo de la misma.

El escenario que se va a probar es el de un usuario que adquiere un sistema RFM, y se registra en la aplicación. Su sistema RFM tiene asignado el código “0000”, que será indicado durante el registro.

La aplicación está publicada en *Internet*, disponible para cualquiera desde el enlace: http://rfmapp.cloudno.de/. La prueba empezará accediendo a la aplicación, tras abrirla, se pulsa el botón “*Registro*” y se introducen los datos del usuario, a continuación, se vuelve a pulsar el botón de “*Registro*”, lo que nos devuelve a la pantalla inicial, donde se introduce el usuario y contraseña del usuario recién creado, y se pulsa “*Acceder*”. Al entrar a la aplicación se abre directamente el menú de productos, que de momento estará vacío.

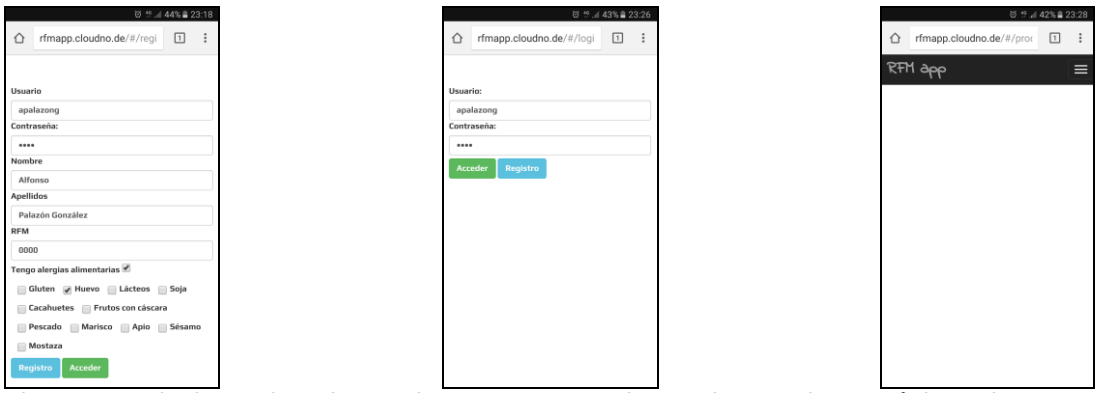

Figura 74 Izquierda: Registro de usuario, Centro: Acceso de usuario. Derecha: Menú de productos vacío

Ahora que el usuario está registrado, se van a guardar productos en su frigorífico y en su despensa. Para la prueba se han grabado once tarjetas con información de diferentes productos, haciendo uso del programa *rfid_write*, detallado en el apartado 4.2.1.

Se realiza el montaje del *Arduino*, tal y como se indicó en la figura 42, se graba el programa *rfid_read*, explicado en el apartado 4.2.2, y se conecta el *Arduino* a *Internet*.

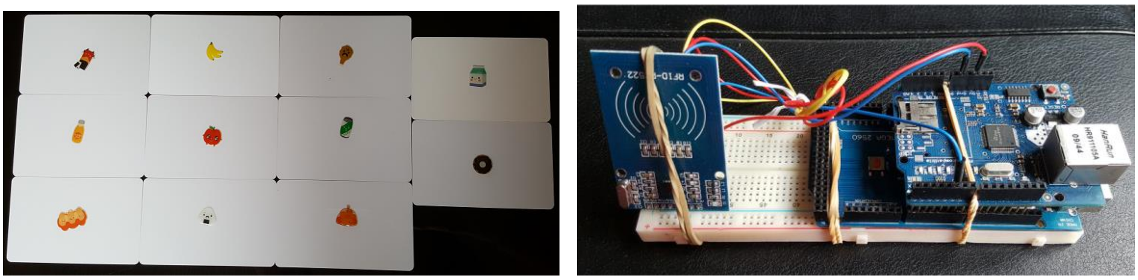

Figura 75 Izquierda: Tarjetas con los productos. Derecha: Montaje del Arduino en protoboard

La lectura se hace en dos tandas, por una parte se leen los productos que irán al frigorífico, estableciendo en el programa de lectura la variable *antenna* con el valor “01”, y más tarde los que se colocarán en la despensa, cambiando el valor de la variable por “02”.

```
Iniciando Ethernet Shield
Esperando a leer productos
Tarjeta Detectada
Producto leido
{"rfm":"0000","antenna":"01","UID":"b1990fa","arduino":"4d75736c6f7320646520706f6c6c6f202020202020202020202020202020202020202020202020202020202020202020202020202
02020303530393130353032323130303430303031373030303031374e4e4e4e4e4e4e4e4e4e4e3230303532303136672020"}
conectado
post realizado
Tarjeta Detectada
Producto leido
{"rfm":"0000","antenna":"01","UID":"81c7fbf9","arduino":"4e6172616e6a6173202020202020202020202020202020202020202020202020202020202020202020202020202020202020
20202030393230353039303036333033303030303030303031363030314e4e4e4e4e4e4e4e4e4e4e3037303632303136672020"}
conectado
```

Figura 76 Monitor serie, lectura de productos y envío al servidor

Comprobamos cómo a medida que leemos las tarjetas, al refrescar el menú de productos de la aplicación, los productos van apareciendo.

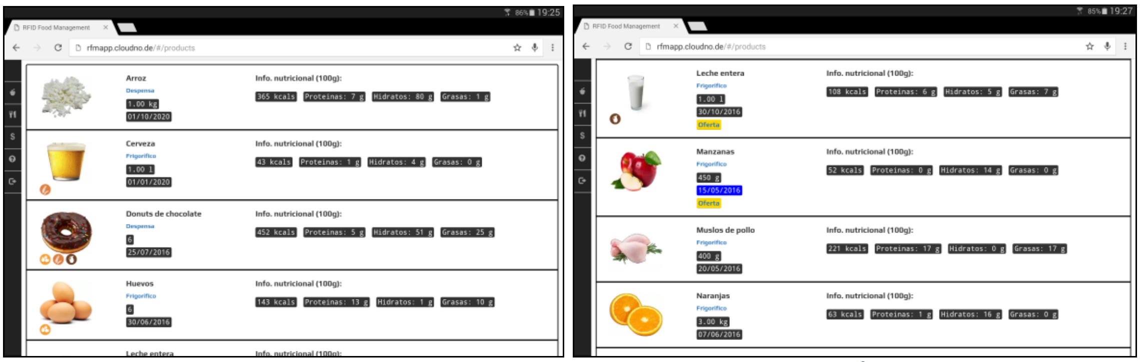

Figura 77 RFM App en una Tablet Samsung Galaxy Note 10.1, menú de productos

El usuario puede ver toda la información de sus productos, si estos tienen algún alérgeno, se indica mediante un icono debajo de la foto del producto.

Los productos que están próximos a caducar o han caducado, tienen su fecha de caducidad en color rojo, de este modo, de un vistazo, se puede saber qué productos van a caducar, y se podrán consumir antes de que esto ocurra, evitando el desperdicio de alimentos. Por último, se indica la información nutricional, el usuario puede tomar conciencia de qué productos son más convenientes para su dieta.

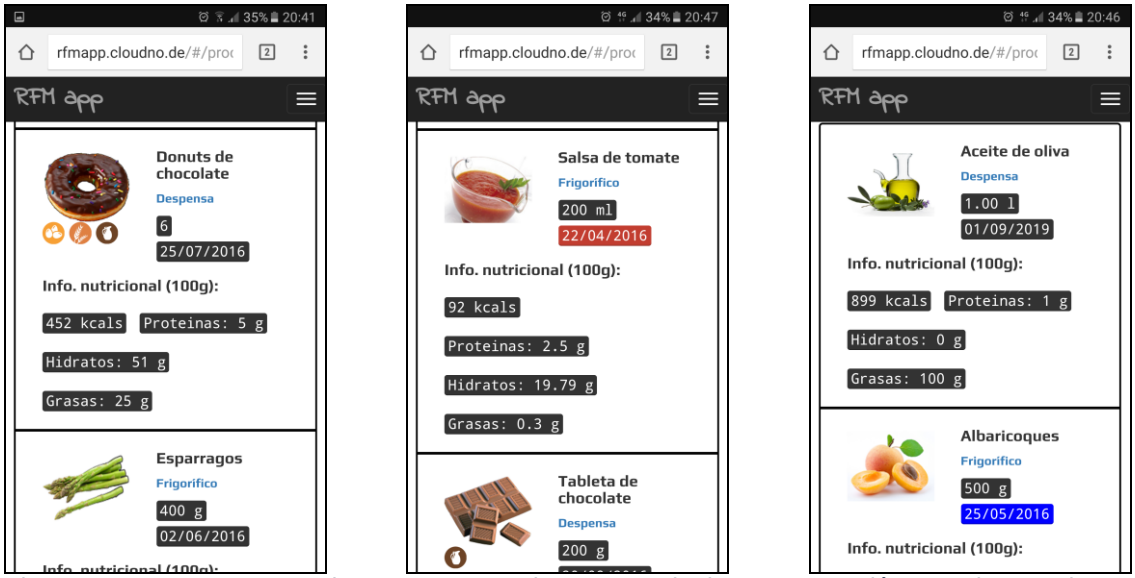

Figura 78 RFM App en smartphone Samsung Galaxy S6. Izquierda: Donut con alérgenos, huevo, gluten y lácteos. Centro: Salsa de tomate caducada, fecha de caducidad en rojo. Derecha: Fecha de envasado en azul

Muchas veces se desperdician alimentos porque no se sabe qué preparar con ellos, cómo combinarlos. La aplicación RFM App, dispone de un catálogo de recetas que pueden ser filtradas en base a varios criterios, aplicables individualmente, o de manera conjunta.

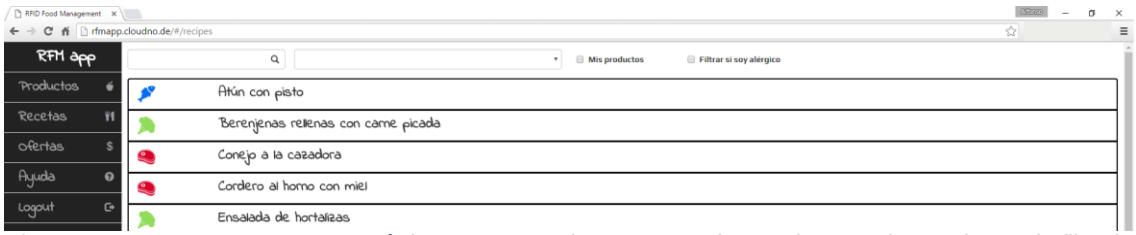

Figura 79 RFM App en un PC. Menú de Recetas, en la parte superior se observan las opciones de filtrado

En primer lugar, la aplicación tiene un cuadro de búsqueda, que permite filtrar por el nombre de la receta, o por el de alguno de sus ingredientes. También se pueden seleccionar recetas según a qué categoría pertenezcan: arroces, ensaladas, sopas, etc.

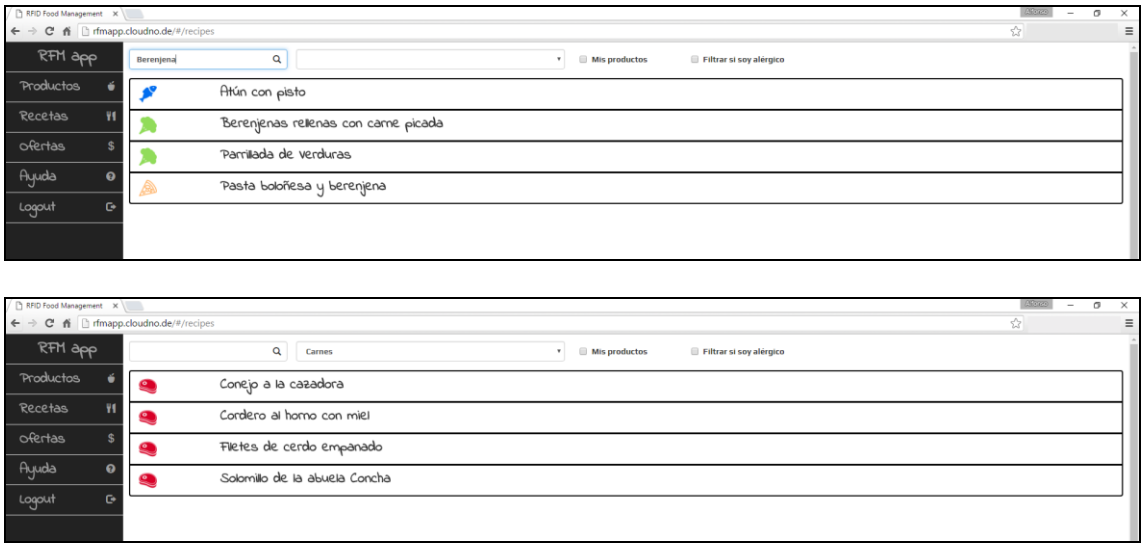

Figura 80 RFM App en un PC. Arriba: Filtrado de recetas que tienen berenjena. Abajo: Filtrado de recetas de la categoría carnes

Lo más interesante es que la aplicación puede recomendar recetas al usuario en base a los productos que tiene disponibles. Esto se vio en detalle en el apartado 5.3.

Por último, se pueden excluir aquellas recetas que contengan algún alimento al que el usuario sea alérgico.

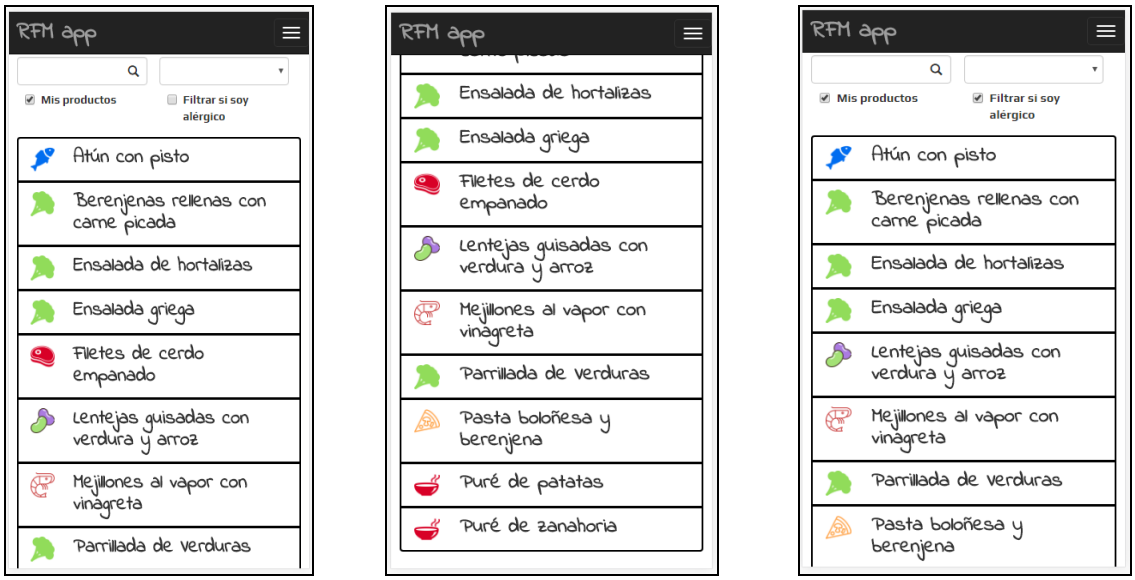

Figura 81 RFM App en un iPhone 6. Izquierda y Centro: recomendación de recetas que se pueden preparar con los productos disponibles. Derecha: Se excluyen las recetas a las que el usuario es alérgico

Para ver el detalle de una receta, basta con pinchar en ella. Cada receta tiene detalle de: presentación, preparación, ingredientes, e información nutricional y de alérgenos.

Figura 82 RFM App en un iPhone 6, detalle de receta. Izquierda: Presentación. Centro: Ingredientes de la receta, en rojo los ingredientes no disponibles, en verde los disponibles. Derecha: Las cantidades de cada ingrediente varían en función del número de personas que se indiquen.

Figura 83 RFM App en un iPhone 6. Izquierda: Preparación de una receta. Centro y Derecha: Información nutricional y de alérgenos, la receta contiene mostaza.

La aplicación cuenta con una interesante fuente de ingresos para el fabricante del sistema RFM. Los supermercados pueden contratar publicidad para mostrar sus ofertas en la aplicación. Todas las ofertas se pueden consultar desde el menú de ofertas, además, si tenemos algún producto que está en oferta, se muestra un enlace a la oferta en el detalle del producto. En último lugar, si no se dispone de algún ingrediente necesario para preparar una receta, y este está en oferta, se muestra un enlace a dicha oferta.

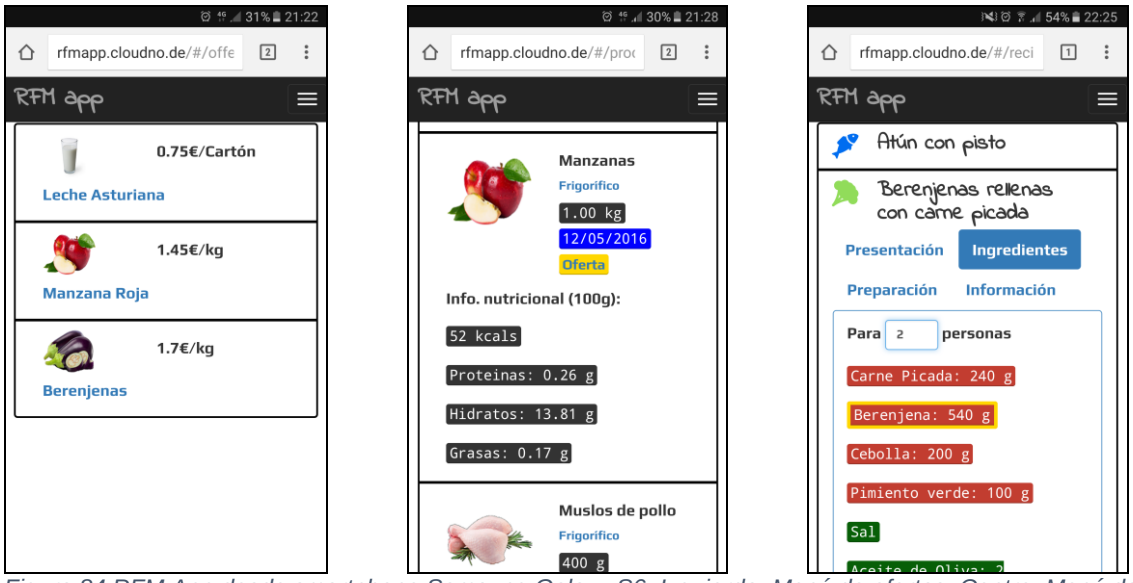

Figura 84 RFM App desde smartphone Samsung Galaxy S6. Izquierda: Menú de ofertas. Centro: Menú de productos, enlace a la oferta de manzanas desde los detalles de las manzanas. Derecha: Menú de recetas, enlace a la oferta de berenjenas desde detalle de ingredientes

6. Conclusiones

Gracias a este trabajo, se han podido ampliar y aplicar algunos de los conocimientos adquiridos durante la titulación. Ha sido muy interesante poder aprender tecnologías como *AngularJS*, o profundizar en el prototipado de circuitos electrónicos mediante *Arduino*, y en el estudio del RFID.

Del estudio de la tecnología RFID se desprende que es ideal para el desarrollo del *Internet* de las cosas. Este nuevo paradigma de computación tendrá un enorme impacto económico en los próximos años, supondrá el 11% del producto interior bruto mundial para el año 2025 según *McKinsey* [61].

El *Internet* de las cosas no es solo importante en lo económico, su efecto en la sociedad puede ser muy positivo en ámbitos como la alimentación, la sanidad, la seguridad vial o la eficiencia energética.

En este contexto, se fijó como objetivo desarrollar un sistema de gestión de alimentos en el hogar mediante RFID, con el fin de ayudar a sus usuarios a reducir el desperdicio de alimentos y llevar una dieta saludable.

En base a los resultados obtenidos, se puede concluir que los objetivos planteados inicialmente se han cumplido satisfactoriamente:

- El primer objetivo se ha llevado a cabo, ya que se ha desarrollado una aplicación *web* multiplataforma, en la que tanto su *front-end*, como *back-end* y base de datos están completamente operativos. Esto queda reflejado en los apartados 3.2 y 5.4. La aplicación alcanza todos los objetivos planteados: permite la consulta de productos disponibles, el acceso a ofertas, y la consulta de recetas. Prueba de esto es el apartado 5.4.

- El segundo objetivo también se ha conseguido, el algoritmo de recomendación de recetas funciona perfectamente, en el 100% de las ocasiones, como se probó en el apartado 5.3.

- El tercer y último objetivo se ha alcanzado, ya que se ha implementado un prototipo de sistema RFID con una placa *Arduino*, que permite grabar información de productos en tarjetas RFID, dicha información es leída y enviada al servidor de la aplicación posteriormente, tal y como se ha plasmado en los apartados 4.2 y 5.2.

En cuanto a la planificación del trabajo, ha habido algunas desviaciones respecto a la planificación inicial. La memoria se ha ido redactando en paralelo al desarrollo del sistema RFM, por lo que ha estado lista antes de tiempo. Como consecuencia, han faltado algunos días para realizar todas las pruebas antes del hito de entrega de la PEC3, y se tuvieron que hacer en los primeros días planificados inicialmente para la redacción de la memoria.

La metodología prevista inicialmente ha sido adecuada, el planteamiento ha sido muy práctico, lo que ha permitido avanzar en el desarrollo de un sistema que ha acabado por estar operativo, según lo esperado.

Este trabajo deja abiertas varias líneas de mejora muy interesantes:

- **Gestión de la dieta del usuario**, la aplicación podría pedir al usuario cuando se registre, su peso y altura, calcular su índice de masa corporal [62] en base a estas variables, y recomendarle una dieta. El usuario iría actualizando su peso, de manera diaria o semanal, y viendo gráficas de su evolución. Si además indica qué alimentos consume, la aplicación le indicaría si está tomando demasiadas grasas, pocas vitaminas, etc.
- **Estudio de patrones de consumo**, la aplicación, podrá encontrar los patrones de consumo de sus usuarios. Conociendo en qué día entra un producto al frigorífico o despensa, y los días que permanece en él, tendrá una estimación de con qué frecuencia consume cada producto el usuario. En base a esto, la aplicación puede avisar cuando un producto esté próximo a consumirse.
- **Monetización de patrones de consumo**, los patrones de consumo anteriormente mencionados, se pueden anonimizar y agregar, para venderlos a supermercados. Un supermercado podría estar interesado en saber qué productos son los más demandados en un determinado barrio, o por las personas dentro de rango de edades, con el objetivo de realizar ofertas personalizadas, y aumentar sus ventas.
- **Creación automática de la lista de la compra**, la aplicación sería capaz de crear una lista de la compra que contenga los productos que están próximos a consumirse, y los que se hayan consumido en los últimos días. A partir de esta propuesta de lista de la compra, el usuario podría añadir o quitar productos, y comprarlos automáticamente a través de *Internet*.
- **Ahorro en la lista de la compra**, una vez creada la lista de la compra, y previo a su compra a través de *Internet*, la aplicación podría hacer una comparativa de esta lista de la compra en diferentes supermercados de la zona, e indicar en cuál de ellos su coste es menor. También se podrían anunciar aquí las ofertas disponibles en la base de datos de la herramienta, previo pago de los supermercados interesados en anunciarse.

7. Glosario

AngularJs: *framework JavaScript* de desarrollo de aplicaciones web en el lado cliente.

Aplicación web: *Software* codificado en un lenguaje soportado por un navegador *web*, en el que es ejecutado.

Arduino: Plataforma de prototipado electrónico de código abierto.

Array: En programación, es una manera de almacenar un conjunto de objetos.

ASCII: *American Standard Code for Information Interchange*. Sistema de codificación de caracteres alfanuméricos que asigna un número del 0 al 127 a cada letra, número o carácter especial recogidos. El ASCII extendido permite hasta 256 caracteres distintos.

Back-end: Capa de acceso a datos en una aplicación.

Bootstrap: Framework de diseño web. Incluye componentes como botones, formularios, carruseles, etc.

BSON: *Binary* JSON. Formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos *Mongodb*.

CSS: *Cascading Style Sheets*. Lenguaje empleado para definir los estilos de un documento HTML.

Deployd: Plataforma de código abierto para diseñar y construir APIs para aplicaciones *web* y móviles.

Ethernet: Estándar de redes de área local para computadores con acceso al medio por detección de portadora, y con detección de colisiones.

Framework: Conjunto de herramientas de desarrollo que facilitan la creación de aplicaciones implementando en librerías las tareas más comunes.

Front-end: Capa de presentación e interacción con el usuario en una aplicación.

Github: Plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones *Git*.

HTML: *HiperText Markup Language*. Lenguaje empleado para definir la estructura y contenido de una página *web*.

HTTP: *Hypertext Transfer Protocol*. Protocolo que permite las transferencias de información en la *web*.

IDE: *Integrated Development Environment*. Entorno de programación consiste en un editor de código y un compilador.

IoT: *Internet of Things*. Interconexión digital de objetos cotidianos con Internet.

Javascript: Lenguaje de programación interpretado. Permite mejorar las interfaces de usuario y crear páginas *web* dinámicas. El código es interpretado por el navegador.

Jquery: Biblioteca de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

JSON: *JavaScript Object Notation*. Formato ligero para el intercambio de datos.

MPA: *Multiple page application*. Aplicación *web* tradicional, cada vez que necesita mostrar o mandar información solicita una nueva página al servidor.

Mongodb: Sistema de base de datos NoSQL de código abierto orientado a documentos.

MVC: Modelo Vista Controlador. Patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación, de la interfaz de usuario, y del módulo encargado de gestionar los eventos y las comunicaciones.

Navegador *web*: Aplicación empleada para acceder a páginas *web*.

NodeJS: Entorno de programación en la capa del servidor basado en el lenguaje de programación *Javascript*.

NoSQL: *Not Only SQL*. Sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales.

Protoboard: Tablero con orificios conectados eléctricamente entre sí, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables el prototipado de circuitos electrónicos.

RFID*: Radio Frequency IDentification*. Sistema de identificación, almacenamiento y recuperación de datos remoto mediante radio frecuencia.

RFM: *RFID Food Management*. Sistema de gestión de alimentos en el hogar desarrollado en el presente trabajo.

RFM App: *RFID Food Management App*. Aplicación *web* desarrollada en el presente trabajo.

RS-232: *Recommended Standard-232C*. Es un estándar para la conexión serial de señales de datos binarias entre un DTE (Equipo terminal de datos) y un DCE (Equipo de terminación del circuito de datos).

Script: Programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

Smartfridge: Frigorífico programado para controlar los productos almacenados en el mediante código de barras o escaneado *RFID*.

Smartphone: Teléfono móvil inteligente. Tienen capacidad para acceder a *Internet* e instalar aplicaciones de terceros.

SPA: *Single Page Application*. Aplicación *web* en la que los recursos se cargan dinámicamente según las acciones del usuario.

SPI: *Serial Peripheral Interface.* Estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

Tablet: Dispositivo de mayor tamaño que un teléfono inteligente, permite navegar por internet o ejecutar aplicaciones.

Wifi: *Wireless Fidelity*, mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

8. Bibliografía

[1] Royte, E. “One-Third of food is lost or wasted: what can be done“, 2014. URL http://news.nationalgeographic.com/news/2014/10/141013-food-waste-national-security-environment-science-ngfood/

[2] Gustavsson, J., & Cederberg,C. “Global food losses and food waste”, pp. 5-17, 2011. URL http://www.fao.org/fileadmin/user_upload/suistainability/pdf/Global_Food_Losses_and_Food_Waste.pdf

[3] World Health Organization. “Global Health Observatory (GHO) data, Overweight and obesity”, 2014. URL http://www.who.int/gho/ncd/risk_factors/overweight/en/index1.html

[4] World Health Organization. “Global Health Observatory (GHO) data, Overweight and obesity, Adults aged 18+”, 2014. URL http://www.who.int/gho/ncd/risk_factors/overweight_text/en/

[5] Butland, B., Jebb, S., Kopelman, P., McPherson, K., Thomas, S., Mardell, J., Parry, V. “Tackling obesities: future choices – Project report”, pp.39, 2014. URL https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/287937/07-1184x-tackling-obesities-future-choices-report.pdf

[6] Nicole L. Novak, M., Kelly D. “Role of Policy and Government in the Obesity Epidemic”, 2012. URL http://circ.ahajournals.org/content/126/19/2345.full

[7] EnginnersGarage. “RFID vs Barcodes”, 2016. URL http://www.engineersgarage.com/contribution/rfid-vs-barcodes

[8] R. Want, "An introduction to RFID technology," in *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25-33, Jan.-March 2006.

[9] Miles, S., Sarma, S., & Williams, J., eds. “RFID Technology and Applications”. Cambridge, GBR: Cambridge University Press, 2008. ProQuest ebrary. Web. 25 March 2016.

[10] David, C., “RFID 101: the next big thing for management”. Management Research News 29:4, pp.154-173. 2006.

[11] White, G., Gardiner, G., Prabhakar, G. P., & Abd Razak, A. “A comparison of barcoding and RFID technologies in practice”. *Journal of Information, Information Technology and Organizations*, 2, pp.119-132. ISSN 1557-1319. 2006.

[12] Weber, H., Romana H. “Internet of Things: legal perspectives”. *Berlin, Heidelberg : Springer Berlin Heidelberg.* ISBN 9781282927445. 2010.

[13] Samsung News. “Samsung Introduces an Entirely New Category in Refrigeration as Part of Kitchen Appliance Lineup at CES 2016”. 2016. URL

https://news.samsung.com/global/samsung-introduces-an-entirely-new-category-in-refrigeration-as-part-of-kitchen-appliance-lineup-at-2016-ces. Último acceso: 27/03/2016

[14] CBS Interactive Inc. “A close look at LG's Smart ThinQ LFX31995ST Refrigerator (hands-on)”. 2013. URL http://www.cnet.com/products/lg-smart-thinq-lfx31995st-refrigerator/. Último acceso: 27/03/2016

[15] Luo, S., Jin, J., & Li, J. (2009). “A smart fridge with an ability to enhance health and enable better nutrition”. Int. J. Multimedia Ubiquitous Eng, 4(2), pp.66-80. 2009.

[16] Rouillard, J. “The Pervasive Fridge. A smart computer system against uneaten food loss”. Seventh International Conference on Systems (ICONS2012), Feb 2012, Saint-Gilles, Reunion, pp.135-140, 2012.

[17] Hanshen, G. & Dong, W. "A Content-aware Fridge based on RFID in smart home for home-healthcare," *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, Phoenix Park, pp.987-990. 2009.

[18] Violino, B. “RFID System Components and Costs”. 2005. URL http://www.rfidjournal.com/articles/view?1336

[19] “Advantages and Disadvantages of Web, Native, and Hybrid Apps”. 2016. URL http://threefourteen.ie/blog/advantages-and-disadvantages-of-web-native-and-hybrid-apps/. Último acceso: 10/04/2016

[20] Shklar, L., & Rosen, R. “Web application architecture”. 2003

[21] Serge S. “Multi page web applications vs.single page web applications”. 2015. URL http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications Último acceso: 11/04/2016

[22] “What's the Difference Between the Front-End and Back-End?“. 2015. URL http://blog.digitaltutors.com/whats-difference-front-end-back-end/. Último acceso: 11/04/2016

[23] W3C. “HTML5 A vocabulary and associated APIs for HTML and XHTML”. 2014. URL https://www.w3.org/TR/2014/REC-html5-20141028/introduction.html#introduction. Último acceso: 01/05/2016

[24] W3C. “CSS3 Introduction”. 2016. URL http://www.w3schools.com/css/css3_intro.asp. Último acceso: 01/05/2016

[25] Mozilla Developer Network. “JavaScript”. 2016. URL https://developer.mozilla.org/es/docs/Web/JavaScript. Último acceso: 02/05/2016

[26] “¿Qué framework o librería de JavaScript elegir para mis desarrollos? 2015. URL https://carlosazaustre.es/blog/frameworks-de-javascript/. Último acceso: 05/05/2016

[27] Google. “AngularJS Doc”. 2016. URL https://angularjs.org/. Último acceso: 05/05/2016

[28] TutorialsPoint. “AngularJS - MVC Architecture”. 2016. URL http://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm. Último acceso: 06/05/2016

[29] Twitter. “Bootstrap 3 Doc”. 2016. URL http://getbootstrap.com/. Último acceso: 06/05/2016

[30] Stackoverflow. “What exactly is RESTful programming?” 2009. URL http://stackoverflow.com/questions/671118/what-exactly-is-restful-programming. Último acceso: 06/05/2016

[31] The Internet Society. “RFC2616, Hypertext Transfer Protocol - HTTP/1.1 ” pp.7. 1999.

[32] Deployd. Inc. “Deployd Doc”. 2016. URL http://deployd.com/. Último acceso: 08/05/2016

[33] Mongodb, Inc. “Mongodb Doc”. 2016. URL https://www.Mongodb.org/. Último acceso: 08/05/2016

[34] “NOSQL vs SQL. Diferencias y cuando elegir cada una”. 2015. URL http://blog.pandorafms.org/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/. Último acceso: 09/05/2016

[35] Node.js Foundation. “Node.js Doc”. 2016. URL https://nodejs.org/en/. Último acceso: 09/05/2016

[36] Deployd. Inc. “Installing Deployd”. 2016. URL http://docs.deployd.com/docs/getting-started/installing-deployd.html. Último acceso: 09/05/2016

[37] Mongodb, Inc. “BSON Types”. 2016. URL https://docs.Mongodb.org/manual/reference/bson-types/. Último acceso: 09/05/2016

[38] Mongodb, Inc. “Data Modeling Introduction”. 2016. URL https://docs.Mongodb.org/manual/core/data-modeling-introduction/. Último acceso: 10/05/2016

[39] Mongodb, Inc. “Data Model Design”. 2016. URL https://docs.Mongodb.org/manual/core/data-model-design/#data-modeling-referencing. Último acceso: 10/05/2016

[40] Nutrient Data Laboratory, Beltsville Human Nutrition Research Center "USDA National Nutrient Database for Standard Reference". 2016. URL https://ndb.nal.usda.gov/ . Último acceso: 10/05/2016

[41] GitHub, Inc. "Github". 2016. URL https://github.com/. Último acceso: 06/06/2016

[42] Palazón, A. "RFM". 2016. URL https://github.com/apalazon86/rfm

[43] Google. "AngularJS Conceptual Overview". 2016. URL https://docs.angularjs.org/guide/concepts#directive. Último acceso: 08/05/2016

[44] Jquery Foundation. "Jquery Doc". 2016. URL https://jquery.com/. Último acceso: 08/05/2016

[45] Google. "AngularJS Understanding Controllers". 2016. URL https://docs.angularjs.org/guide/controller. Último acceso: 08/05/2016

[46] Google. "AngularJS $Route". 2016. URL https://docs.angularjs.org/api/ngRoute/service/$route. Último acceso: 09/05/2016

[47] Arduino. "What is Arduino". 2016. URL https://www.arduino.cc/en/Guide/Introduction#. Último acceso: 09/05/2016

[48] Arduino. "Arduino Software". 2016. URL https://www.arduino.cc/en/Main/Software. Último acceso: 11/05/2016

[49] Arduino "Arduino Mega". 2016. URL https://www.arduino.cc/en/Main/ArduinoBoardMega2560. Último acceso: 11/05/2016

[50] WIZnet Co. W5100 Datasheet "". 2008. URL https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf

[51] SparkFun Electronics. "Serial Peripheral Interface (SPI)". 2016. URL https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. Último acceso: 01/05/2016

[52] MIFARE. "MIFARE RC522 Datasheet". 2014. URL http://www.nxp.com/documents/data_sheet/MFRC522.pdf

[53] MIFARE. "MIFARE Classic 1K Datasheet". 2011. URL http://www.puntoflotante.net/CARD-TAG-MF1S503.pdf

[54] ASCII. "Caracteres ASCII estándar". 2016. URL http://ascii.cl/es/. Último acceso: 16/05/2016

[55] Balboa, M. “MFRC522”. 2016. URL https://github.com/miguelbalboa/rfid. Último acceso: 30/05/2016

[56] Arduino. “Installing Additional Arduino Libraries”. 2016. URL https://www.arduino.cc/en/Guide/Libraries . Último acceso: 30/05/2016

[57] Palazón, A. “rfmArduino”. 2016. URL https://github.com/apalazon86/rfmArduino . Último acceso: 06/06/2016

[58] Codebender. “Serial Monitor”. 2016. URL https://codebender.cc/static/serialmonitor. Último acceso: 06/06/2016

[59] RapidTables.com. “Hex to ASCII text converter”. 2016. URL http://www.rapidtables.com/convert/number/hex-to-ascii.htm. Último acceso: 02/05/2016

[60] International Organization for Standardization. “ISO/IEC 14443-3”. 2001. URL http://jpkc.szpt.edu.cn/2007/sznk/UploadFile/biaozhun/iso14443/14443-3.pdf.

[61] McKinsey Global Institute. “Unlocking the potential of the Internet of Things”. 2015. URL. http://www.mckinsey.com/business-functions/business-technology/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world.

[62] Organización de Consumidores y Usuarios. “Índice de masa corporal”. 2016. URL http://www.ocu.org/alimentacion/adelgazar/calculadora/indice-de-masa-corporal. Último acceso: 03/06/2016

9. Anexos

9.1 Manual de usuario de RFM App

9.1.1 Registro, acceso, barra de navegación

La aplicación RFM App está accesible desde http://rfmapp.cloudno.de/ , esta es la pantalla de inicio:

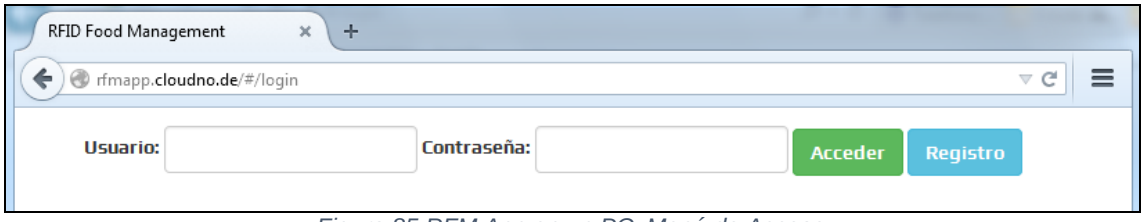

Figura 85 RFM App en un PC. Menú de Acceso

Si es la primera vez que se accede a la aplicación, hay que pulsar sobre el botón “Registro”, lo que abrirá el formulario de registro:

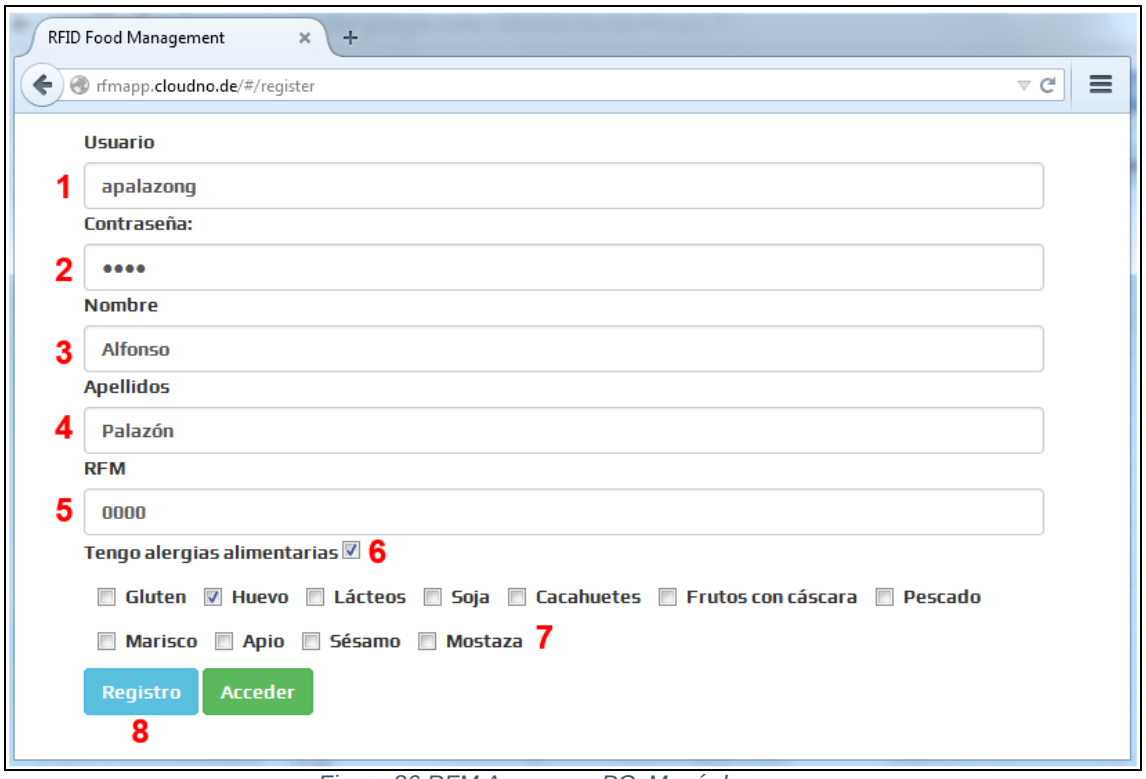

Figura 86 RFM App en un PC. Menú de acceso

En el formulario de registro, se debe indicar el usuario (1) y contraseña (2) con los que se accederá a la aplicación, y el nombre (3) y apellidos (4) del usuario. Además, el fabricante dará un código llamado código RFM (5), mediante el que el usuario quedará relacionado con el sistema RFM que ha adquirido.

Por último hay una casilla que se debe marcar si el usuario tiene una alergia alimentaria (6), en este caso se muestran diferentes alérgenos (7) que el usuario deberá marcar según proceda. Tras pulsar el botón “Registro” (8),

somos devueltos a la pantalla inicial. En la pantalla inicial se indica el usuario y contraseña, y se accede a la aplicación, por defecto al menú de productos, que se puede ver a continuación:

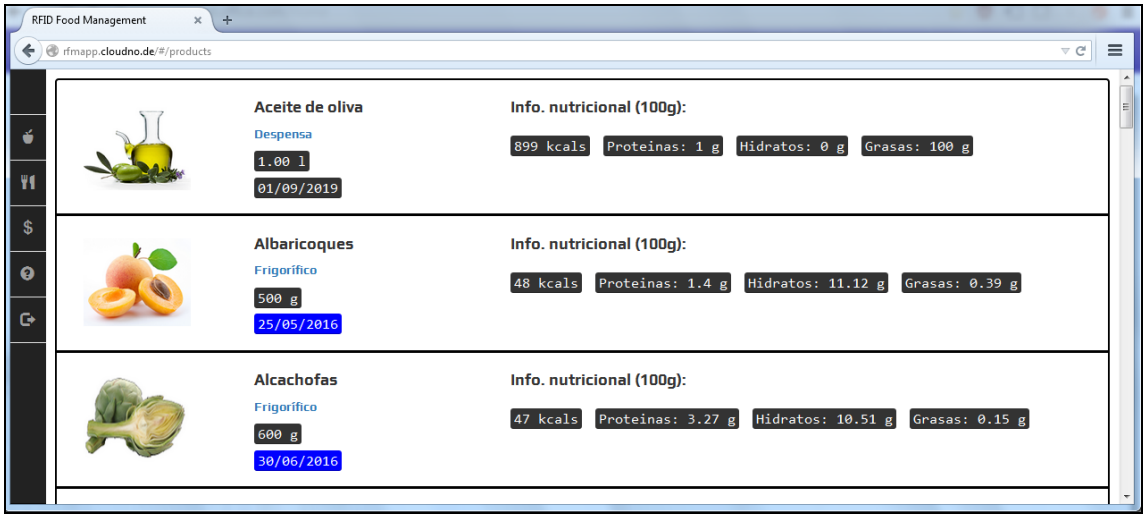

Figura 87 RFM App en un PC. Menú de productos

En la parte izquierda se tiene una barra lateral con una serie de iconos, pulsando sobre ellos se navega por los distintos menús de la aplicación. La barra lateral está siempre expandida en monitores de mayor resolución. En este caso, se tiene un monitor con poca resolución, se expande al pasar el ratón sobre ella:

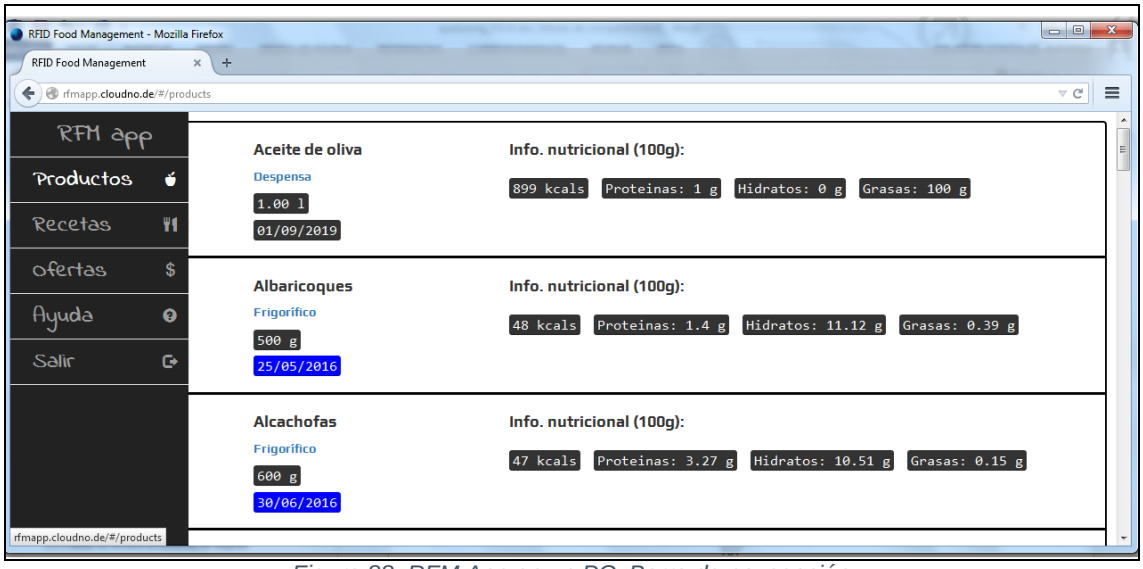

Figura 88 RFM App en un PC. Barra de navegación

En el caso de las pantallas de menor resolución, como la de un *smartphone*, la barra de navegación desaparece del lateral, y es sustituida por un botón en la esquina superior derecha, tras pulsar en el botón se muestra la barra de navegación completa:

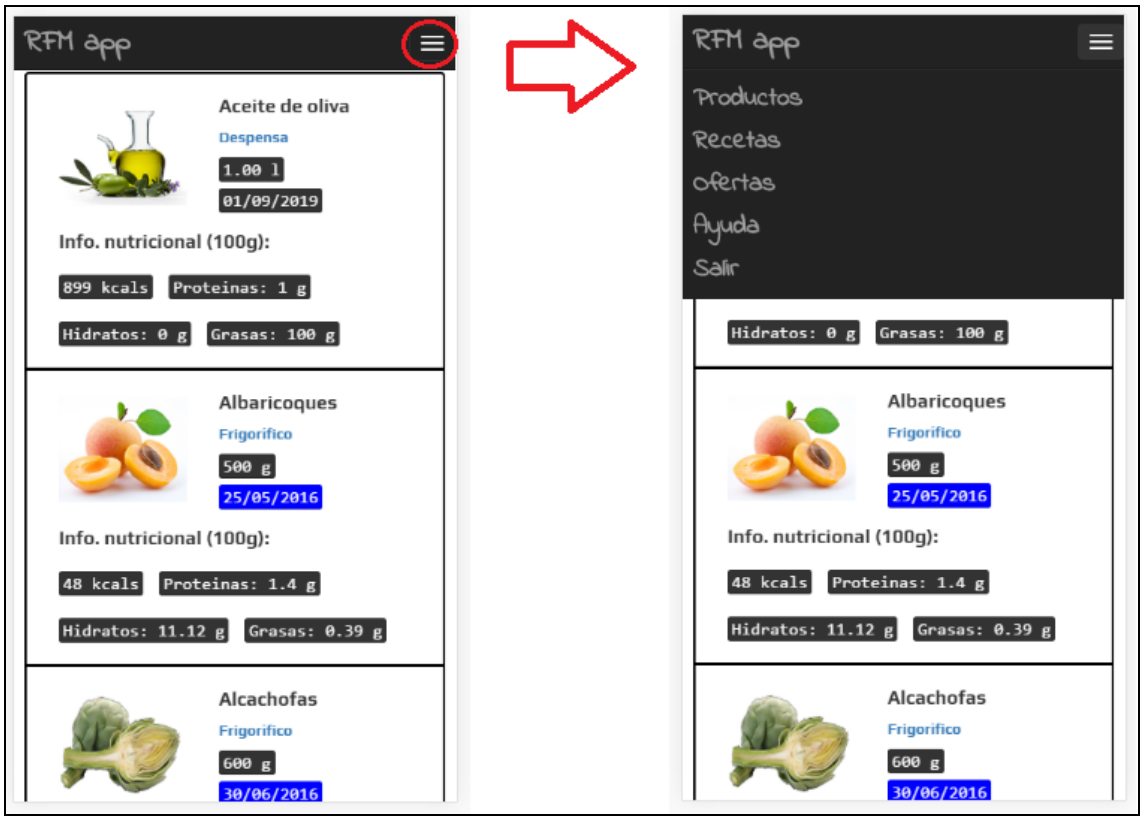

Figura 89 RFM App en un iPhone 6. Barra de navegación

9.1.2 Productos

El menú de productos aparece por defecto al acceder a la aplicación, también es accesible desde cualquier otro menú a través de la barra de navegación, pulsando sobre el icono . Este menú contiene todos los productos de que dispone el usuario en su frigorífico y despensa. El detalle de cada producto es el siguiente:

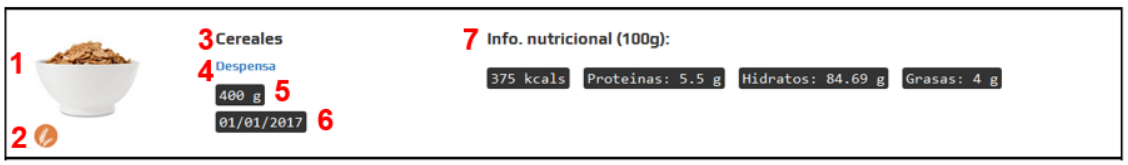

Figura 90 RFM App en un PC. Detalle de un producto

En la parte izquierda se tiene una fotografía del producto (1), justo debajo, los alérgenos (2) que contiene, el listado completo se muestra más tarde.

En el centro, se tiene el nombre del producto (3), el lugar donde está ubicado (4), que puede ser frigorífico o despensa, la cantidad de producto (5), y una fecha (6). La fecha tiene diferente significado dependiendo del color de fondo, si el fondo es azul, la fecha es aquella en la que el producto fue envasado. Si el fondo es negro o rojo, la fecha mostrada es la de caducidad, el fondo rojo rojo indica que el producto ha caducado o va a hacerlo en los próximos dos días.

Por otra parte, se muestra la información nutricional (7) en 100 gramos de producto, se indica el aporte energético en kcals, los gramos de proteína, hidratos de carbono, y grasas.

Los iconos de los diferentes alérgenos son los siguientes:

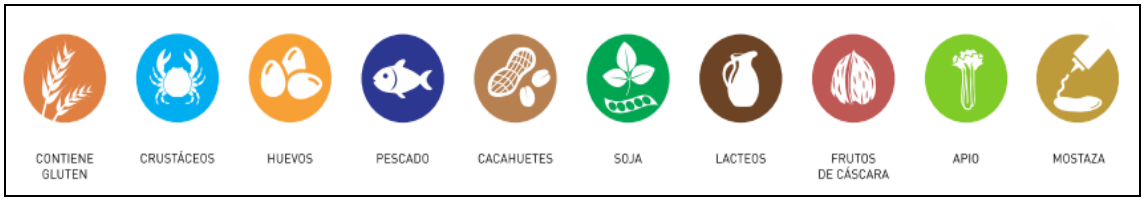

Figura 91 Alérgenos de productos y recetas

Por último, si se dispone de alguna oferta para uno de los productos de los que disponemos, se muestra un enlace a la oferta debajo de la fecha de caducidad/envasado:

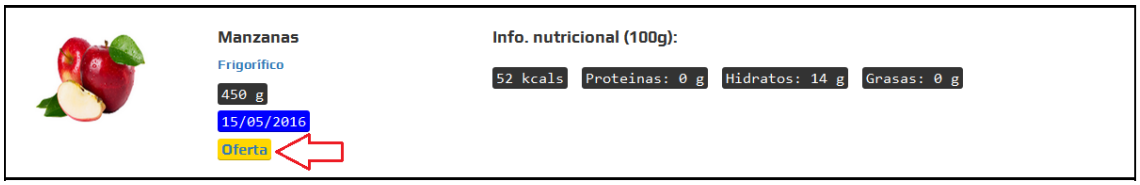

Figura 92 RFM App en un PC. Enlace a una oferta desde el menú de productos

9.1.3 Recetas

Al pulsar sobre el icono  de la barra de navegación, se abre el menú de recetas, que vemos a continuación:

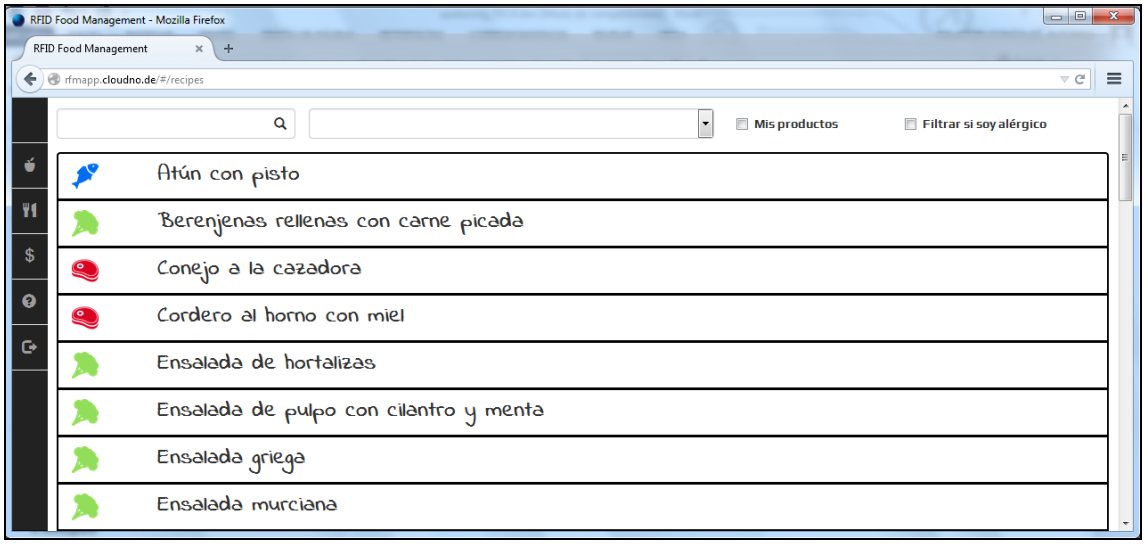

Figura 93 RFM App en un PC. Menú de recetas

En la parte superior se dispone de cuatro controles para realizar la recomendación y filtrado de recetas, esto será explicado más tarde. Debajo de estos controles, se muestra el listado de recetas, cada receta tiene un icono que indica el tipo de receta del que se trata, junto al nombre de la misma:

Icono	Tipo de receta
	Arroces
	Carnes
	Ensaladas y Verduras
	Legumbres
	Mariscos
	Pastas y Pizzas
	Pescados
	Postres
	Setas y Hongos
	Sopas y Cremas

Figura 94 Tipos de receta

Para ver el detalle de una receta, se pulsa sobre ella:

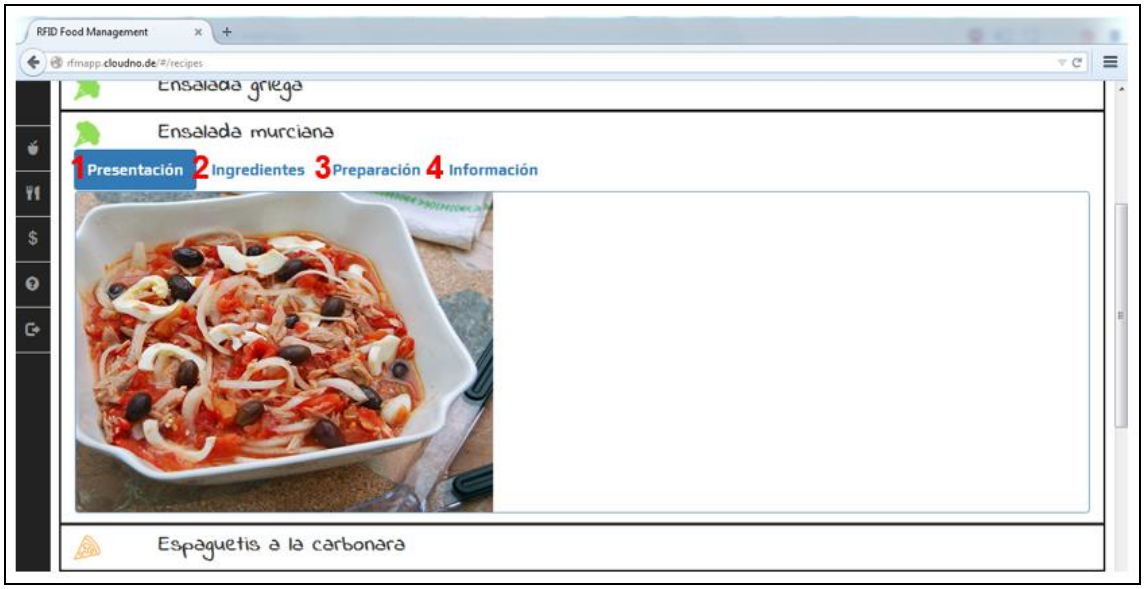

Figura 95 RFM App en un PC. Detalles de una receta

El detalle se muestra desde desde cada una de las cuatro pestañas mostradas en la imagen anterior.

La primera pestaña (1), muestra una fotografía con la presentación de la receta, desde el resto de pestañas se accede a los ingredientes que componen la receta (2), a cómo se prepara la receta (3), y por último a su información nutricional y de alérgenos (4).

Tras hacer pulsar sobre la pestaña de ingredientes se muestra lo siguiente:

Figura 96 RFM App en un PC. Ingredientes de una receta

En la parte superior, se muestra para cuántas personas es la receta (1), este número se puede variar, y las cantidades de cada ingrediente se modificarán de manera proporcional como podemos observar en la siguiente captura:

Figura 97 RFM App en un PC. Ingredientes de una receta, se modifica el número de personas, y se cambian las cantidades de cada ingrediente de manera automática

Debajo del número de personas, se tiene el listado de ingredientes, y en qué cantidad son necesarios para preparar la receta. Si un ingrediente se muestra de color rojo (2) es porque entre nuestros productos disponibles no se encuentra este ingrediente, o no se encuentra en la cantidad necesaria para el número de personas seleccionado. Por el contrario, si se dispone de la cantidad necesaria de un determinado ingrediente, este se muestra en color verde (3).

Si alguno de los ingredientes no disponibles está en oferta, se muestra el ingrediente con fondo rojo y borde dorado, si se pulsa el ingrediente, se abre la oferta en una nueva pestaña del navegador:

Figura 98 RFM App en un PC. Enlace a una oferta desde los ingredientes de una receta

La tercera pestaña muestra las instrucciones de preparación de la receta:

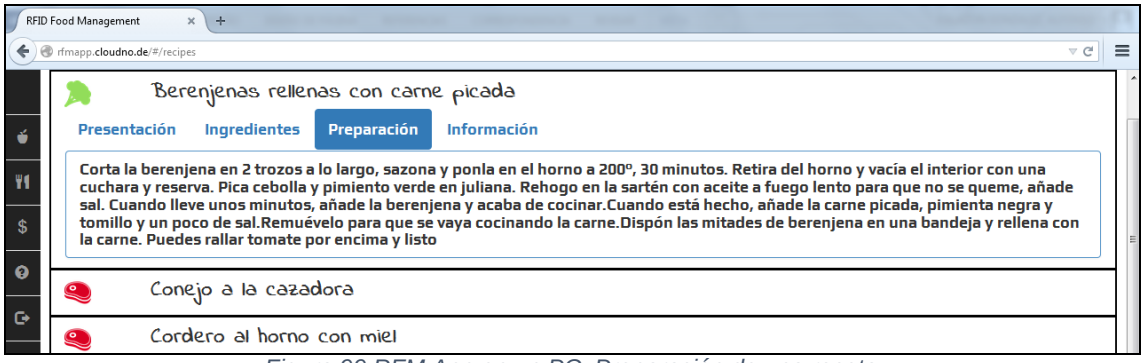

Figura 99 RFM App en un PC. Preparación de una receta

La cuarta y última pestaña contiene la información nutricional y de alérgenos:

Figura 100 RFM App en un PC. Información nutricional y de alérgenos de una receta

En la parte superior (1) se muestra la información nutricional de una ración, su aporte energético en kcals, y los gramos de proteínas, hidratos de carbono y grasas. Si la receta contiene algún alérgeno, se muestra en la parte inferior (2), en caso contrario, este apartado se oculta.

Las recetas se pueden filtrar, en base a diferentes criterios, en la parte superior del menú de recetas hay cuatro controles para realizar este filtrado:

Figura 101 RFM App en un PC. Controles de filtrado de una receta

El primer control (1) permite introducir un texto de búsqueda, si el texto aparece en el nombre de la receta, o en el de algún ingrediente de la misma, se mostrará dicha receta. En el siguiente ejemplo se muestran todas aquellas recetas que contienen berenjena:

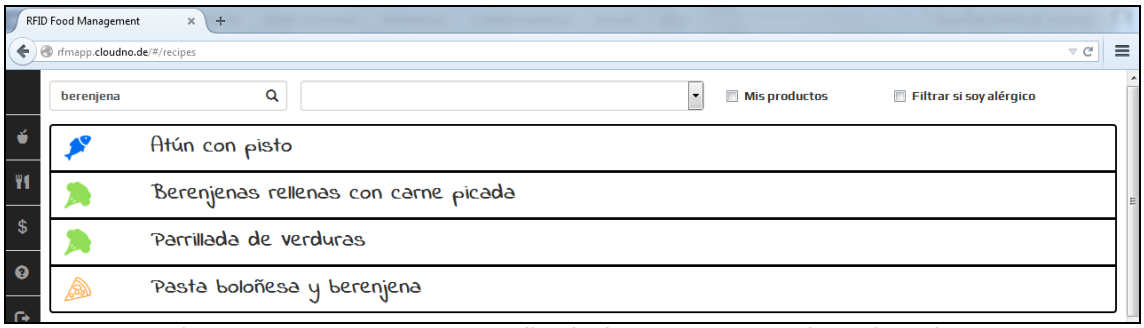

Figura 102 RFM App en un PC. Filtrado de recetas que contienen berenjena

El segundo control es una lista desplegable que muestra todos los tipos de receta. Si se deja en blanco se muestran las recetas de cualquier tipo, si se elige un valor del desplegable, se muestran solamente las recetas cuyo tipo coincida con el valor seleccionado. En el siguiente ejemplo se muestran solo las recetas del tipo “Pastas y Pizzas”:

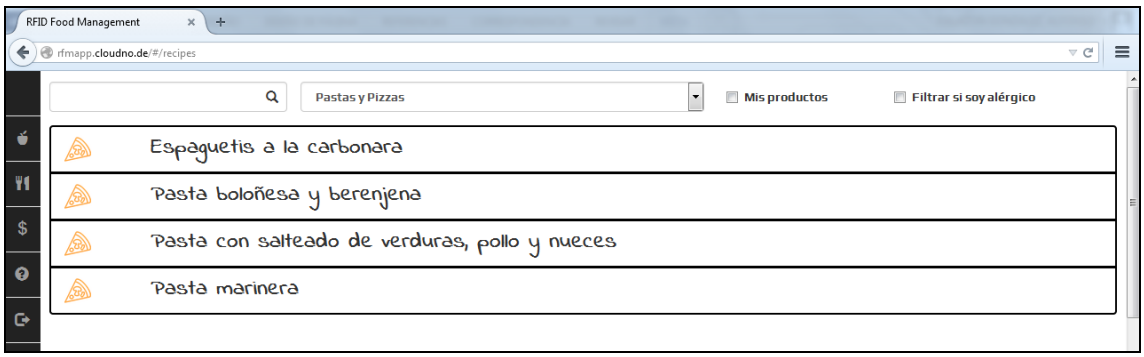

Figura 103 RFM App en un PC. Filtrado de recetas del tipo “Pasta y Pizzas”

Cuando se active el tercer control, la casilla “Mis productos”, se mostrarán solo aquellas recetas que podemos preparar con los productos disponibles. Toda receta da un peso a cada ingrediente, si los pesos de los ingredientes disponibles son mayores a los no disponibles, se muestra la receta.

En el siguiente ejemplo se muestran las recetas que se pueden preparar con los productos de que se dispone:

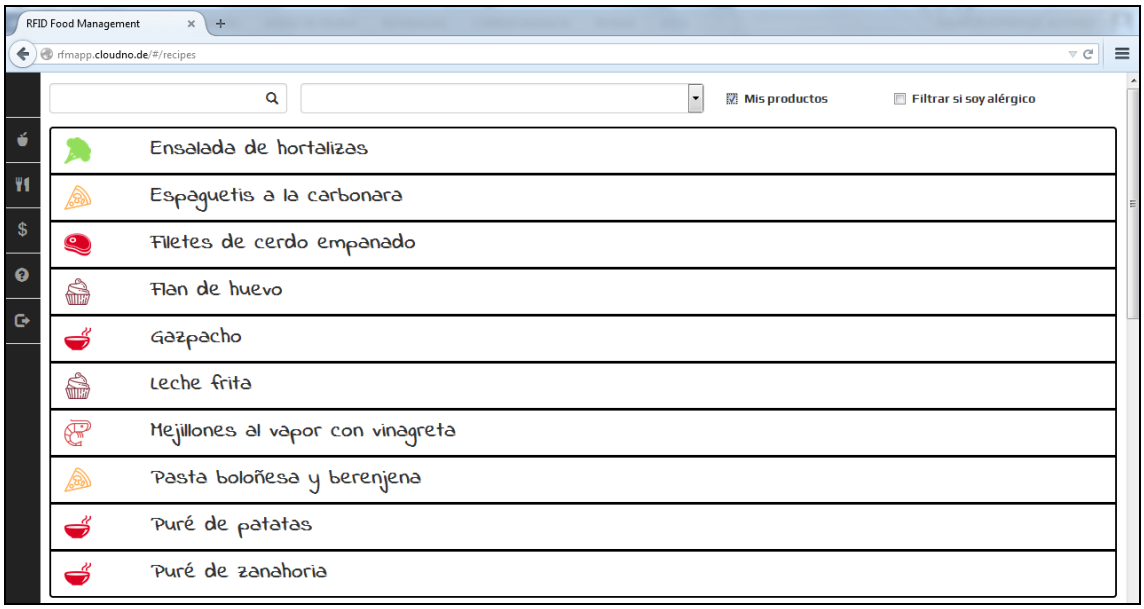

Figura 104 RFM App en un PC. Filtrado de recetas en base a los productos disponibles

El cuarto y último control, la casilla “Filtrar si soy alérgico”, permite eliminar del listado de recetas aquellas que contienen algún ingrediente al que el usuario actual es alérgico.

En el siguiente ejemplo, se van a mostrar todas las recetas que contienen huevo:

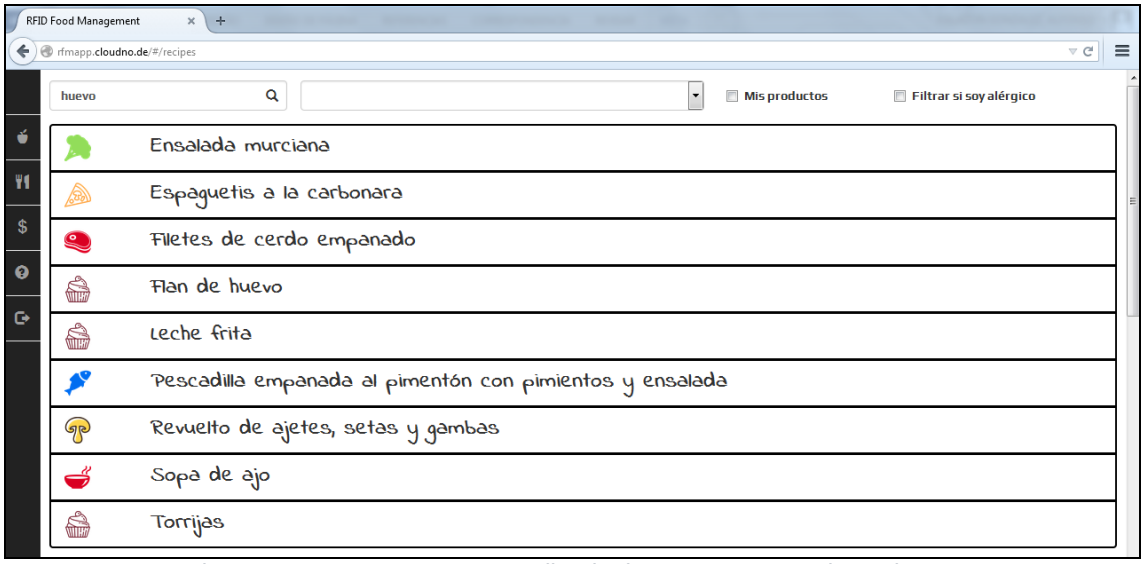

Figura 105 RFM App en un PC. Filtrado de recetas que contienen huevo

El usuario actual indicó cuando se registró que es alérgico al huevo, por lo que si marca la casilla “Filtrar si soy alérgico”, no se mostrará ninguna receta:

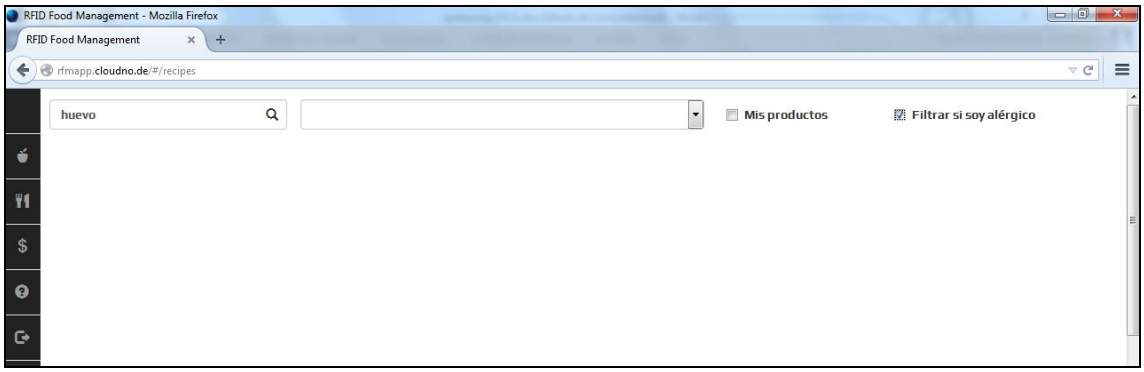

Figura 106 RFM App en un PC. Filtrado de recetas que contienen huevo y a las que no se es alérgico

Los cuatro controles de filtrado se pueden emplear en cualquier combinación para conseguir el filtrado de recetas deseado.

9.1.4 Ofertas

Al pulsar sobre el icono  de la barra de navegación, se accede a las ofertas. En este menú se muestran todas las ofertas de productos que se tienen en la aplicación:

Figura 107 RFM App en un PC. Menú de ofertas

Para cada oferta se muestra la imagen del producto en oferta (1), el precio por unidad (2), y una descripción de la oferta (3), la descripción de la oferta es un enlace, por lo que si se pulsa en ella, se abrirá la oferta en una nueva pestaña del navegador.

9.1.5 Ayuda

Al pulsar sobre el icono  de la barra de navegación, se accede a la ayuda de la aplicación.

9.1.6 Salir

Al pulsar sobre el icono  de la barra de navegación se sale de la aplicación.