E.T.S. de Ingeniería Industrial,

Informática y de Telecomunicación

Sistema informático de gestión de productos para superficies comerciales

Grado en Ingeniería Informática

Trabajo Fin de Grado

Nombre del autor: Alvaro Carrero Mariezcurrena

Nombre del director/es: Eduardo Alfaro Larragueta

José Javier Astrain Escola

Pamplona, fecha de defensa: 30 de junio de 2016

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Quisiera agradecer a Eduardo Alfaro Larragueta y a José Javier Astrain Escola haber dirigido este TFG, así como la disposición y ayuda recibidos para su realización.

También me gustaría agradecer el apoyo recibido durante todos estos años de carrera a mis padres y hermana, a mi pareja Patricia Jiménez, y a mis compañeros de promoción, de manera especial a Alberto Ginesta, Irene Gómez y Daniel Díez.

Resumen

Esta aplicación tiene como objetivo mejorar el sistema de gestión de una superficie comercial. Para ello se proporciona una interfaz al cliente y otra a los empleados.

El cliente puede, mediante la aplicación, constituir un listado de una compra que quiera realizar. Al realizarla, los productos de la lista que hayan sido comprados se borraran automáticamente y los que no, permanecerán.

Los reponedores pueden ver en tiempo real en qué lugares es necesario reponer y se les permite introducir incidencias de robos y pérdidas de productos.

Por último a los gerentes se les permite ver la cantidad de cada producto existente en la superficie comercial, pudiendo realizar pedidos desde ese mismo lugar. Además pueden dar de alta/baja productos, editarlos y dar de alta a nuevos empleados. También se les permite ver información relativa a ventas e ingresos de todos sus productos.

Palabras clave

Supermercado, superficie comercial, carro, carrito, lista, compra, producto, cliente, automático.

Índice

Introducción

Visión general

A día de hoy las superficies comerciales gastan mucho dinero y recursos en la gestión de sus negocios, y no de la forma más eficiente posible. Se invierte mucho tiempo en recopilar estadísticas de los productos más vendidos, el margen de beneficio que da cada producto, la inversión en personal, etc.

Este apartado solamente teniendo en cuenta lo que concierne exclusivamente a la gestión de recursos de la superficie comercial, porque además, seguro que muchos clientes se han encontrado con colas inmensas a la hora de realizar los pagos en el cajero, con que el producto que está buscando está agotado, o simplemente que queden en el almacén pero nadie haya repuesto las estanterías. Y en los tiempos tecnológicos tan avanzados en los que vivimos, no es posible no dar una alternativa al cliente para evitar que se siga haciendo la lista de la compra con papel y bolígrafo.

Viendo esta problemática, este proyecto está pensado para dar un mejor servicio al cliente evitando que se agoten los productos o que no se hayan repuesto y dándole la opción de elaborar una lista con los productos existentes en esa superficie comercial concreta.

Además se gestionarán todas esas compras de manera que los reponedores puedan conocer que productos son los que tienen que reponer y los encargados de realizar los pedidos tengan la información real al momento del estado del stock de los productos, pudiendo realizar los pedidos de forma automática.

Objetivos

Se pretende desarrollar una aplicación que ayude al cliente a realizar la compra. El cliente podrá definir su propia lista de la compra con los productos reales disponibles y la aplicación le guiará por el supermercado hasta los productos que desea adquirir. Además, ofrecerá al cliente una serie de servicios de valor añadido como información sobre la temperatura de los productos, los precios de cada uno de ellos, el total que supondría la lista, etc. Se pretende ofrecer una herramienta que facilite al cliente su compra de una manera no invasiva y bajo demanda.

Esta aplicación también dispone de un apartado para empleados en la que se pretende, después de gestionar los productos comprados por el cliente, dar información detallada a los trabajadores según su puesto de trabajo, como por ejemplo, los lugares donde haría falta reponer, el stock de los productos para poder realizar nuevos pedidos, ganancias y beneficios de los diferentes productos, etc.

Por tanto se trata de realizar un sistema que gestione los bienes de un supermercado. Para ello hay 2 partes diferenciadas: la parte centrada exclusivamente en el cliente y la que se centra en la gestión de los recursos del supermercado, pero de los que también se beneficia el cliente. Este sistema necesitaría una instalación hardware en la superficie comercial que se explicará a continuación.

Lo que se pretende es la implementación de una aplicación que permita a un cliente registrarse en ella. Una vez esté el cliente identificado, realice a través de la aplicación una lista de la compra seleccionando diferentes cantidades de diferentes productos.

Una vez realizada la lista, el cliente accedería al supermercado y cogería un carrito de la compra. En estos carritos se instalaría un arduino en cada uno de ellos que permite reconocer cada uno de los productos que se introducen en él. Un arduino es una pequeña placa, un pequeño sistema de procesamiento, que hará de controlador entre el dispositivo del cliente, el contenido del carrito y el sistema encargado de la gestión haciendo que éstos se comuniquen de forma que la información sea consistente. Este

arduino lo primero que haría sería conectarse al dispositivo móvil del cliente, por Bluetooth por ejemplo, para así identificar a la persona que va a realizar la compra. Bluetooth es una tecnología que permite la transmisión de datos entre diferentes dispositivos por radiofrecuencia, sin necesidad de estar conectados físicamente con cable, por lo que se antoja como una solución idónea para la aplicación.

Una vez identificado el cliente, éste iría introduciendo productos en el carro. El arduino cada vez que detectase un producto lo comunicaría al punto de acceso y éste lo almacenaría en una tabla en memoria, ya que si cada vez que se inserta un producto en el carro hay que acceder a tabla, contando con los numerosos clientes de una superficie comercial puede que no fuese viable.

Cuando el cliente pase por caja, ese contenido en la tabla en memoria se volcaría a un histórico de base de datos, en el que se compararían los productos comprados con

los productos en la lista de la compra, y se eliminarían los comprados y se mantendrían los que no.

A su vez los reponedores tendrían acceso al estado de las ubicaciones respecto a cantidad de productos se refiere, lo que produciría más eficiencia en sus horas de trabajo.

Por otro lado permitiría a los encargados del supermercado tener información real sobre las ventas, tener conocimiento de la cantidad de producto que queda en el almacén en tiempo real de manera que puedan automatizar pedidos bajo algún parámetro, como por ejemplo, que se haga un pedido cuando las unidades bajen de un valor concreto.

Estado del arte

Existen diferentes aplicaciones que permiten realizar una lista de la compra pero se piensa que no son del todo prácticas para el usuario. Las aplicaciones de hoy en día permiten la realización de listas de la compra con productos genéricos, es decir, al no ser aplicaciones asociadas a ninguna empresa, no tienen la posibilidad de ofrecer diferentes marcas del mismo producto, ya que pueden no existir en todos los supermercado. Eso hace que la aplicación a desarrollar mejore en este aspecto a las ya existentes, porque hay mucha gente que de un producto quiere una marca concreta, no cualquiera. En algunas aplicaciones también es posible añadir a la lista algún producto que se tenga pero que se quiera comprar más unidades pasando el código de barras del producto por delante de la cámara del móvil. Se comprueba que no está muy avanzado ya que multitud de productos de marcas conocidas no son encontradas al pasar el código de barras.

Por otro lado, una vez hecha la compra, es el propio usuario quien tiene que eliminar a mano los productos comprados de la lista por lo que si al usuario se le olvida

es probable que en la siguiente compra vuelva a comprar productos que ya tenía. Las acciones que se realizan automáticamente dan al usuario un punto más de satisfacción.

Respecto a la gestión de la superficie comercial, las empresas suelen utilizar ERPs, software de gestión. Los ERP son sistemas de información gerenciales que manejan toda la información relativa a la producción, distribución, bienes, etc. Generalmente son aplicaciones contrastadas y de cierta fiabilidad pero con unas licencias bastante costosas para las empresas. Este software está pensado para personas con responsabilidad dentro de una empresa por lo que no todos los empleados tendrían acceso a él.

Por parte de los reponedores no existe software para el mejor desenvolvimiento de su trabajo.

Contexto tecnológico

En este apartado se hablará sobre que herramientas y tecnologías se han utilizado en el proyecto, y que nos permite realizar cada tecnología.

El software utilizado consiste en Visual Studio 2015 ya que es una herramienta muy integrada en las empresas del sector TIC, especialmente las dedicadas a utilizar .NET. Con él se ha realizado el desarrollo de la aplicación. También se ha utilizado Team Foundation Server ya que está disponible con el Visual Studio, por lo que se evita tener que integrar con cualquier otro software. Ha permitido realizar la gestión de versiones de la aplicación durante el periodo de desarrollo de la aplicación. Para la gestión de los datos se ha utilizado SQL Server 2014 Management Studio. Al ser software de la misma marca, se asegura una buena integración entre herramientas. Ha permitido el almacenamiento y la gestión de la base de datos, la cual se ha creado con el editor Notepad++.

A partir de ahora se verán las tecnologías utilizadas para la realización del proyecto y las razones de la elección de las más importantes:

AngularJS

AngularJS es un framework de JavaScript que ha permitido realizar la SPA (Single Page Application). Se ha elegido angular porque permite construir una aplicación siguiendo un patrón MVC, una buena forma de estructurar el código, y porque hace que la lógica de negocio y la manipulación del DOM queden totalmente separadas. Además ha permitido llevar muchos de los servicios al lado del cliente, por lo que hace que la aplicación web sea mucho más ligera. Se ha utilizado también porque permite el 2 way binding, que hace que se los cambios en los datos en una vista hagan que cambie también en el controlador y viceversa.

.NET

Se ha escogido .NET por su alto grado de implantación en las empresas y por su amplia utilización en aplicaciones web similares. Con .NET se ha utilizado para el acceso a los datos ADO.NET Entity Framework, mientras que también se ha utilizado Web API para crear la API REST (servicios HTTP).

BootStrap

Utilizar BootStrap ha permitido realizar una aplicación web responsive, que asegure su correcta visualización en otros dispositivos móviles, como pueden ser móviles o tablets. Este framework permite que la interfaz de la aplicación no varíe dependiendo del dispositivo en el que se esté usando, por lo que permite a los clientes poder acceder a ella desde cualquier aparato electrónico.

Otras tecnologías

Se ha utilizado SQL como lenguaje de acceso a base de datos propio de SQL Server, C# como lenguaje de programación orientado a objetos, ya que está estandarizado como parte del framework .NET, Entity Framework y Web API para el acceso a datos y como API para servicios web respectivamente, y CSS para darle estilos a la aplicación.

Base de Datos

En este apartado se verá cómo se ha pensado y construido la base de datos. Para ello lo primero ha sido realizar un análisis semántico con el cual poder describir como tendría que ser la BBDD. Analizando la semántica de la BBDD se pueden obtener las entidades necesarias para poder cubrir el problema y los atributos necesarios en cada una de ellas. Una vez se tienen todas las entidades necesarias, habrá que establecer el tipo de relaciones entre ellas de manera que se le dé sentido al problema que se quiere resolver. Cuando se tenga todo esto hecho ya se tendrá el modelo Entidad-Relación, el cual se tendrá que pasar al modelo relacional para tener la base de datos a punto para poder empezar con la implementación de la aplicación.

Análisis semántico

- Cada cliente puede introducir varios productos en su lista de la compra, mientras que un producto puede estar en varias listas de varios usuarios.
- De la misma manera un cliente puede comprar uno o varios productos y cada producto puede ser comprado por uno o más clientes.
- De cada lista de la compra y de cada compra interesa conocer la cantidad de producto y el total que ello supone.
- De cada cliente se guardará la información básica personal además de un nombre de usuario y contraseña que se utilizará para la identificación de cada uno de ellos en la aplicación.
- De cada producto interesa saber si el producto está disponible para el público, la cantidad que se dispone de ese producto en la superficie comercial, la cantidad que entra de ese producto en cada balda y los precios tanto de compra como de venta para poder obtener los márgenes de beneficio de cada uno de ellos.

- Cada producto tiene un baremo diferente, mientras que se puede utilizar un mismo baremo para varios productos.

- Cada baremo servirá para saber hasta qué cantidad de producto se tiene que pedir al proveedor cuando se dé un hecho puntual, por tanto para cada uno de ellos interesa saber hasta qué cantidad hay que pedir al proveedor.

- Cada producto pertenece a una categoría diferente mientras que en cada categoría puede haber más de un producto.

- De cada categoría interesa la descripción de la misma, para poder categorizar los productos.

- En cada ubicación del supermercado sólo puede haber un único producto mientras que cada producto puede ponerse en varias ubicaciones.

- Cada ubicación se conocerá por su sección, número de pasillo, estantería y balda. Además es interesante conocer la cantidad de producto que hay en cada balda, para poder tener información y poder transmitírsela al reponedor.

- De los empleados se quiere conocer la información básica, el tipo de puesto que desempeña (gerente o reponedor), y tal y cómo se ha pedido con el cliente, un nombre de usuario y contraseña para poder identificar a cada uno de ellos.

- Cada empleado puede meter incidencias tanto de robos como de pérdidas de varios productos. A su vez, puede haber varias incidencias de un mismo producto por parte de varios empleados. De cada una de las incidencias se quiere conocer la fecha, la cantidad perdida o robada y una breve descripción de la incidencia.

- Por último interesa saber el histórico de compras que ha habido en el supermercado. Se sabe que cada cliente puede comprar varios productos y que a su vez un producto puede estar en la compra de varios clientes. De cada compra interesa conocer la fecha en la que se produce, la cantidad de producto que se ha comprado y el total de la compra.

Entidades y atributos

Como entidad se tiene a **cliente**, que como información básica se obtiene el nombre, apellido, fecha de nacimiento y su localidad. Además se almacenará su nombre de usuario y contraseña, por lo que dicha entidad quedaría de la siguiente manera:

Clientes
IdCliente
Nombre
Apellido1
FechaNac
Localidad
NomUsuario
Contrasena

Producto es otra de las entidades que se reconocen en esta semántica. Para cada producto se quiere saber su nombre, si está disponible o no para el cliente, la cantidad que existe en el supermercado y la cantidad que entra en cada balda. También se quiere conocer su precio de compra al proveedor y su precio de venta al cliente. Nunca se permitirá eliminar productos de la base de datos ya que el permitirlo podría suponer varios problemas. Podría suceder que un cliente haya incluido un producto en su lista de la compra, y que al eliminarlo de nuestra base de datos, se elimine de su lista también. Este supuesto seguramente provocaría malestar entre los clientes ya que no encontrarían razón para su explicación. Además el hecho de eliminar un producto supondría alterar los datos de ventas de la superficie comercial, tales como estadísticas

de venta, históricos, etc. Por ello se utilizará el atributo “Disponible”, que permitirá que un producto tenga una marca de borrado pero siempre será parte de la información.

Productos
IdProducto
NomProducto
Disponible
CantidadPorBalda
PrecioCompra
PrecioVenta

De cada producto interesa saber el **baremo**, del cual se quiere saber hasta qué cantidad de producto se tiene que pedir. Esto permitirá que el encargado de realizar los pedidos, una vez tenga los datos del estado de cada producto, pueda realizar un pedido automático de x unidades, siendo x las unidades la diferencia entre la cantidad existente en la superficie comercial y la cantidad máxima que se pretende tener.

Baremos
TipoBaremo
StockMaximo

De cada **categoría** se quiere saber el nombre de cada una de ellas, lo que permitirá categorizar todos los productos para que a la hora de que un cliente vaya hacer la lista de la compra, tenga un acceso más cómodo y lógico a los productos que requiera.

Categorias
IdCategoria
DescCategoria

Las ubicaciones permitirán conocer el estado real de cada uno de los lugares del establecimiento comercial. Las ubicaciones se refieren a la parte de la superficie comercial a la que tiene el acceso el usuario (nunca a la parte del almacén). El supermercado estará dividido en secciones como si fueran cuadrantes en un mapa, cada una de las secciones tendrá identificados de manera única todos sus pasillos. En cada

uno de los pasillos se tendrán identificadas todas las estanterías mientras que cada estantería tendrá también un número definido de baldas igualmente identificadas. Dando por supuesto que el tamaño de cada balda es el mismo a cualquier otro, se puede disponer de información real del estado de cada una de las baldas de la superficie comercial. Así se puede saber el número de unidades de x producto que hay en dicho lugar. El tener todas las baldas del mismo tamaño no supone ningún problema ya que un mismo producto puede colocarse en más de una, por lo que elimina el problema de tener que introducir el mismo volumen de producto en todas las baldas.

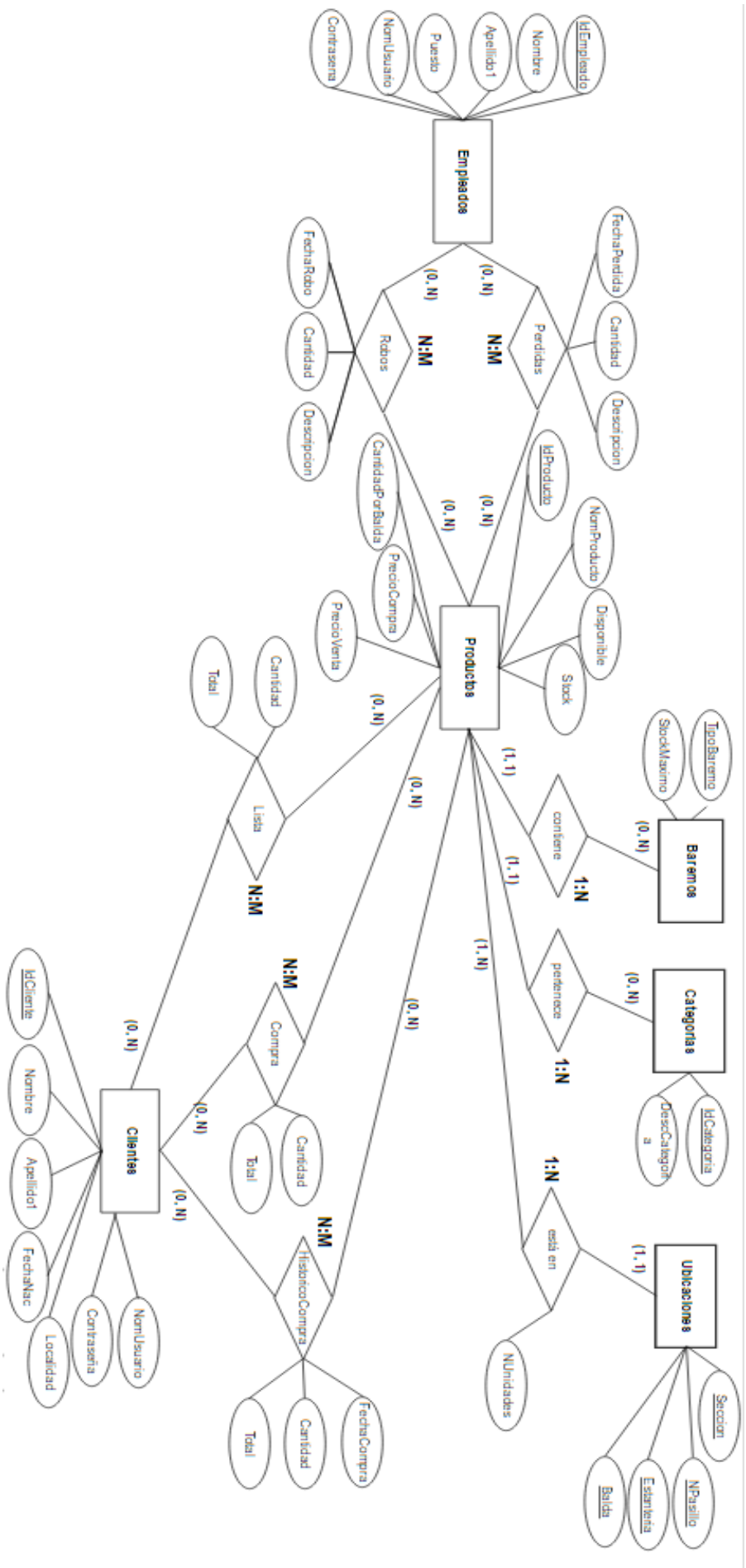
Ubicaciones
Seccion
NPasillo
Estanteria
Balda
NUnidades

Por último se pude localizar los **empleados**, de los cuales se quiere conocer su información básica además del puesto que desempeñan y del nombre de usuario y contraseña. Esto hace que un empleado pueda tener acceso a partes de la aplicación que ninguna otra persona pueda. Además, dependiendo del puesto del empleado, tendrá acceso a diferente información. Si el usuario es un reponedor, tendrá acceso a ver el estado de los productos en cada una de sus ubicaciones mientras que si es un gerente, tendrá acceso a información de pedidos, estadísticas monetarias, etc.

Empleados
IdEmpleado
Nombre
Apellido1
Puesto
NomUsuario
Contrasena

Entidad-Relación

Viendo las entidades que se conocen hasta ahora y la semántica de la base de datos, se puede representar mediante un diagrama entidad-relación de la siguiente manera:

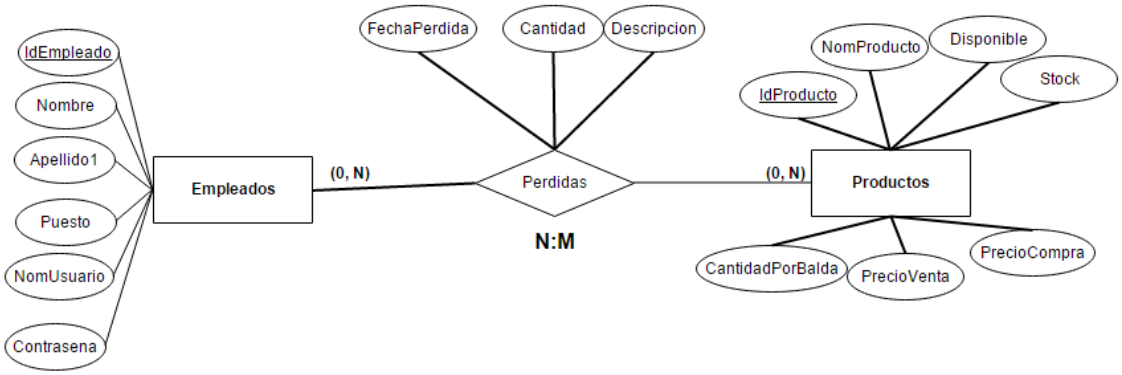

Empleados
IdEmpleado
Nombre
Apellido1
Puesto
NomUsuario
Contrasena
Robos
FechaRobo
Cantidad
Descripcion
Perdidas
FechaPerdida
Cantidad
Descripcion
Productos
IdProducto
NomProducto
Disponible
Stock
PrecioVenta
PrecioCompra
CantidadPorBalda
Barrenos
TipoBarreno
StockMaximo
contiene
Categorias
IdCategoria
DescCategoria
pertenece
Ubicaciones
Seccion
NPasillo
Estanteria
Balda
esta en
NUnidades
Clientes
IdCliente
Nombre
Apellido1
FechaNac
Localidad
Contrasena
NomUsuario
Lista
Cantidad
Total
Compra
Cantidad
Total
HistoricoCompra
FechaCompra
Cantidad
Total
(0, N)
(1, 1)
1:N
N:M

Modelo relacional

Para poder generar los scripts en SQL y ejecutarlos en SQL Server faltaría tener el modelo relacional de este diagrama entidad-relación. De esa manera se obtendrán todas las tablas necesarias con sus atributos necesarios:

Se empezará con las relaciones N:M seguidas de las relaciones 1:N.

Relación Empleados – Pérdidas – Productos



En esta relación habrá 3 tablas, la de Empleados y la de Productos tal y como están en la imagen, y una tercera que será pérdidas que contendrá los atributos propios, y las claves primarias de las dos tablas que relaciona como claves foráneas. En ésta tercera tabla se introducirá un identificador de la pérdida que actuará como clave primaria, ya que con las claves primarias de empleados y productos no se puede asegurar la unicidad de las tuplas porque un mismo empleado puede insertar varias tuplas del mismo producto.

Se necesita contemplar las pérdidas ya que es imposible que a ningún reponedor no se le rompa ningún producto, o que algún cliente sin quererlo pueda romper alguna unidad. Si no se hubiese contemplado esta posibilidad, los datos serían inconsistentes ya que en la base de datos se tendrían reflejadas más unidades de las que realmente habría.

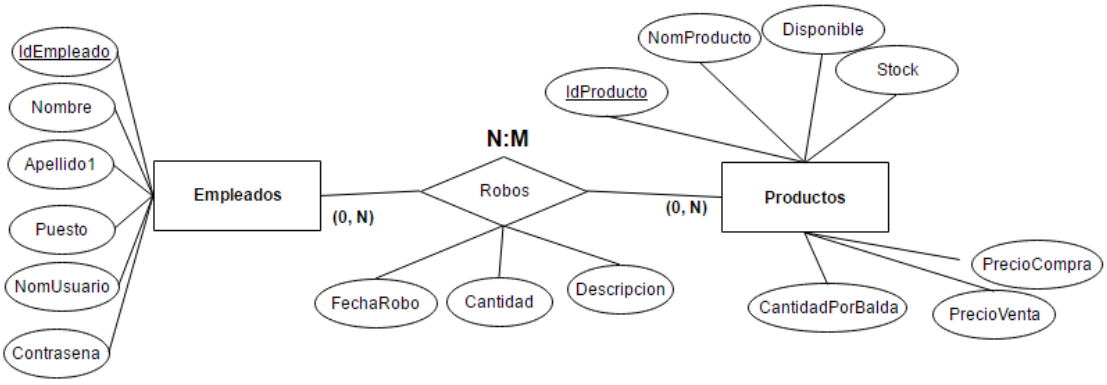
Por tanto las 3 tablas quedarían de la siguiente manera:

Empleado
IdEmpleado **(PK)**
Nombre
Apellido1
Puesto
NomUsuario
Contrasena

Perdidas
IdPerdida **(PK)**
IdEmpleado **(FK)**
IdProducto **(FK)**
FechaPerdida
Cantidad
Descripcion

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta

Relación Empleados – Robos – Productos



En esta relación habrá 3 tablas, la de Empleados y la de Productos tal y como están en la imagen, y una tercera que será robos que contendrán los atributos propios, y las claves primarias de las dos tablas que relaciona como claves foráneas, al igual que en el anterior caso. En ésta tercera tabla se introducirá un identificador del robo que actuará como clave primaria, ya que con las claves primarias de empleados y productos no se puede asegurar la unicidad de las tuplas porque un mismo empleado puede insertar varias tuplas del mismo producto.

Este es un caso similar al de pérdidas, en el que también se ha de suponer que en algún momento pueda producirse un robo. En un principio este dato se obtendría a través de los reponedores y los históricos de ventas. Se da por hecho que un producto que no se ha roto, que no está en ningún carrito, y que no está en stock, es que ha sido robado, por lo que estos datos los tendría que pasar como una incidencia el reponedor. La única manera de poder contabilizarlo es sabiendo el número de unidades de las que se disponen tanto en almacén, en las baldas como en los carritos. La diferencia de productos sería la cantidad de productos robados que tendría que realizar el control el reponedor al final del día por ejemplo.

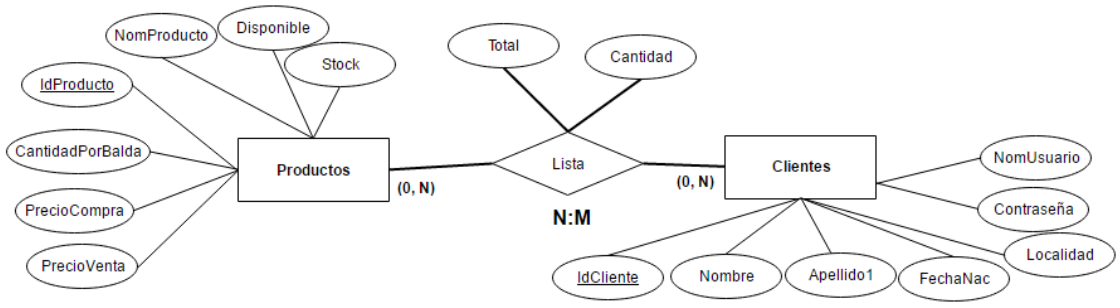
Por tanto las 3 tablas quedarían de la siguiente manera:

Empleado
IdEmpleado **(PK)**
Nombre
Apellido1
Puesto
NomUsuario
Contrasena

Robos
IdRobo **(PK)**
IdEmpleado **(FK)**
IdProducto **(FK)**
FechaRobo
Cantidad
Descripcion

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta

Relación Productos – Listas – Clientes



De esta relación también saldrán 3 tablas ya que existe una relación N:M. Las tablas de productos y clientes quedarán como se refleja en el entidad-relación, y la tabla Listas cogerá las claves primarias de productos y clientes y las tendrá como claves primarias y extranjeras. Además de esos atributos, también tendrá los propios de la relación, como son total y cantidad. Aquí es donde se alojan los datos de las listas de la compra que realizan los usuarios.

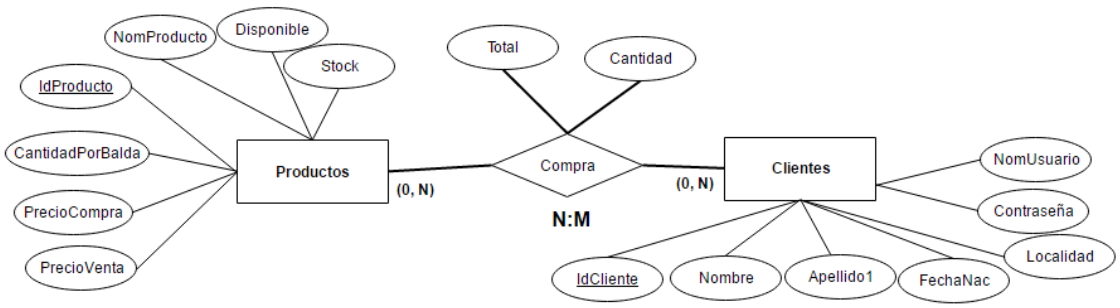
Por tanto las 3 tablas quedarían de la siguiente manera:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta

Listas
IdCliente **(PK) (FK)**
IdProducto **(PK) (FK)**
Cantidad
Total

Clientes
IdCliente **(PK)**
Nombre
Apellido1
FechaNac
Localidad
NomUsuario
Contrasena

Relación Productos – Compras – Clientes



En esta relación se da exactamente el mismo caso que en el anterior. De esta relación saldrá una nueva tercera tabla Compras. Compras contendrá como clave primaria las claves primarias de las tablas a las que relaciona, es decir, las claves primarias de productos y clientes. A su vez esos atributos serán claves extranjeras de los mismos atributos de productos y clientes. Para finalizar se añadirán los atributos propios de la relación, como son cantidad y total.

Ésta tabla es la que simulará la tabla en memoria explicada anteriormente. En ella se almacenarán todos los productos introducidos en los carros de las compras de manera que mediante el identificador del cliente se puede saber de quién es cada producto.

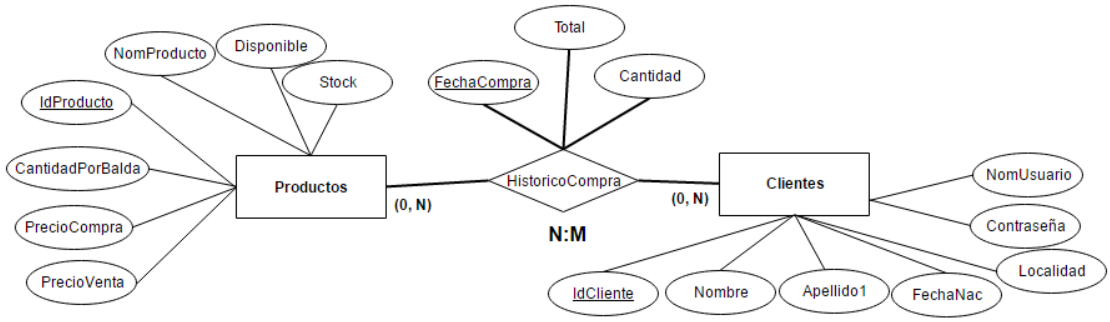
Por tanto las 3 tablas quedarían de la siguiente manera:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta

Compras
IdCliente **(PK) (FK)**
IdProducto **(PK) (FK)**
Cantidad
Total

Clientes
IdCliente **(PK)**
Nombre
Apellido1
FechaNac
Localidad
NomUsuario
Contrasena

Relación Productos – HistoricoCompras – Clientes



Esta es la última relación N:M que hay en el diagrama E/R. Por tanto se volverá a crear una nueva tabla HistoricoCompras. Esta nueva tabla contendrá las claves principales de Productos y Clientes como su propia clave primaria. Además también tendrá como clave primaria el propio atributo de la relación FechaCompra, ya que no se puede asegurar la unicidad de las tuplas únicamente con los dos primeros atributos como clave primaria.

En esta tabla se almacenarán las compras realizadas por los usuarios. Aquí se volcarán los datos que había en la tabla Compras al pasar por caja. Como un cliente podría realizar varias compras en el mismo día, se utilizará el atributo *FechaCompra* como parte de la clave primaria, así evitando datos inconsistentes.

Por tanto la tabla HistoricoCompras tendría una clave primaria compuesta de 3 atributos (FechaCompra, IdCliente, IdProducto), y estos dos últimos, a su vez, serán claves extranjeras de productos y clientes. Además esta nueva tabla tendrá como atributos la cantidad y el total.

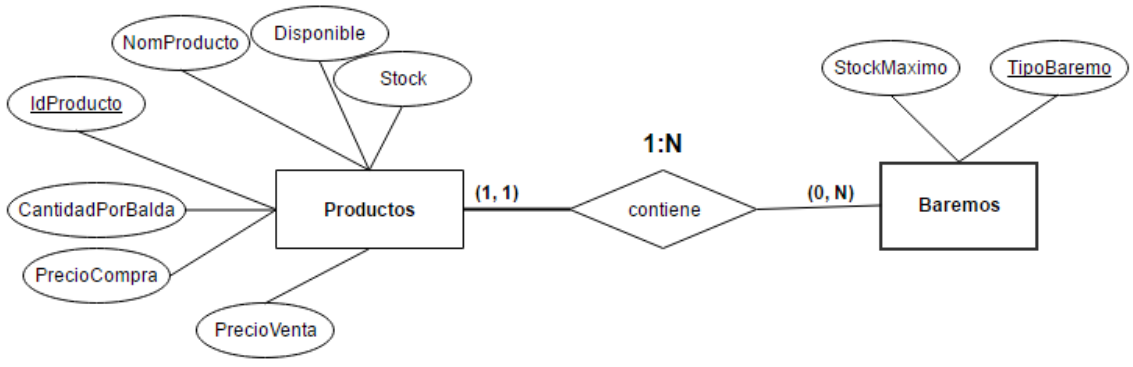
Así quedarían las 3 tablas:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta

HistoricoCompras
FechaCompra **(PK)**
IdCliente **(PK) (FK)**
IdProducto **(PK) (FK)**
Cantidad
Total

Clientes
IdCliente **(PK)**
Nombre
Apellido1
FechaNac
Localidad
NomUsuario
Contrasena

Relación Productos – contiene – Baremos



En la siguiente relación se puede apreciar que ya se ha acabado con las relaciones N:M. Esta ya es una relación 1:N por lo que no se generará una nueva tabla en la base de datos. Se puede apreciar que cada producto tiene un único baremo mientras que un baremo lo pueden seguir varios productos. Como consecuencia de esa relación habrá que pasar la clave principal de baremos a la tabla productos como clave extranjera, y así poder mantener la relación entre estas dos entidades.

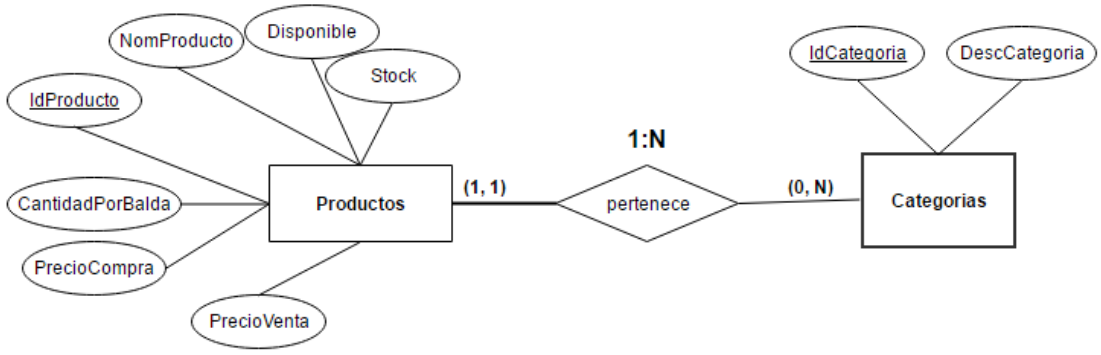
Los baremos se utilizarán para especificar la cantidad máxima que se quiere tener en la superficie comercial de cada producto.

Las 2 tablas quedarían de la siguiente manera:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta
TipoBaremo **(FK)**

Baremos
TipoBaremo **(PK)**
StockMaximo

Relación Productos – pertenece – Categorías



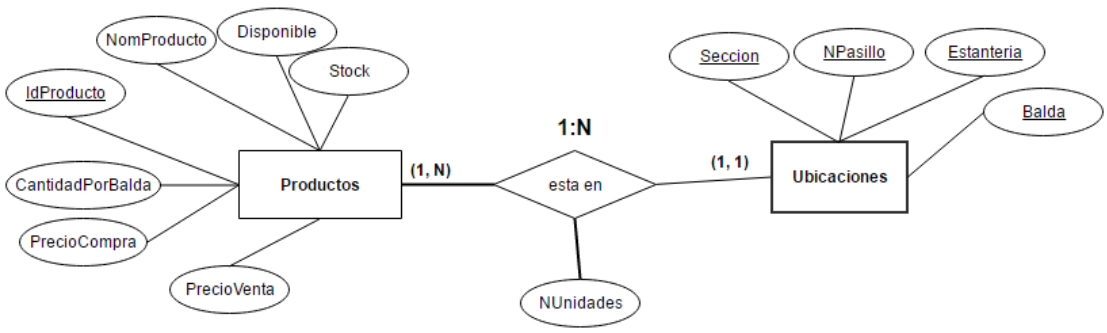
Esta relación 1:N es idéntica a la relación anterior por lo que no se generará una nueva tabla en la base de datos. Se puede apreciar que cada producto tiene una única categoría, mientras que una categoría la pueden tener varios productos. Como consecuencia de esa relación habrá que pasar la clave principal de categorías a la tabla productos como clave extranjera, y así poder mantener la relación entre estas dos entidades.

Las 2 tablas quedarían de la siguiente manera:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta
TipoBaremo
IdCategoria **(FK)**

Categorias
IdCategoria **(PK)**
DescCategoria

Relación Productos – está en – Ubicaciones



Esta relación 1:N es contraria a la relación anterior. Seguirá sin generarse una nueva tabla para esta relación. Se puede apreciar que cada ubicación tiene un único producto, mientras que un producto puede estar en varias ubicaciones. Como consecuencia de esa relación habrá que pasar la clave principal de productos a la tabla ubicaciones como clave extranjera, y así poder mantener la relación entre estas dos entidades.

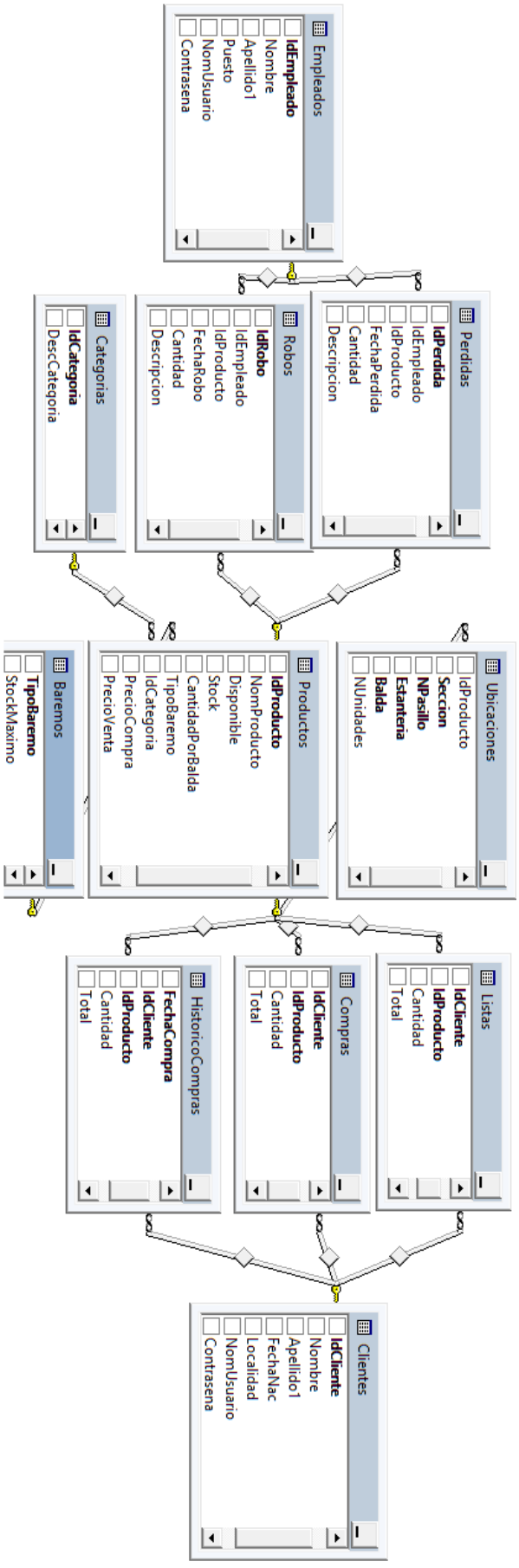
Esto permite que se puedan utilizar más ubicaciones para unos productos que para otros como resulta lógico.

Las 2 tablas quedarían de la siguiente manera:

Productos
IdProducto **(PK)**
NomProducto
Disponible
Stock
CantidadPorBalda
PrecioCompra
PrecioVenta
TipoBaremo
IdCategoria

Ubicaciones
Seccion **(PK)**
NPasillo **(PK)**
Estanteria **(PK)**
Balda **(PK)**
IdProducto **(FK)**
NUnidades

Por tanto la estructura de la base de datos y sus relaciones con las claves foráneas quedarán de la siguiente manera:


Empleados
IdEmpleado
Nombre
Apellido1
Puesto
NomUsuario
Contrasena
Perdidas
IdPerdida
IdEmpleado
IdProducto
FechaPerdida
Cantidad
Descripcion
Robos
IdRobo
IdEmpleado
IdProducto
FechaRobo
Cantidad
Descripcion
Categorias
IdCategoria
DescCategoria
Ubicaciones
IdProducto
Seccion
NPasillo
Estanteria
Balda
NUnidades
Productos
IdProducto
NomProducto
Disponible
Stock
CantidadPorBalda
TipoBaremo
IdCategoria
PrecioCompra
PrecioVenta
Baremos
TipoBaremo
StockMaximo
Listas
IdCliente
IdProducto
Cantidad
Total
Compras
IdCliente
IdProducto
Cantidad
Total
HistoricoCompras
FechaCompra
IdCliente
IdProducto
Cantidad
Total
Clientes
IdCliente
Nombre
Apellido1
FechaNac
Localidad
NomUsuario
Contrasena

Se puede apreciar que se ha combinado todo lo visto hasta ahora, comprobando las claves primarias, claves foráneas, y en la última imagen con las propias relaciones. Una vez conseguido esto, se puede decir que existe una base de datos relacional que garantiza la no duplicidad de los datos y que garantiza la integridad referencial, de forma que si se elimina un registro, se eliminan también los registros relacionados.

Una vez se tenga el modelo relacional, quedará asignar a los propios atributos los tipos de datos que serán, el tamaño de los mismo, si pueden ser nulos o no, etc.

Al asignar estas propiedades a los datos habrá que tener en cuenta varios aspectos. Uno de ellos es que hay que asegurarse de que a una clave foránea hay que asignarle las mismas propiedades que a la clave primaria a la que hace referencia. Si no se hace de esta manera, probablemente surjan problemas de integridad al introducir registros en la base de datos. Otro de los aspectos a tener en cuenta es que se van a asignar propiedades a los datos en los que luego se van a almacenar valores desde la propia aplicación. Por tanto, aunque no se haya implementado nada de la aplicación aún, es preferible tener una visión de lo que se va a querer hacer desde la aplicación en lo que respecta a la base de datos, para no tener que estar constantemente cambiando las propiedades de los atributos en la base de datos.

Una vez que se han tenido en cuenta estos aspectos, los datos de la base de datos quedan con los siguientes tipos de datos:

Baremos

	Nombre de columna	Tipo de datos	Permitir val...
▶🔑	IdCategoria	int	☐
	DescCategoria	varchar(300)	☐
			☐

Categorias

Nombre de columna	Tipo de datos	Permitir val...
TipoBaremo	int	☐
StockMaximo	int	☐
		☐

Clientes

Nombre de columna	Tipo de datos	Permitir val...
IdCliente	int	☐
Nombre	varchar(20)	☐
Apellido1	varchar(30)	☐
FechaNac	date	☑
Localidad	varchar(25)	☐
NomUsuario	varchar(20)	☐
Contrasena	varchar(20)	☐
		☐

Compras

Nombre de columna	Tipo de datos	Permitir val...
IdCliente	int	☐
IdProducto	int	☐
Cantidad	int	☐
Total	decimal(6, 2)	☐
		☐

Empleados

Nombre de columna	Tipo de datos	Permitir val...
IdEmpleado	int	☐
Nombre	varchar(30)	☐
Apellido1	varchar(30)	☐
Puesto	varchar(20)	☐
NomUsuario	varchar(20)	☐
Contrasena	varchar(20)	☐
		☐

HistoricoCompras

Nombre de columna	Tipo de datos	Permitir val...
FechaCompra	datetime	☐
IdCliente	int	☐
IdProducto	int	☐
Cantidad	int	☐
Total	decimal(6, 2)	☐
		☐

Listas

Nombre de columna	Tipo de datos	Permitir val...
IdCliente	int	☐
IdProducto	int	☐
Cantidad	int	☐
Total	decimal(6, 2)	☐
		☐

Perdidas

Nombre de columna	Tipo de datos	Permitir val...
IdPerdida	int	☐
IdEmpleado	int	☐
IdProducto	int	☐
FechaPerdida	date	☐
Cantidad	int	☐
Descripcion	varchar(200)	☐

Productos

Nombre de columna	Tipo de datos	Permitir val...
IdProducto	int	☐
NomProducto	varchar(30)	☐
Disponible	varchar(1)	☐
Stock	int	☐
CantidadPorBalda	int	☐
TipoBaremo	int	☐
IdCategoria	int	☐
PrecioCompra	decimal(6, 2)	☐
PrecioVenta	decimal(6, 2)	☐

Robos

Nombre de columna	Tipo de datos	Permitir val...
IdRobo	int	☐
IdEmpleado	int	☐
IdProducto	int	☐
FechaRobo	date	☐
Cantidad	int	☐
Descripcion	varchar(200)	☐

Ubicaciones

Nombre de columna	Tipo de datos	Permitir val...
IdProducto	int	☐
Seccion	varchar(20)	☐
NPasillo	varchar(3)	☐
Estanteria	varchar(2)	☐
Balda	varchar(1)	☐
NUnidades	int	☐

Una vez esté la base de datos implementada, se empezará a explicar la aplicación y la conexión entre aplicación y BBDD.

Aplicación web

La aplicación web realmente está construido en 3 proyectos en 1.

El primer proyecto es una librería de clases (supermercadoDAL) que haría las funciones de la capa de acceso a datos.

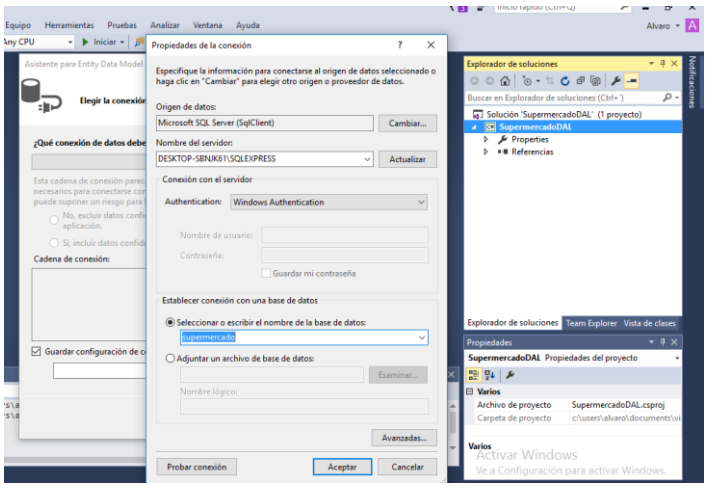
El segundo proyecto es otra librería de clases (supermercadoDTO) que sólo contiene clases de objetos DTO. Estos objetos son los que se encargan de transportar los datos entre el frontend y la capa de acceso a datos.

El tercer y último proyecto es un proyecto ASP.NET, donde se realiza toda la lógica de negocio de la aplicación.

Capa de acceso a datos

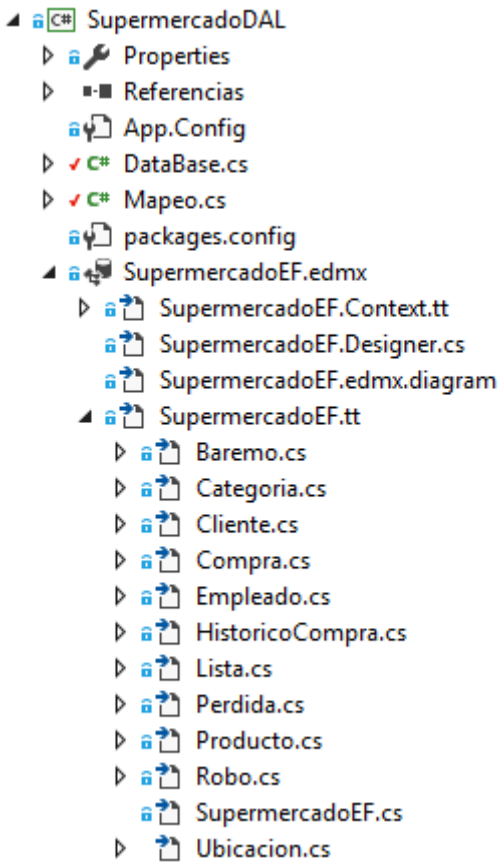
La función de este proyecto es el acceso a la base de datos, para lo que se ha utilizado Entity Framework.

Lo que se ha hecho es crear un proyecto ADO.NET Entity Data Model y en el cual crear una conexión a la base de datos creada anteriormente en SQL Server.



Al realizar esta conexión se obtiene un archivo .edmx, que es quien mapea la base de datos en el proyecto y quien da acceso a ellos. Dentro de él se pueden apreciar todas las tablas y atributos de la base de datos con quien se quiere interactuar.

Una vez se obtiene el mapeo de la base de datos SQL Server en este primer proyecto de la aplicación, se crean dos clases más estrechamente relacionadas con los datos. Uno de ellos se utilizará para el acceso a datos (DataBase.cs) y el otro se utilizará para hacer el mapeo entre las clases de la base de datos y los objetos de las clases DTO que se verán en el siguiente proyecto (Mapeo.cs).



El fichero DataBase lo que hace es centralizar en él todas las funciones que necesiten comunicación directa con la base de datos. Para ello se instancia la conexión a la base de datos y a través de ese objeto se almacenan y se obtienen datos. Estos métodos en sí mismos no tienen ninguna lógica, se llaman desde la tercera parte del proyecto pero el tenerlos en este apartado hace que se abstraigan las funcionalidades de la aplicación.

Se puede decir algo parecido del fichero Mapeo, que su función es la de convertir objetos instanciados de la propia base de datos a objetos DTO que se utilizarán más adelante y viceversa.

Por lo que se puede ver, este primer proyecto se centra en el acceso a datos. De esta manera se consigue separar totalmente el acceso a base de datos, y a su vez, no se verá ningún acceso más en el resto de la aplicación.

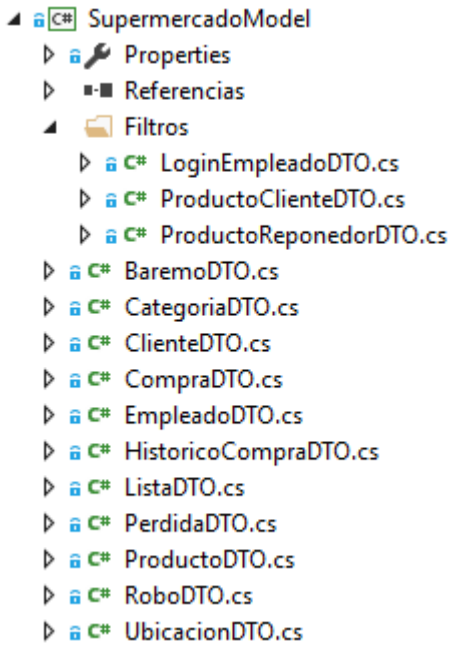
Se verá su función detalladamente al explicar el último proyecto, ya que todos van relacionados.

Transporte de datos

Este segundo proyecto solamente contiene todas las clases DTO de las tablas de la base de datos. Estos DTOs se usarán para transferir datos entre los otros 2 proyectos (capa de base de datos y capa de presentación).

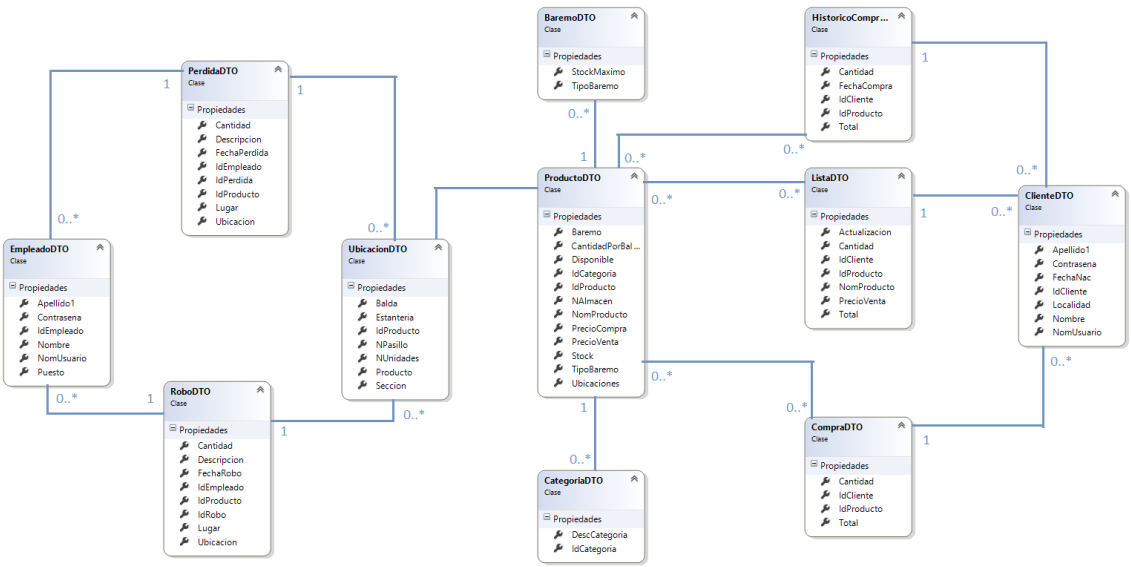
Los DTOs se crean para hacer que se independice más de las entidades del dominio, además de permitir que cuando se muestren los datos mediante este tipo de objetos y el usuario los cambie, no cambien también en la base de datos, al menos sin tenerlo previsto. Por tanto, estas clases representan a otros objetos de las entidades de la base de datos.

Se tendrá una clase DTO por cada entidad que haya en la base de datos, por lo que el segundo proyecto tendría este aspecto:



Se puede observar que se han utilizado filtros. Estos han servido para poder representar a un objeto de una entidad concreta sin necesidad de tener la información completa de la misma. En resumen, permiten transportar parte de la información de un objeto, permitiendo así que las transferencias sean más rápidas al contener menos datos.

Por tanto este proyecto contiene clases que representan a las clases de la base de datos. Así quedaría el diagrama de clases de los DTOs:

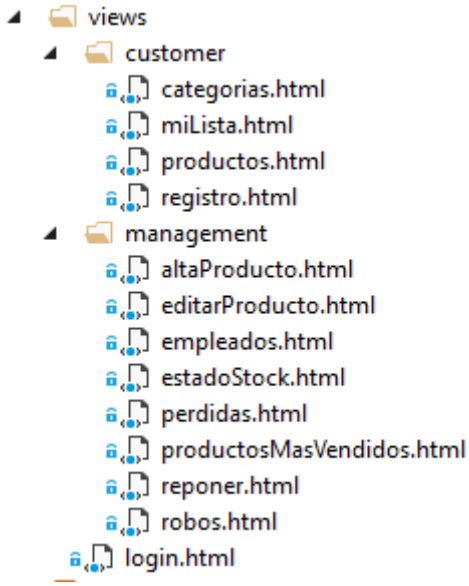


Capa de presentación

En este apartado se encuentra toda la lógica de la aplicación y está estructurado para separar los ficheros por funcionalidad. Se ha desarrollado en un proyecto ASP.NET Web API.

En un paquete está la parte de AngularJS, donde está estructurado de forma que sigue un patrón MVC. En él lógicamente se contienen las vistas por un lado (DOM), los controladores de las vistas por otro lado (controladores JavaScript) y el "modelo" (factorías de angular) finalmente.

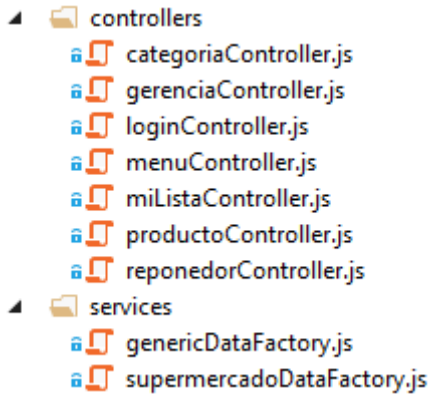
Las vistas están separadas de forma que por un lado están las que refieren a la parte del cliente, y por otro lado las que están relacionadas con los empleados. La única que se sale de esta lógica es la de login, que es común para los dos tipos.



Realmente el modelo de angular no contiene los datos, sino que es el que se encarga de obtener los datos, por tanto si uno se centra únicamente en Angular, se podría considerar el modelo de esa parte.

Los controladores de angular no pueden almacenar información de forma persistente, cada vez que se cargan estos controladores, los datos que pudiese contener se eliminan. Sin embargo las factorías utilizan el patrón Singleton, lo que significa que cada vez que se instancie un objeto factoría, no se creará uno nuevo sino que se

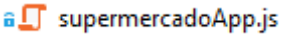
devolverá el ya existente y los datos no se perderán nunca durante la ejecución de la aplicación. La factorías son un tipo de servicio que se utilizarán en el proyecto como medio de comunicación entre los distintos controladores, pudiéndose guardar datos relevantes de la aplicación y diferentes funciones para obtener diferentes datos. Esto lo que permite es no tener que implementar la misma función en diferentes controladores en el caso de necesitar la misma información en varios de ellos.



Se puede observar que existe un segundo servicio genericDataFactory. Aquí es donde está implementada la API de angular donde también tenemos los métodos GET, POST y REMOVE. Estos métodos de la API son los que realizarán las peticiones a los ApiController de Web API. Los POST se harán en JSON ya que son más ligeros para el intercambio de datos.

En estas funciones se utilizan los servicios $q y $html. El servicio $q contiene la funcionalidad de las promesas. A través de $q se obtiene la promesa que tiene 3 estados: pendiente, resuelta o rechazada. Se va a combinar este servicio con el servicio $http, que es el que permite hacer peticiones AJAX al servidor. El servicio $http realiza la petición de la url de la Web API. Si la petición ha sido exitosa la promesa será resuelta y devolverá lo obtenido, mientras que si ha sido errónea, la promesa se volverá a estado rechazada.

Por último está el fichero con el que se inicia la aplicación web en angular.



Este archivo permite gestionar las rutas internas de la aplicación web. En él se especifican las rutas válidas por la aplicación, en la que a cada ruta le asigna un controlador y una vista, por lo que es quién hace que se cargue la vista en el index y quien dice quién es el controlador que se va a encargar de gestionar la vista. Se ha definido en él una ruta por defecto en el caso de que se intente acceder a alguna ruta interna no válida.

Este proyecto a parte de tener el apartado de Angular, contiene todos los controladores de Web API. Estos controladores proveen a la aplicación de los servicios web necesarios (API REST) para comunicar con la capa de acceso a datos como se había explicado anteriormente. Estos controladores se utilizarán para el paso de datos entre la capa de acceso a datos y AngularJS, por lo que se utilizará GET para la obtención de datos, POST para inserción y actualización de datos y DELETE para el borrado de datos.

Se han construido varios controladores para el acceso a datos de diferentes tipos de objetos pero básicamente el funcionamiento de todos ellos es el mismo por lo que se verá mejor con algún ejemplo más adelante.

Funcionalidad

A partir de ahora se conocerá la forma en la que ha sido implementada la aplicación y cómo se comunican todas las clases entre sí.

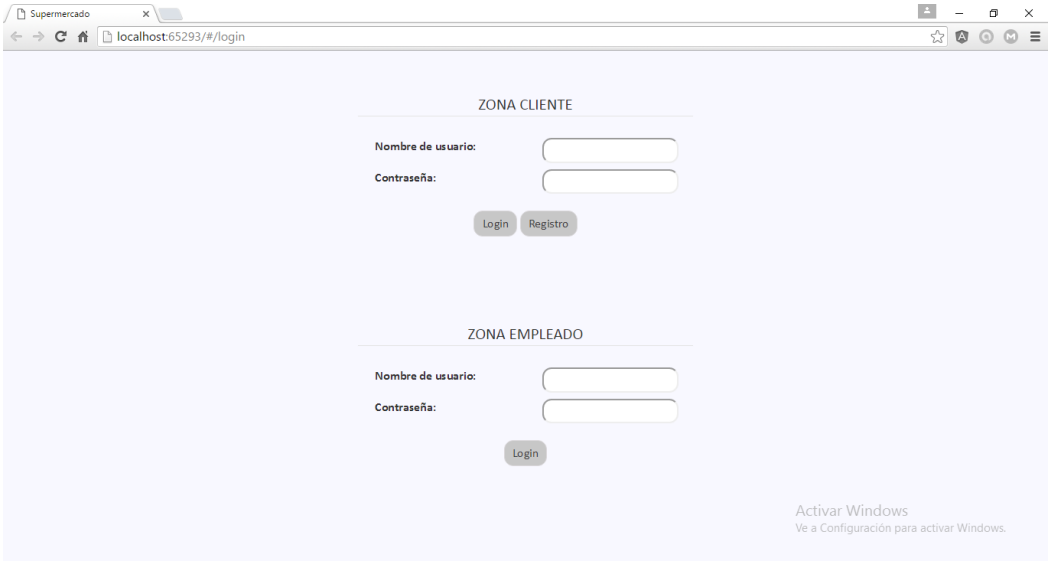
El proceso es muy similar en todo momento por lo que se mostrarán los casos más representativos de la aplicación, aunque sí se explicarán las medidas adoptadas para el correcto funcionamiento de toda la aplicación.

Login

Al iniciar la aplicación se puede observar una vista para el login. Al introducir tanto el nombre de usuario como la contraseña lo primero que se hace es comprobar si el usuario ha introducido algún carácter diferente a minúsculas, mayúsculas o números. De ser así, se deshabilita el botón para guardar los datos en base de datos, por tanto se evita la inyección de código malicioso por parte de un usuario malintencionado.

Si el nombre y contraseña contienen caracteres válidos, se hace una consulta en base de datos en la que se obtiene la tupla que coincida con ese nombre de usuario y contraseña. Si los campos existen, se devuelve el identificador del cliente en caso de hacer el login en la zona de clientes, o el objeto completo empleado ya que interesa conocer a parte del su id, el puesto que desempeña para poder mostrarle los datos del reponedor o del gerente. Si ese nombre de usuario y contraseña no existen, se devuelve un identificador inválido para poder mostrar un mensaje informativo por pantalla.

Una vez devuelto el id o el objeto empleado DTO, en el caso del cliente se redirige a la pantalla de categorías, en el de reponedores a la pantalla de productos a reponer y en el de los gerentes a la pantalla del estado del stock.



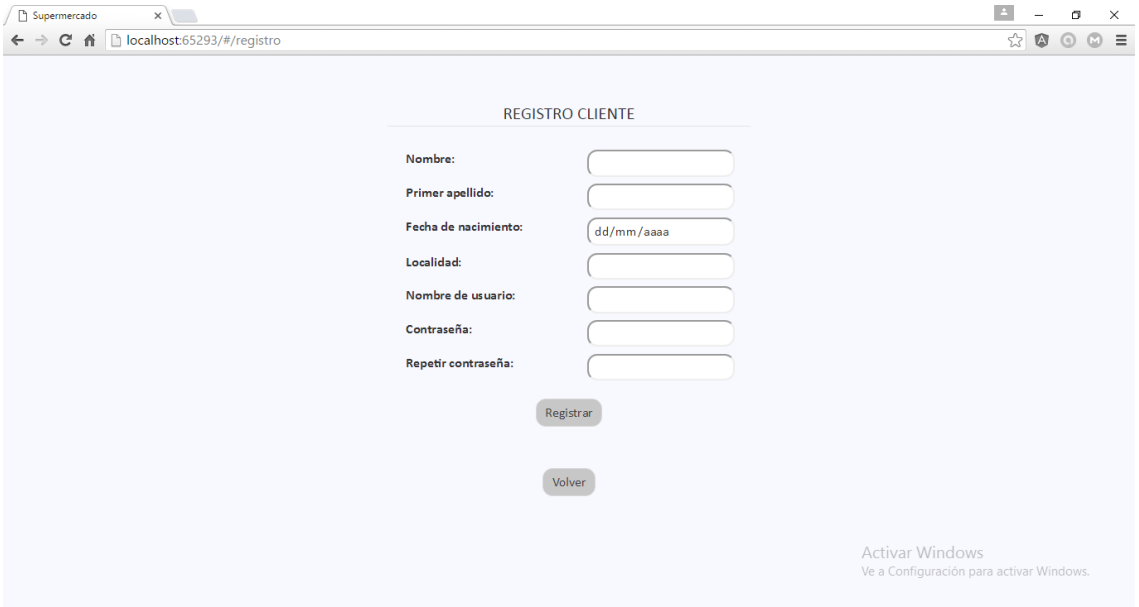
Apartado cliente

Registro

Si le pincha en el botón de registro redirige a esa página donde el usuario deberá introducir una serie de datos. Todos estos datos también llevan un control de lo que el usuario introduce. Exceptuando la fecha que obviamente deberá contener una fecha menor a la del día actual, y el nombre de usuario y contraseña que admitirán letras y números, los demás campos solamente admitirán letras. En todos los campos se comprueba que el número de caracteres introducido no supera nunca el número de caracteres máximo establecido en la base de datos para cada uno de los atributos. Además se obligará a introducir dos veces la contraseña para asegurarse de que se ha introducido lo que se quería. Todos los campos son obligatorios excepto la fecha de nacimiento. En caso de que la fecha no sea vacía, se realiza un parseo de la misma en el controlador de la vista para adecuarla del formato que se introduce en el formulario al formato que debe tener para ingresar en la base de datos.

Todas estas comprobaciones se hacen en el controlador de la vista, ya que si alguno de los datos no son válidos no es necesario hacer ninguna petición a la base de datos. De no ser así, el controlador realiza la petición a la factoría mientras que éste,

realiza otra petición al sistema de promesas de AngularJS. Éste llama a la Web API buscada y es este quien se encarga de llamar al método correspondiente en el fichero de base de datos. En este método primero se comprueba que el nombre de usuario que se quiere registrar no existe, ya que debe ser único. En el caso de existir se vuelve a la vista de registro devolviendo un identificador inválido y mostrando un mensaje de error, mientras que si el nombre no estaba usado, el registro termina con normalidad devolviendo el id del nuevo cliente y realizando seguidamente un login con los datos insertados en el registro para no hacer al usuario tener que loguearse después de registrarse.



Categorías

En esta vista se muestran las categorías obtenidas de la base de datos. Cada categoría contiene un botón para mostrar los productos relacionados con dicha categoría.

El proceso de la redirección a esta vista hasta la obtención de los datos es la siguiente:

Al realizar el login correctamente se redirige a la ruta interna de categorías a través del servicio $location. Mediante este servicio desde el login se redirige a la ruta de categorías pasándole como argumento el identificador del cliente. Como todas las vistas están integradas en el index.html e index.html tiene asociado un fichero supermercadoApp.js como iniciador del proyecto de angular, irá a dicho fichero en busca de esa ruta.

Cuando encuentre esa ruta en el fichero, se mostrará la vista relacionada a esa ruta con un controlador asociado. En esta ruta está especificado el nombre de la ruta y el nombre y número de cada uno de los parámetros que se le pasarán. Se cargará la vista (categorías.html) en el propio index.html y se ejecutará el controlador asociado, en este caso *ListCategoriasController.* Una vez aquí, a través del servicio $routeParams, se permite obtener los valores de los parámetros pasados utilizando el nombre especificado para ellos en el fichero de gestión de direcciones. Se ejecutará el contenido del controlador, en este caso una función llamada *obtenerCategorias()* de la factoría *SupermercadoDataFactory.* En esta función se llamará a la función get() de GenericDataFactory pasándole como argumento la dirección de la Web API a la que quiere acceder *('api/Categoria'*). Aquí se utilizan las promesas como ya se ha explicado antes, si la promesa es errónea, no se devolverá nada ya que no se habrá encontrado la Web API buscada mientras que si es correcta, redirigirá a la función Get de la Web API *'api/Categoria'*.

En esta función se instancia a la clase *DataBase* y se llama a la función *obtenerCategorias().* Como se puede observar ya se ha accedido desde el frontend a la capa de acceso a datos.

Una vez aquí se instancia a la conexión de la base de datos y a través de ella se realiza la consulta para obtener los datos requeridos. Las categorías obtenidas son objetos de la clase *Categoria* de la base de datos, por lo que antes de devolverlos los transformaremos en objetos DTO para devolverlos de forma regresiva hasta el controlador de la vista actual.

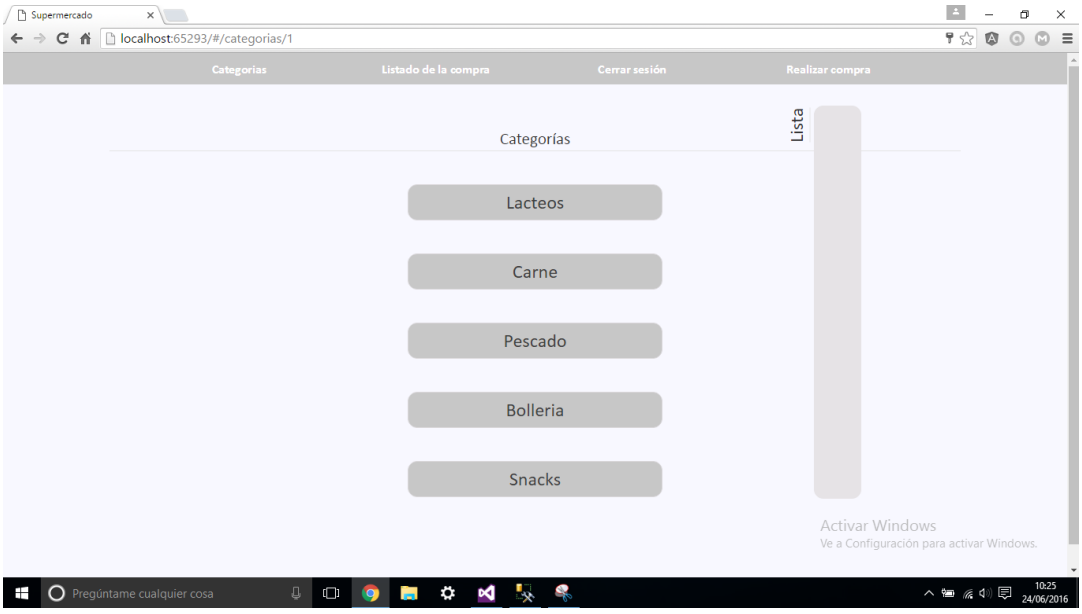
Para transformar estos objetos a objetos DTO se utiliza una función de la clase *Mapeo* que como ya se ha explicado anteriormente, su función es transformar los objetos en objetos DTO y viceversa. Por tanto se llama a la función correspondiente de esa clase *Mapeo.CargarCategoriasACategoriasDTO(categorias)* pasándole por argumento los objetos *Categoria* y recibiendo *CategoriaDTO*.

En esa función se obtiene una lista de categorías por lo que recorriendo esa lista y categoría a categoría, se va llamando a otra función de esa misma clase que realiza el mapeo objeto a objeto.

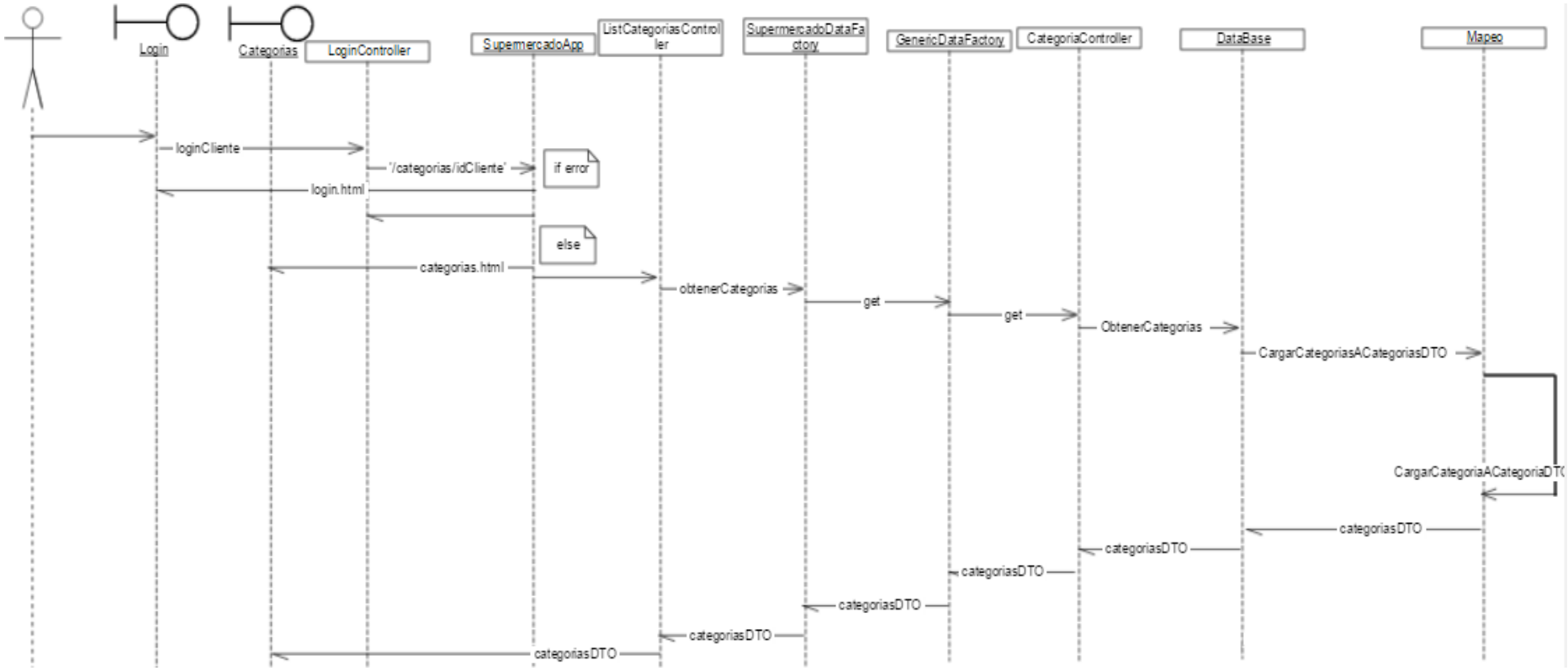
De esta manera al terminar, esta lista de objetos DTO vuelve hasta el controlador donde se almacena en una variable del scope. Accediendo a esa variable del controlador desde la vista se hace posible mostrar los datos por pantalla al usuario.

La última comprobación que se hace es si la lista que se ha traído contiene elementos. Si la lista es vacía se mostrará un mensaje por pantalla advirtiendo de la inexistencia de categorías.

Recordar que los datos almacenados en el scope del controlador se eliminarán al recargar el controlador.



Este proceso se verá mejor con un diagrama de secuencia:


Login
Categorias
LoginController
SupermercadoApp
ListCategoriasController
SupermercadoDataFactory
GenericDataFactory
CategoriaController
DataBase
Mapeo
loginCliente
'/categorias/idCliente'
if error
login.html
else
categorias.html
obtenerCategorias
get
get
ObtenerCategorias
CargarCategoriasACategoriasDTO
CargarCategoriaACategoriaDT
categoriasDTO
categoriasDTO
categoriasDTO
categoriasDTO
categoriasDTO
categoriasDTO

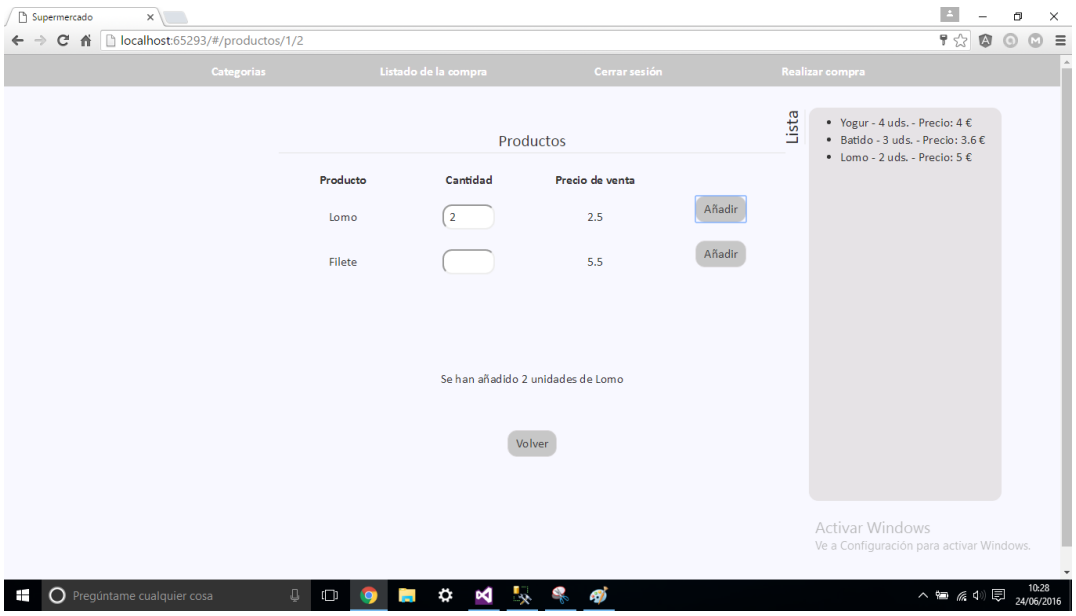
Productos

El proceso es exactamente el mismo que el anterior, con la diferencia que en esta vista se muestran los productos alojados en la base de datos que sean del tipo de la categoría seleccionada. Es lo primero que hace el controlador al cargarse, con el identificador de la categoría seleccionada, desde la capa de acceso a datos realiza una consulta en la que obtiene una lista de todos los productos que cumplan la condición de ser parte de esa categoría. Aquí el usuario podrá añadir a la lista una cantidad determinada por él mismo de algunos de los productos. Estas unidades están controladas para que nunca sean menores que 1 ni mayores que 10. Al cargarse la vista, estos campos están vacíos y si el usuario decide darle a añadir al carrito sin haber seleccionado la cantidad antes, el valor de esta variable por defecto es 0. Al no querer aceptar este valor en la lógica de negocio, se comprueba en el controlador que su valor no sea ese, porque si lo es, no se accederá a la capa de acceso a datos. Mientras no se escoja una cantidad determinada no se hará nada. Al agregar algún producto de forma correcta en la lista se muestra un mensaje haciendo ver que la selección ha sido añadida. El proceso es exactamente igual que el anterior, la única diferencia es que además de obtener los productos de una categoría, se aloja en la base de datos los productos seleccionados por POST.

Las únicas diferencias son que se hace pasando un objeto como argumento en vez de dos identificadores y que al llegar a la capa de acceso a datos, lo primero que se hace es el mapeo de un objeto DTO a un objeto de la clase Productos de la base de datos ya que se quiere guardarlo ahí. También se tiene en cuenta que cuando un usuario introduce una cantidad determinada de un producto, si ya existe ese producto en la lista se suman las cantidades para tener únicamente un producto con la suma de las cantidades, y no varios productos duplicados, ya que produciría la inconsistencia de los datos.

Al mostrar la lista de productos por pantalla, también se controla que la lista no sea vacía. De ser así se muestra un mensaje avisando que no se han encontrado productos.

En esta vista también existe un botón de volver que permite volver a la página de categorías.



Lista

En las páginas de categorías y de productos, de la parte lateral derecha se despliega una lista con los productos, cantidades y totales que el usuario tiene añadidos en la base de datos. Esta lista está controlada por otro controlador diferente al que gestiona el contenido de la vista. El controlador de la lista sabe cuándo tiene que mostrarse y cuando no porque el controlador que gestiona el contenido y él mismo se comunican a través de la factoría *SupermercadoDataFactory*. En la factoría los datos si son persistentes aunque se cambie de ruta por lo que es el lugar ideal para la comunicación entre controladores. Al cargar los controladores de categorías o de productos, lo primero que se hace es llamar a una función en la factoría que hace que una variable de la propia factoría se ponga a *true*, mientras que el controlador de la lista, mediante el servicio $interval, comprueba cada medio segundo el estado de esa variable. Si esa variable está a *true* la primera vez obtiene mediante el mismo sistema utilizado anteriormente los productos que están alojados en la tabla listas de la base de datos que correspondan con el identificador del cliente que esté usando la aplicación.

Una vez que han obtenido los productos y se muestran en esa lista desplegable, no se vuelven a obtener los productos hasta que el usuario no inserta otro producto desde la vista explicada anteriormente. Cuando eso ocurre, el controlador de productos cambia el estado de otra variable de la factoría, y como el controlador de la lista accede cada medio segundo a comprobar el estado de esta segunda variable, si está a *true* vuelve a obtener los productos.

De esta manera se asegura que cuando un cliente añade un producto a la lista de la compra, ésta se actualiza al momento (como mucho en medio segundo, algo prácticamente inapreciable).

Esta lista se ha pensado para que el cliente pueda ver que lleva apuntado en ella mientras sigue agregando más productos, ya que se considera algo poco práctico cambiar de vista para ver que lleva en la lista y volver a los productos para continuar con ella.

Menú

El menú es otra parte de la vista que lo dirige otro controlador diferente. En este caso se vuelven a utilizar variables en la factoría para mostrarlo u ocultarlo, por tanto se vuelve a utilizar el servicio $interval para que cada 3 décimas se compruebe si se ha de mostrar el menú. Si es así se obtiene de la propia factoría el identificador del cliente para disponer de su información. El menú contiene 4 opciones entre las que se encuentran “Categorías”, “Listado de la compra”, “Cerrar Sesión” y “Realizar Compra”.

La opción categoría redirige al listado de categorías visto anteriormente.

La opción listado de la compra permite ver lo mismo que en el listado mostrado anteriormente pudiendo ver el precio total de la lista completa, pudiendo modificar la cantidad de los productos y pudiendo eliminar productos de la lista. El controlador de esta vista lo primero que hace es obtener de base de datos los productos agregados a la lista de la compra del cliente que está navegando. Estos datos se cargan en el controlador de la vista y permite mostrarlos. Además el usuario puede cambiar la

cantidad de cualquier producto y se puede apreciar cómo el precio total del carrito va cambiando a su vez. Esto es posible ya que una de las características principales de AngularJS es el two way binding. Por tanto al darle a modificar producto, se envía a través de POST el producto modificado DTO y al llegar a la capa de acceso a datos se realiza una consulta en la que se obtenga el producto en cuestión y se modifique la cantidad a guardar. Guardando los cambios quedaría actualizada la lista de la compra del cliente. En esta vista se controla que las cantidades varíen entre 1 y 10, ya que si el usuario quiere eliminar el producto, dispone de otro botón para ello.

Al pulsar el botón eliminar, se realiza una petición DELETE con el identificador del cliente y del producto, con los cuales en la capa de acceso a datos se realiza el borrado del objeto que contenga el id enviado para ello.

La opción Cerrar sesión redirige la vista a la del login y al no ser persistentes los datos en los controladores, los identificadores que antes se pasaban por url ahora ya se han eliminado por lo que se aprovecha para realizar la función de cerrar sesión.

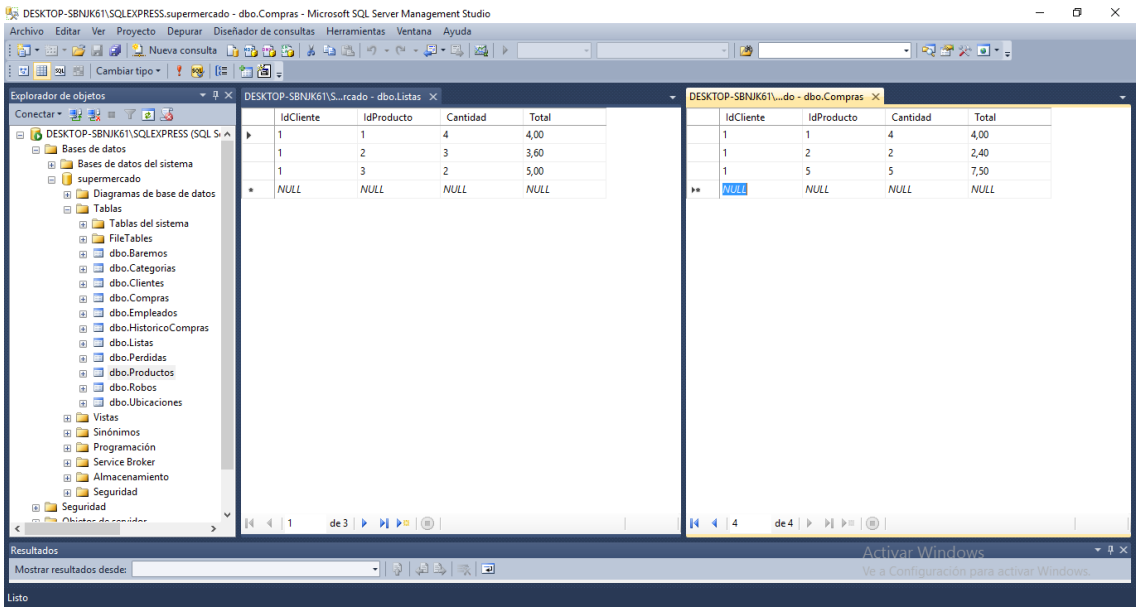
Por último, la opción de realizar compra, es un botón que simula el paso por el cajero en un supermercado. Para realizar esta simulación primero se deberá elaborar una lista de la compra con los productos que se deseen. Una vez realizada la lista de la compra se insertarán a mano en la tabla *Compras* los productos que "se tienen" dentro del carrito de la compra, siendo esto la simulación de ir a hacer la compra e introducir los productos en el carro. Y por último la simulación de pasar por caja la realiza este botón.

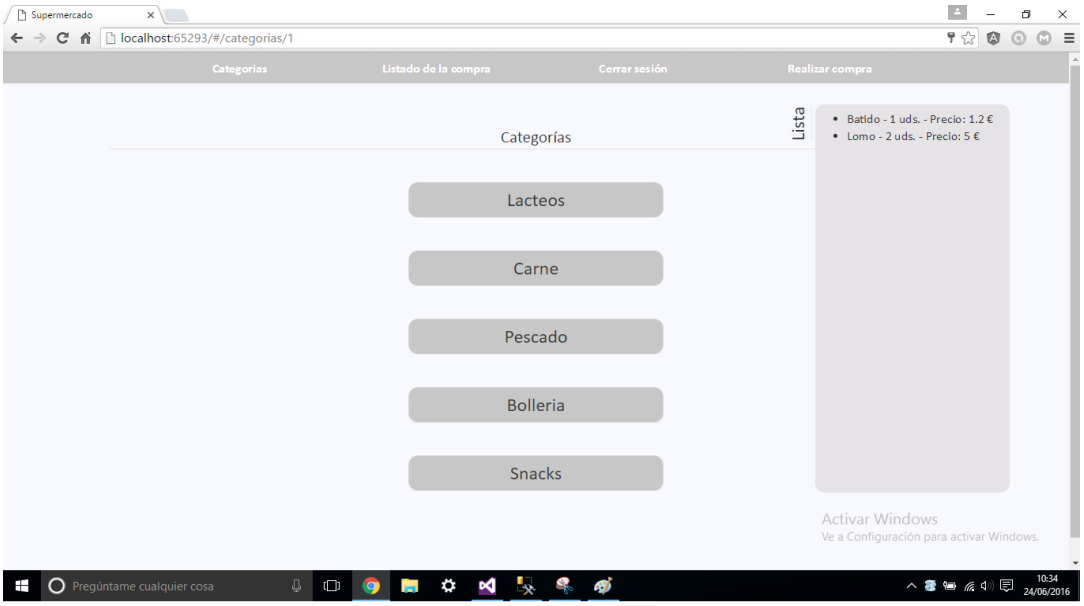
Lo que realiza el botón es a través de la factoría y del Web API, acceder a la capa de acceso a datos.

Allí lo primero que se hace es obtener todas las compras realizadas (productos que se han alojado en la tabla compras de la base de datos como simulación de una compra real), y por cada uno de los productos realizar una serie de comprobaciones.

Lo primero que se hace es restar la cantidad comprada en la ubicación donde están colocados los productos. A su vez se resta también en el stock del producto, ya que ahí se tienen contabilizados tanto los productos en almacén como en estanterías. Seguidamente se comprueba si ese producto estaba en la lista de la compra del cliente. De no estar no pasa nada, la compra ha sido descontada y el cliente compraría algo que no tenía apuntado, de estar apuntado en la lista se hacen dos comprobaciones. Si la cantidad de producto comprada es mayor o igual que la reflejada en la lista, el producto sería eliminado de la lista de la compra, pero si la cantidad comprada es menor, se descontarían las unidades y la cantidad del producto en la lista sería actualizada con la diferencia. A su vez se actualizaría el precio total de esos productos.

Una vez hecho este proceso y habiendo vuelto al controlador del menú, se cambia el valor de una variable en la factoría a *true*, para que el controlador de la lista desplegable pueda actualizarse con el $interval ya mencionado y pueda volver a mostrar los datos actualizados al cliente.





Apartado empleado

Una vez logueados en la parte de empleado de login, si el puesto del empleado es reponedor se mostrará información relevante para él, haciendo lo mismo en el caso de que el puesto de trabajo sea de gerente.

Se empezará explicando la funcionalidad del apartado del reponedor.

Reponedor

Productos a reponer

Al acceder a este apartado lo primero que se hace es cambiar el valor de las variables utilizadas en la factoría para el mostrado de diferentes elementos. Se cambian de tal forma que la lista de la compra quede oculta, al igual que el menú del cliente. En este caso se activa la variable para poder mostrar un menú diferente al del cliente.

Al realizar estas acciones, se muestra un listado de productos a reponer. Este proceso se hace de igual manera a la explicada anteriormente. Cuando el empleado se loguea, mediante el servicio $location se carga la nueva vista y se localiza el controlador asociado a esa vista. Al encontrar el controlador se ejecuta su contenido, que consiste en llamar a una función en SupermercadoDataFactory que este a su vez llama al método GET de GenericDataFactory en busca de la Web API pasada por argumento. Mediante el sistema de promesas se llama al GET del Web API y éste llama al método correspondiente del fichero DataBase en la parte de la capa de acceso a datos.

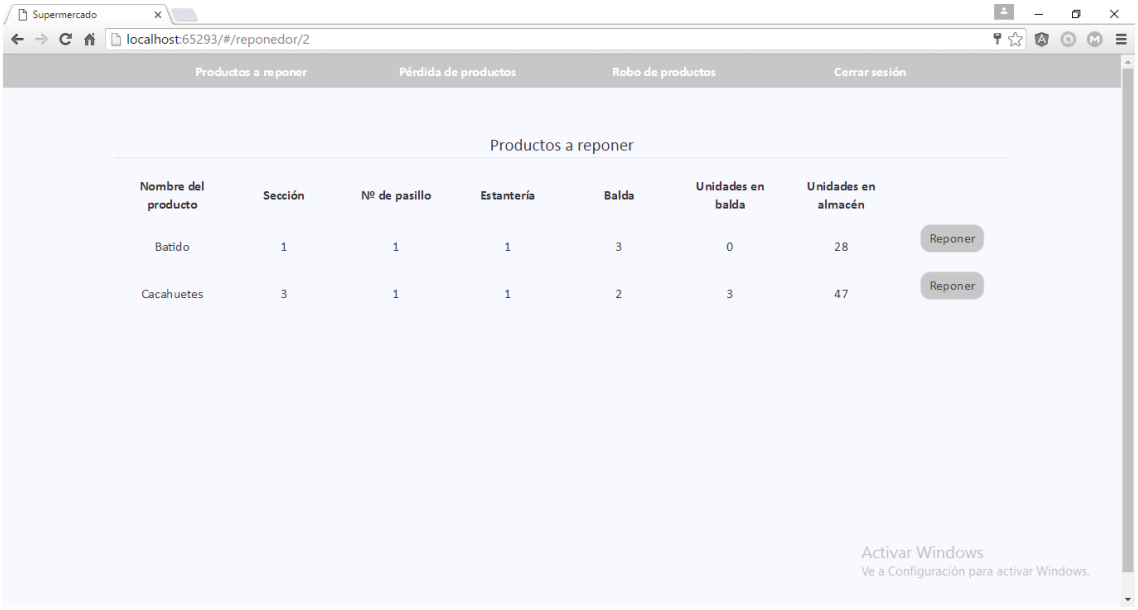
En este método mediante una consulta a base de datos, se obtienen todas las ubicaciones del supermercado en la que el número de unidades actuales en la balda sea menor al 20% de su capacidad. Por tanto la lista que devuelve la consulta se mapea para obtener Ubicaciones DTO y devolver esta lista al controlador de la vista.

Si esa lista devuelta no contiene ningún elemento se muestra un mensaje de que no hay productos a reponer, mientras que si existe algún elemento debajo del 20% de su capacidad, se mostrará al reponedor información de donde se encuentra ese producto, de que producto se trata y de cuantas unidades quedan en almacén.

Una vez que el reponedor haya repuesto la balda, dando por supuesto que siempre que se vaya a reponer una balda se va a rellenar por completo, el reponedor pinchará sobre un botón para informar de que esa ubicación ya ha sido repuesta. Una vez que se informe del trabajo realizado, ese producto desaparecerá de la lista al no

cumplir la condición de que aparezcan los productos con un número de unidades inferior al 20% de su capacidad. Esto se hace volviendo a obtener los productos que cumplan la condición cada vez que un empleado reponga una estantería.

Esta vista, al contrario que con la lista de la compra, no actualiza sus datos en la vista en el momento de que se sucedan las compras de los clientes. Este apartado tiene el objetivo de que un reponedor pueda en un momento determinado consultar si alguna ubicación está cercana a vaciarse, y si no es así, dedicarse a realizar otras tareas. No es necesario una actualización de datos en pantalla en tiempo real porque el reponedor no se va a pasar el tiempo mirando la aplicación para ver si surgen cambios.



Menú

El menú mostrado al reponedor funciona de la misma manera que el menú del cliente. Se muestra o se oculta en función de una variable en la factoría, por lo tanto en este apartado se muestra. Este menú gestionado por su controlador permite esto accediendo al contenido de esa variable.

Permite redirigir entre la vista anteriormente descrita de productos a reponer, permite ir a las vistas de informar de una pérdida o de un robo, y permite también el cierre de sesión que funciona de la misma manera que el cierre de sesión del cliente.

Este menú también utiliza el servicio $interval, al igual que en el menú de clientes comprueba cada 3 décimas si el valor de esa variable almacenada en la factoría ha cambiado y en el momento de hacerlo, muestra el menú y recoge el identificador del empleado que ha iniciado sesión.

Pérdida/Robo productos

Estas dos vistas son idénticas entre ellas por lo que se explicarán a la vez.

De cada una de estas acciones interesa saber cuál es el producto en cuestión, si la incidencia ha sucedido en el almacén o en la parte accesible al cliente y si ha sido en esta última parte en qué lugar se encuentra. La ubicación solo se pedirá si la incidencia ha ocurrido fuera del almacén.

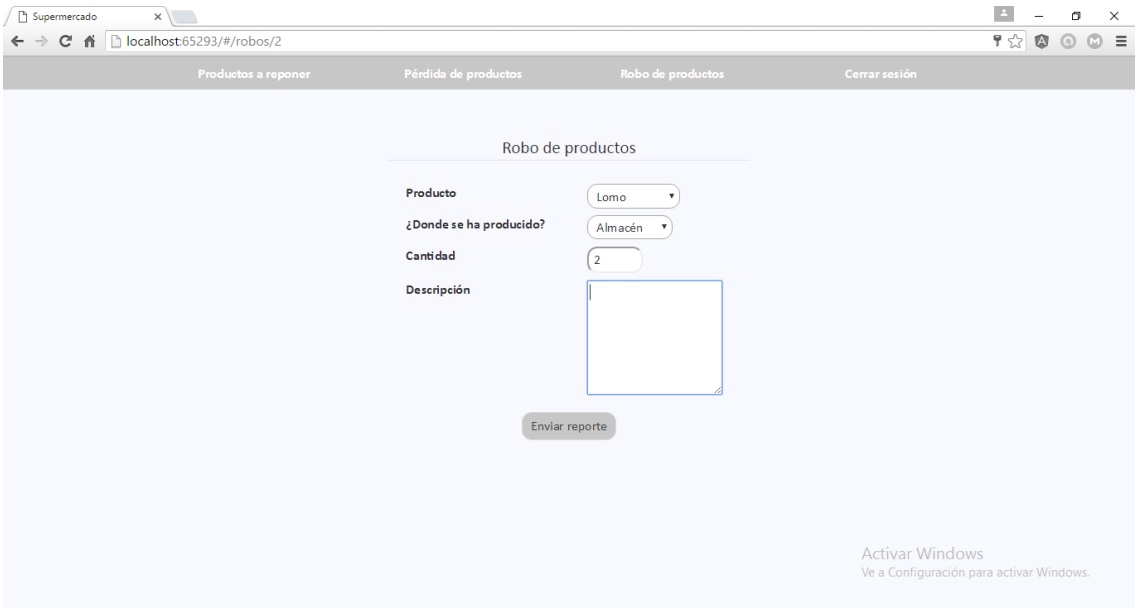
Una vez se tengan estos datos introducidos se permitirá introducir la cantidad de producto perdido/robado, donde se controla que el número de unidades esté entre 1 y la cantidad máxima del lugar indicado en la incidencia.

Por último al introducir la cantidad se muestra la opción de introducir una descripción de la incidencia antes de almacenarla en base de datos. Se controla que la descripción no contenga mayor número de caracteres que los permitidos para ese campo en la base de datos. Mediante directivas angular es posible controlar también el tipo de caracteres introducidos en los campos de texto, como descripción por ejemplo. Se utilizan patrones que hacen que cuando el contenido del campo no los cumple se realiza el deshabilitado del botón, por lo que se evita la inyección de código SQL que permita al usuario poder obtener cualquier información de la base de datos.

Se ha tenido en cuenta que el usuario pueda cambiar algún valor escogido para alguno de los campos requeridos una vez tenga todos rellenos, es decir, podría darse el caso de que el reponedor seleccione un producto determinado en una ubicación, seleccione un número determinado de ese producto y le ponga una descripción. En ese caso, sin enviarlo, puede darse la situación de que decida cambiar el producto, por lo que en estos casos el control que se realiza es el borrado de todos los datos relacionados

con el producto escogido anteriormente. Si no se tuviese en cuenta, podría darse el caso de intentar meter en base de datos un reporte indicando que se han perdido 20 unidades de un producto concreto ubicado en una zona determinada cuando realmente en esa ubicación había menos de 20 unidades y además no se encuentra ese producto en ese lugar.

Al enviar esta información se almacenará siguiendo el procedimiento de siempre al acceder a base de datos, y lo que hará es descontar esa cantidad de producto perdido/robado en el lugar indicado en el reporte.



Gerente

Gestión de productos

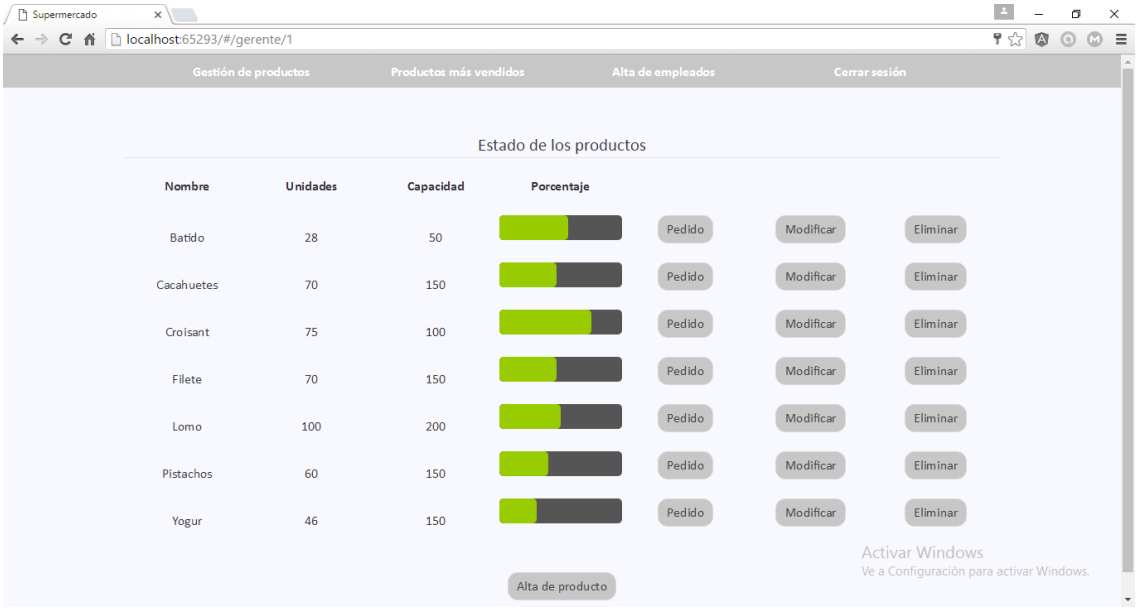
Al acceder a este apartado redirige a la gestión de productos. Lo primero que hace el controlador de esta vista es cambiar la variable indicada en la factoría para realizar el mostrado del menú de gerencia. Una vez hecho esto, accediendo a los datos obtiene un listado de todos los productos ordenados alfabéticamente y que permite ver al usuario el número de unidades de cada producto en toda la superficie (almacén + estanterías), y el stock máximo que se puede tener. El porcentaje de estos datos se muestran

mediante un gráfico integrado en la aplicación para que a la persona logueada le resulte más cómodo y porque refleja la realidad del stock de forma más visual que los datos. Se tiene la posibilidad de realizar un pedido para cada producto del número de unidades que equivale a la diferencia entre lo que hay y lo que se puede tener (la diferencia entre el stock actual, y el tipo de baremo que tiene cada producto). Al realizar esta acción se simula el pedido por lo que se añadirá en base de datos la cantidad que supone la diferencia de esas dos cantidades. Si en el supermercado ya hay el número máximo de producto permitido, esta opción se deshabilitará y no se dará la opción de hacer el pedido de ese producto (un pedido de cero productos).

Además en este apartado existe un CRUD de productos. La modificación de algún producto supondría solamente la posibilidad de modificar el tipo de baremo de ese producto, y el precio de compra y de venta del mismo. Se controla que el baremo escogido sea uno de entre los que existen en la base de datos y que el precio de venta nunca sea inferior al precio de compra del producto.

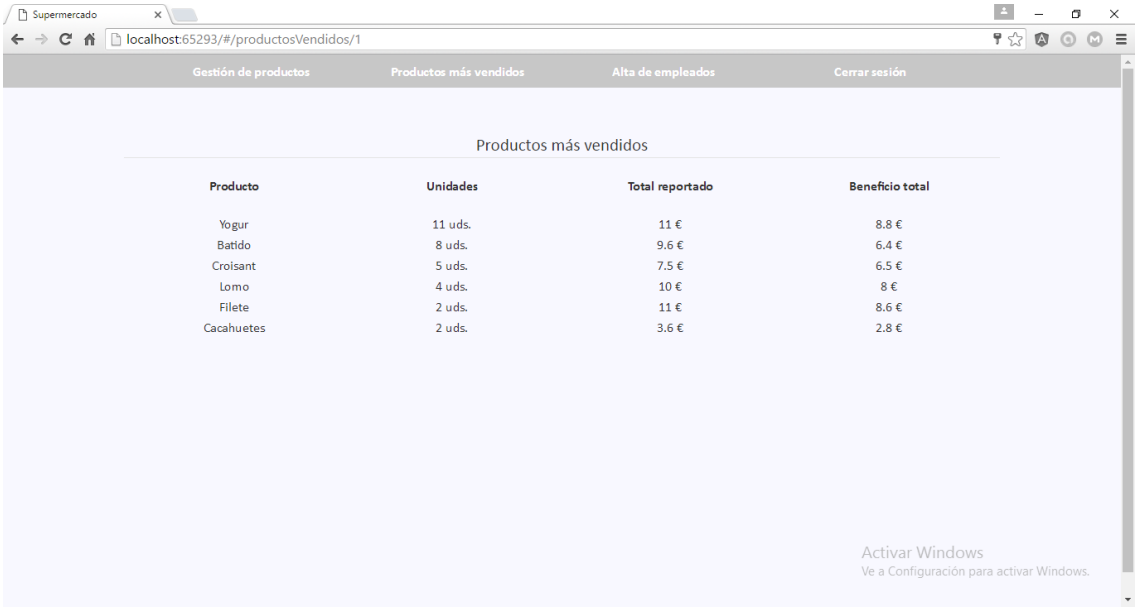
Al eliminar un producto, realmente no se realiza un borrado, ya que no interesa que la empresa pierda información. En el fondo es una actualización de un atributo del producto, en la que se pondrá en no disponible. Ese producto no existirá más a ojos del cliente, pero se podrá utilizar para ver información de ventas por parte de la gerencia.

Al realizar el alta de un producto se requieren todos los campos ya que son obligatorios en base de datos. Se controla que la longitud de los atributos no exceda lo permitido en base de datos, que los precios puesto sean siempre positivos y el de venta nunca menor que el de compra y que la cantidad por balda del producto sea entre 1 y 50. De esta manera se asegura que los datos no producirán errores en BBDD. Se controla también que todos los campos de texto contengan únicamente letras y números, de lo contrario no se podrá realizar el alta.



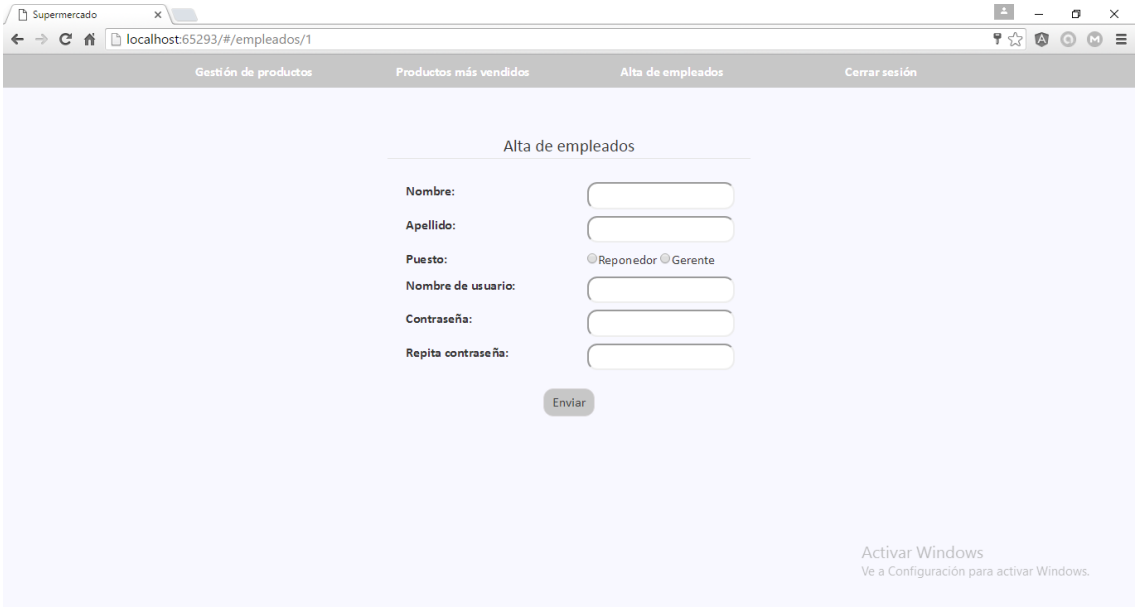
Productos vendidos

En esta vista se muestran los productos ordenados por unidades vendidas. Aunque no se asegure que el producto más vendido sea el más rentable, se puede decir que es el más exitoso. De cada uno de los productos se muestra la cantidad de unidades vendidas, el ingreso total que se ha obtenido con él y el beneficio que supone teniendo en cuenta el dinero invertido en su adquisición. Es una vista meramente informativa.



Alta de empleados

El controlador de esta vista se encarga de gestionar nuevas altas de empleados. De la misma manera que anteriormente, se controla que todos los campos hayan sido rellenados, de tal forma que cada campo no exceda de un número determinado de caracteres. Se controla también que las cadenas introducidas no contengan caracteres extraños, que las contraseñas introducidas coincidan entre ellas y que el nombre de usuario seleccionado no exista. También se controlan los caracteres introducidos para evitar la inyección de código.



Problemas con el proyecto

Los problemas que se han tenido con el proyecto no han sido muy numerosos. El primero de ellos es el desconocimiento de alguna de las tecnologías utilizadas antes de iniciar el proyecto como puede ser AngularJS. Sí que se había visto algo de .NET y un poco más de SQL, pero se ha tenido que invertir bastante tiempo en el aprendizaje de las mismas.

El siguiente problema que se ha apreciado es que al empezar a aprender estas tecnologías, no se ven las soluciones de la manera que se ven cuando uno sabe más de ellas, por lo que seguramente se hubiese estructurado el código de una manera más limpia y coherente en algunas ocasiones.

Conclusiones y trabajo futuro

Este proyecto podría darle otro enfoque a la gestión de las superficies comerciales. De esta manera los empleados pueden obtener información real en tiempo real con el objetivo de que sus horas trabajadas sean más eficientes.

Además se ofrece un mejor servicio a los clientes que podría servir también para su fidelización, sobre todo si les parece un sistema cómodo de utilizar en la que se les ayude a gestionar sus compras.

Aun así hay varias casuísticas que no se han trabajado en este proyecto y que mejorarían el producto. Se puede suponer que un producto concreto pueda ocupar varias baldas contiguas ya que se estima que ese producto va a ser muy reclamado por los clientes. Si un comprador coge un producto de cualquiera de esas baldas, al introducirlo en el carrito, se descontará una unidad de ese producto en cualquiera de esas baldas. Esto en principio no supone un gran problema ya que es posible que cuando los productos comiencen a agotarse al reponedor le aparezca que tiene que reponer un producto en una ubicación determinada pero que realmente la ubicación que haya que reponer sea la contigua. El problema puede darse cuando algún producto está en varias ubicaciones separadas, como puede darse con los productos que están en su ubicación habitual, pero que además decide ponerse también al lado de los cajeros para su promoción. En esta situación no estaría controlada esta casuística, no es posible saber de dónde se ha cogido un producto por lo que el sistema en este sentido es mejorable.

Tampoco se ha contemplado el caso de que no todos los productos van por unidades. En este aspecto, la aplicación gestiona los productos por unidades, donde una unidad podría ser 1 yogur, 1 paquete de yogures, etc. El problema reside en los productos a granel por ejemplo, en la que es imposible la gestión a través de esta aplicación.

Una solución pasaría por que la superficie comercial decidiese quitar los productos a granel y pasar a vender ese tipo de productos como empaquetados, pero en la actualidad, el caso de las carnicerías o pescaderías, seguramente impedirían esta implantación por el pensamiento social sobre este tipo de productos envasados.

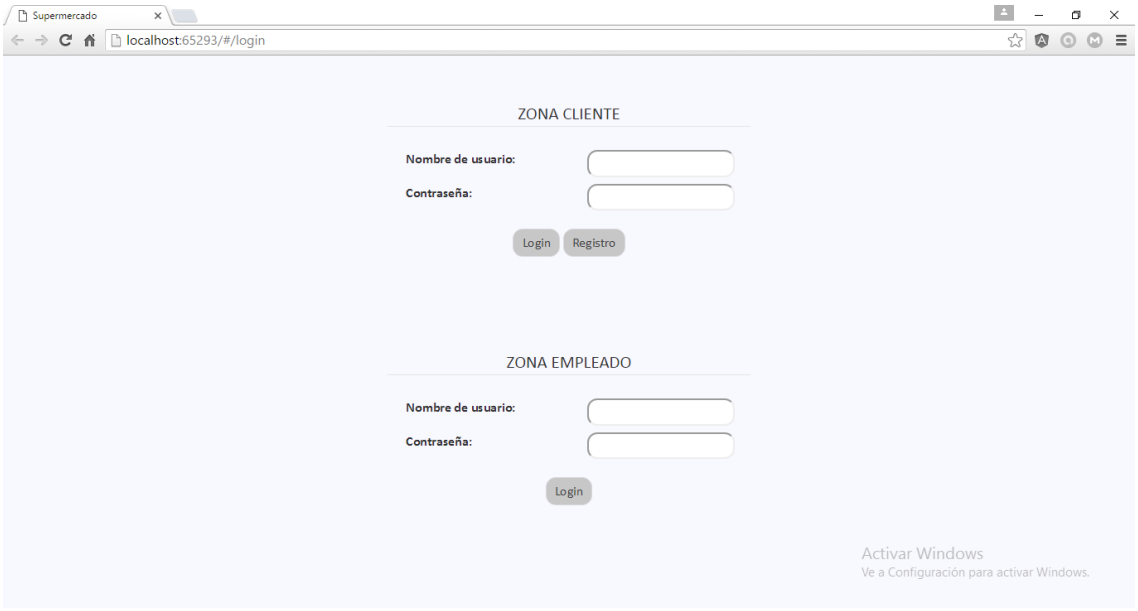
Bibliografía online

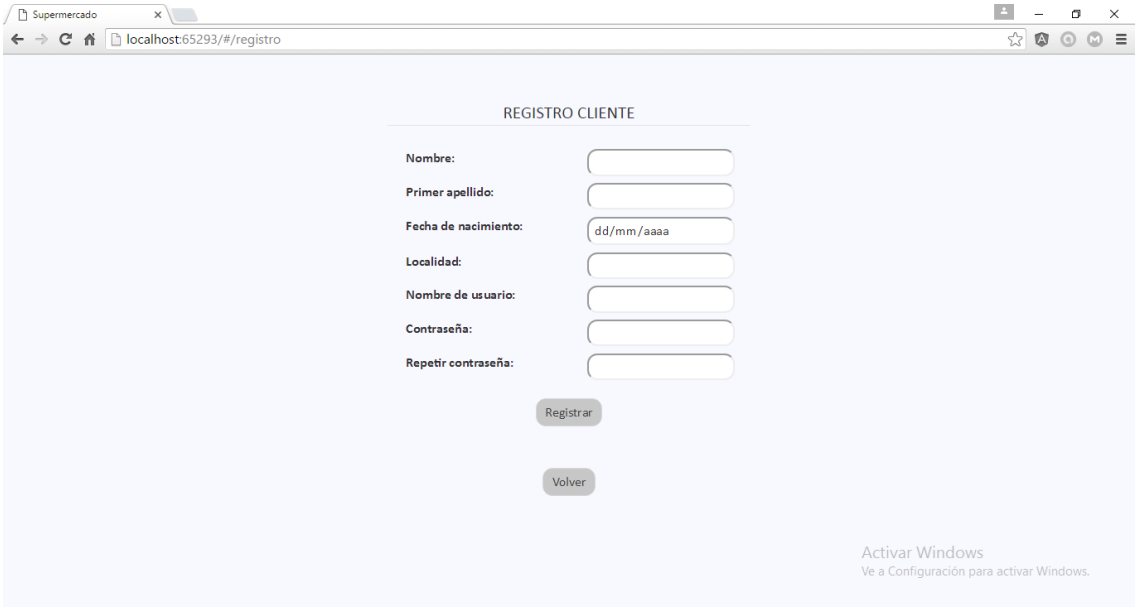
- Curso de AngularJS - http://eduardo-alfaro.herokuapp.com
- Curso de AngularJS - http://www.cursoangularjs.es
- API de AngularJS - https://docs.angularjs.org/api
- Ejemplos prácticos .NET y AngularJS - https://www.youtube.com
- Entity Framework - http://blogs.msdn.com/b/adonet/archive/2010/07/19/absolue-beginners-guide-to-entity-framework.aspx
- Web API - https://msdn.microsoft.com/es-es/library/hh833994(v=vs.108).aspx

Anexo – Manual de usuario

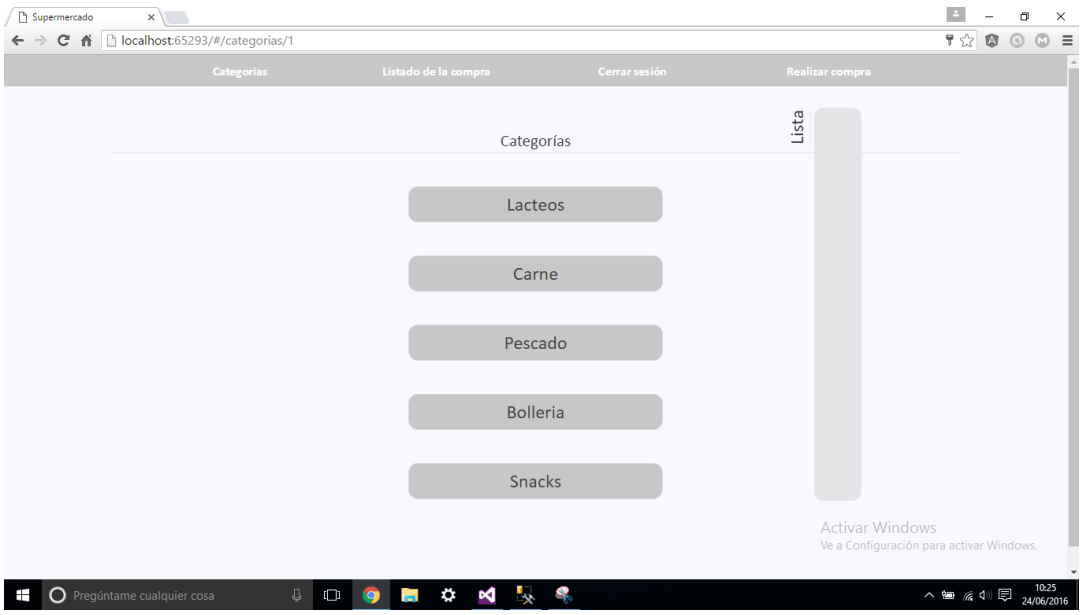
Cliente

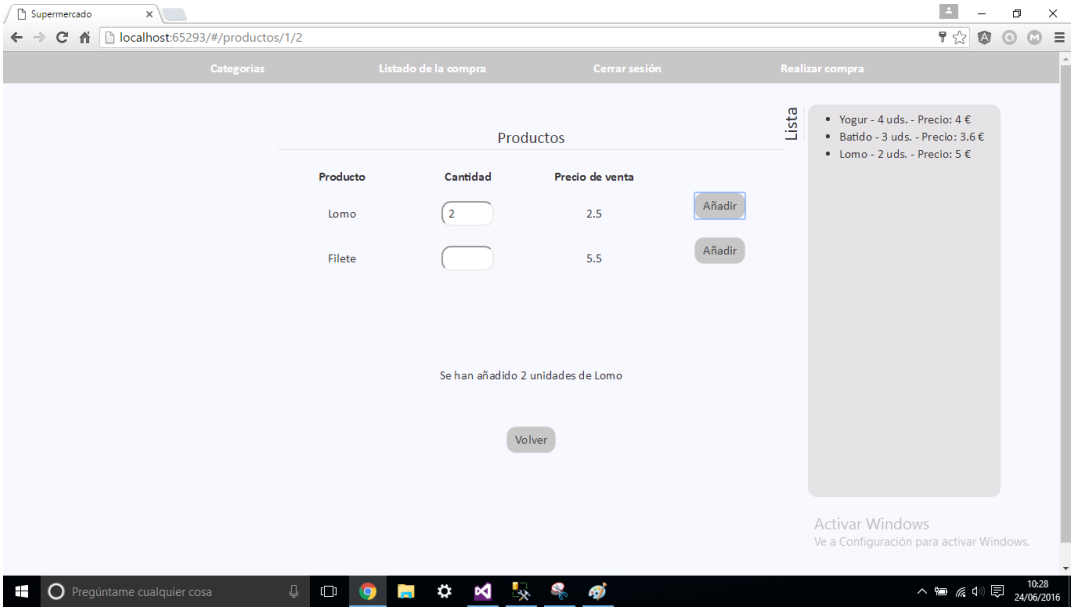
Si el cliente no dispone de una cuenta de usuario en la aplicación, tiene la posibilidad de registrarse.



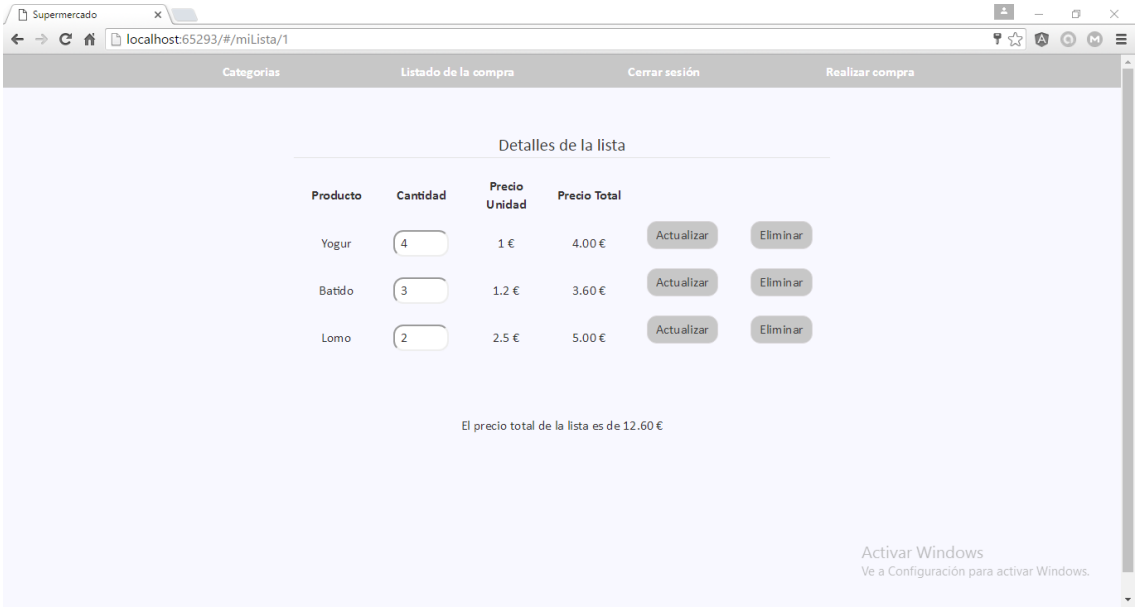


Una vez registrado y logueado, se observan las diferentes categorías de las que dispone el supermercado y accediendo a una de ellas, se pueden ver los diferentes productos contenidos en esa categoría.



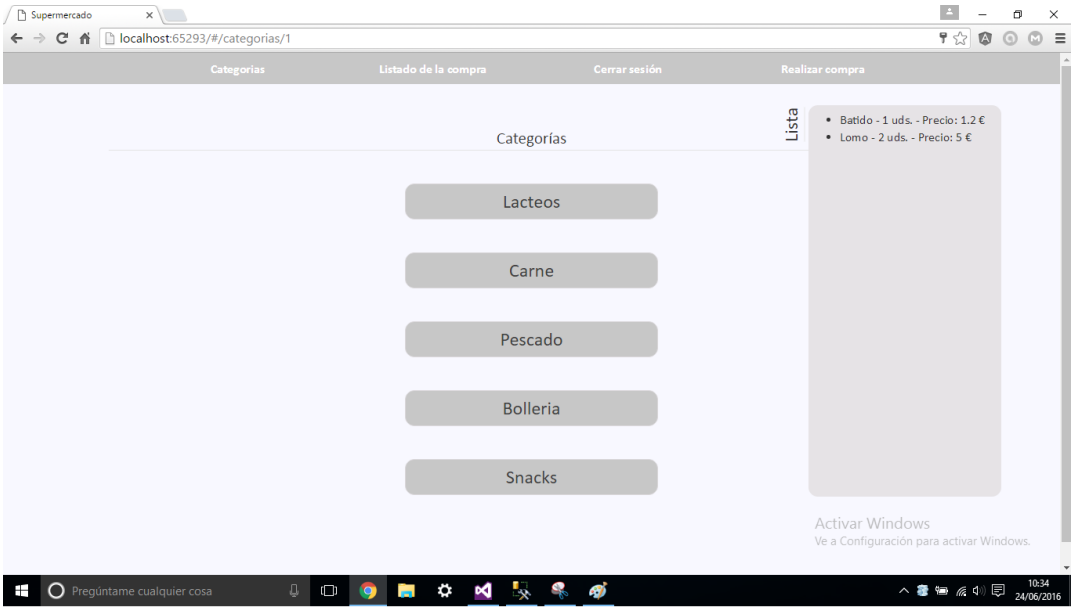


En ese lugar se pueden añadir diferentes productos a la lista de la compra, que se puede observar en una pestaña desplegable en la parte derecha de la pantalla. Al añadir los productos van apareciendo en la lista. Si se quiere eliminar o modificar la cantidad de algunos de los productos que se tienen en la lista, yendo a través del menú a la opción *Listado de la compra* se puede hacer.



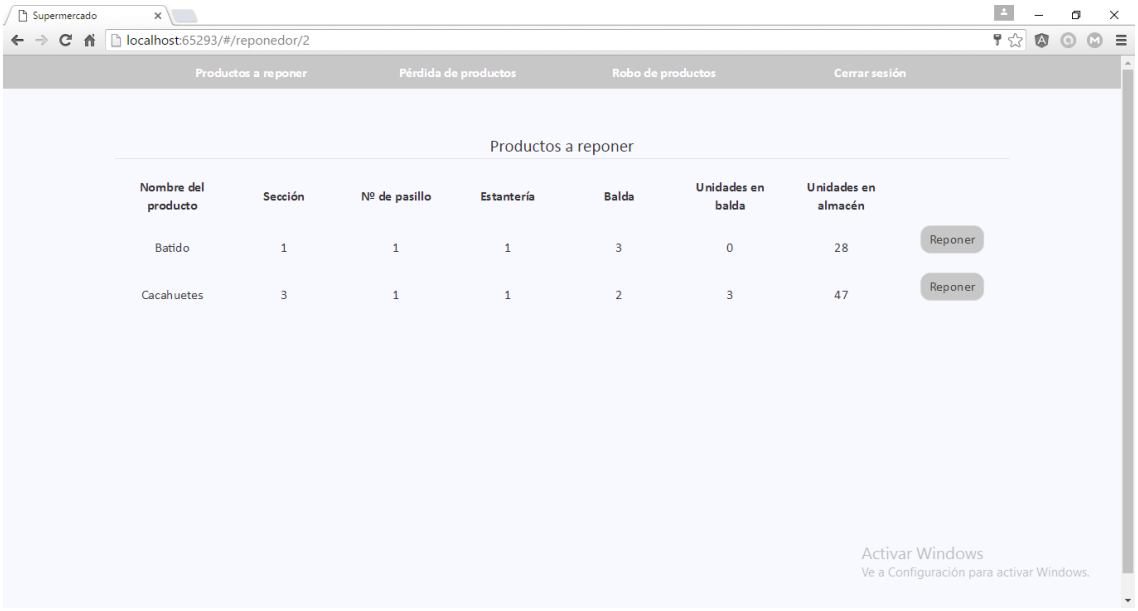
Si se cierra la sesión y cualquier otro día se vuelve a iniciar, se observa que la lista sigue siendo la misma, por lo que la lista no se pierde.

Cuando se realice el pago de los productos al realizar la compra, se eliminarán automáticamente de la lista los productos que se hayan comprado y estén en ella. Si alguno de esos productos estaba en la lista pero se ha comprado en menor cantidad, se actualizará la cantidad en la lista de la compra.

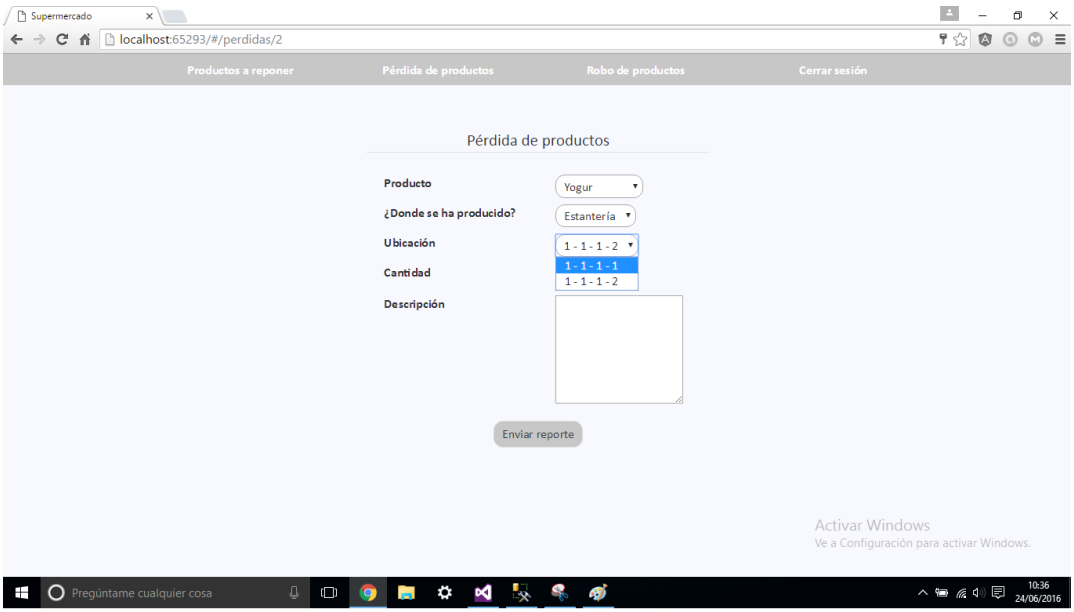


Reponedor

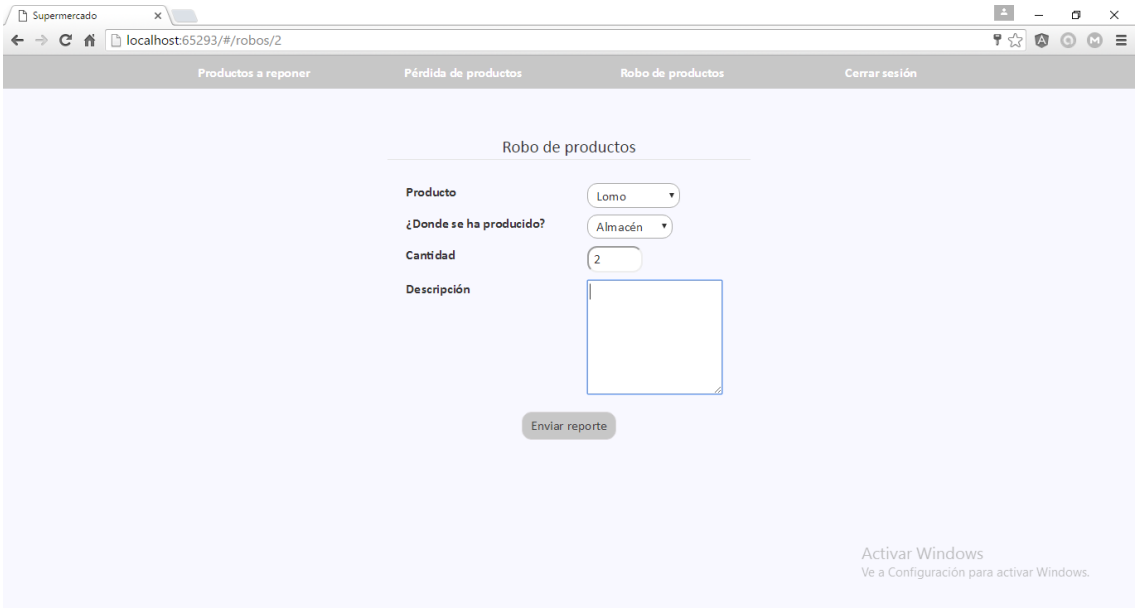
Al loguearse en el apartado de empleados, se obtiene un listado de los productos que están pendientes de reponer con su ubicación y la cantidad de unidades que hay en el almacén. Una vez se realice este trabajo, se notificará pulsando el botón *Reponer*. Cuando algún producto aparezca en esta lista hay que reponerlo por completo.



Si en algún momento se produce la rotura de algún producto, en el apartado de *Pérdida de productos* habría que rellenar una incidencia informando de dicha pérdida de productos.



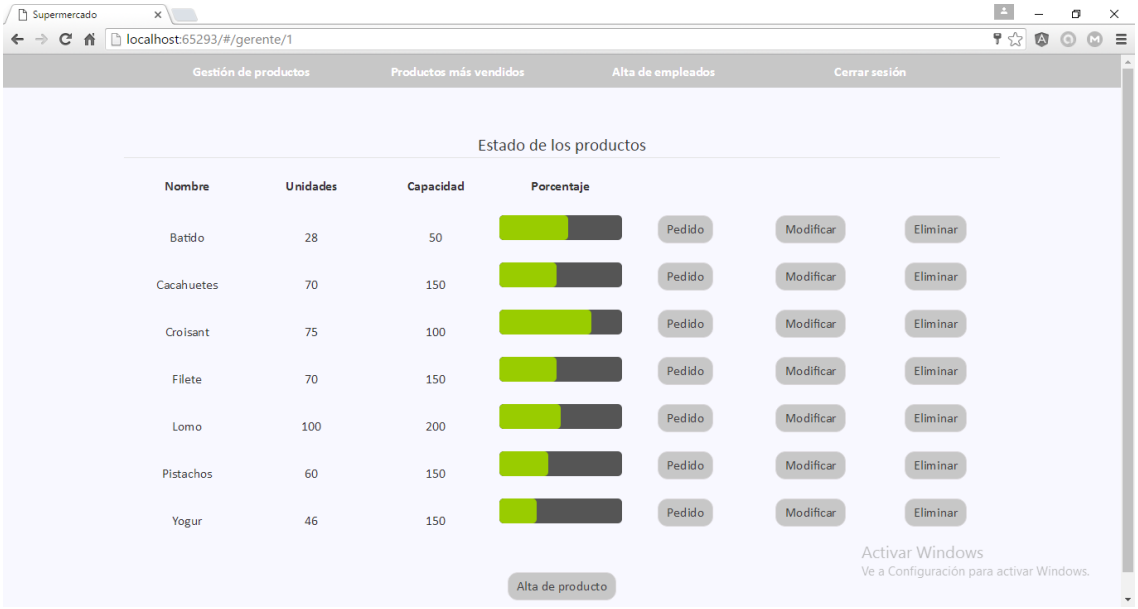
Lo mismo ocurrirá cuando se perciba que falta algún producto. Esto se gestionará según indique la empresa (semanalmente, diariamente, etc.). Al realizar el recuento de los productos en balda y en almacén, si se percibe que no corresponden las unidades con las que deberían, la diferencia de producto se mete como incidencia de robo en la pestaña de *Robo de productos*.



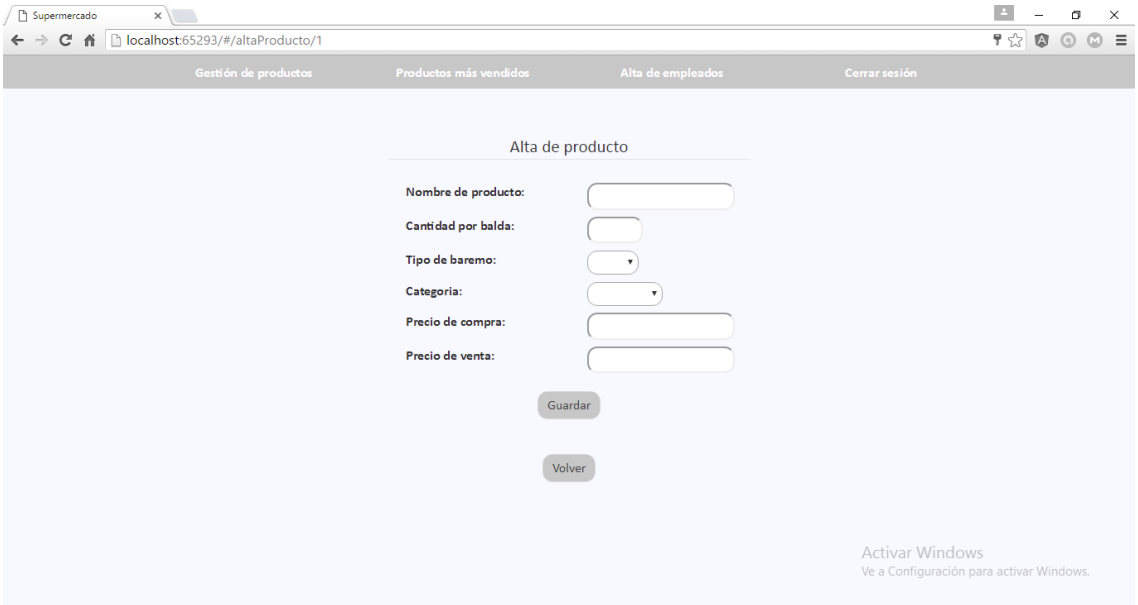
Cuando se acabe de realizar estas tareas se puede cerrar la sesión.

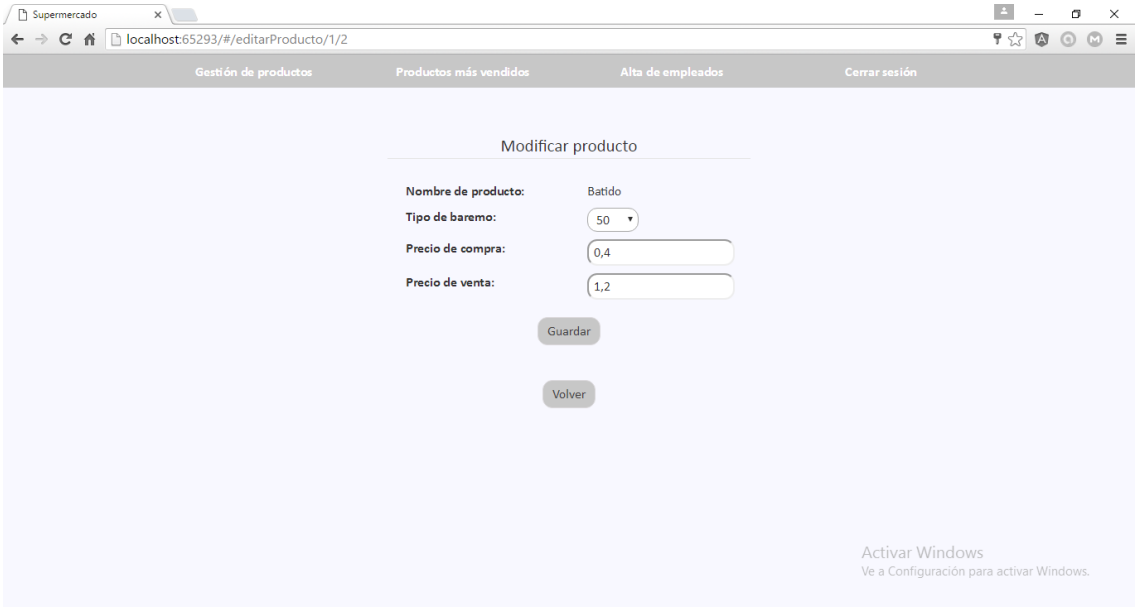
Gerente

Al realizar el login se muestra un listado de productos disponibles en la superficie comercial indicando las unidades existentes y las unidades máximas que pueden tenerse en el local. Se muestra el porcentaje del estado de los productos mediante un gráfico. Si se considera se puede realizar un pedido del producto en base a los baremos indicados para cada uno de ellos.

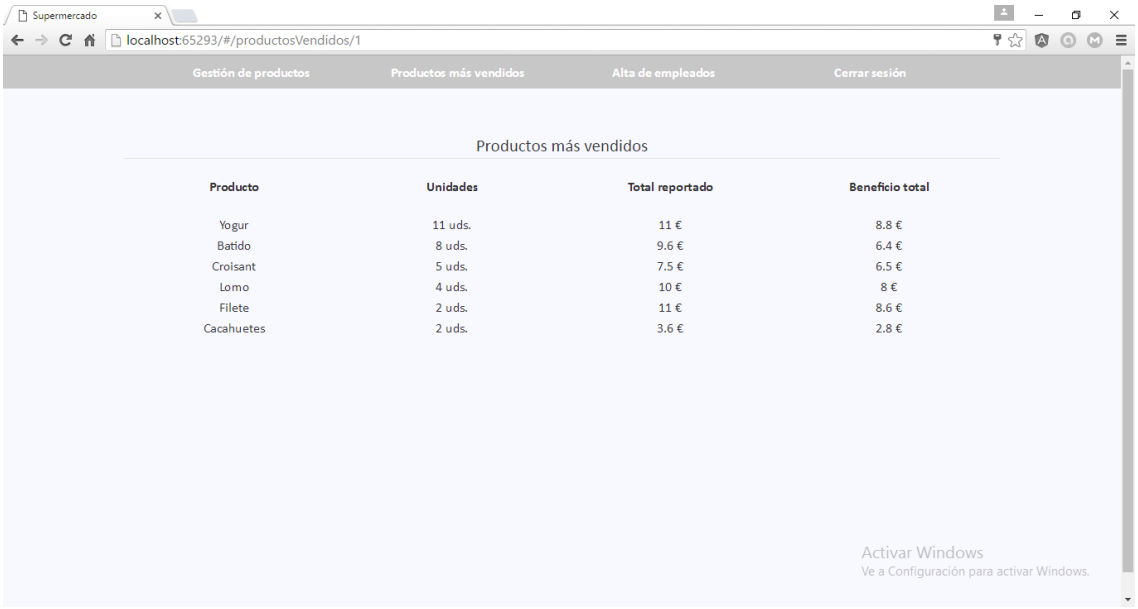


Es posible dar de alta nuevos productos, modificar los existentes o eliminarlos si no se van a vender más en el supermercado, pinchando en los botones correspondientes. Las modificaciones sirven para cambiar precios y el tipo de baremo del producto.





Si se accede al apartado de *Productos más vendidos* puede verse un listado de los productos ordenados de manera que el primero que aparezca es el más vendido y el último el que menos. También puede verse las unidades vendidas, y el ingreso/beneficio obtenido con cada uno de ellos.



En el apartado *Alta de empleados* se puede dar de alta a un nuevo empleado cumplimentando los datos requeridos.

Terminada la gestión se puede cerrar la sesión.