

C.P.R. Liceo “La Paz”

Proyecto Fin de Ciclo

Desarrollo de Aplicaciones Multiplataforma

Autor : Gabriel Rodríguez Díaz  
Tutor: Jesús Ángel Pérez-Roca Fernández

ÍNDICE

Contenido

[Resumen 3](#_Toc74339996)

[Palabras Clave 4](#_Toc74339997)

[Motivación 7](#_Toc74339998)

[Estado del arte 8](#_Toc74339999)

[Viabilidad Tecno-Económica 9](#_Toc74340000)

[Diagramas 13](#_Toc74340001)

[Detalles técnicos 33](#_Toc74340002)

[Bibliografía 36](#_Toc74340003)

[Dependencias 36](#_Toc74340003)

[Conclusiones 3](#_Toc74340004)7

[Futuras líneas de investigación 37](#_Toc74340005)

RESUMEN

La aplicación consiste en una solución **multiplataforma**, **escalable** y **modular** para una empresa dedicada a venta de productos, ya sea de forma mayorista o minorista. También sirve para empresas cuyo modelo de negocio sea la fabricación de productos.

Aunque se puede utilizar cualquiera de sus módulos de forma aislada si así se desea.

La aplicación cuenta con módulos de gestión de productos distribuidos en secciones, lo que hace posible su funcionamiento aunque la empresa entre en nuevos mercados.

Respecto a los productos, se almacenan su precio, su coste y su tipo de IVA, el beneficio obtenido… Facilitando así la contabilidad. También se almacenan otros datos relevantes: referencia, modelo, etc…

Se almacenan pedidos, cuyas líneas contienen el producto y las unidades de éste; el cliente y el día en que se ha realizado.

Se puede mantener la jerarquía de la empresa mediante los perfiles. Un empleado tiene asignado un perfil y las funciones que podrá desempeñar en la aplicación variarán en función del perfil que tenga asignado.

Mediante la aplicación se puede consultar la información en gráficos y también se pueden generar facturas.

La base de datos cumple con la [RLOPD artículo 93.3](https://www.boe.es/buscar/pdf/2008/BOE-A-2008-979-consolidado.pdf) asegurando así la confidencialidad de las credenciales de los usuarios almacenadas en la base datos.

PALABRAS CLAVE

**Sistema informático:** Acorde a Sebastian K. Boell y Dubravka Cecez-Kecmanovic, podemos definición sistema informático de la siguiente forma: Un sistema informático es un proceso de sistema cuyos procesos y actividades están orientados al proceso de información: capturar, transmitir, almacenar, recuperar, manipular y mostrar información.

En este documento nos centraremos especialmente en el punto de vista tecnológico y en el punto de vista de los procesos informáticos.

**Escalabilidad:** Que la aplicación sea escalable significa que es capaz de soportar una serie de cambios en el modelo de negocio.

A nivel técnico esto implica que cualquier campo de selección dentro de un formulario recogerá información de base de datos.

[**JavaFX**](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784)**:**  Consiste en una tecnología que permite integrar una serie de paquetes multimedia en tu aplicación Java. Es una alternativa más estética (aunque menos completa) a su precursor, Java Swing\*[[1]](#footnote-1).

Como su nombre indica, se utilizan archivos XML (.fxml) para diseñar las GUI. También permite el uso de CSS.

**SceneBuilder:** Herramienta complementaria a JavaFX que permite diseñar la GUI de forma gráfica arrastrando componentes (drag and drop).

**JFoenix:** Librería de JavaFX que implementa componentes basados en el estándar [Material Design](https://material.io/design/introduction) de Google.

**SQL Server:** Es un sistema gestor de base de datos relacionales desarrollado por Microsoft y que utiliza el lenguaje TransactSQL\*[[2]](#footnote-2). El lenguaje TSQL nos permite añadir lógica de negocio en la base de datos, lo que lo hace un gestor muy potente.

En este caso se han explotado funciones propias de SQL Server mediante procedimientos almacenados.

**Procedimiento almacenado:** Consiste en una función almacenada en la propia base de datos que se ejecuta utilizando el motor de la misma. Permite añadir lógica de negocio a la aplicación desde la propia base de datos.

**Vista (BD):** Una vista es una tabla virtual formada a partir de los resultados de una consulta. Al igual que en una tabla normal, sobre ésta se pueden ejecutar queries.

**Singleton\***[[3]](#footnote-3)**:** Es un patrón de diseño que nos permite crear una instancia de un objeto durante toda la vida de la aplicación. En este caso se ha utilizado para almacenar de forma global la información del usuario que ha iniciado sesión.

MODELO UML DEL PATRÓN SINGLETON

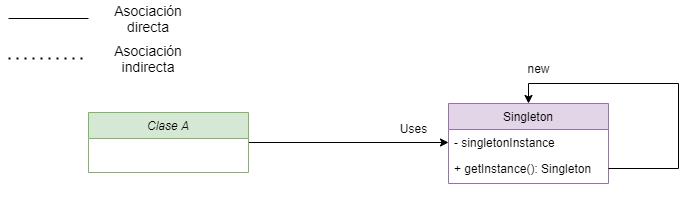


Imagen de elaboración propia usando la herramienta [Draw.io](https://app.diagrams.net)

[**DAO**](https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/#:~:text=Para%20esto%2C%20tenemos%20el%20patrón,al%20resto%20de%20la%20aplicación.)**:** DataAccessObject.Consiste en un patrón de diseño que propone la división de la lógica de negocio de la lógica de acceso a datos. Dejando así un código más limpio donde la clase DAO se encargaría exclusivamente de las operaciones CRUD trabajando con POJOs. El uso de POJOs implica a su vez la implementación del patrón de diseño [DTO](https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/).

MODELO UML DEL PATRÓN DAO

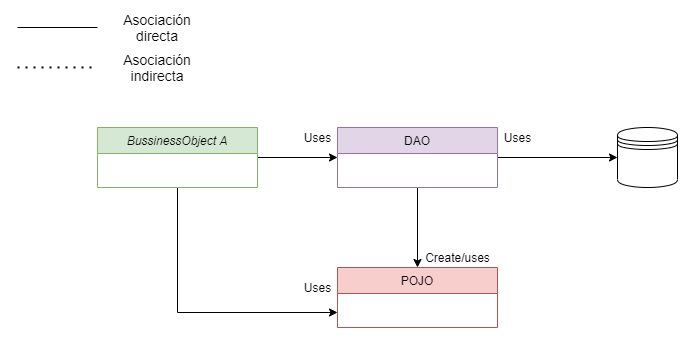


Imagen de elaboración propia usando la herramienta [Draw.io](https://app.diagrams.net)

[**Hashing & Salting**](https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/)**:** Son dos técnicas de criptografía. El hashing consistiría en aplicar una función de cifrado a una cadena de caracteres.

Ésta podría ser descifrada mediante Rainbow Tables[[4]](#footnote-4), con lo cual se le aplica Salting: Es decir, a la cadena inicial se le concatena otra cadena aleatoria para cifrar aún más la información.

[**Overload**](https://www.sitepoint.com/quick-guide-to-polymorphism-in-java/#:~:text=Static%20Polymorphism%3A,%2Forder%2Fnumber%20of%20parameters.&text=For%20the%20third%20and%20fourth,change%20of%20order%20of%20parameters.): Es una característica de Java que nos permite alcanzar el polimorfismo estático.

En la aplicación se explota en los POJOs instanciados en para las clases DAO. Se permite así utilizar distintas instancias de los POJOs en función de los parámetros requeridos por la lógica de negocio.

[**MVC**](https://www.oscarblancarteblog.com/2014/07/21/patron-de-diseno-modelo-vista-controlador-mvc/)**:** Consiste en un patrón de diseño que nos permite separar la GUI de la lógica de negocio. Para comunicar ambos componentes se utilizan clases Controller.

[**IText**](https://itextpdf.com/es/products/itext-7)**:** IText es una herramienta de generación de archivos ofimáticos para Java. En el desarrollo se ha usado la versión IText 7

[**Bootstrap**](https://getbootstrap.com)**:** Es un framework inicialmente pensado para diseño web que facilita el desarrollo de aplicaciones responsive y otorga una serie de componentes para mayor productividad.

[**JDBC**](https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html): Se trata de una API que permite la ejecución de operaciones sobre base de datos utilizando Java. Necesita un driver adecuado a su SGBD.

MODELO UML DEL PATRÓN MVC

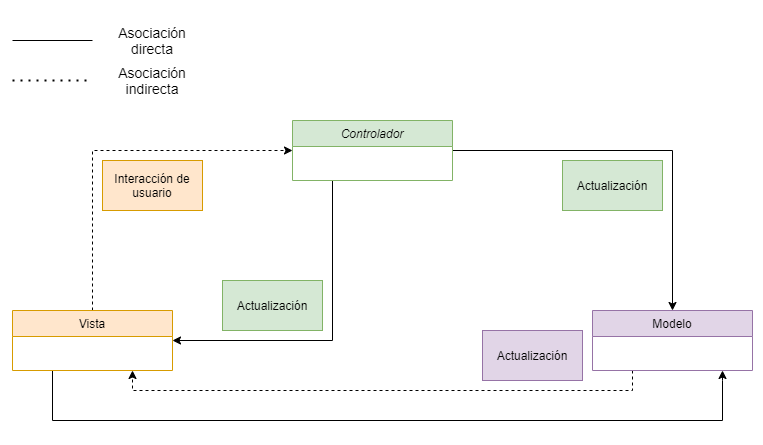


Imagen de elaboración propia usando la herramienta [Draw.io](https://app.diagrams.net)

MOTIVACIÓN

Tanto siendo alumno de FP medio como FP superior realicé mi FCT en VegaGestión S.L., una consultora tecnológica de Viveiro. Esta empresa trabaja con el sistema Ahora ERP.

Allí fue donde vi por primera vez un ERP funcionando y descubrí la capacidad que tenían de aumentar la productividad de una empresa.

En mi estancia allí descubrí un problema recurrente en los ERP. Este es la cantidad de GB que ocupa tan sólo para instalarlo. Además de eso, al ser herramientas tan potentes, en muchos casos tienen más funcionalidades de las necesarias y se ven tablas vacías en la base de datos.

Con mi aplicación pretendo crear una solución sencilla: Una herramienta que permita llevar a cabo una digitalización sin necesidad de utilizar programas con un excedente de funcionalidades y que además tenga una interfaz sencilla e intuitiva.

De ahí que todos los menús y los controles funcionen igual en todo el programa.

Creo que los desarrolladores debemos estar dispuestos a dar soporte, pero siempre intentar que no sea necesario en ningún caso. Evitemos los errores y las interfaces poco intuitivas.

ESTADO DEL ARTE

La aplicación, a pesar de no ser tan completa como otras soluciones del mercado, no deja de ser un ERP.

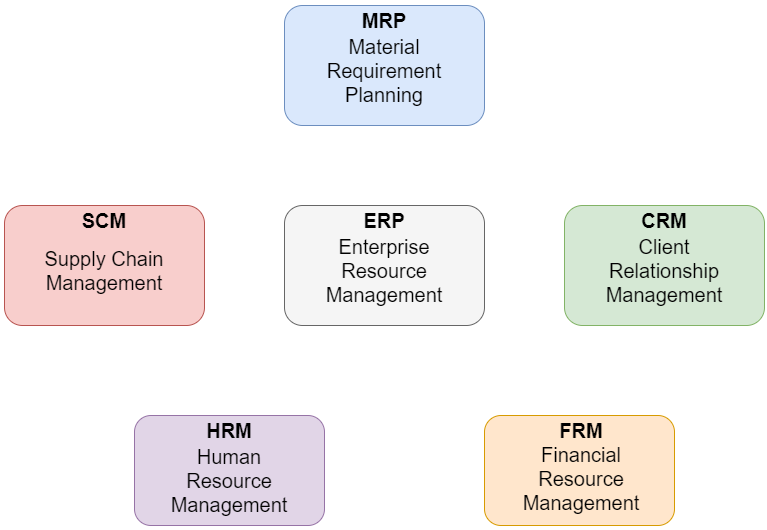


Imagen de elaboración propia usando la herramienta [Draw.io](https://app.diagrams.net)

Los ERP existen aproximadamente desde los años 90 y en el mercado hay muchas soluciones. Pasando de las más potentes y asentadas como SAP a otras más humildes como Syspro.

Existen incluso ERP web como [Flexygo](https://www.flexygo.com), de la empresa valenciana Ahora.

El elemento común en todas estas soluciones es que son herramientas muy pesadas que requieren de técnicos especializados para implementarlas.

Desde la app se gestionan empleados, precios de venta, costes, pedidos, productos y stocks. Así que podríamos decir que cuenta con los principales módulos para empezar a ser considerada un ERP.

La alternativa que ofrece la aplicación aquí expuesta es la sencillez de la misma, la facilidad de instalación con tutoriales guiados y lo ligera que es.

Otras aplicaciones como Ahora ERP trabajan con bases de datos de varios GB y contienen multitud de tablas que no siempre se usan, como ya se ha mencionado en apartados anteriores.

VIABILIDAD TECNO-ECONÓMICA

Figurémonos que este producto se vende con un modelo de negocio similar a una empresa consultora tecnológica.

Supongamos también que el modus operandi (en un supuesto general) de la susodicha empresa es el siguiente:

EJEMPLO DE FUNCIONAMIENTO

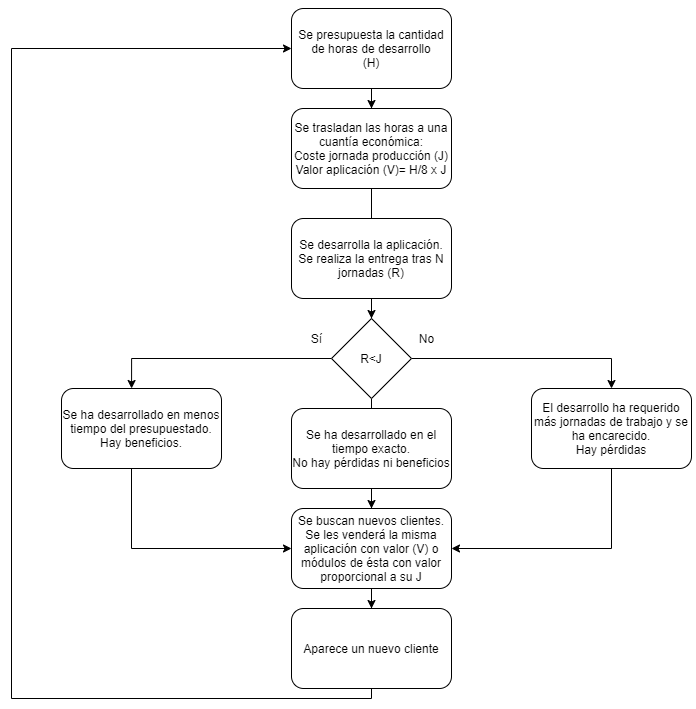


Imagen de elaboración propia usando la herramienta [Draw.io](https://app.diagrams.net)

Basándonos en el modelo expuesto arriba, la aplicación no tiene por qué ser rentable en primera instancia. Aunque sería lo recomendable para maximizar el beneficio, ésta será rentable a largo plazo. Esto siempre y cuando los clientes demanden aplicaciones similares y se pueda reutilizar código.

Para ello se recomendaría hacer una serie de estudios de mercado previos.

El presupuesto del desarrollo de este programa fue de 26 horas. Tomando como referencia la programación que figura en el DOG[[5]](#footnote-5).

Haciendo una serie de búsquedas en páginas de empleo, pongamos, por ejemplo [indeed.com](https://es.indeed.com/career/desarrollador-junior/salaries) vemos que el salario de un programador junior oscila entre los 18.000 y 20.000€ euros brutos al año.

Averigüemos ahora el valor de cada jornada de producción (J)

J = 18.000 ÷ 12 ÷ 30 = 50€

Calculemos ahora el coste figurado de la aplicación (V)

V = 26 ÷ 8 X J = 162,5€

Sin embargo, este programa ha llevado más horas de las presupuestadas, consecuentemente, se buscaría la rentabilidad a largo plazo propiciada por la modularidad en el código de la aplicación.

Estas serían las bases sobre las que se ha trabajado respecto a la viabilidad económica en la producción. Pasemos ahora a otras formas de facturación.

Este producto, recordemos, ofrecido por una consultoría, es un sistema informático. Todo sistema informático es vulnerable y está sujeto a errores en mayor o menor medida.

La consultoría informática es una empresa que, además de crear aplicaciones, lleva el soporte de las mismas.

El modelo de negocio planteado consistiría en una licencia de mantenimiento que se cobraría en todo caso. En esta licencia entraría la solución de errores.

Por otro lado existe la bolsa de horas. Dicha bolsa de horas se vendería a cierto precio, y se consumirían las horas en cada petición de soporte para modificar características de la aplicación.

Con este modelo de negocio ya tendríamos una nueva fuente de ingresos a corto, medio y largo plazo, pues funcionaría durante todo el ciclo de vida de la aplicación.

Otra fuente de ingresos serían las formaciones. Con la aplicación se entregarán unos cursos en vídeo donde se explica cómo usar la aplicación. Para aquellas personas que requieran de una enseñanza personalizada se ofertarán clases para aprender a manejar el software.

A continuación se desglosarán los requisitos hardware de la implementación. Opcionalmente, la consultora podrá obtener beneficio del coste de los dispositivos.

**Requisitos hardware**

Atendiendo a las recomendaciones mínimas de Microsoft y aumentando el baremo para un rendimiento aceptable.

**Servidor:**

OS: Windows Server 2019

SGBD: SQL SERVER 2017

|  |  |  |
| --- | --- | --- |
| **Componente** | **Características** | **Mínimo recomendado** |
| Memoria RAM | 16GB DDR4 | 6GB |
| SSD  Disco duro | 256GB  1TB | 50GB |
| CPU | Intel Xeon E-2124 3,3GHz  4 núcleos, 4 hilos | CPU x64 2GHz |
| Adaptador de Red | Tarjeta de red 1Gb | Tarjeta de red 1Gb |
| Fuente de alimentación | Fuente de alimentación 650w |
| Refrigeración | 2 ventiladores de 120mm |
| Placa base | Adaptada al socket de la CPU |
| GPU | Integrada de la CPU |
| Caja | Genérica ATX |

Esta sería una configuración recomendada para una máquina que vaya a funcionar como servidor de una aplicación.

El precio de las licencias de Windows Server 2019 y SQL SERVER 2017 oscilaría los 275€ sin IVA

No obstante el precio es muy variable. Seguramente se podrían adquirir claves válidas por menos precios.

El precio de la máquina rondaría los 425€ sin IVA. Cantidad también variable.

El total de la configuración serían 700€ aproximadamente.

**Máquina cliente:** Para el cliente serviría cualquier portátil de uso cotidiano o cualquier ordenador de empresa.

**Viabilidad de cara al futuro:**

La aplicación, como se detallará en el apartado de líneas futuras de desarrollo, contará con un backoffice. Dicho backoffice se presupuestará de una forma más realista y será, por tanto, otra fuente de ingresos del producto.

Otras líneas futuras de desarrollo sería la creación de dos app móviles:

* Venta de una aplicación de Android e IOS orientada a los clientes, desde donde podrían hacer sus pedidos.
* Venta de un port de la aplicación a Android e IOS. El haber usado JavaFX nos facilitará el trabajo, pues la integración con el [*Proyecto Gluon*](https://gluonhq.com/products/mobile/javafxports/) permite realizar este tipo de tareas con facilidad.

DIAGRAMAS

En este apartado se mostrarán los diagramas y mockups que se han utilizado en las fases de diseño iniciales de la aplicación.

Nota: Las imágenes .png están adjuntas en la carpeta de la documentación.

A continuación se desglosarán los nombres de los diagramas en orden de aparición:

* **Diagrama ER**: El diagrama entidad-relación nos mostrará las diferentes entidades y las relaciones entre ellas. En el diagrama se han excluido los atributos y las claves en pos de una mayor facilidad de lectura.

Los elementos excluidos se mostrarán en el diagrama de base de datos.

* **Diagrama de base de datos:** Este diagrama representa las entidades y las relaciones entre éstas. Además, también se reflejan los atributos de cada entidad y las restricciones de cada columna.

En este apartado también aparecerán las cuatro vistas que contiene la base datos.

* **Diagrama de clases:** En este apartado se representará la estructura de las clases mediante lenguaje UML.

La aplicación utiliza tres patrones de diseño: DAO, MVC y Singleton (Se han mostrado los diagramas de los tres anteriormente). Ahora se verán integrados en la aplicación.

* **Diagramas de casos de uso:** Mediante los diagramas de casos de uso se mostrarán una serie de casos generales que son válidos en toda la aplicación.
* **Mockups:** Finalmente se mostrarán los mockups que se han utilizado para desarrollar la aplicación. Los mockups se han creado con SceneBuilder, pues el hecho de ser una herramienta "drag and drop” facilita tanto la creación de GUI como el visualizado de mockups.

MODELO ENTIDAD RELACIÓN

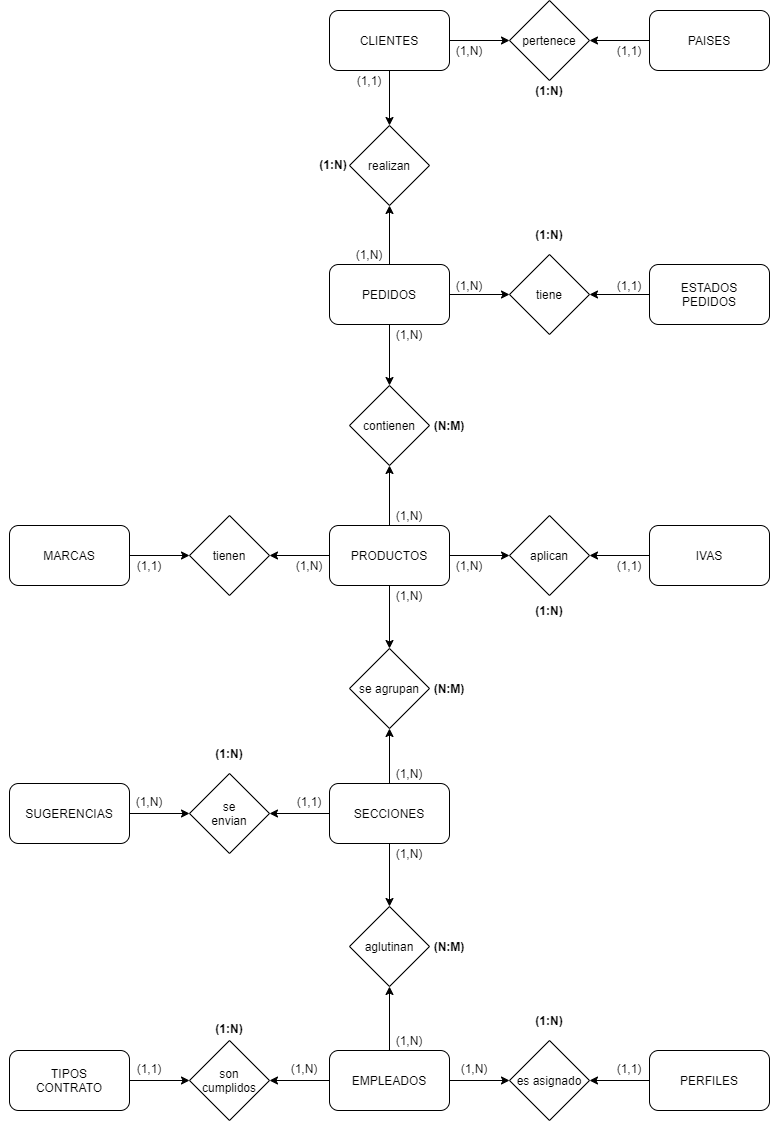
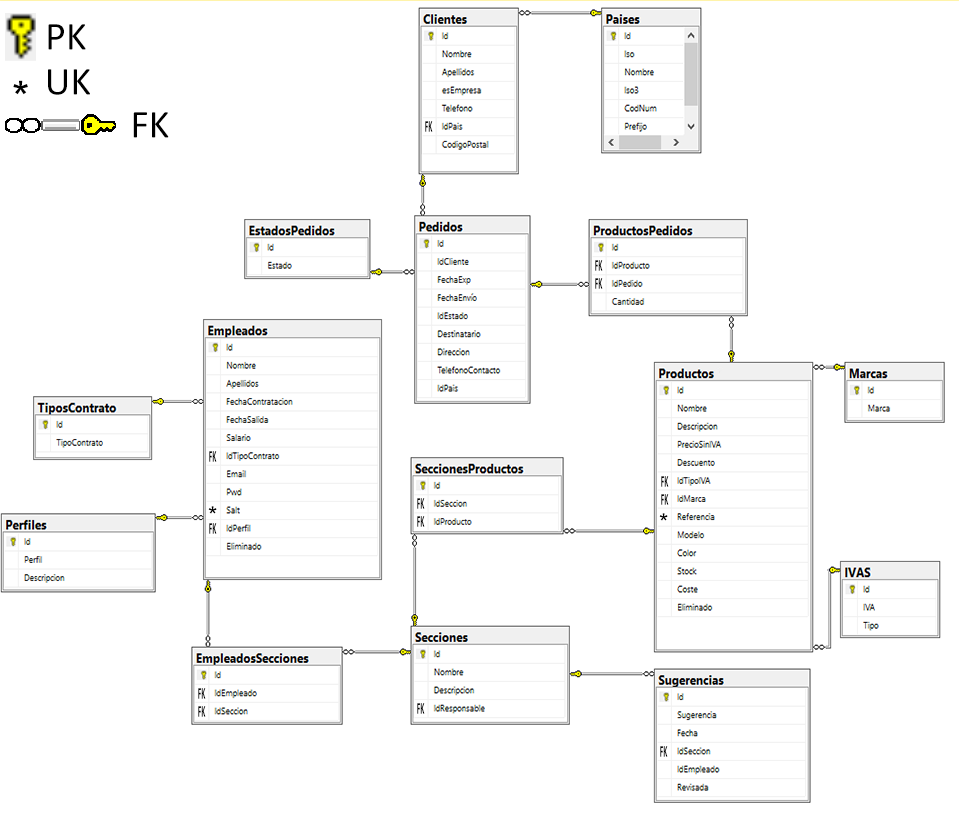


DIAGRAMA DE BASE DE DATOS



En este diagrama se han suprimido los tipos de datos para una mejor lectura. Aclaraciones respecto a los tipos de datos:

* En campos de fecha se ha usado el tipo datetime. Esto es debido a que se trata de una solución escalable, pues nos permitirá almacenar horas de ser requerido.
* En campos que tratan con cantidades monetarias se ha usado el tipo money.
* En campos booleanos se ha usado el tipo smallint.
* Para números de teléfono se ha usado varchar(15). La razón es la escalabilidad: Usar varchar abre las puertas a incluir el prefijo en el registro telefónico.

Los tipos de datos que puedan generar controversia se mostrarán a continuación:

|  |  |  |
| --- | --- | --- |
| **Tabla** | **Columna** | **Tipo de dato** |
| Clientes | EsEmpresa | smallint |
| ProductosPedidos | Cantidad | int |
| Productos | Referencia | varchar(150) |
| Eliminado | smallint |
| Descuento | float |
| Stock | int |
| Empleados | Salario | money |
| Sugerencias | Revisada | smallint |

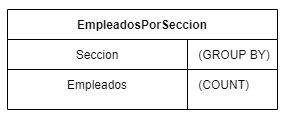
El caso de las columnas Pwd y Salt es particular. Son dos columnas que siempre funcionan juntas.

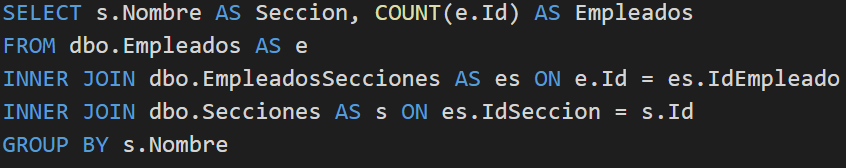
El hecho de que se usen funciones de cifrado implica que, necesariamente, el tipo de dato de Pwd sea binary(64); por otro lado, la columna Salt funciona con nvarchar(36).

Tanto Salt como Email utilizan el tipo de dato nvarchar a fin de aceptar caracteres UNICODE.

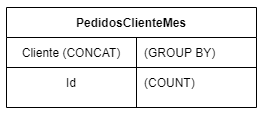
A continuación se mostrarán las cuatro vistas que tiene la base de datos:

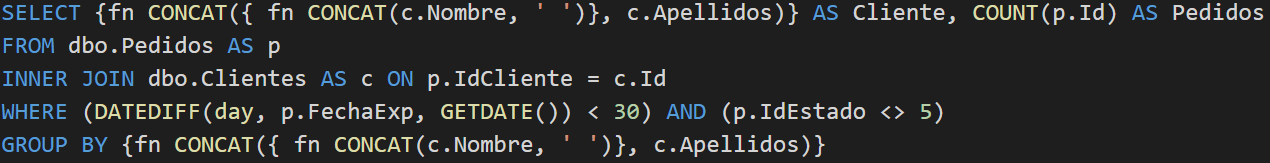
**Empleados por sección:**



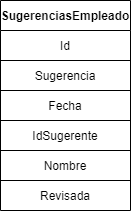


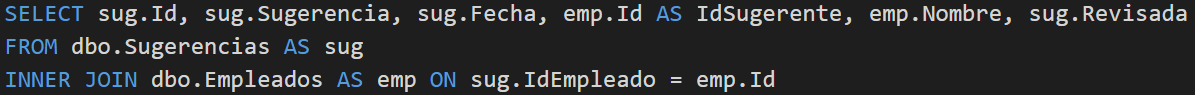
**Pedidos mensuales por cliente:**



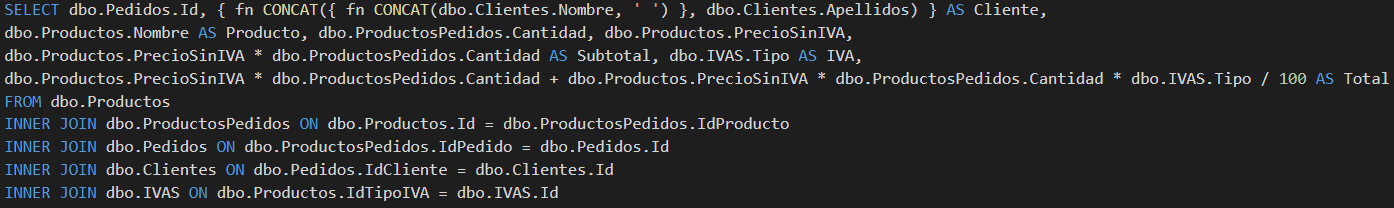


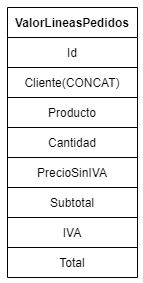
**Sugerencias por cliente:**

****



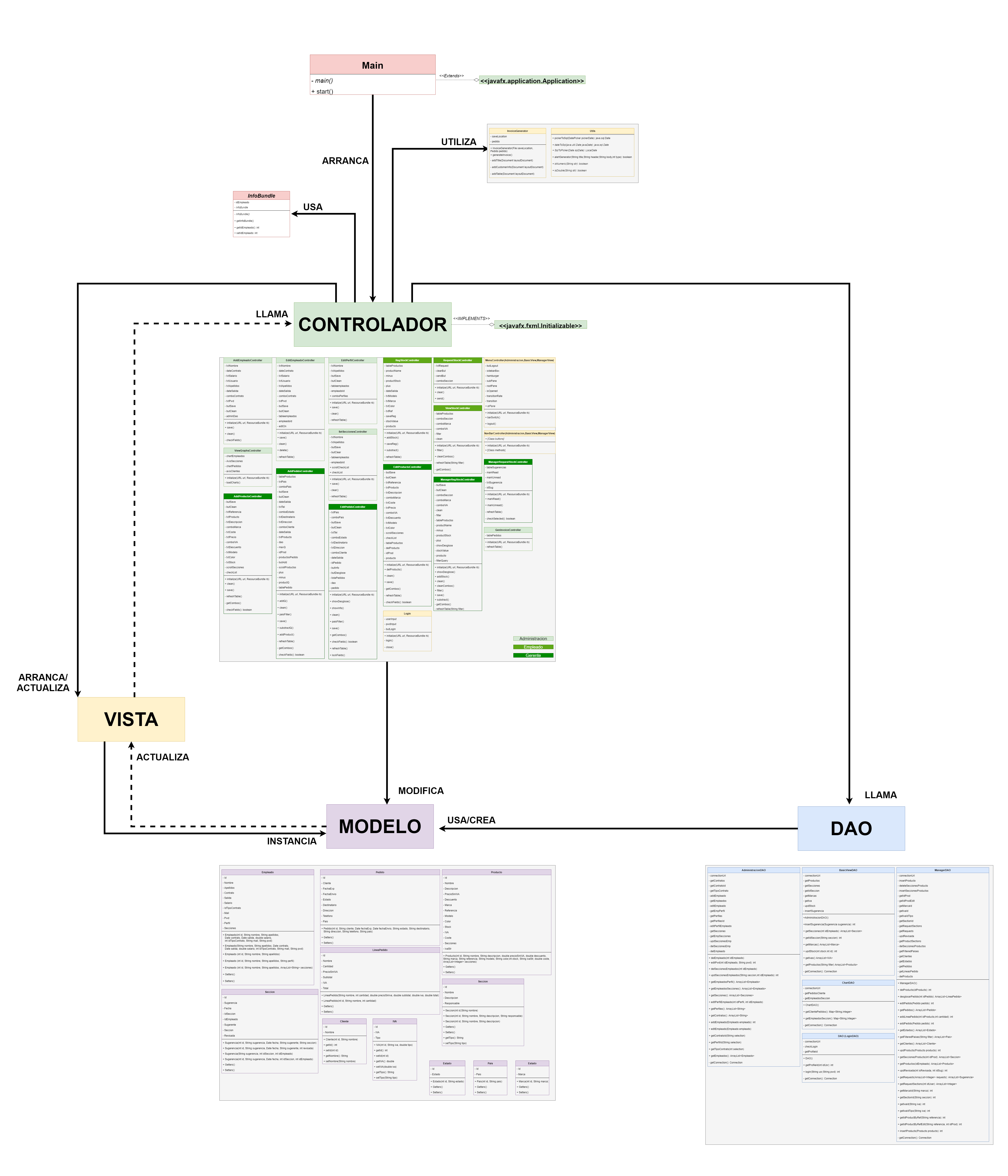
**Valor de cada línea de un pedido:**





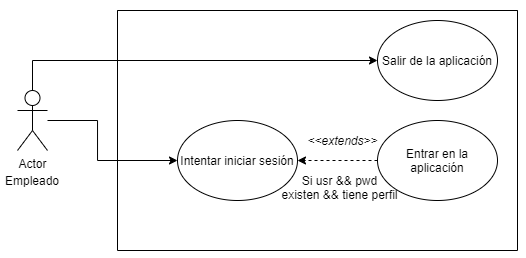
En la siguiente página se mostrará el diagrama de clases de la aplicación.

DIAGRAMA DE CLASES

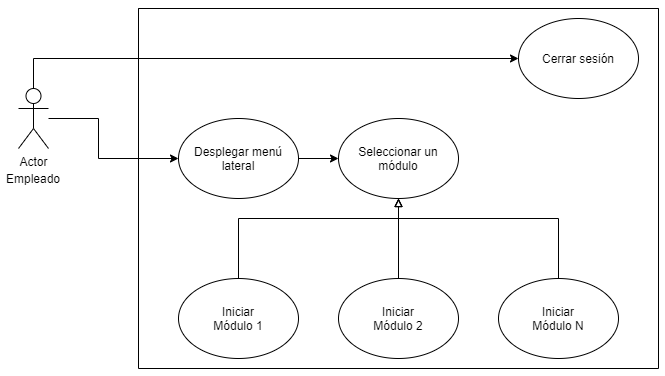


A continuación se mostrarán los diagramas de casos de uso

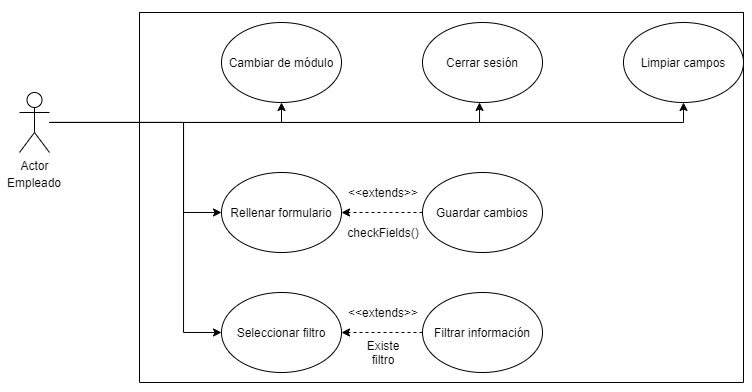
CASO DE USO: LOGIN



CASO DE USO: MENÚ



CASO DE USO: FORMULARIO GENÉRICO



Este último representa un funcionamiento general de los módulos. Es un caso genérico, pues algunos formularios tienen más funciones que las mostradas en el diagrama.

Estas acciones se detallarán en el último apartado: **Mockups.**

Para finalizar se mostrarán los mockups de los archivos FXML. Antes de continuar se hará una serie de aclaraciones previas:

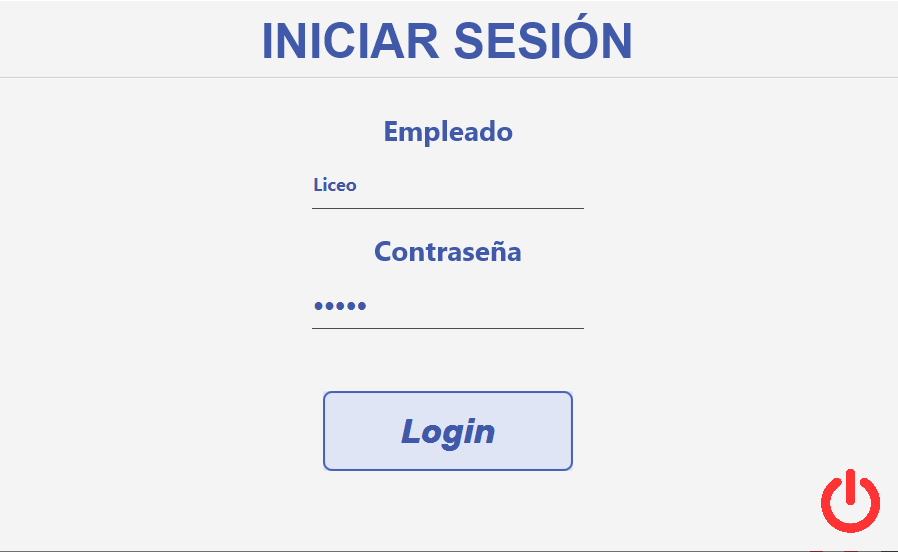
* La aplicación no es responsive. Es decir, no se adaptará dinámicamente a las distintas resoluciones de pantalla.
* Todas las ventanas son redimensionables, aunque no se recomienda cambiar el tamaño.

Esto es así para evitar posibles problemas con monitores pequeños o más antiguos.

* Los cuadros de alerta se crean mediante objetos instanciados desde una clase utilidad. No se dedicará especial atención a los mockups de los cuadros modales, pues son cuadros genéricos sin css.
* Todas las ventanas son de 900x550 excepto el menú de gerente, que es de 1170x742

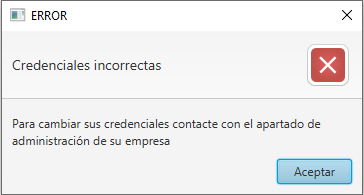
Una vez tratadas dichas cuestiones, pasemos a ver los mockups.

MOCKUP: LOGIN

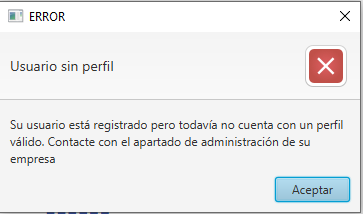


En este formulario se lleva a cabo el inicio de sesión tal y como se muestra en el caso de uso. El botón de abajo a la derecha cierra el programa.

En caso de introducir unas credenciales incorrectas mostraría el siguiente cuadro.

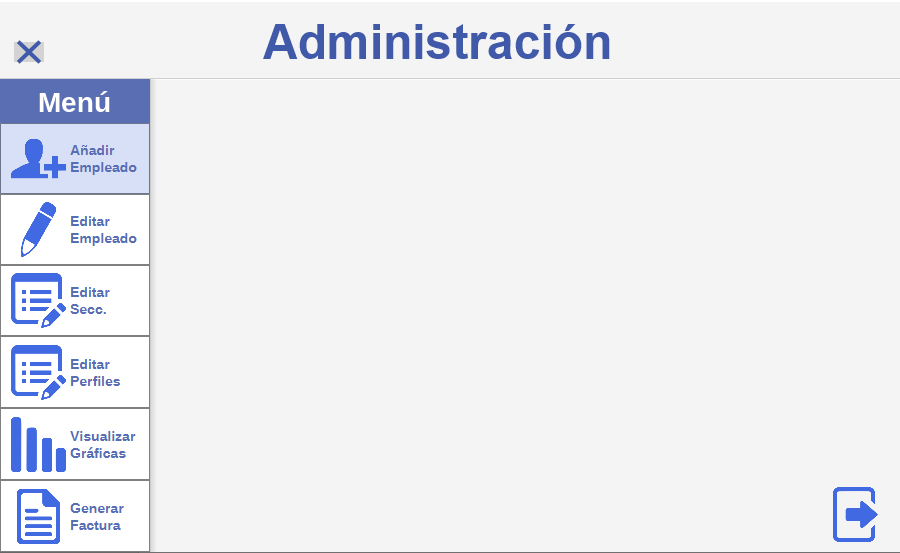


También puede darse el caso de que el empleado esté registrado y no tenga ningún perfil asignado, en ese caso, saldría este otro.



Ahora se detallarán los mockups del apartado de administración

MOCKUP: MENÚ ADMINISTRACIÓN



Cuando se hace click en alguno de los botones de la sidebar se carga en la parte central el módulo elegido. Esto es algo común en todos los departamentos.

El botón de abajo a la derecha nos sirve para cerrar sesión.

MOCKUP: AÑADIR EMPLEADO



MOCKUP: EDITAR EMPLEADO



En los formularios de edición siempre se procede de la misma manera. Se hace doble click en la tabla para seleccionar el objeto a editar.

MOCKUP: EDITAR SECCIONES



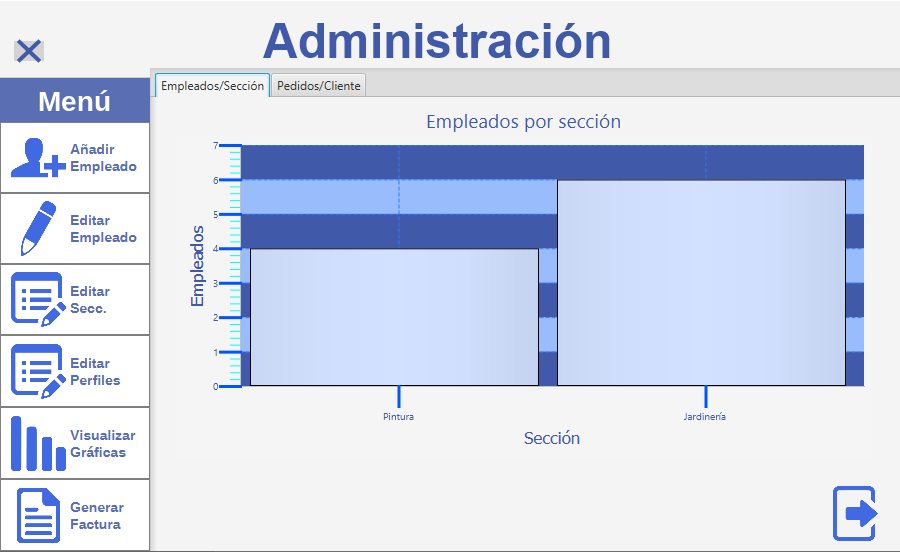
Este formulario funciona con un ScrollPane que contiene los CheckBox de secciones. Al elegir un empleado se escogen las secciones a las que pertenece.

MOCKUP: EDITAR PERFIL



Este caso funciona con la misma lógica que el anterior., simplemente se utiliza un ComboBox.

MOCKUP: VER GRÁFICAS

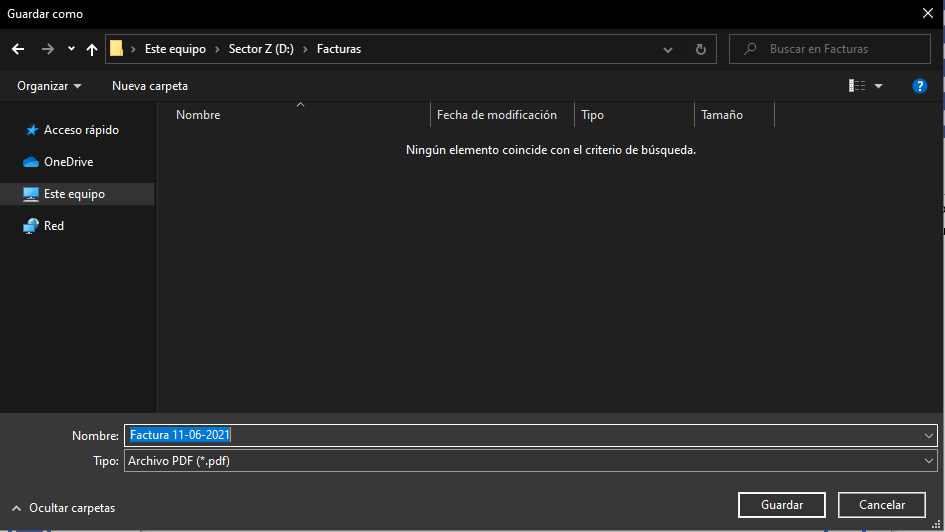


El formulario de arriba contiene un TabPane en donde se puede elegir la gráfica que se desee consultar. Ambas gráficas son BarChart.

MOCKUP: GENERAR FACTURA



En cuanto a la generación de facturas hemos de seguir el mismo procedimiento que con el resto de tablas. Doble click en el registro que deseemos. Esto nos abrirá un FileChooser para guardar nuestra factura en .pdf



Estos son todos los módulos del apartado de administración. A continuación se detallarán los del módulos perfil Mozo de almacén.

MOCKUP: VISUALIZAR STOCK

En este módulo se pueden aplicar filtros tal y como se especifica en los casos de uso. El botón de la goma serviría para borrar el filtro y mostrar la consulta por defecto.

MOCKUP: REGULARIZAR STOCK



En este apartado se puede seleccionar un producto para modificar su stock. Se modifica con los símbolos (+) y (-). El floppy disk permite guardar los cambios.

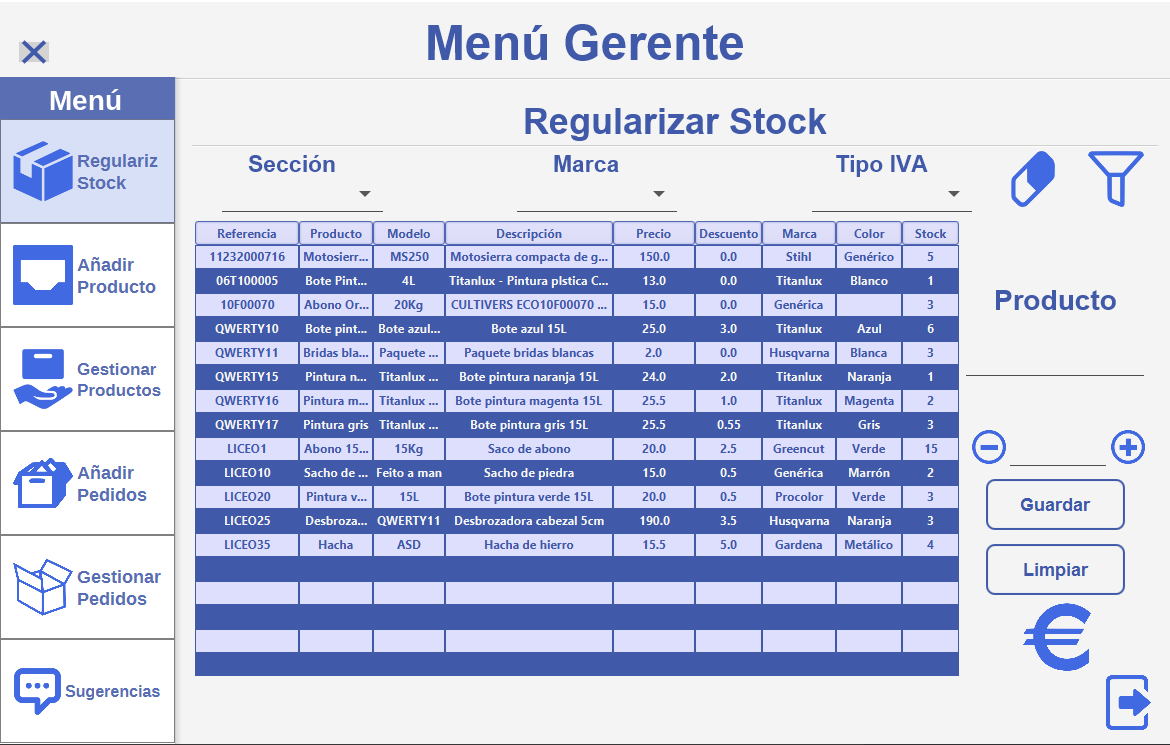
MOCKUP: SUGERENCIA STOCK



En este módulo se envían sugerencias a los gerentes de la sección seleccionada. Sólo se muestran las secciones a las que pertenece el usuario.

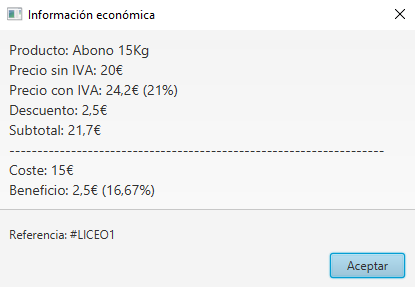
Para finalizar se mostrarán los mockups del apartado de gerencia.

MOCKUP: REGULARIZAR STOCK (GERENCIA)



En este módulo se combinan dos de las funcionalidades de Mozo de almacén: Regularizar stock y ver stock.

El botón del euro permite ver información económica del producto.

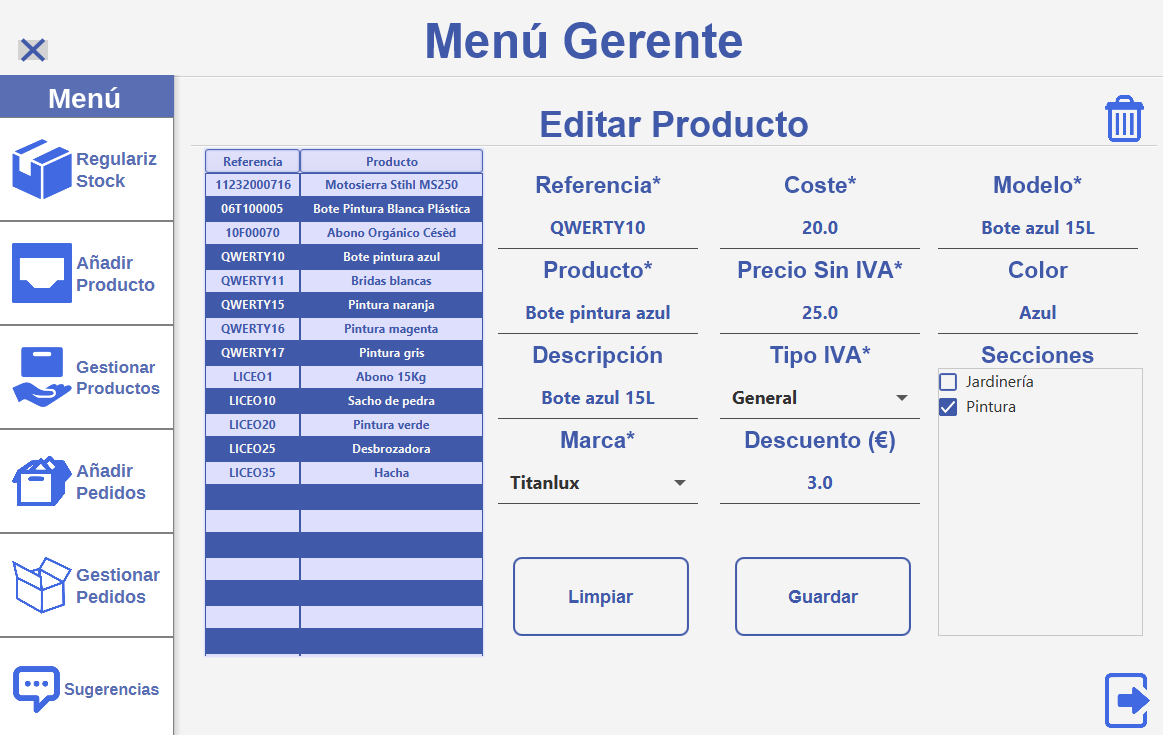


MOCKUP: AÑADIR PRODUCTO

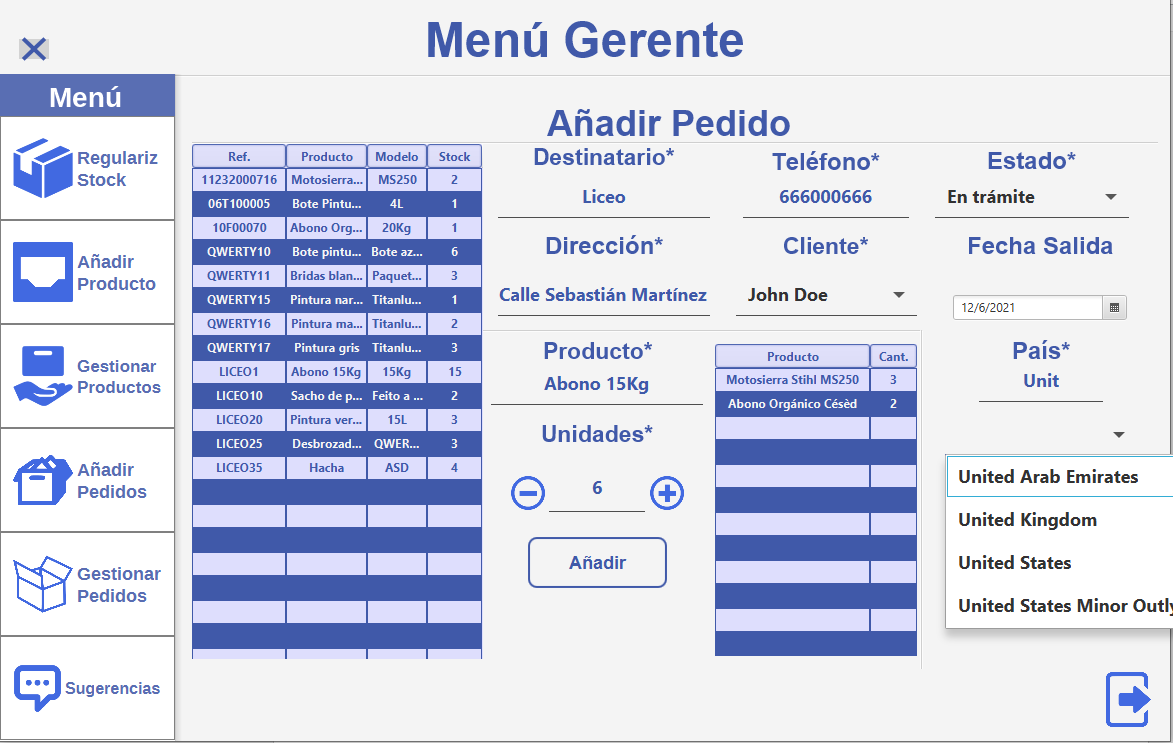


Este módulo combina varios componentes ya vistos para crear un nuevo producto.

MOCKUP: EDITAR PRODUCTO

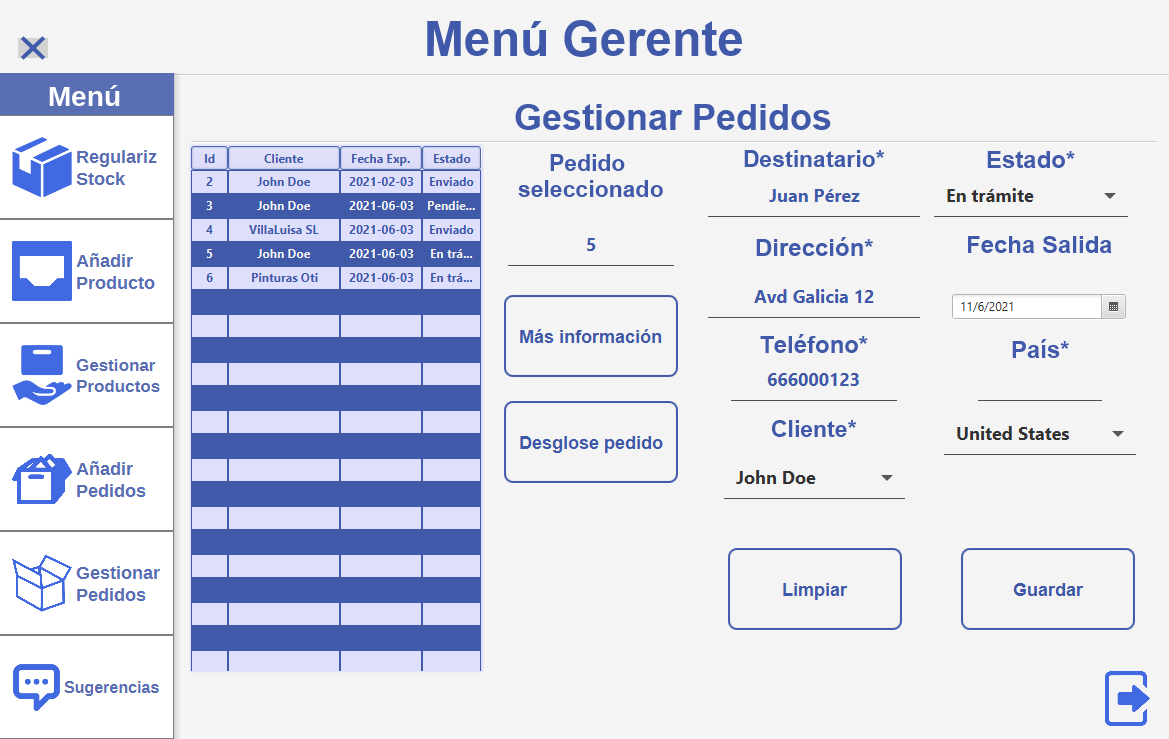


MOCKUP: AÑADIR PEDIDO

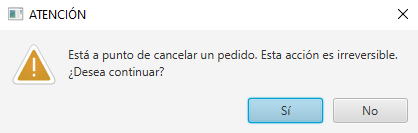


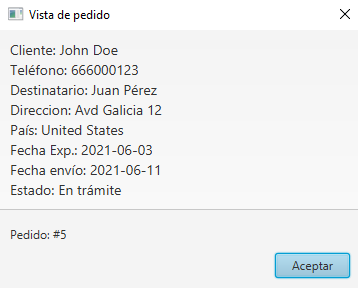
En este módulo se pueden añadir líneas al pedido. Se selecciona el producto de la tabla izquierda y se añade con el stock deseado a la tabla de la derecha usando el botón “Añadir”. También tiene un filtro por país que se activa al presionar Enter.

MOCKUP: GESTIONAR PEDIDOS

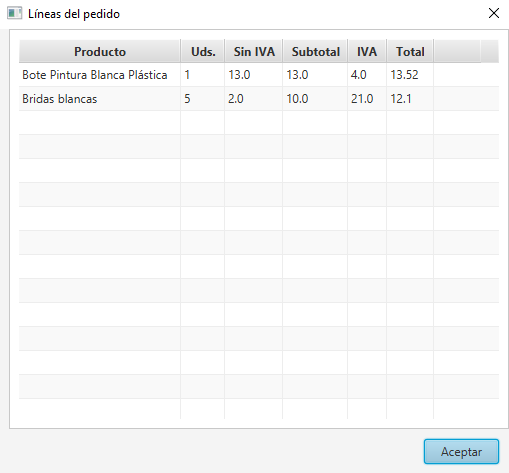


Este formulario permite editar los pedidos existentes. No se pueden cambiar sus líneas. En caso de que el estado sea “Cancelado” nos avisará de que es una acción irreversible.



El botón de “Más información” muestra información general del pedido. 

Finalmente el botón “Desglose pedido” muestra las líneas del pedido.

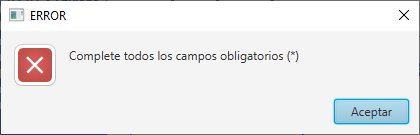


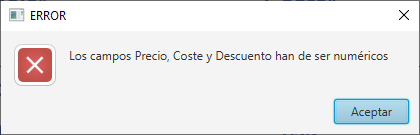
MOCKUP SUGERENCIAS

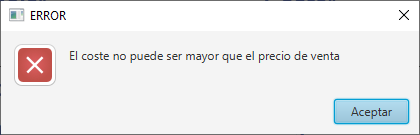
Este formulario muestra las sugerencias y permite marcarlas como leídas o no leídas.

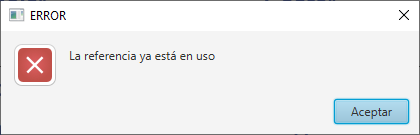
En muchos formularios de la aplicación se comprueba, , que los datos introducidos sean válidos. Todas estas comprobaciones lanzan cuadros modales con alertas en caso de no pasar las validaciones.

Algunos ejemplos serían:





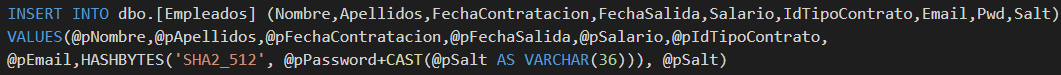




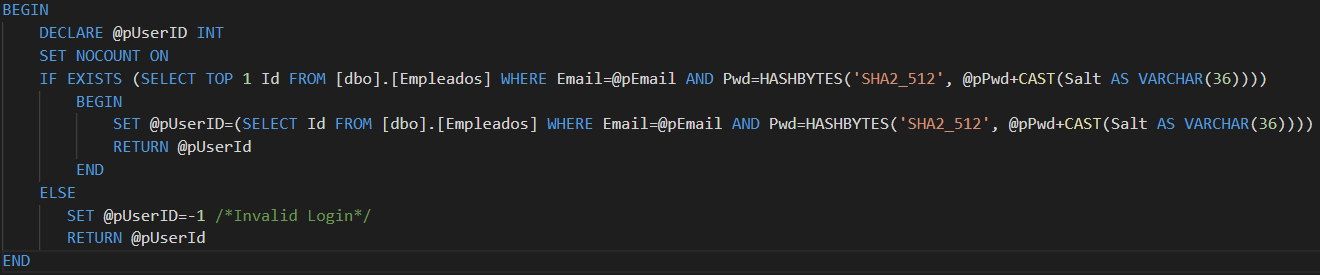
DETALLES TÉCNICOS

A continuación se expondrán una serie de detalles relevantes de forma resumida sobre el código de la aplicación.

* Para la creación de empleados se ha usado un procedimiento almacenado que aplica hashing y salting a las credenciales.



* Este sería el procedimiento para iniciar sesión:



* Hay ciertos métodos que se ejecutan desde la clase utilidad Utils. El ejemplo más recurrente es alertGenerator();



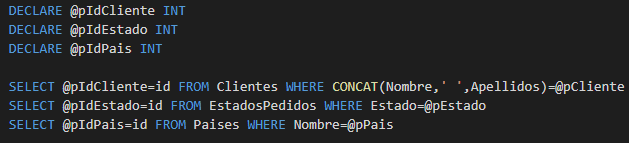
* Para generar una factura también se utiliza una clase utilidad. En este caso la clase InvoiceGenerator.
* El patrón Singleton se utiliza para almacenar en todo momento la id del empleado que ha iniciado sesión.
* El método barSwitch() se utiliza de forma recurrente para abrir y cerrar el SidePane con los botones



* Durante el desarrollo del proyecto se ha utilizado la herramienta online [Trello](https://trello.com).

El uso de tableros Kanban es uno de los pilares fundamentales de la metodología Scrum de desarrollo.

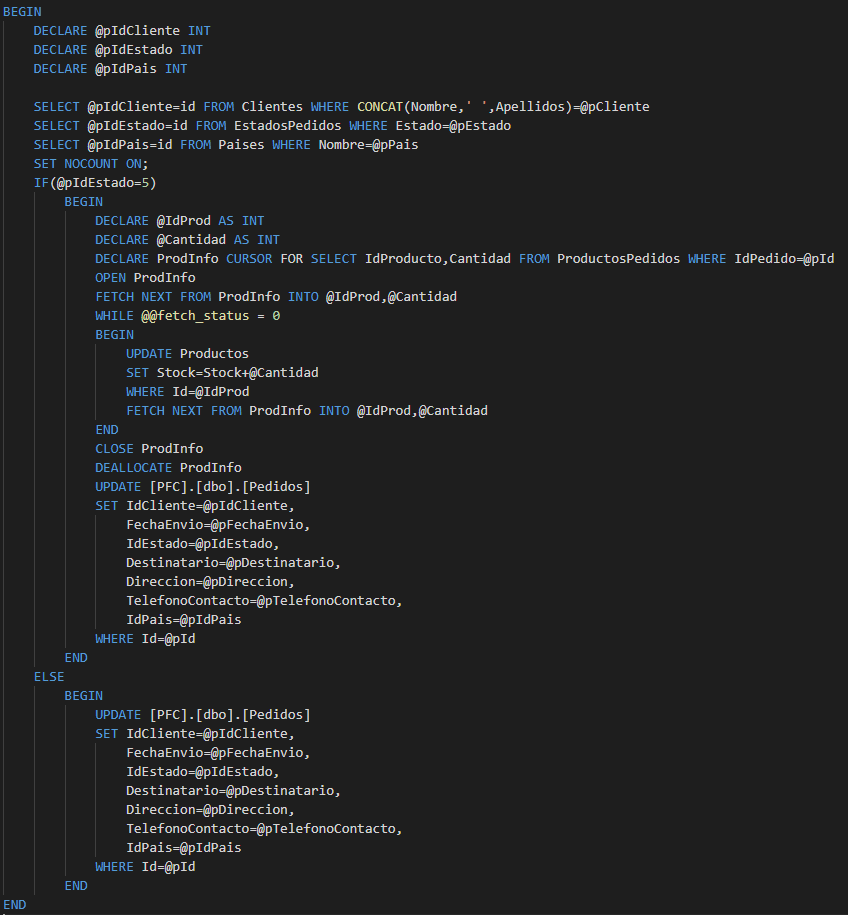
* En la aplicación se recurre a procedimientos almacenados para reducir el número de consultas. Se explota de la siguiente forma:



Se obtienen en el procedimiento los valores Id en función del String recogido en en la clase Controlador.

En este caso se evita ejecutar tres consultas para obtener la id.

* El procedimiento almacenado más destacable es el utilizado para editar pedidos. En este caso se explotan al completo los beneficios de los procedimientos usando cursores.



* A fin de no perder trazabilidad, en la base de datos nunca se eliminará un registro, siempre quedará marcado como no disponible. Se sustituye así la operación DELETE por UPDATE en estos casos.

BIBLIOGRAFÍA

* [Paper Sistema Informático](https://www.researchgate.net/publication/271588444_What_is_an_Information_System/link/54cd5d350cf298d6565d5877/download)
* [Script Tabla de países](https://gist.github.com/abroadbent/6233480)
* Configuración SQLServer JDBC:
  + [Conexión SQLServer JDBC](https://docs.microsoft.com/es-es/sql/connect/jdbc/step-3-proof-of-concept-connecting-to-sql-using-java?view=sql-server-ver15)
  + [Propiedades de conexión](https://docs.microsoft.com/en-us/previous-versions/sql/legacy/ms378988(v=sql.90)?redirectedfrom=MSDN)
  + [URL de conexión](https://www.codejava.net/java-se/jdbc/connect-to-microsoft-sql-server-via-jdbc)
* [Beneficios de los Stored Procedures](https://blog.titansoft.com.sg/2017/06/11/benefit-sp-over-inline-query/)
* [Ejecutar Stored Procedures en JDBC](http://www-mips.unice.fr/Doc/Java/Tutorial/jdbc/basics/sql.html#:~:text=Calling a Stored Procedure from,with an open Connection object.)
* [Vídeo: Tutorial Stored Procedures](https://www.youtube.com/watch?v=LbW4-fqLBjE&t=652s)
* Hashing & Salting:
  + [Vídeo: ¿Por qué usar hash y salt?](https://www.youtube.com/watch?v=I2pblc9FdmI)
  + [Explicación salt](https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/)
* [Alerts JavaFX](https://code.makery.ch/blog/javafx-dialogs-official/)

DEPENDENCIAS

* [JavaFX](https://mvnrepository.com/artifact/org.openjfx/javafx-controls/13)
* [Jfoenix](https://mvnrepository.com/artifact/com.jfoenix/jfoenix/9.0.10)
* [Bootstrap](https://mvnrepository.com/artifact/org.kordamp.bootstrapfx/bootstrapfx-core/0.3.0)
* [JDBC SQLServer](https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc/9.2.1.jre8)
* [IText 7](https://mvnrepository.com/artifact/com.itextpdf/itext7-core/7.0.4)

CONCLUSIONES

Esta aplicación es un proyecto que se ha ido desarrollando tras varios meses. Como es natural, el proyecto ha ido evolucionando y mejorando a medida que se avanzaba.

Durante estos meses de trabajo he aprendido muchas cosas nuevas y se han visualizado formas de proceder mejores que las ya implementadas.

Está claro que de rehacerse el proyecto desde cero, la estructura sería mucho mejor y habría más margen de maniobra para hacer un código más limpio que el actual.

Sin embargo, creo que se puede considerar un proyecto bastante completo y con algunas características técnicas muy interesantes.

Se han explotado las características de JavaFX y se le ha sacado mucho provecho al motor de SQLServer. Creo que la elección de tecnologías ha sido muy buena.

Independientemente de la valoración que se le pueda dar desde el punto de vista más objetivo y técnico creo que este proyecto ha servido para, además de crear una aplicación funcional, hacerme salir de la zona de confort y buscar nuevos problemas y obligarme a enfrentarme a ellos, pensar e investigar. En mi opinión, es ahí cuando mejoras como desarrollador

LÍNEAS FUTURAS DE INVESTIGACIÓN

Anteriormente en este documento se ha planteado la necesidad de continuar el desarrollo de la aplicación para hacerla aún más completa. En este apartado se irá desglosando cada punto a mejorar de forma individual.

* **Creación de un backoffice:** El hecho de que el la base de datos del programa tenga 15 tablas implica un tiempo de desarrollo mayor, como es obvio.

En esta primera versión se ha lanzado la aplicación sin CRUD completo para muchas de estas tablas.

Así pues, uno de los puntos a focalizar para las próxima versión consistirá en añadir un backoffice funcional en su totalidad para toda la base de datos.

* **Creación de un port para móvil:** Durante las próximas versiones se estudiará la posibilidad de realizar un port de la aplicación para Android e IOS utilizando, como ya se ha mencionado anteriormente, las herramientas de Gluon.
* **Creación de una aplicación para clientes:** Una herramienta muy recomendable y que podría aumentar la facturación sería la posibilidad de ofrecer una aplicación móvil para que los clientes puedan comprar desde sus teléfonos.
* **Asegurar credenciales:** Un problema muy destacable de la aplicación y que se debería de corregir cuanto antes es la seguridad de las credenciales de base de datos.

Tal y como está ahora, cualquier persona con acceso al código fuente podría obtener acceso a la base de datos. Se han de buscar herramientas que solucionen este problema.

Aún así ya pesar de la brecha de seguridad, durante el desarrollo se consideró como entorno poco peligroso el de cualquier potencial cliente, con lo cual no se lanzó la versión con las mejoras de seguridad ya implementadas.

* **Creación de tests unitarios:** De cara a versiones más estables de la aplicación se ha de priorizar la creación de tests unitarios que reflejen la robustez del programa.

1. [Comparativa entre Java Swing y JavaFX](https://www.educba.com/java-swing-vs-java-fx/) [↑](#footnote-ref-1)
2. [Comparativa entre SQL y TSQL](https://www.geeksforgeeks.org/difference-between-sql-and-t-sql/) [↑](#footnote-ref-2)
3. Blancarte, O.J. 2016: Introducción a los patrones de diseño: Un enfoque práctico, Autoeditado\*, pp. (68/69), disponible en: [Reactiveprogramming.io](https://reactiveprogramming.io/books/design-patterns/es), [Amazon](https://www.amazon.es/Introducción-los-patrones-diseño-práctico/dp/1539619214) [↑](#footnote-ref-3)
4. Son tablas que contienen los resultados de una función de cifrado (hash) para cada cadena de caracteres. [↑](#footnote-ref-4)
5. [DOG Nº109 Miércoles 8 de junio de 2011. Anexo VI. Tabla 1,Fila 16](https://www.xunta.gal/dog/Publicados/2011/20110608/AnuncioC3F1-250511-2127_es.html) [↑](#footnote-ref-5)