

Proyecto final Desarrollo de Aplicaciones Multiplataformas

Índice

1 Enunciado.....	4
2 Alcance.....	6
3 Requerimientos detallados.....	7
3.1 Compilación del código fuente.....	7
3.2 Ejecución del formato Jar.....	8
4 Viabilidad.....	9
5 Proyecto.....	10
5.1 Fases de proyecto.....	10
5.2 Plan de proyecto.....	11
6 Diseño de la aplicación.....	12
6.1 Estructura de directorios.....	12
6.2 Patrones utilizados.....	12
6.3 Vistas.....	14
6.3.1 Login.....	14
6.3.2 Base.....	14
6.3.3 Usuario.....	15
6.3.4 Productos.....	16
6.3.5 Clientes.....	17
6.3.6 Proveedores.....	18
6.3.7 Facturas.....	19
6.3.8 Recibos.....	20
6.4 Clases.....	21
6.4.1 Daos.....	21
6.4.2 Entidades.....	21

6.4.3 Casos de uso.....	22
6.4.4 Utils.....	22
7 Idiomas.....	23
8 Base de datos.....	23
9 Seguridad.....	25
10 Fuentes de información.....	26

1 Enunciado

Esta prueba práctica servirá para realizar el proyecto de desarrollo de una aplicación informática, tanto por lo que respecta a la documentación que hay que generar en las fases del proyecto como por lo que se refiere a la implementación de la aplicación: los programas, la base de datos, etc.

Supongamos que eres el jefe de proyecto, diseñador y programador del desarrollo de aplicaciones de una empresa de servicios informáticos.

Una empresa, cuyo sector debes elegir, os ha encargado desarrollar una aplicación. Debes inventar qué aplicación y funcionalidades os ha pedido, pero, como mínimo, debe tener las características y las funcionalidades del ejemplo que se describe a continuación:

Base de datos

La base de datos debe tener, al menos, 6 tablas con 3 relaciones. Por ejemplo, una aplicación sencilla para captar pedidos de cliente de una pequeña tienda en línea podría tener las siguientes tablas mínimas: clientes, productos, pedidos, línea de pedido, facturas y línea de factura.

Autenticación

Antes de mostrar cualquier página o sector, nuestra aplicación debe pedir al usuario que se autentique mediante un formulario de usuario y contraseña que se validará contra la base de datos (tabla adicional de Usuarios).

Formularios

Al menos, un formulario. En el ejemplo, tendríamos un formulario para que el cliente actualice sus datos.

Páginas

Al menos, 5 páginas o sectores, además de la de autenticación. En el ejemplo:

1. Una página o sector inicial con la lista de productos, con botón para añadir en un carrito de la compra, es decir, un pedido (si no está creado para la sesión) y una línea de pedido con el producto seleccionado.

2. Una página o sector para consultar y actualizar el pedido actual y las líneas de pedido. En el momento de aceptar, se creará la factura.
3. Una página o sector para el seguimiento de los pedidos (creado, en transporte, entregado).
4. Una página o sector para consultar las facturas.
5. Una página o sector para que el cliente actualice sus datos.

Debemos disponer de un menú que permita acceder a cada una de las páginas de nuestra aplicación. Nuestra aplicación puede estar programada en cualquier lenguaje como, por ejemplo, en Java para la lógica de negocio y MySQL para la gestión de la base de datos.

Si se utilizan imágenes asociadas a alguna entidad, en nuestro ejemplo serían las fotos de los productos. Estas se deben guardar en el sistema operativo; es decir, no se guardarán imágenes en la base de datos, sino que se creará un campo en la tabla para decir dónde está guardada y qué fichero es.

En los requerimientos anteriores no se ha hablado de, por ejemplo, un formulario para actualizar los productos o para darse de alta como cliente (solo para actualizarlo). No importa; digamos que la parte de administración del contenido no la vamos a desarrollar todavía, pues sería una segunda fase del proyecto. En tal situación, la creación de productos y clientes, la modificación de productos y cualquier actualización de tablas no realizada mediante nuestra aplicación se efectuarán directamente sobre la base de datos, por ejemplo, con MySQL WorkBench.

Presentación

Al finalizar el proyecto debemos presentar los siguientes archivos:

- a) Un documento que contenga el análisis de nuestro proyecto. Como mínimo este documento debe contener los siguientes apartados:

- | | |
|------------------------------|----------------------------|
| 1. Alcance | 5. Viabilidad técnica |
| 2. Requerimientos detallados | 6. Viabilidad económica |
| 3. Viabilidad | 7. Plan del proyecto |
| 4. Fases del proyecto | 8. Diseño de la aplicación |

9. Casos de uso

12. Base de datos

10. Mockups (vistas)

13. Clases

11. Idiomas

14. Seguridad

- b) La carpeta con los ficheros del proyecto. Tanto los archivos de programación como el resto de recursos gráficos o documentos.
- c) La carpeta con las imágenes asociadas a las entidades, como las fotos de cada producto.
- d) Un archivo con la base de datos (por ejemplo, una exportación de PHPMyAdmin o de MySQL Workbench). Debería ser un .sql o algún fichero que se pueda importar fácilmente para que el evaluador pueda generar la base de datos.

2 Alcance

Esta aplicación tiene por objeto la práctica de programación para demostrar los conocimientos informáticos adquiridos a lo largo del curso de Desarrollo de Aplicaciones Multiplataforma.

Se ha planteado como una aplicación a utilizar para gestionar el inventario de un establecimiento comercial permitiendo:

- El registro de productos
- El registro de clientes y proveedores
- El registro de facturas pagadas para la compra de productos
- El registro e impresión de facturas para la venta de productos
- El acceso restringido a través de la validación de usuarios con contraseña

La aplicación está completamente desarrollada en Java, incorpora las fuentes que utiliza y cuenta con una base de datos embebida (Apache Derby). Por lo que es una aplicación completamente multiplataforma. Aunque su uso está principalmente orientado para escritorios de trabajo y NO para dispositivos táctiles.

3 Requerimientos detallados

3.1 Compilación del código fuente

Para la compilación de la aplicación informática a partir del código fuente es necesario tener instalado:

- **Oracle Java JDK 8¹**: el lenguaje de programación utilizado
- **Apache Maven 3.6.3²** (opcional, recomendado), un gestor de paquetes y administrador de proyecto para Java que descarga las dependencias indicadas en el fichero "pom.xml".

En caso de no utilizar Maven para descargar las dependencias, estas deberán ser agregadas manualmente al proyecto:

- **Hibernate 5.4.3³**: Hibernate es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación.
- **Apache Derby 10.14.2.0⁴**: Apache Derby es un sistema gestor de base de datos relacional escrito en Java que puede ser empotrado en aplicaciones Java y utilizado para procesos de transacciones online. NOTA
- **Apache log4j 2⁵**: es una biblioteca open source desarrollada en Java, permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución. Log4j permite filtrar los mensajes en función de su importancia. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos.

1 Java JDK 8: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

2 Apache Maven 3.6.3: <https://maven.apache.org/>

3 Hibernate 5.4.3: <https://hibernate.org/orm/releases/5.4/>

4 Apache Derby 10.14.2.0: https://db.apache.org/derby/derby_downloads.html Derby dispone de un modo embebido que almacena la base de datos en ficheros dentro de un directorio, aparte, también dispone de un modo cliente/servidor. En esta aplicación se utiliza el driver JDBC embebido para simplificar el uso y distribución del proyecto.

5 Apache log4j 2.12.0: <https://logging.apache.org/log4j/2.x/download.html>

- **Apache PDFBox 2.0.18⁶**: librería open source desarrollada en Java, permite trabajar con documentos PDF bien sea creando nuevos documentos, manipulando documentos existentes o permitir la extracción de información de los mismos.
- **Apache Commons-lang 3.9⁷**: librería open source desarrollada en Java, añade nuevos métodos para la "java.lang" API.
- **Apache Commons-io 2.6⁸**: librería open source desarrollada en Java, añade nuevos métodos para las funcionalidades IO.
- **JavaFX 8**: framework para el desarrollo de la interfaces gráficas que se encuentra incorporado en el Oracle Java JDK 8, pero que se retiró del mismo a partir de las versiones 9+ ni en los OpenJDK.

NOTA: La aplicación ha sido desarrollada y testada con Oracle Java JDK 8. Se ha proporcionado compatibilidad con Oracle Java JDK 11 y OpenJDK 11, se ignoran las versiones Java no LTS, añadiendo las dependencias necesarias de JavaFX 11⁹ al "pom.xml" del proyecto (aunque no está completamente probada). En caso de utilizar las versiones 11 y de no utilizar Maven, tendría que incluirse la librería manualmente.

3.2 Ejecución del formato Jar

El empaquetado Jar ha sido probado satisfactoriamente con el Java JRE¹⁰ versión 8 Update 241 de 14 de Enero de 2020 tanto para Windows 10 como en Linux (distribución Ubuntu 18.04).

NOTA: la base de datos en Apache Derby, contenida en el directorio "gestor_database", debe estar localizada en el mismo directorio que el archivo "Jar" de la aplicación. En caso de no estar presente se generará una base de datos sin contenido.

6 Apache PDFBox 2.0.18: <https://pdfbox.apache.org/download.cgi>

7 Apache Commons-lang 3.9: https://commons.apache.org/proper/commons-lang/download_lang.cgi

8 Apache Commons-io 2.6: https://commons.apache.org/proper/commons-io/download_io.cgi

9 JavaFX 11: <https://gluonhq.com/products/javafx/>

10 Java JRE: <https://www.java.com/es/download/>

4 Viabilidad

Estamos utilizando el supuesto de que una empresa cliente “Beds” decide establecer un local comercial para la venta de productos de pertenecientes a dormitorios y consulta a una empresa externa para el desarrollo de una aplicación que les ayude a gestionar los clientes, proveedores, productos, facturas y llevar un registro del material comprado y del inventario.

La empresa desarrolladora de aplicaciones informáticas, al recibir el cliente, realiza una serie de preguntas destinadas a conocer las necesidades del cliente y reconocer qué tipo de solución se debe aplicar. También para determinar la viabilidad técnica, económica y operativa.

Viabilidad técnica

El analista debe averiguar si es posible actualizar o incrementar los recursos técnicos actuales de tal manera que satisfagan los requerimientos bajo consideración.

Viabilidad económica

Los recursos a considerar son:

- El tiempo requerido a invertir.
- El costo de realizar un estudio de sistemas completo.
- El costo del tiempo de los empleados de la empresa.
- El costo estimado del hardware y el costo estimado del software comercial o del desarrollo del software.

Viabilidad operativa

Depende de los recursos humanos disponibles para el proyecto e implica determinar si el sistema funcionará y será utilizado una vez que se instale.

Si los usuarios están contentos con el sistema actual, no tienen problemas con su manejo y por lo general no están involucrados en la solicitud de un nuevo sistema.

Por el contrario, si los usuarios mismos han expresado la necesidad de un sistema que funcione la mayor parte del tiempo, de una manera más eficiente y accesible, hay más probabilidades de que a la larga el sistema sea más utilizado.

5 Proyecto

5.1 Fases de proyecto

Para realizar el proyecto informático se procederá a las siguientes fases.

Iniciación

Determinar los objetivos a cumplir de la aplicación informática. Llevar un registro en base de datos de los productos, clientes proveedores, facturas y recibos.

Determinar el presupuesto económico que se calculará a partir de los medios necesarios y del número de empleados que van a participar en la planificación, desarrollo, prueba y redacción de la documentación de la aplicación requerida.

Periódicamente se establecerán puntos de control para comprobar que se están cumpliendo los objetivos planteados.

Planificación

Definición de los eventos que el proyecto debe realizar, cuando y en qué orden.

- El registro de productos
- El registro de clientes y proveedores
- El registro de facturas pagadas para la compra de productos
- El registro e impresión de facturas para la venta de productos

El acceso restringido a través de la validación de usuarios con contraseña.

Enumeración de las etapas (tareas) en las que se irá desarrollando el proyecto y ordenación a lo largo del tiempo.

Ejecución

Programar la aplicación planificada siguiendo el orden previsto e intentando cumplir los tiempos marcados.

Control

Se irá realizando el control de la aplicación mientras se está desarrollando la aplicación.

Finalización

Una vez terminada la programación y el testeo de la aplicación. Se realizará un repaso reflexionando sobre los hechos conseguidos, la inversión realizada y los desvíos producidos.

5.2 Plan de proyecto

El plan de proyecto es definido a partir de las fases de proyecto explicadas anteriormente.

- **Declaración del plan:** desarrollo de aplicación informática para local comercial que necesita llevar un registro de sus clientes y proveedores y de las transacciones realizadas con sus productos.
- **Especificaciones de diseño:** se realizará una aplicación sencilla, intuitiva y práctica para sus usuarios con una base de datos embebida para favorecer la portabilidad de la misma. Posteriormente los desarrolladores impartirán una charla a los usuarios para que conozcan el manejo de la aplicación.
- **Procedimiento:** se diseñará la base de datos en base a las necesidades del cliente, posteriormente se planifica los modelos a redactar para trabajar con la misma. Mientras tanto un diseñador se encargará de redactar las vistas mostrándolas al cliente para que este de su conformidad al diseño optado. Posteriormente la actividad de la aplicación se separará en distintos casos de uso para realizar su programación por separado. Mientras tanto se irá controlando que la aplicación cumple con los objetivos marcados.
- **Ejecución de las pruebas:** tras impartir a los usuarios una charla sobre el manejo de la aplicación, se supervisará unos días a los usuario para comprobar que al aprendizaje correctamente su manejo y que la aplicación cumple con los objetivos marcados.

6 Diseño de la aplicación

6.1 Estructura de directorios

.	Raíz del proyecto
├─ gestor_database	Base de datos de Apache Derby
├─ src	
└─ main	
└─ java	Código fuente en Java
└─ daos	Clases que realizan las conexiones a la BD
└─ entities	Clases que definen las entidades/tablas de la BD
└─ start	Clase Main
└─ stories	Casos de uso
└─ utils	Métodos utilitarios y otras clases
└─ resources	Ficheros estáticos
└─ fonts	Fuentes de texto utilizadas por la aplicación
└─ images	Imágenes
└─ META-INF	Aloja el persistence.xml para la conexión a la BD
└─ properties	Ficheros de configuración
└─ stylesheets	Ficheros css utilizados por las vistas
└─ views	Vistas fxml de javafx
└─ templates	Plantillas para las vistas javafx u otros documentos
└─ sql	Scripts SQL
└─ pom.xml	Indica las las dependencias y operaciones a realizar a Maven

6.2 Patrones utilizados

Modelo – Vista – Controlador (MVC)

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

La aplicación se ha desarrollado desde el principio con el patrón MVC en mente. Desde el principio el “Main” lo único que hace es cargar un fichero fxml (Vista) y este accede a un caso de uso (Controlador), el cual recoge las interacciones del usuario con las vistas y se comunica con la base de datos a través de los Daos y las entidades (los Modelos).

Singleton

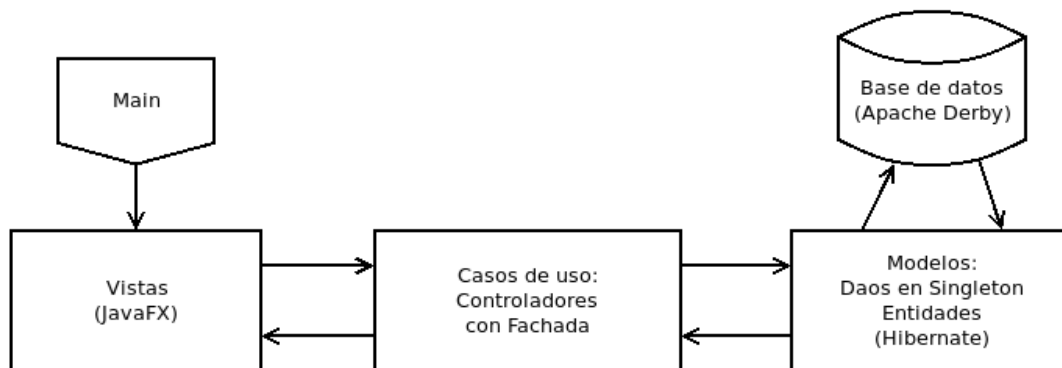
El singleton o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

Este patrón se ha utilizado para los Daos.

Fachadas

Fachada (Facade) es un tipo de patrón de diseño estructural. Viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos.

Se ha optado por el uso de fachadas en los casos de uso para aligerar código del controlador y separar las funcionalidades. De la manera planteada en cada caso de uso el Controlador tan solo interactúa con las vistas mientras que la fachada se encarga de la lógica del negocio y de interactuar con los modelos.



6.3 Vistas

Por cada página relevante en la aplicación se ha creado una vista "FXML" de JavaFX y su caso de uso correspondiente para manejarla. Estas vistas son:

6.3.1 Login

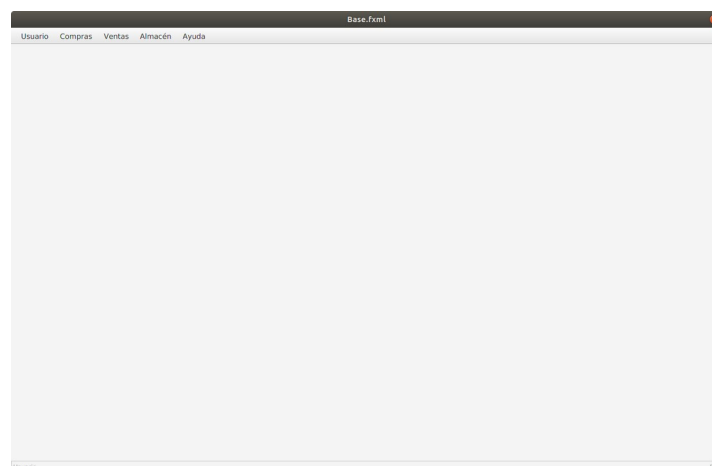
Solicita a la persona que va a utilizar la aplicación que se identifique con un usuario y contraseña que estará alojado en base de datos.

Una vez haya iniciado sesión, se marca al usuario en base de datos como que está en sesión y se cierra la ventana para abrir la ventana de la aplicación.



6.3.2 Base

Esta vista y caso de uso actúan de plantilla para el resto de la aplicación y proveen de una ventana con un menú con los accesos a las diferentes páginas y un contenedor para alojar las mismas.



6.3.3 Usuario

Permite al usuario, que ha iniciado sesión, la modificación de sus datos o el cambio de contraseña.

Usuario

NIF/CIF/NIE*:

Nombre*:

Apellidos*:

Dirección*: Calle Nº
 Piso Puerta Código Postal
 Municipio Provincia

Contacto*: Móvil Teléfono Fijo
 Correo electrónico

Cuenta Bancaria*:

Contraseña antigua:

Contraseña nueva:

Repetir contraseña nueva:

Modificar

Modificar

Los distintos campos del formulario son validados previamente a su modificación en base de datos para evitar la entrada de campos erróneos y se marcan con un * los obligatorios. De esta manera:

- **Nif/Nie, Nombre, Apellidos, Dirección, Email, Móvil y Cuenta Bancaria** son obligatorios.
- El **Nif/Nie** se fuerza a que sean 9 caracteres (sin guiones) y se verifica que la letra de control sea la correspondiente.
- El **número de la dirección** y el de planta tendrá que ser un número entero.
- El **correo electrónico** será verificado para comprobar que sea válido.
- La **cuenta bancaria** tendrá que ser un IBAN español válido.
- La modificación de la **contraseña** requiere la contraseña actual y añadir la nueva contraseña dos veces.

6.3.4 Productos

Permite registrar nuevos productos a la base de datos para que puedan ser añadido a recibos de compras y facturas de ventas, así como buscar productos existentes para ver su información, eliminar el producto o modificarlo. NOTA: los productos se registran en base de datos como que no existen unidades en almacén (0), para poder añadir unidades tendrá que registrarse a través de recibos, una vez hayan unidades de un productos en almacén podrá generarse facturas de venta.

The screenshot shows a web application window titled 'Productos.fxml'. It contains a form on the left for adding products and a table on the right for listing them.

Form Fields:

- Código barras*:
- Marca*:
- Nombre*:
- Categoría*:
- Descripción:
- Precio de venta*:

Buttons:

- Limpiar (Clear)
- Modificar (Modify)
- Registrar (Register)

Table:

Id	Nombre	Marca	Categoría	Uds
Tabla sin contenido				

Additional Buttons:

- Limpiar (Clear)
- Editar (Edit)
- Eliminar (Delete)

- Son obligatorios el **código de barras**, la **marca del producto**, el **nombre comercial** y el **precio de venta**.
- El **precio de venta** solo admite números decimales.
- Para poder realizar la **búsqueda** tendrá que escogerse una categoría y/o una marca.
- Una vez elegido un producto buscado se activarán los botones "Editar" y "Eliminar".
- Al presionar "Editar" se lanzan los datos al formulario de productos, se desactiva el botón "registrar" y se activa el botón "modificar"

6.3.5 Clientes

Permite dar alta a los clientes para poder ser indicados en las facturas, buscarlos para poder consultar sus datos, modificarlos o eliminar el cliente.

The screenshot shows a web application window titled 'Clientes.fxml'. It contains a form on the left for adding a new client and a table on the right for listing clients.

Formulario de Alta de Clientes:

- NIF/CIF/NIE*:** Campo de texto obligatorio.
- Nombre*:** Campo de texto obligatorio.
- Apellidos*:** Campo de texto obligatorio.
- Dirección:** Campos para Calle, Nº, Piso, Puerta, and Código Postal.
- Municipio:** Campo de texto.
- Provincia:** Campo de texto.
- Contacto:** Campos para Móvil, Teléfono Fijo, and Correo electrónico.
- Cuenta Bancaria:** Campo de texto.

Botones de Acción: Limpiar, Modificar, Dar Alta.

Tabla de Clientes:

Apellidos	Nombre	CP
Tabla sin contenido		

Botones de Acción de la Tabla: Limpiar, Editar, Borrar.

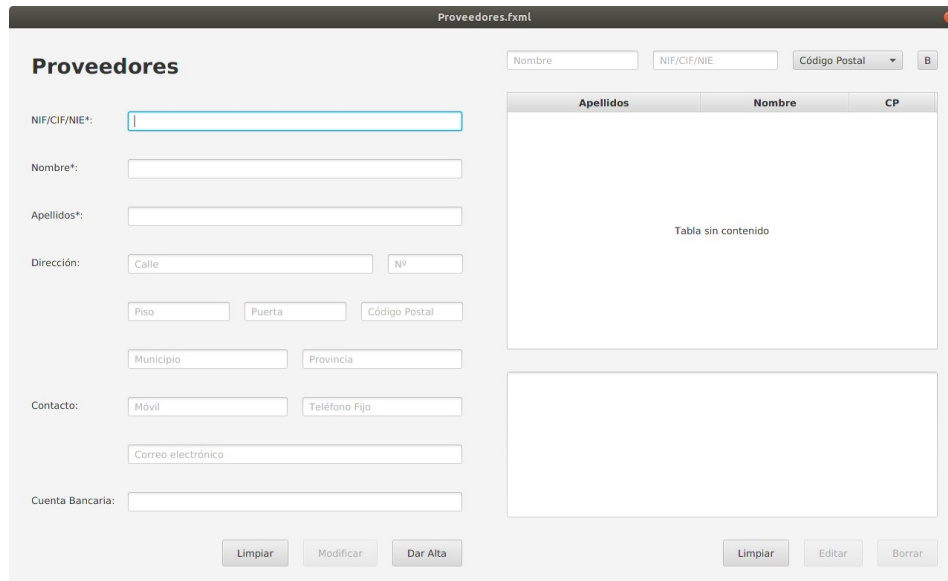
En el formulario de alta de clientes:

- Es obligatorio solo el **Nif/Cif/Nie** y el **nombre**.
- El **Nif/Cif/Nie** será validado para comprobar si es un número de identificación válido.
- El **número de la dirección**, y el **piso** se verificará que sean número enteros.
- En el **código postal** se comprobará si son 5 dígitos.
- El **correo electrónico** se comprobará si es un patrón válido de emails.
- La **cuenta bancaria** deberá ser un IBAN español (empezar con ES y seguir con 22 dígitos).

Para realizar la búsqueda deberá indicarse el nombre o el Número de Identificación o el Código Postal. Al seleccionar un cliente buscado se mostrará su información y se activarán los botones "editar" y "borrar". Al presionar "Editar" se lanzan los datos al formulario de productos, se desactiva el botón "registrar" y se activa el botón "modificar".

6.3.6 Proveedores

Permite dar alta a los proveedores para poder ser indicados en las recibos, buscarlos para poder consultar sus datos, modificarlos o eliminar el proveedor.



En el formulario de alta de clientes:

- Es obligatorio solo el **Nif/Cif/Nie** y el **nombre**.
- El **Nif/Cif/Nie** será validado para comprobar si es un número de identificación válido.
- El **número de la dirección**, y el **piso** se verificará que sean número enteros.
- En el **código postal** se comprobará si son 5 dígitos.
- El **correo electrónico** se comprobará si es un patrón válido de emails.
- La **cuenta bancaria** deberá ser un IBAN español (empezar con ES y seguir con 22 dígitos).

Para realizar la búsqueda deberá indicarse el nombre o el Número de Identificación o el Código Postal. Al seleccionar un proveedor buscado se mostrará su información y se activarán los botones “editar” y “borrar”. Al presionar “Editar” se lanzan los datos al formulario de productos, se desactiva el botón “registrar” y se activa el botón “modificar”.

6.3.7 Facturas

Permite emitir facturas e imprimirlas (a PDF). También se pueden buscar facturas antiguas para consultarlas o volver a imprimirlas.

The screenshot shows a web application window titled 'Facturas.fxml'. The interface is divided into two main sections. The left section contains a form for creating a new invoice with the following fields: 'Identificación nº:' (text input), 'Cliente:' (dropdown menu), 'Productos:' (dropdown menu with a '+' button), 'Base imponible:' (text input), 'Impuestos:' (text input with '7.00' pre-filled), and 'Total:' (text input). Below these fields are three buttons: 'Limpiar', 'Emitir', and 'Imprimir'. The right section contains two tables. The top table has columns 'Nº Factura', 'Fecha', 'Cliente', and 'Importe', and its content area displays 'Tabla sin contenido'. The bottom table has columns 'ID', 'Nombre', 'Marca', 'Uds.', and 'Precio', and its content area also displays 'Tabla sin contenido'. At the bottom right of the right section are two more buttons: 'Limpiar' and 'Imprimir'.

En el formulario de emisión de facturas:

- El **número de factura** se genera automáticamente (... , F00011, F00012, F00013, ...).
- El **cliente** se seleccionará entre los existentes en la base de datos.
- Es obligatorio seleccionar un **producto** ya existente en base de datos e indicar la cantidad a vender (se verificará si existe tal cantidad en almacén).
- Los productos añadidos se presentarán en la **tabla** y se calculará la **base imponible**, **impuestos** y **total** a facturar de los productos presentes en ella. También se activará el botón "Emitir".
- Al **emitir** una factura se activa el botón "imprimir" y se desactiva "emitir". También se restan los productos facturados del almacén.

Se podrán buscar facturas antiguas por su número o por cliente (las que se hayan emitido con cliente). Al seleccionar una factura buscada se mostrarán sus datos y se podrá imprimir de nuevo.

6.3.8 Recibos

Permite registrar facturas pagadas a proveedores, para la compra de material del almacén, como recibos. También buscar recibos antiguos para consultarlos.

En el formulario de emisión de recibos:

- Es obligatorio indicar el **número del recibo**, el **proveedor** y los **productos** con sus respectivas **cantidades y precios de compra**.
- El **proveedor** y los **productos** se escogerán entre los existentes en base de datos.
- Se verificará que la cantidad es un número entero y el precio es un número decimal.
- Los productos añadidos se presentarán en la **tabla** y se calculará la **base imponible, impuestos y total** a facturar de los productos presentes en ella. También se activará el botón "Emitir".

Se podrán buscar recibos antiguos por su número o por proveedor. Al seleccionar un recibo buscado se mostrarán sus datos.

6.4 Clases

6.4.1 Daos

Los Daos (Data Access Objects) suministran una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, en este caso, la base de datos almacenada con Apache Derby.

Se ha creado un Dao común, "CommonDao.java", con los métodos de conexión a la base de datos y los básicos CRUD (Create, Read, Update, Delete) para interactuar con la base de datos.

También se ha creado un Dao por cada entidad (o tabla). Cada uno de estos Dao trabaja solo con su tabla correspondiente y extiende el "CommonDao" para disponer de sus métodos, aparte de incorporar sus métodos propios.

6.4.2 Entidades

Una entidad de persistencia (entity) es una clase de Java ligera, cuyo estado es persistido de manera asociada a una tabla en una base de datos relacional. Las instancias de estas entidades corresponden a un registro (conjunto de datos representados en una fila) en la tabla. Normalmente las entidades están relacionadas a otras entidades, y estas relaciones son expresadas a través de meta datos objeto/relacional.

Se ha creado una clase común abstracta "CommonEntity.java" que fuerza a las entidades que van a heredar la presencia de algunos métodos (get y set de Id) así como sobrescribir (@Override) el comportamiento de otros métodos que utilizarán.

Las demás entidades, cada una de ellas:

- Extienden el "CommonEntity"
- Definen una tabla en la base de datos
- Con sus parámetros definen las columnas de la tabla y sus propiedades
- Cuenta con getters y setters de cada parámetros
- Algunas entidades añaden métodos adicionales para la devolución formateada.

6.4.3 Casos de uso

Por cada vista relevante se ha creado un caso de uso (stories). Cada caso de uso se compone de un controlador y una fachada para simplificar el controlador y separar funcionalidades. De tal manera que el controlador se encarga de interactuar con la vista y la fachada de la lógica de negocio.

NOTA: los casos "base" y "login" son cortos por lo que se ha optado por no utilizar fachadas.

Controlador

El controlador se encarga de:

- Parametrizar los campos de la vista fxml en los que necesita actuar.
- Implementar el "Initializable" de JavaFX y sobrescribir el método "initialize()" para la ejecución de instrucción nada más abrir la página.
- Implementar la fachada en el constructor.
- Definir los métodos llamados a través de la vista. Por ejemplo: la vista "Cliente" cuenta con el botón "Alta Cliente" que ejecuta el método "buscarCliente()" al hacer click.
- Recoger y "settear" los datos de la vista.

Fachada

Se ha optado por utilizar fachadas para aligerar el Controlador y separar funciones, de tal manera que el controlador se limita a interactuar con la vista mientras que la fachada provee métodos que interactúan con los modelos u otros métodos de tratamiento de datos (lógica de negocios).

6.4.4 Utils

En el directorio "utils" se han añadido un conjunto de clases y métodos que se utilizan a lo largo de toda la aplicación pero que no caen dentro de las categorías antes mencionadas:

- **Alerta:** permite lanzar un modal tipo "alert" con una sola instrucción.
- **ProductoEnFactura:** un objeto que se creará para introducir productos temporalmente en las tablas de las vistas de Facturas y Recibos.

- **Validar:** conjunto de métodos “estáticos” y “booleanos” para comprobar que el “string” de entrada cumple unas condiciones (si es entero, decimal, correo, etc).
- **ValidarNifCif:** clase estática para validar si el string recibido es un Nif/Cif/Nie válido.
- **Ventana:** permite lanzar una ventana de JavaFX abriendo la vista indicada cargando la plantilla o no.

7 Idiomas

La aplicación ha sido desarrollada en español y sin soporte multi-idioma.

8 Base de datos

Al ejecutar la aplicación la base de datos, en caso de no existir, es generada automáticamente por Hibernate a partir de las entidades definidas en “resources/META-INF/persistence.xml” según las especificaciones JPA. En el “persistence” además también se indica que utilice Apache Derby como gestor de base de datos a partir del paquete instalado por Maven (o añadido manualmente en el IDE).

La razón por la que se ha utilizado “JPA, Hibernate” en lugar “JDBC” son que permiten un desarrollo más rápido y nos abstraen de especificaciones propias de los diferentes sistemas de gestión de bases de datos ya que basta con indicar el que se utilice en el “persistence”.

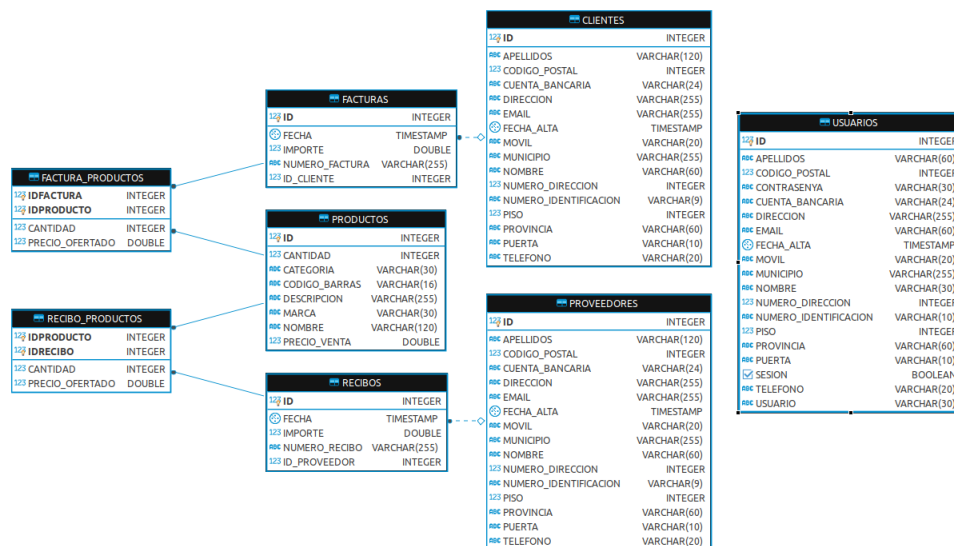


Diagrama de la base de datos en Apache Derby

La base de datos se compone por las tablas:

- **Usuario:** con los datos del empleado que utiliza la aplicación además del nombre de usuario, contraseña y un boolean de sesión para marcar que es quien está usando la aplicación.
- **Productos:** producto y cantidad que se tiene en almacén.
- **Clientes:** los compradores que se incorporarán a las facturas.
- **Proveedores:** los suministradores de productos y que se incorporarán a los recibos.
- **Facturas:** los documentos que registran la venta de productos a los clientes.
- **FacturaProductos** (entidad asociativa): lista de productos que se añaden a la factura indicando la cantidad vendida de productos y el precio de venta de la fecha a la que se realizó la transacción.
- **Recibos:** los documentos que registran la compra de productos a los proveedores.
- **ReciboProductos** (entidad asociativa): lista de productos que se añaden al recibo indicando la cantidad comprada de productos y el precio de compra de la fecha a la que se realizó la transacción.

Las relaciones que hay entre entidades son:

- **Clientes – Facturas** (una a muchas): un cliente puede estar en muchas facturas, pero una factura solo puede estar dirigida a un cliente.
- **Proveedores – Recibos** (una a muchas): un proveedor puede estar en muchos recibos, pero un recibo solo puede haber sido expedido por un proveedor.
- **Facturas – Productos** (asociativa): una factura puede contener muchos productos y un producto puede estar en muchas facturas PERO se pueden haber vendido distintas cantidades de productos en las distintas facturas y el precio de venta de los productos varía con el tiempo.
- **Recibos – Productos** (asociativa): un recibo puede contener muchos productos y un producto puede estar en muchos recibos PERO se pueden haber comprado distintas cantidades de productos en los distintos recibos y el precio de compra de los productos varía con el tiempo

Estas relaciones asociativas podrían parecer de tipo muchos-a-muchos en un principio, pero la necesidad de añadir propiedades adicionales a la relación hace necesario utilizar entidades asociativas con relaciones uno-a-muchos con los productos a las facturas o recibos. Ejemplo: la factura con id 5 tiene un producto con id 2 que se vendió a 3,99 €/ud, tres productos 5 a 18,99 €/ud y dos productos 6 a 5,99 €/ud.

IdFactura	IdProducto	Cantidades	PrecioOfertado
5	2	1	3.99
5	5	3	18.99
5	6	2	5.99
6	2	1	3.99
6	10	5	1.00

9 Seguridad

Se ha **restringido el acceso** a la aplicación al requerir a los usuarios que se autentiquen para poder empezar a usarla.

Los **datos que se almacenan en la base de datos** están guardados en binarios que solo son accesibles desde la propia aplicación o desde un cliente de un gestor de bases de datos compatible con Apache Derby.

De todas maneras serían accesibles para cualquier persona con los conocimientos adecuados y tenga acceso al escritorio de trabajo. También habría riesgo de pérdida ante una posible daño en el disco duro. Para una mayor seguridad de la base de datos sería conveniente el almacenamiento en servidores remotos que requiriesen autenticación para su acceso.

10 Fuentes de información

Según ISO 690:2010 que proporciona las directrices básicas para la preparación de referencias bibliográficas de materiales publicados.

- Java Tutorials - Java Learning Resources - HowToDoInJava, 2019. HowToDoInJava [online]
- JavaFX « Java, 2019. Java2s.com [online]
- Overview (JavaFX 13), 2019. Openjfx.io [online]
- Documentation - 5.4 - Hibernate ORM, 2019. Hibernate.org [online]
- PORTER, BRETT, 2019, Maven – Users Centre. Maven.apache.org [online]. 2019. [Accedido 30 Diciembre 2019]. Disponible en: <https://maven.apache.org/users/index.html>
- RODRÍGUEZ GONZÁLEZ, CÉSAR, 2019, cesar-rgon/simple-kf2-server-launcher. GitHub [online]. 2019. [Accedido 30 Diciembre 2019]. Disponible en: <https://github.com/cesar-rgon/simple-kf2-server-launcher>
- JANSSEN, THORBEN, 2016, Java 8 Support in Hibernate 5. 1. Thoughts on Java Library.
- JANSSEN, THORBEN, 2016, Native queries with Hibernate. 1. Thoughts on Java Library.
- Java Persistence API, 2019. En.wikipedia.org [online],