

**GESTIÓN DE ALMACENES DAM**

Una caricatura de una persona

Descripción generada automáticamente con confianza media

**Realizado por:**

Azahara Blanco Rodríguez

Daniel Rodríguez Fernández

Daniel Alejandro Carmona Rodríguez

Jeremy Ramos Segura

Jan Carlos Ramos Martínez

**04/05/2022**

|  |  |  |  |
| --- | --- | --- | --- |
| **Proyecto** | GDAM | | |
| **Entregable** | Especificación de Requisitos | | |
| **Autor** | GDAM GROUP S.L. | | |
| **Versión/Edición** | V1.1 | **Fecha Versión** | 27/04/2022 |
| **Aprobado por** |  | **Fecha Aprobación** | DD/MM/AAAA |
|  |  | **Nº Total de Páginas** | 20 |

|  |  |
| --- | --- |
| **PROJECT MANAGER** | |
| Azahara Blanco Rodríguez |

|  |  |
| --- | --- |
| **PROJECT OWNER AND PLANNING MANAGER** | |
| Daniel Rodríguez Fernández |

|  |  |
| --- | --- |
| **QUALITY ASSURANCE MANAGER** | |
| Daniel Alejandro Carmona Rodríguez |

|  |  |
| --- | --- |
| **DATA BASE ADMINISTRATOR MANAGER** | |
| Jeremy Ramos Segura |

|  |  |
| --- | --- |
| **UI DESIGN MANAGER** | |
| Jan Carlos Ramos Martínez |

Tabla de contenido

[Tabla de contenido 3](#_Toc104946788)

[1. Introducción 4](#_Toc104946789)

[2. Problema Propuesto: 5](#_Toc104946790)

[3. Requisitos 8](#_Toc104946791)

[3.1. Requisitos Funcionales 8](#_Toc104946792)

[3.2. Requisitos No Funcionales 9](#_Toc104946793)

[3.3. Requisitos de Información 11](#_Toc104946794)

[4. Diseño e Implementación 13](#_Toc104946795)

[4.1. Diagramas 13](#_Toc104946796)

[4.1.1. Diagrama de Clases 13](#_Toc104946797)

[4.1.2. Diagramas de Casos de Uso 14](#_Toc104946798)

[4.2. Casos de Uso 17](#_Toc104946799)

[4.3. Historias de Usuario 35](#_Toc104946800)

[5. Uso de Interfaces de Usuario 40](#_Toc104946801)

[5.1. Interfaz de Cliente 40](#_Toc104946802)

[5.2. Interfaz de Manager 44](#_Toc104946803)

[5.3. Interfaz de Empleado 52](#_Toc104946804)

[6. Evaluación y Análisis 57](#_Toc104946805)

[6.1. CLASES POJO 57](#_Toc104946806)

[6.2. REPOSITORY 65](#_Toc104946807)

[6.3. CONTROLADORES 78](#_Toc104946808)

[7. Conclusiones 121](#_Toc104946809)

[7.1. Fases: 121](#_Toc104946810)

[7.2. Tareas asignadas por persona: 122](#_Toc104946811)

[7.3. Costes 124](#_Toc104946812)

[8. Futuras Implementaciones 127](#_Toc104946813)

[9. Bibliografía 128](#_Toc104946814)

# Introducción

A continuación, exponemos un proyecto de gestión de evaluaciones y alumnado basado en un módulo como, nos indican que está destinado para el módulo de programación, resuelto a través de un desarrollo de software.

Este proyecto está dirigido por Azahara Blanco Rodríguez y el equipo formado por Jeremy Ramos Segura, Daniel Alejandro Carmona Rodríguez, Daniel Rodríguez Fernández y Jan Carlos Ramos Martínez, desarrollado a partir de una propuesta de un trabajo de fin del módulo de programación y entornos de desarrollo.

Este proyecto esta destinado a la gestión de un almacén y E-Commerce donde los clientes pueden realizar pedidos.

En los siguientes apartados detallaremos, los requisitos iniciales de este proyecto, realización de cada una de las partes y otros apartados de interés.

# Problema Propuesto:

Hemos sido contratados para realizar un sistema de gestión de almacenes, vamos a realizarlo para gestionar de forma genérica los Supplier, Reception, Article, Order, User y Customer.

Comienzo describiendo cada clase y la funcionalidad de cada clase:

**User:**

Son aquellos que pueden acceder al sistema, solo los vamos a utilizar para loggearnos, tenemos dos perfiles:

*Atributos de la clase: userName(String), password(String), name(String), surnames(String), photo(String), email(String), dateCreatedAt(Local Date), typeUser(Manager – Employer).*

**Manager:**

Sus funciones son las siguientes: crear Article, actualizar un article, consultar todos los Article, consultar un article por código de article, crear User, eliminar User, consulta de todos los User, consulta de un User por su nombre de User. Crear Supplier, modificar Supplier, listar todos los Supplier, crea una nueva Reception, eliminar una Reception, consultar todas las Receptions, crear Order, borrar Order, consultar todos los Order, consultar un Order por su código de Order, crear store, consultar una store por código de store, consultar todas las Store, imprimir ordenes de Order o de Receptions en HTML, exportar e importar los datos del programa, copia de seguridad del programa.

**Empleado:**

Sus funcione serán más limitadas, sus funciones serán las siguientes: crear un Order, borrar un Order, buscar un Order por código de Order, mostrar todos los Order, listar todos los Article, buscar un article por código de article, crear una Reception, consultar Receptions, generar la orden de Order o de Reception en HTML.

**Article:**

Son todos los elementos físicos que tenemos almacenados en nuestro almacén, y podemos dar salida a través de los Order o de entrada cuando no haya existencias disponibles o deseemos aumentar el stock.

*Atributos de la clase: PIC(UUID), article(String), description(String), location(String), price(double), stock(int), isActive(Boolean).*

*\*\* PIC à Product Identification Code \*\**

**Supplier**

Son aquellos que nos permiten aumentar la cantidad de stock de un article. Pueden suministrar cualquier tipo de article, es decir que un article puede ser suministrado por Supplier diferentes.

*Atributos de la clase: SIC(UUID), nameSuplier(String), direction(String), telephoneNumber(String), email(String)*

*\*\* SIC à Suplier Identification Code \*\**

**Store**

Son aquellos los cuales hacen un Order de Article. Son propias de la empresa, por lo tanto, no se les aplica facturación.

*Atributos de la clase: STIC(UUID), name(String), cif(String), direction(String), telephoneNumber(String), email(String).*

*\*\* STIC à Store Identification Code \*\**

**Reception**

A través de la Reception podemos proveer nuestro almacén de existencias de Article. Las Receptions pueden crearlas tanto los Manager como los Empleados. Se puede hacer cuando necesitemos más cantidad de un article o falta de existencias. El Supplier de una Reception puede suministrar cualquier tipo de article.

Las Receptions no se pueden modificar, en caso de que este mal se tiene que borrar y se crea de nuevo la Reception, esta opción solo está disponible para el perfil de Manager.

Las Receptions están formadas de líneas de Reception.

Una vez que se finaliza la Reception se introduce el método de pago y nos devolverá la Orden de Reception en un archivo HTML.

*Atributos de la Clase Reception: RIC(UUID), SICSuplier(UUID), Carrier(String), Cost(double).*

|  |  |  |  |
| --- | --- | --- | --- |
| RIC | Supplier | Carrier | Cost |

*\*\* RIC à Reception Identification Code \*\**

*Atributos de la Clase Line Reception: RLIC(UUID), pic(UUID), articleName(String), load(int), unitPrice(double), totalPrice(Double)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RLIC | pic | articleName | load | unitPrice | totalPrice |

*\*\* RLIC à Reception Line Identification Code \*\**

**Order**

Cuando un cliente no hace una solicitud de Order, podremos crear un Order tanto el Manager como el Empleado podrán crear Order. Una vez que se realiza un Order se descuenta la cantidad del stock del article.

Si hacemos un Order y no hay existencias de un article tenemos varias opciones, o se salga de crear Order, o nos informe de que no hay cantidades existentes y modifiquemos el Order.

Un Order está formado por el Order y las líneas de Order que lo componen.

Una vez se finaliza el Order se genera una Orden de Order en HTML.

*Atributos de la Clase Order: OIC(UUID), customer(Lista de Clientes), Price(double).*

*\*\* OIC à Order Identification Code \*\**

*Atributos de la Clase Línea Order: OLIC(UUID), article(Lista de Article), load(int), unitPrice(double),*  *totalPrice(double).*

*\*\* OLIC à Order Line Identification Code \*\**

**Tool**

El manager tiene la opción de exportar e importar los datos del programa en un JSON. Dentro del menú crearemos estas dos opciones.

*--Nuevas actualizaciones--*

Tras varias reuniones realizadas con el cliente después de haber realizado, los requisitos funcionales propuestos por el cliente han cambiado, ahora tendremos una vista para que los clientes puedan realizar los pedidos directamente sobre el programa. Ahora no va a ver tiendas si no que tendremos clientes, que engloba todo tipo de usuario final.

Los nuevos requisitos funcionales y de información se detalla más adelante en este mismo documento, con las nuevas actualizaciones que incorpora el cliente.

# Requisitos

## Requisitos Funcionales

|  |  |  |
| --- | --- | --- |
| **Cod** | **Requisito** | **Check** |
| RF1 | El programa debe permitir registrarse como cliente. | ✔ |
| RF2 | El programa debe permitir identificarse como cliente. | ✔ |
| RF3 | El programa debe permitir modificar un cliente ya existente. | ✔ |
| RF4 | El programa debe permitir consultar un cliente por su código | ✔ |
| RF5 | Poder consultar todos los clientes existentes en el programa. | ✔ |
| RF6 | Debe permitir crear un nuevo empleado en el programa. | ✔ |
| RF7 | Debe permitir modificar un empleado ya existente en el programa. | ✔ |
| RF8 | Poder consultar un empleado por su código de empleado. | ✔ |
| RF9 | Posibilidad de consultar todos los empleados existentes del programa. | ✔ |
| RF10 | Debe permitir identificarse como empleado en el programa. | ✔ |
| RF11 | Poder crear nuevos productos y almacenarlos en el programa. | ✔ |
| RF12 | Tener la posibilidad de modificar productos ya existentes en el programa. | ✔ |
| RF13 | Tener la posibilidad de consultar un producto almacenado en el programa por su código. | ✔ |
| RF14 | Tener la posibilidad de consultar todos los productos almacenados en el programa. | ✔ |
| RF15 | Poder crear un pedido en el programa. | ✔ |
| RF16 | Poder consular un pedido existente, por su código de pedido, en el programa. | ✔ |
| RF17 | Consultar todos los pedidos existentes en el programa | ✔ |
| RF18 | El programa debe permitir imprimir la factura de un pedido | ✔ |
| RF19 | Poder identificarse en el programa con el perfil de manager. | ✔ |
| RF20 | Poder crear una nueva recepción en el programa | ✔ |
| RF21 | Poder crear proveedores en el programa | ✔ |
| RF22 | Tener la posibilidad de modificar un proveedor existente en el programa. | ✔ |
| RF23 | Tener la posibilidad de buscar un proveedor por su código en el programa. | ✔ |
| RF24 | Poder consultar una lista completa de los proveedores existentes en el programa. | ✔ |
| RF25 | Tener la posibilidad de consultar todas las recepciones existentes en el programa | ✔ |
| RF26 | Tener la posibilidad de identificarse como manager | ✔ |
| RF27 | Tener la posibilidad de tener herramientas que nos permitan exportar los datos del programa | ✔ |
| RF28 | Poder tener la posibilidad de importar los datos desde un fichero externo a nuestro programa. | ✔ |

## Requisitos No Funcionales

|  |  |  |
| --- | --- | --- |
| **Cod** | **Requisito No Funcional** | **Check** |
| RNF1 | El código de identificación de Article estará formado por las siglas “PIC” seguido de un UUID aleatorio. | ✔ |
| RNF2 | Todos los User recogidos dentro del sistema deben tener asociados a ellos un User y contraseña. | ✔ |
| RNF3 | Todos los precios deben ser de tipo double, con dos decimales y moneda europea. | ✔ |
| RNF4 | El programa tendrá asociada una base de datos migrada en SQLlite. | ✔ |
| RNF5 | El código de identificación de Supplier estará formado por las siglas “SIC” seguido de un UUID aleatorio. | ✔ |
| RNF6 | El código de identificación de store estará formado por las siglas “STIC” seguido de un UUID aleatorio. | ✔ |
| RNF7 | El código de identificación de una Reception estará formado por las siglas “RIC” seguido de un UUID aleatorio. | ✔ |
| RNF8 | El programa cuenta con una copia de seguridad de este, la cual se exportará en un archivo JSON | ✔ |
| RNF9 | El código de identificación de línea de una Reception estará formado por las siglas “RLIC” seguido de un UUID aleatorio. | ✔ |
| RNF10 | El código de identificación de un Order estará formado por las siglas “OIC” seguido de un UUID aleatorio. | ✔ |
| RNF11 | El código de identificación de línea de un Order estará formado por las siglas “OLIC” seguido de un UUID aleatorio. | ✔ |
| RNF12 | El formato de los emails que se almacenen debe tener el siguiente formato: x@x.x | ✔ |
| RNF13 | El formato de los números telefónicos tiene que ser el siguiente:   * Han de comenzar con dígitos entre 6 y 9. * Seguido de, tendrán que disponer de 8 dígitos del 0 al 9. | ✔ |
| RNF14 | Para poder realizar el Order de un article la cantidad solicitada tiene que ser menor o igual a la existente. | ✔ |
| RNF15 | La localización de un article tendrá el siguiente formato:  (F-C-A).   * F será Fila que será entre la A y Z * C será Columna número entre 0 y 9 * A será Altura número entre 0 y 9 | ✔ |
| RNF16 | Sí un article no está activo, no se podrá realizar el Order de uno de ellos. | ✔ |
| RNF17 | El fichero de orden de Order debe ser en formato HTML | ✔ |
| RNF18 | El fichero de orden de Order debe contener Article adquiridos con su cantidad precio unitario y precio de la línea de Order. En el pie de página indicará fecha y hora de generación. | ✔ |
| RNF19 | El programa tiene que ser 100% legible para todos los User que accedan al mismo. | ✔ |
| RNF20 | El programa tiene que ser portátil con lo cual podremos transportarlo a cualquier instancia. | ✔ |
| RNF21 | El fichero de orden de Reception debe ser en formato HTML | ✔ |
| RNF22 | El fichero de orden de Reception debe contene Article adquiridos con su cantidad precio unitario y precio de la línea de Order. En el pie de página indicará fecha y hora de generación. | ✔ |
| RNF23 | El programa cuenta con una pestaña de Acerca de donde irá información complementaria del programa tales como: versión, desarrolladores etc... | ✔ |
| RNF24 | Cada User tendrá un perfil con las opciones disponibles según su perfil, puede ser Manager, Employer o Costumer.  Opciones de Manager:   * Crear Employer. * Crear Costumer. * Modificar Employer. * Modificar Costumer. * Consultar todos los clientes. * Consultar un cliente. * Consultar todos los empleados. * Consultar un empleado * Crear articles. * Consultar un articles. * Consultar todos los articles * Modificar un article. * Crear Supplier. * Modificar Supplier. * Crear una receptions. * Consultar todas las Receptions. * Consultar un order. * Consultar todos los orders. * Imprimir reception. * Imprimir order. * Exportar data. * Importar data.   Opciones de Employer:   * Crear Costumer. * Modificar Costumer. * Lista todas los Costumer. * Consulta de un Costumer. * Añadir Reception. * Lista todas las Receptions. * Añadir artículos. * Lista todos los Article. * Consulta por código de article. * Lista todos los Order. * Consulta por código de Order. * Imprimir Receptions. * Imprimir Orders.   Opciones del Costumer:   * Registrarse * Crear Order * Consultar Article. * Consultar Orders * Consultar sus datos. * Modificar sus datos. * Imprimir tickets de pedidos. | ✔ |
| RNF25 | El campo password de los User debe tener al menos una longitud de 8 caracteres, contener mayúsculas, minúsculas, números y símbolos. | ✔ |
| RNF26 | El campo de pago de una Reception puede ser PayPal o tarjeta | ✔ |
| RNF27 | Nuestro programa tiene que haber cumplido con los test específicos realizados por el conjunto de desarrolladores antes de su, distribución. Debe estar testeado a nivel de repositorio con (Mockito) y a nivel unitario con JUNIT. | ✔ |
| RNF28 | Debajo del botón de entrar los usuarios no registrados tendrán un enlace que les permite registrarse en el sistema. | ✔ |

## Requisitos de Información

|  |  |  |
| --- | --- | --- |
| **Cod** | **Requisito de Información** | **Check** |
| RI1 | Article:   * PIC * article * description * location * price * stock * isActive * photo | ✔ |
| RI2 | Supplier:   * SIC * nameSuplier * direction * telephoneNumber * email | ✔ |
| RI3 | Costumer:   * CIC * Nick * password * name * surname * cif * direction * telephoneNumber * email * photo * createdAt * isActive | ✔ |
| RI4 | Reception:   * RIC * Supplier * Carrier * Cost | ✔ |
| RI5 | Line Reception:   * RLIC * article * load * unitPrice * totalPrice * BelongsReception | ✔ |
| RI6 | Order:   * OIC * customer * Price * methodPay | ✔ |
| RI7 | Línea Order:   * OLIC * article * load * unitPrice * totalPrice * BelongsOrder | ✔ |
| RI8 | Employee:   * EIC * nick * password * name * surname * nif * email * photo * isManager * createdAt * isActive | ✔ |

# Diseño e Implementación

## Diagramas

En este apartado nos encontraremos distintos tipos de Diagramas, los diagramas nos ayudarán a interpretar mucho mejor la estructura y composición de nuestro programa, así como a la hora de programar tener un diseño de la arquitectura que va a seguir.

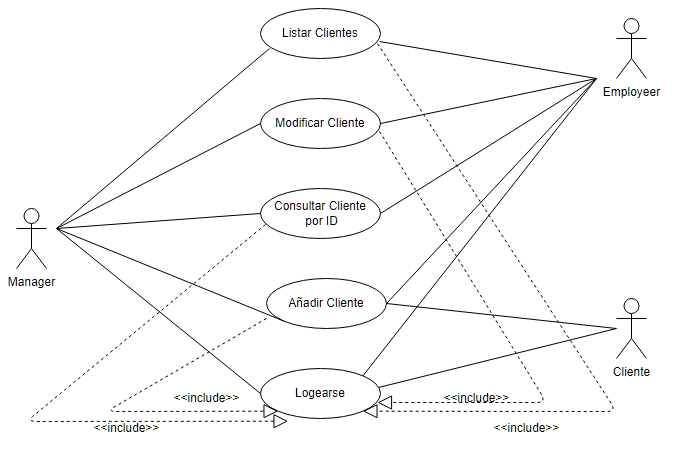
## Diagrama de Clases

Captura de pantalla de un celular con letras

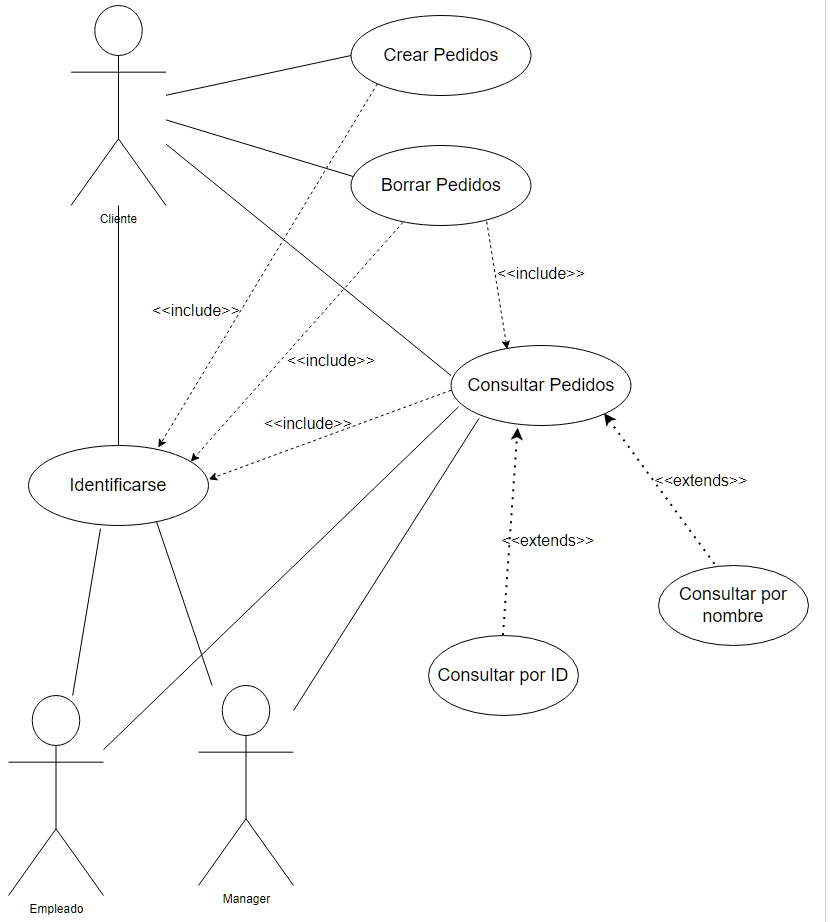
Descripción generada automáticamente con confianza media

## Diagramas de Casos de Uso

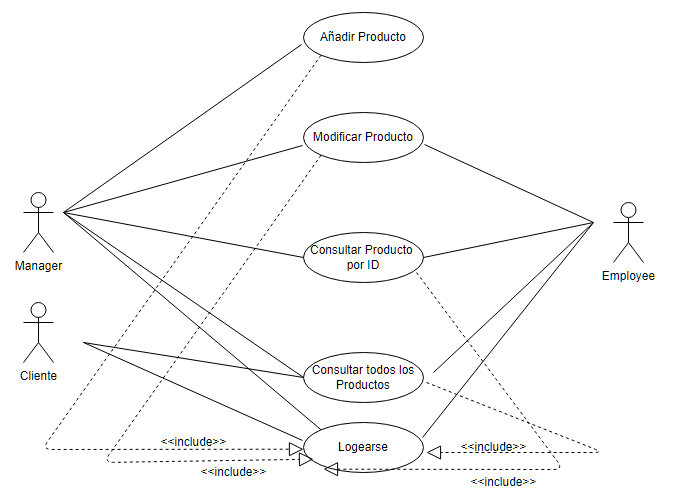
***Cliente***



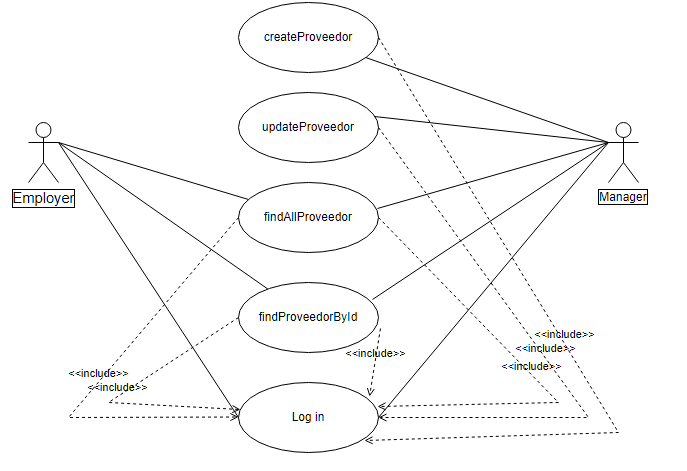
***Pedidos***



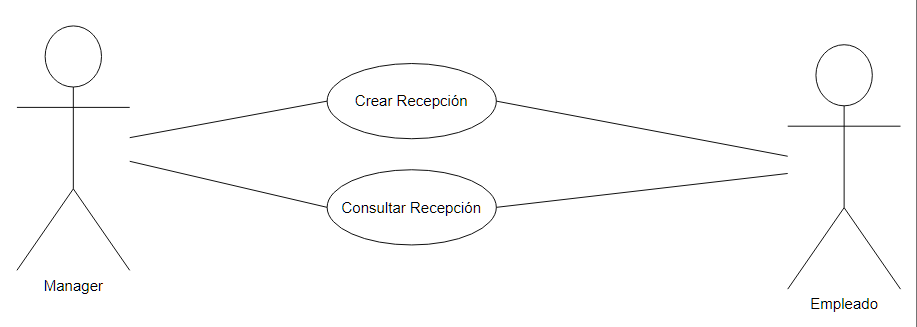
***Productos***



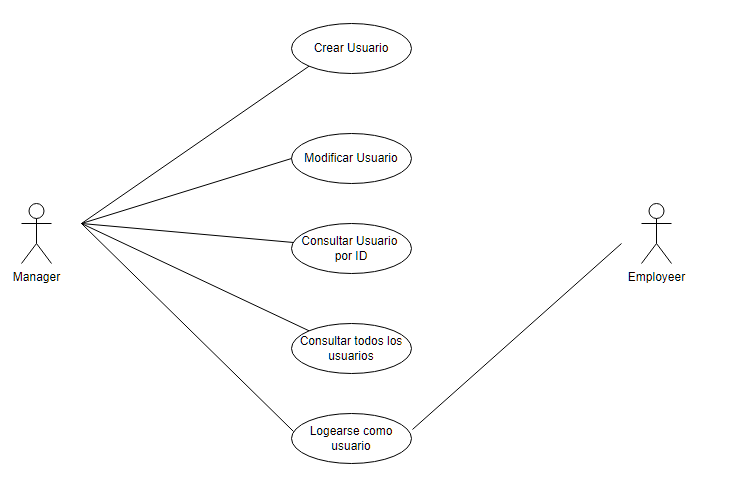
***Proveedor***



***Recepción***



***Usuarios***



## Casos de Uso

***Cliente***

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU01\_Loguear*** | |
| **Actores:** | | Cliente | |
| **Precondiciones:** | | Se requiere que el cliente se encuentre ya registrado en el sistema. | |
| **Postcondiciones:** | | El cliente accede a la pantalla principal. | |
| **CU01\_Loguear** | | |
| 1 | El sistema lanza una ventana para iniciar sesión. | | |
| 2 | El usuario indicara en los campos su nombre de usuario y su contraseña. | | |
| 3 | El sistema comprobara los datos introducidos con los datos del sistema. | | |
| 3.1 | Si los datos introducidos son correctos ir a paso 4. | | |
| 3.2 | Si los datos introducidos son incorrectos el sistema le indicará un mensaje de error y volverá al paso 2. | | |
| 4 | El sistema permitirá entrar al usuario a la pantalla principal y finaliza el caso de uso loguear. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Registrarse*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el cliente seleccione la opción de registrarse. |
| **Postcondiciones:** | | El cliente se ha registrado en el sistema. |
| **CU02\_Registrarse** | | |
| 1 | El sistema lanza una ventana para introducir los datos personales. | |
| 2 | El usuario indicara en los campos sus datos personales. | |
| 3 | El usuario indica que quiere guardar los datos. | |
| 4 | El sistema comprobara si todos los datos son correctos o no se encuentre ningún campo vacío. | |
| 4.1 | Si todos los campos están rellenos y son correctos. Ir a paso 5. | |
| 4.2 | Si algún campo se encuentra vacío o es incorrecto, el sistema mandara un mensaje de error mostrando los posibles errores cometidos. Ir a paso 2. | |
| 5 | El sistema mostrar un mensaje con algunos de los datos introducidos del usuario y esperara confirmación de los datos. | |
| 6 | El usuario indicara una opción. | |
| 6.1 | Si el usuario confirma los datos ir a paso 7. | |
| 6.2 | Si el usuario no confirma los datos ir a paso 2. | |
| 7 | Los datos serán almacenados en el sistema y podrá acceder a la aplicación. Finaliza el caso de uso registrarse. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU03\_Consultar Producto*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el cliente se haya logueado correctamente. |
| **Postcondiciones:** | | El cliente ha consultado el producto. |
| **CU03\_Consultar Producto** | | |
| 1 | El sistema lanza la pantalla principal. | |
| 2 | El usuario podrá ver directamente los productos. | |
| 3 | El usuario selecciona el producto que desea consultar. | |
| 4 | El sistema le mostrar la información del producto seleccionado. | |
| 5 | Finaliza el caso de uso consultar producto. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU04\_Consultar Productos*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el cliente se haya logueado correctamente. |
| **Postcondiciones:** | | El cliente ha consultado los productos. |
| **CU04\_Consultar Productos** | | |
| 1 | El sistema lanza la pantalla principal. | |
| 2 | El usuario podrá ver directamente los productos. | |
| 3 | El usuario indicara en el campo de búsqueda el producto a consultar. | |
| 3.1 | Si el producto existe, ir al paso 4. | |
| 3.2 | Si el producto no existe, no saldrá ningún producto. Ir a paso3. | |
| 4 | El sistema le mostrara en la pantalla principal el o los productos con dicho nombre indicado. | |
| 5 | Finaliza el caso de uso consultar productos. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU05\_Consultar Perfil*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el cliente se haya logueado correctamente. |
| **Postcondiciones:** | | El cliente ha consultado su perfil. |
| **CU05\_Consultar Perfil** | | |
| 1 | El usuario seleccionara la opción para ver su perfil. | |
| 2 | El sistema le mostrara una nueva ventana con toda la información de usuario. | |
| 3 | Finaliza el caso de uso consultar perfil. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU06\_Modificar Perfil*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el cliente haya seleccionado la opción para ver su perfil. |
| **Postcondiciones:** | | El cliente ha modificado su perfil. |
| **CU06\_Modificar Perfil** | | |
| 1 | El usuario indicara la opción para modificar su perfil. | |
| 2 | El sistema le mostrara la ventana con los datos en los campos para poder modificar sus datos. | |
| 3 | El usuario introduce los datos nuevos en los campos. | |
| 4 | El usuario indicara que quiere guardar los datos. | |
| 5 | El sistema comprobara los datos nuevos. | |
| 5.1 | Si los datos introducidos son correctos. Ir a paso 6. | |
| 5.2 | Si los datos introducidos son incorrectos, el sistema le mostrara un mensaje de error. Ir a paso 3. | |
| 6 | El sistema le mostrara los datos nuevos introducidos en la ventana de su perfil. | |
| 7 | Finaliza el caso de uso modificar perfil. | |

***Pedidos Clientes***

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU01\_Crear Pedidos*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el usuario haya iniciado sesión en el sistema. |
| **Postcondiciones:** | | El usuario en cuestión crea un pedido. |
| **CU01\_Crear Pedidos** | | |
| 1 | El sistema le muestra todos los productos disponibles. | |
| 2 | El usuario indica el producto seleccionado y lo añade al carrito. | |
| 3 | El sistema muestra productos existentes seleccionados. | |
| 4 | El usuario selecciona un producto y elige la cantidad. | |
| 4.1 | Sí el producto no se encuentra en stock o no esta disponible, no se le permite añadir al carrito. | |
| 5 | El usuario indica que quiere finalizar el pedido. | |
| 5.1 | Si todo es correcto ir al paso 6. | |
| 5.2 | Sí hay algún fallo con los productos el sistema informara al usuario mostrando un error. | |
| 6 | Se inicia el método de pago frecuente y se muestra los tramites pertinentes. | |
| 7 | El sistema lanza un aviso de éxito completado y muestra un documento HTML donde se refleja los detalles del pedido. | |
| 8 | Finaliza la sección de creación de pedidos. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Consultar Pedidos*** |
| **Actores:** | | Cliente |
| **Precondiciones:** | | Se requiere que el Cliente este registrado en el sistema. |
| **Postcondiciones:** | | El usuario en cuestión consulta él pedido. |
| **CU02\_Consultar Pedidos** | | |
| 1 | El usuario selecciona la opción para consultar sus pedidos. | |
| 2 | El sistema le muestra al usuario la lista de los pedidos realizados por el usuario. | |
| 3 | El usuario selecciona el pedido que quiere consultar. | |
| 4 | El sistema le mostrara en una venta la información del pedido que ha seleccionado. | |
| 5 | El sistema vuelve a la pantalla de inicio. Ir al paso 1 | |
| 5.1 | Sí el usuario desea consultar otro producto ir al paso 2. | |
| 5.1 | Sí el usuario no desea consultar ningún producto ir al paso 6. | |
| 6 | Finaliza la sección de Consulta de Productos | |

***Productos***

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU01Crear Producto*** | |
| **Actores:** | | Manager, empleado. | |
| **Precondiciones:** | | Se requiere que el usuario autorizado esté registrado en el sistema. | |
| **Postcondiciones:** | | Existe un producto nuevo en el sistema | |
| **CU01\_Crear Producto** | | |
| 1 | El sistema lanza el mensaje de bienvenida. | | |
| 2 | El sistema muestra el menú de operaciones. | | |
| 3 | El usuario indica la opción de Artículos. | | |
| 4 | El usuario accede a la opción para crear un producto. | | |
| 5 | El usuario indica los datos. | | |
| 6 | El sistema comprueba que el producto no está ya introducido en el sistema. | | |
| 6.1 | Si el sistema encuentra que hay un producto con el mismo identificador o que los datos no están bien introducidos, informa de un error al crear el producto, pasa al campo 5. | | |
| 6.2 | Si el sistema no encuentra que hay un producto con el mismo identificador, ni que los datos estén mal introducidos pasa al paso 7. | | |
| 7 | El sistema muestra al usuario la lista de los productos actualizada. | | |
| 8 | Finaliza la sección de creación de tienda. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU02\_Modificar Producto*** | |
| **Actores:** | | Manager, empleado | |
| **Precondiciones:** | | Se requiere que el usuario autorizado esté registrado en el sistema. | |
| **Postcondiciones:** | | Los datos de un producto se modifican y almacenan. | |
| **CU02\_Modificar Producto** | | |
| 1 | El usuario autorizado inicia con sus datos al sistema. | | |
| 2 | El sistema muestra al usuario autorizado el menú de operaciones. | | |
| 3 | El usuario autorizado accede a la opción de Artículos. | | |
| 4 | El sistema le muestra la lista de productos al usuario autorizado. | | |
| 5 | El usuario autorizado selecciona uno de los productos y entra al menú de modificar productos. | | |
| 6 | El usuario autorizado modifica los datos del producto. | | |
| 7 | El sistema verifica que los nuevos datos sean correctos. | | |
| 7.1 | Si los datos modificados son válidos se procede al paso 8. | | |
| 7.2 | Si los datos modificados no son válidos, da un mensaje de error y pasa al paso 5. | | |
| 8 | El sistema muestra la lista de todos los productos. | | |
| 9 | Finaliza la sección de Modificar producto. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU03\_Consultar Productos*** | |
| **Actores:** | | Manager, Empleado | |
| **Precondiciones:** | | Se requiere que el usuario autorizado esté registrado en el sistema. | |
| **Postcondiciones:** | | El usuario autorizado habrá consultado los productos disponibles. | |
| **CU03\_Consultar Productos** | | |
| 1 | El usuario elige la opción de Artículos. | | |
| 2 | El sistema le mostrara todos los artículos almacenados en el sistema. | | |
| 4 | El usuario autorizado una vez haya consultado, selecciona la opción de salir. | | |
| 5 | Finaliza la sección de Consulta de productos. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU04\_Consultar Producto*** | |
| **Actores:** | | Manager, Empleado | |
| **Precondiciones:** | | Se requiere que el usuario autorizado esté registrado en el sistema. | |
| **Postcondiciones:** | | Un producto ha sido consultado por un usuario autorizado. | |
| **CU04\_Consultar producto** | | |
| 1 | El usuario elige la opción de Artículos. | | |
| 2 | El sistema le mostrara todos los artículos almacenados en el sistema. | | |
| 3 | El usuario indicara el artículo que desea buscar. | | |
| 3.1 | Si existe algún producto en el sistema, muestra los productos existentes y pasa al paso 4. | | |
| 3.1 | Si no existe el producto en el sistema no le mostrara nada e ir a paso 3. | | |
| 4 | El sistema muestra la lista de productos. | | |
| 5 | El usuario selecciona el producto que desea consultar y selecciona la opción para ver los detalles. | | |
| 6 | El sistema le mostrara una nueva ventana con los datos del producto seleccionado. | | |
| 7 | Finaliza la sección de consulta de Producto. | | |

***Proveedor***

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU01\_Consultar\_Proveedores*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el manager se encuentre ya registrado en el sistema. |
| **Postcondiciones:** | | Listado proveedores. |
| ***CU01\_Consultar\_Proveedores*** | | |
| 1 | El usuario elige la opción de Proveedores. | |
| 2 | El sistema le mostrara todos los proveedores almacenados en el sistema. | |
| 3 | El usuario una vez haya consultado, selecciona la opción de salir. | |
| 4 | Finaliza la sección de Consulta de Proveedor. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Consultar\_Proveedor*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el manager se encuentre ya registrado en el sistema. |
| **Postcondiciones:** | | Consultado proveedor. |
| ***CU02\_Consultar\_Proveedor*** | | |
| 1 | El usuario elige la opción de Proveedores. | |
| 2 | El sistema le mostrara todos los proveedores almacenados en el sistema. | |
| 3 | El usuario indicara el proveedor que desea buscar. | |
| 3.1 | Si existe algún proveedor en el sistema, muestra los proveedores existentes y pasa al paso 4. | |
| 3.2 | Si no existe el proveedor en el sistema no le mostrara nada e ir a paso 3. | |
| 4 | El sistema muestra la lista de proveedores. | |
| 5 | Finaliza la sección de consulta de Proveedores. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU03\_Modificar\_Proveedor*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el manager se encuentre ya registrado en el sistema. |
| **Postcondiciones:** | | Modificado supplier |
| ***CU03\_Modificar\_Proveedor*** | | |
| 1 | El usuario elige la opción de Proveedores. | |
| 2 | El sistema le mostrara todos los proveedores almacenados en el sistema. | |
| 3 | El sistema le muestra la lista de proveedores al usuario autorizado. | |
| 4 | El usuario autorizado selecciona uno de los proveedores y entra a la opción para modificar el proveedor. | |
| 5 | El sistema le mostrara una nueva ventana con los datos a modificar del proveedor. | |
| 6 | El usuario autorizado modifica los datos del proveedor. | |
| 7 | El sistema verifica que los nuevos datos sean correctos. | |
| 7.1 | Si los datos modificados son válidos se procede al paso 8. | |
| 7.2 | Si los datos modificados no son válidos, da un mensaje de error y pasa al paso 5. | |
| 8 | El sistema muestra la lista de todos los productos. | |
| 9 | Finaliza la sección de modificar Proveedores. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU04\_Crear\_Proveedor*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el manager se encuentre ya registrado en el sistema. |
| **Postcondiciones:** | | Crear supplier |
| ***CU04\_ Crear \_Proveedor*** | | |
| 1 | El usuario elige la opción de Proveedores. | |
| 2 | El sistema le mostrara todos los proveedores almacenados en el sistema. | |
| 3 | El sistema le muestra la lista de proveedores al usuario autorizado. | |
| 4 | El usuario autorizado selecciona uno de los proveedores y entra a la opción para crear un nuevo proveedor. | |
| 5 | El sistema le mostrara una nueva ventana para introducir los datos del nuevo proveedor. | |
| 6 | El usuario introducirá los datos nuevos del nuevo proveedor. | |
| 7 | El sistema verifica que los datos sean correctos. | |
| 7.1 | Si los datos son válidos se procede al paso 8. | |
| 7.2 | Si los datos no son válidos, da un mensaje de error y pasa al paso 5. | |
| 8 | El sistema muestra la lista de todos los productos. | |
| 9 | Finaliza la sección de crear Proveedor. | |

***Clientes***

|  |  |  |
| --- | --- | --- |
|  | | ***CU01\_Crear Cliente*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema. |
| **Postcondiciones:** | | Existe un cliente nuevo en el sistema |
| **CU01\_Crear\_Cliente** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de clientes. | |
| 3 | El sistema mostrará los clientes y las operaciones que se pueden realizar. | |
| 4 | El usuario manager indicara que desea agregar un nuevo cliente. | |
| 5 | El sistema mostrará la información que se debe cumplimentar para agregar un nuevo usuario. | |
| 6 | El usuario manager cumplimentara la información solicitada y aceptara. | |
| 6.1 | Sí el sistema encuentra que hay campos vacíos informa de un error al crear el cliente, pasa al campo 6 | |
| 6.2 | Sí el sistema no encuentra que hay datos obligatorios vacíos del cliente pasa al paso 7. | |
| 7 | El sistema muestra los clientes del sistema. | |
| 8 | Finaliza la sección de creación de clientes. | |

|  |  |  |
| --- | --- | --- |
|  | | ***CU02\_Modificar Cliente*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema y exista un cliente. |
| **Postcondiciones:** | | Los datos de un cliente se modifican y almacenan. |
| **CU02\_Modificar\_Cliente** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de clientes. | |
| 3 | El sistema mostrará los clientes y las operaciones que se pueden realizar. | |
| 4 | El usuario selecciona el cliente que desea modificar e indica que desea realizar la operación de modificar. | |
| 5 | Se muestra al usuario los datos actuales del cliente a modificar. | |
| 6 | El usuario introduce los datos que desee modificar del cliente. | |
| 7 | El sistema comprueba los datos introducidos por el usuario. | |
| 7.1 | Sí el sistema encuentra que hay campos vacíos informa de un error al crear el cliente, pasa al campo 6 | |
| 7.2 | Sí el sistema no encuentra datos vacíos de carácter obligatorio del cliente, pasa al paso 7. | |
| 7 | Finaliza la sección de Modificar Cliente. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU03\_Consultar\_Clientes*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema y haya clientes en el sistema dados de alta. |
| **Postcondiciones:** | | El usuario habrá consultado los clientes existentes en el sistema. |
| **CU03\_Consultar\_Clientes** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de clientes. | |
| 3 | El sistema mostrará los clientes existentes que hay almacenados en el sistema. | |
| 4 | Finaliza la sección de Consulta de clientes. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU04\_Consultar\_Cliente*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema y haya clientes en el sistema dados de alta. |
| **Postcondiciones:** | | Los datos de un cliente han sido consultados por un manager. |
| **CU04\_Consultar\_Cliente** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de clientes. | |
| 3 | El sistema mostrará los clientes existentes que hay almacenados en el sistema. | |
| 4 | El manager introducirá el nombre del cliente o el CIC de cliente que desea consultar. | |
| 5 | El sistema comprueba que el nombre de usuario/CIC de usuario está en el sistema. | |
| 5.1 | Sí el sistema no encuentra un cliente asociado al nombre de usuario/CIC de cliente, en la pantalla no se mostrará ningún dato, pasa al campo 4. | |
| 5.2 | Sí el sistema encuentra un cliente asociado al nombre de usuario/CIC de cliente, pasa al paso 6. | |
| 6 | El sistema muestra el usuario y sus datos almacenados en sistema. | |
| 7 | Finaliza la sección de consulta cliente. | |

***Empleados***

|  |  |  |
| --- | --- | --- |
|  | | ***CU01\_Crear Empleado*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema. |
| **Postcondiciones:** | | Existe un empleado nuevo en el sistema |
| **CU01\_Crear\_Empleado** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de empleados. | |
| 3 | El sistema mostrará los empleados y las operaciones que se pueden realizar. | |
| 4 | El usuario manager indicara que desea agregar un nuevo empleado. | |
| 5 | El sistema mostrará la información que se debe cumplimentar para agregar un nuevo usuario. | |
| 6 | El usuario manager cumplimentara la información solicitada y aceptara. | |
| 6.1 | Sí el sistema encuentra que hay campos vacíos informa de un error al crear el empleado, pasa al campo 6 | |
| 6.2 | Sí el sistema no encuentra que hay datos obligatorios vacíos del empleado, pasa al paso 7. | |
| 7 | El sistema muestra los usuarios del sistema. | |
| 8 | Finaliza la sección de creación de empleado. | |

|  |  |  |
| --- | --- | --- |
|  | | ***CU02\_Modificar Empleado*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema y exista un empleado. |
| **Postcondiciones:** | | Los datos de un usuario se modifican y almacenan. |
| **CU02\_Modificar\_Empleado** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario manager selecciona la opción de empleados. | |
| 3 | El sistema mostrará los empleados y las operaciones que se pueden realizar. | |
| 4 | El usuario selecciona el empleado que desea modificar e indica que desea realizar la operación de modificar. | |
| 5 | Se muestra al usuario los datos actuales del empleado a modificar. | |
| 6 | El usuario introduce los datos que desee modificar del empleado. | |
| 7 | El sistema comprueba los datos introducidos por el usuario. | |
| 7.1 | Sí el sistema encuentra que hay campos vacíos informa de un error al crear el empleado, pasa al campo 6 | |
| 7.2 | Sí el sistema no encuentra datos vacíos de carácter obligatorio del empleado, pasa al paso 7. | |
| 7 | Finaliza la sección de Modificar empleado. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID:** | | ***CU03\_Consultar\_Empleados*** | |
| **Actores:** | | Manager | |
| **Precondiciones:** | | Se requiere que el Manager esté registrado en el sistema y haya empleados en el sistema dados de alta. | |
| **Postcondiciones:** | | El usuario habrá consultado los empleados existentes en el sistema. | |
| **CU03\_Consultar\_Empleados** | | | |
| 1 | El sistema lanza el menú de operaciones. | | |
| 2 | El usuario manager selecciona la opción de empleados. | | |
| 3 | El sistema mostrará los empleados existentes que hay almacenados en el sistema. | | |
| 4 | Finaliza la sección de Consulta de empleados | | |
| ID: | CU04\_Consultar\_Empleado | | |
| Actores: | Manager | | |
| Precondiciones: | Se requiere que el Manager esté registrado en el sistema y haya empleados en el sistema dados de alta. | | |
| Postcondiciones: | Los datos de un empleado han sido consultados por un manager. | | |
| **CU04\_Consultar\_Empleado** | | | |
| 1 | | | El sistema lanza el menú de operaciones. |
| 2 | | | El usuario manager selecciona la opción de empleados. |
| 3 | | | El sistema mostrará los empleados existentes que hay almacenados en el sistema. |
| 4 | | | El manager introducirá el nombre de usuario o el EIC de empleado que desea consultar. |
| 5 | | | El sistema comprueba que el nombre de usuario/EIC de usuario está en el sistema. |
| 5.1 | | | Sí el sistema no encuentra un empleado asociado al nombre de usuario/EIC de empleado, en la pantalla no se mostrará ningún dato, pasa al campo 4. |
| 5.2 | | | Sí el sistema encuentra un empleado asociado al nombre de usuario/EIC de empleado, pasa al paso 6. |
| 6 | | | El sistema muestra el usuario y sus datos almacenados en sistema. |
| 7 | | | Finaliza la sección de consultar empleado. |

***Imprimir Tickets***

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU01\_Imprimir Recepción*** |
| **Actores:** | | Manager y empleado. |
| **Precondiciones:** | | Se requiere que el Manager o empleado haya iniciado sesión. |
| **Postcondiciones:** | | Se imprime el ticket de la recepción seleccionada. |
| **CU01\_Imprimir Recepción** | | |
| 1 | El usuario accede a la consulta de recepciones. | |
| 2 | El usuario selecciona la recepción que desea imprimir. | |
| 3 | El sistema le avisa al usuario que ha sido impreso el html. | |
| 4 | Finaliza la sección de Imprimir recepción. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Imprimir Orden*** |
| **Actores:** | | Manager y empleado |
| **Precondiciones:** | | Se requiere que el Manager o empleado haya iniciado sesión. |
| **Postcondiciones:** | | Se imprime el ticket del pedido seleccionado. |
| **CU01\_Imprmir Orden** | | |
| 1 | El usuario accede a la consulta de pedidos. | |
| 2 | El usuario selecciona el pedido que desea imprimir. | |
| 3 | El sistema le avisa al usuario que ha sido impreso el html. | |
| 4 | Finaliza la sección de Imprimir orden. | |

***Importar y Exportar***

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU01\_Exportar datos*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager haya iniciado sesión. |
| **Postcondiciones:** | | Los datos son exportados |
| **CU01\_Exportar Datos** | | |
| 1 | El sistema le muestra una ventana con todas las opciones. | |
| 2 | El usuario indica que desea exportar los datos. | |
| 3 | El sistema exporta los datos. | |
| 4 | El sistema le indica que los datos han sido exportados correctamente. | |
| 5 | Finaliza la sección de Exportar datos. | |

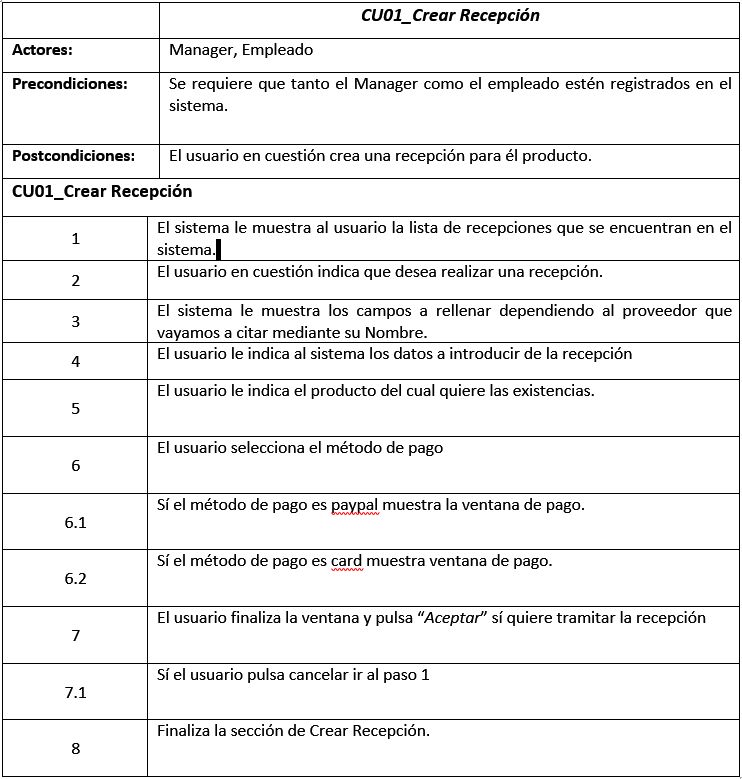
|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Importar Datos*** |
| **Actores:** | | Manager |
| **Precondiciones:** | | Se requiere que el Manager haya iniciado sesión. |
| **Postcondiciones:** | | Los datos son importados. |
| **CU02\_Importar Datos** | | |
| 1 | El sistema le muestra una ventana con todas las opciones. | |
| 2 | El usuario indica que desea importar los datos. | |
| 3 | El sistema importa los datos. | |
| 4 | El sistema le indica que los datos han sido importados correctamente. | |
| 5 | Finaliza la sección de Importar datos. | |

***Pedidos***

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU01\_Consultar pedidos*** |
| **Actores:** | | Manager, Empleado |
| **Precondiciones:** | | Se requiere que el Manager o empleado haya iniciado sesión. |
| **Postcondiciones:** | | Se han consultado los pedidos. |
| **CU01\_Consultar Pedidos** | | |
| 1 | El usuario elige la opción de Pedidos. | |
| 2 | El sistema le mostrara todos los pedidos almacenados en el sistema. | |
| 3 | El usuario autorizado una vez haya consultado, selecciona la opción de salir. | |
| 4 | Finaliza la sección de Consulta de pedidos. | |

|  |  |  |
| --- | --- | --- |
| **ID:** | | ***CU02\_Consultar\_Pedido*** |
| **Actores:** | | Manager, Empleado |
| **Precondiciones:** | | Se requiere que el Manager o empleado haya iniciado sesión. |
| **Postcondiciones:** | | Los datos de un pedido han sido consultados por un manager o empleado. |
| **CU02\_Consultar\_Pedido** | | |
| 1 | El sistema lanza el menú de operaciones. | |
| 2 | El usuario selecciona la opción de pedidos. | |
| 3 | El sistema mostrará los pedidos existentes que hay almacenados en el sistema. | |
| 4 | El manager introducirá el nombre del cliente o el OIC de pedido que desea consultar. | |
| 5 | El sistema comprueba que el nombre de usuario/OIC de pedido está en el sistema. | |
| 5.1 | Sí el sistema no encuentra un cliente asociado al nombre de usuario/OIC de pedido, en la pantalla no se mostrará ningún dato, pasa al campo 4. | |
| 5.2 | Sí el sistema encuentra un cliente asociado al nombre de usuario/OIC de pedido, pasa al paso 6. | |
| 6 | El sistema muestra el usuario y sus datos almacenados en sistema. | |
| 7 | El usuario indica el pedido que quiere consultar y selecciona la opción para ver los detalles. | |
| 8 | El sistema le mostrara los detalles del pedido y sus líneas de pedido. | |
| 9 | Finaliza la sección de consultar pedido. | |

***Recepción***



Tabla

Descripción generada automáticamente

## Historias de Usuario

***Cliente***

|  |
| --- |
| Historia de usuario |
| Registrar cliente |
| Roll de usuario |
| Manager, Employeer, Cliente |
| Objetivo |
| Poder introducir las características de un cliente nuevo en el sistema. |
| Beneficio |
| Añadir un nuevo cliente que podrá hacer pedidos en el sistema. |

|  |
| --- |
| Historia de usuario |
| Consultar cliente |
| Roll de usuario |
| Manager, Employeer |
| Objetivo |
| Poder obtener información de un cliente en concreto. |
| Beneficio |
| Poder modificarlo o ver su información. |

|  |
| --- |
| Historia de usuario |
| Consultar todos los clientes |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder obtener información de todos los clientes. |
| Beneficio |
| Poder seleccionar un cliente , ver los clientes disponibles o compararlo con otros. |

|  |
| --- |
| Historia de usuario |
| Modificar cliente |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder modificar las características de un cliente. |
| Beneficio |
| No tener que borrar un cliente y crearlo cuando sea necearía una modificación de este. |

***Productos***

|  |
| --- |
| Historia de usuario |
| Crear producto |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder introducir las características de un producto nuevo en el sistema. |
| Beneficio |
| Para poder usarlo en las recepciones y pedidos |

|  |
| --- |
| Historia de usuario |
| Consultar producto |
| Roll de usuario |
| Manager, Employeer, Cliente |
| Objetivo |
| Poder obtener información de un producto en concreto. |
| Beneficio |
| Poder modificarlo o utilizarlo en otras funciones. |

|  |
| --- |
| Historia de usuario |
| Consultar todos los productos |
| Roll de usuario |
| Manager, Employeer, Cliente |
| Objetivo |
| Poder obtener información de todos los productos. |
| Beneficio |
| Poder seleccionar un producto , ver los productos disponibles o compararlo con otros. |

|  |
| --- |
| Historia de usuario |
| Modificar producto |
| Roll de usuario |
| Manager |
| Objetivo |
| Pode modificar las características de un producto. |
| Beneficio |
| No tener que borrar un producto y crearlo cuando sea necearía una modificación de este. |

***Usuario***

|  |
| --- |
| Historia de usuario |
| Crear Usuario |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder introducir las características de un usuario nuevo en el sistema. |
| Beneficio |
| Añadir un nuevo usuario que podrá hacer modificaciones en los archivos del sistema. |

|  |
| --- |
| Historia de usuario |
| Consultar Usuario |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder obtener información de un usuario en concreto. |
| Beneficio |
| Poder modificarlo o ver su información. |

|  |
| --- |
| Historia de usuario |
| Consultar todos los Usuarios |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder obtener información de todos los usuario. |
| Beneficio |
| Poder seleccionar un usuario , ver los usuarios disponibles o compararlo con otros. |

|  |
| --- |
| Historia de usuario |
| Modificar Usuario |
| Roll de usuario |
| Manager |
| Objetivo |
| Poder modificar las características de un usuario. |
| Beneficio |
| No tener que borrar un usuario y crearlo cuando sea necearía una modificación de este. |

***Importar y Exportar***

|  |
| --- |
| Historia de usuario |
| Imprimir Reception |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Una vez realizadas la recepción imprimimos los datos de esta. |
| Beneficio |
| Expedir las recepciones realizadas por cada cliente. |

|  |
| --- |
| Historia de usuario |
| Imprimir Orden |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Obtener copia escrita de las ordenes realizadas por cliente |
| Beneficio |
| Expedir las ordenes realizadas por cada cliente. |

|  |
| --- |
| Historia de usuario |
| Importar Datos |
| Roll de usuario |
| Manager |
| Objetivo |
| Obtener los datos almacenados tanto en base de datos como en ficheros. |
| Beneficio |
| Tener los datos actualizados de todo el sistema. |

|  |
| --- |
| Historia de usuario |
| Exportar data |
| Roll de usuario |
| Manager |
| Objetivo |
| Guardar los datos en la base de datos como también en ficheros de texto. |
| Beneficio |
| Tener varias fuentes de almacenamiento de la información del sistema. |

***Pedidos***

|  |
| --- |
| Historia de usuario |
| Crear Pedidos |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| El usuario introduzca las características necesarias para la creación del pedido en curso. |
| Beneficio |
| Añadir un nuevo pedido al almacén para ser procesados. |

|  |
| --- |
| Historia de usuario |
| Borrar Pedidos |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Poder Eliminar un pedido en curso consultando por ID. |
| Beneficio |
| Eliminar el pedido deseado. |

|  |
| --- |
| Historia de usuario |
| Consultar Pedidos |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Poder listar y mostrar todos los pedidos existentes en el sistema que se consultarán por ID o por el Nombre |
| Beneficio |
| Ver todos los pedidos efectuados y realizados por los empleados. |

***Recepción***

|  |
| --- |
| Historia de usuario |
| Crear Recepción |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| El usuario introduzca las características necesarias para la creación de la recepción en curso. |
| Beneficio |
| Solicitar y añadir una nueva recepción. |

|  |
| --- |
| Historia de usuario |
| Consultar Recepciones |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Poder listar y mostrar todos las recepciones en curso que se encuentran en el sistema |
| Beneficio |
| Ver todas las recepciones activas y en curso actualmente. |

***Proveedor***

|  |
| --- |
| Historia de usuario |
| Crear Proveedor |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| El mánager introduce los datos necesarios para la creación del proveedor. |
| Beneficio |
| Añadir un nuevo proveedor para el suministro de producto y mantener nuestro stock |

|  |
| --- |
| Historia de usuario |
| Modificar Proveedor |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Actualizar los datos del proveedor. |
| Beneficio |
| Mantener los datos del proveedor actualizados, y modificar datos erróneos. |

|  |
| --- |
| Historia de usuario |
| Consultar todos los Proveedor |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Mostrar la lista de todo os proveedores que tenemos en nuestra base de datos. |
| Beneficio |
| Ver todo los proveedores actuales en nuestra base de datos. |

|  |
| --- |
| Historia de usuario |
| Consultar un Proveedor |
| Roll de usuario |
| Manager, Empleado |
| Objetivo |
| Mostrar un proveedor en concreto. |
| Beneficio |
| Facilitar la búsqueda de un proveedor, de manera que sea más rápido encontrarlo. |

# Uso de Interfaces de Usuario

## Interfaz de Cliente

Para identificarnos como clientes en el sistema tenemos la opción de introducir las credenciales que ya tuviésemos, introduciendo usuario y contraseña, después pulsaremos en “Login”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En el caso de que no tengamos usuario en el sistema, haremos click sobre Sing up.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ahora rellanamos el formulario, completando todos los campos y aceptando las condiciones del software y una finalicemos, hacemos click sobre “Guardar”.

Interfaz de usuario gráfica

Descripción generada automáticamente

Al pulsar sobre guardar, nos mostrará un informe con todos los datos introducidos sobre cliente, los comprobamos y pulsamos sobre aceptar.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ahora que tenemos nuestros datos de usuario, los introducimos en la pantalla de Login.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Una vez estamos logueados en el sistema, se nos abre la ventana principal podemos encontrar todos los productos que podemos añadir a nuestro carrito. En el menú izquierdo podemos consultar nuestros datos de usuario y los pedidos realizados.

Interfaz de usuario gráfica, Calendario

Descripción generada automáticamente con confianza media

Al pulsar sobre mi perfil, podemos consultar los datos de usuario y en el caso de que lo desee podemos modificar los datos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar los datos de usuario, hacemos click sobre “Modificar datos”, se nos abre una nueva ventana donde introducimos los nuevos datos de usuario una vez finalicemos hacemos click sobre “Guardar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para añadir un producto a nuestro carrito de la compra, hacemos click sobre el producto.

Calendario

Descripción generada automáticamente

Al hacer click sobre el producto, se nos abre una nueva ventana, para añadir al stock pulsamos sobre “Añadir al carrito”. En caso de que no haya stock disponible de un producto este botón estará deshabilitado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Sí pulsamos sobre “Mis pedidos” podemos consultar los pedidos que hemos realizado.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Sí seleccionamos un pedido y a continuación, “Imprimir Ticket” nos genera un ticket de la compra.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## Interfaz de Manager

Para identificarnos como Manager en el programa, nuestro Administrador de Base de Datos, nos debe haber facilitado un usuario o nuestro Manager superior nos haya dado de alta en el sistema.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez estemos identificados en el programa, podremos acceder a diferentes funcionalidades desde el menú de operaciones del Manager.

Interfaz de usuario gráfica

Descripción generada automáticamente

Operaciones sobre artículos, hacemos click sobre artículos, nos mostrará una tabla con todos los artículos que tenemos en nuestro almacén, desde esta ventana podremos crear artículos, modificarlos, consultar artículos por PIC y mostrar los detalles de un artículo.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Al pulsar sobre el cuadro de búsqueda sobre PIC de un artículo, nos mostrará los datos filtrados, según el PIC introducido.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Crear un nuevo producto para ello hacemos click sobre “Crear Nuevo”, nos abre una nueva ventana donde introducimos los datos del artículo. Para terminar de registrarlo pulsamos en “Aceptar”.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Para consultar los detalles de un producto, seleccionamos el producto en tabla de datos y después hacemos click sobre “Ver detalles”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar un producto, seleccionamos un producto y a continuación pulsamos en modificar producto. Ahora introducimos los nuevos datos del producto y hacemos click sobre “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones consultar, crear y modificar los empleados. Para ello hacemos click sobre “Empleados”. Ahora podemos ver una tabla con el registro de todos los empleados y también los botones de operaciones para crear, modificar y consultar un empleado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para consultar un empleado, el cuadro de texto podemos introducir su número de EIC o su Nick de usuario, una vez introducido el dato pulsamos en intro.

Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente

Para agregar un nuevo empleado, pulsamos sobre “Crear nuevo”, aquí se nos abre una nueva ventana donde debemos introducir los datos del nuevo usuario. Una vez estén rellenos todos los campos, indicar sí tiene perfil de manager y por último, hacemos click en “Aceptar”.

Interfaz de usuario gráfica

Descripción generada automáticamente

Para modificar un empleado, seleccionamos el empleado que queremos modificar y pulsamos sobre “Modificar”. Ahora modificamos los campos que deseamos una vez hayamos realizado las modificaciones sobre el usuario hacemos click en “Aceptar”.

Interfaz de usuario gráfica

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones consultar, crear y modificar los empleados. Para ello hacemos click sobre “Clientes”. Ahora podemos ver una tabla con el registro de todos los clientes y también los botones de operaciones para crear y modificar un cliente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para consultar un cliente, el cuadro de texto podemos introducir su número de CIC, una vez introducido el dato pulsamos en intro.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Para agregar un nuevo cliente, pulsamos sobre “Crear nuevo”, aquí se nos abre una nueva ventana donde debemos introducir los datos del nuevo usuario. Una vez estén rellenos todos los campos, indicar sí tiene perfil de manager y por último, hacemos click en “Aceptar”.

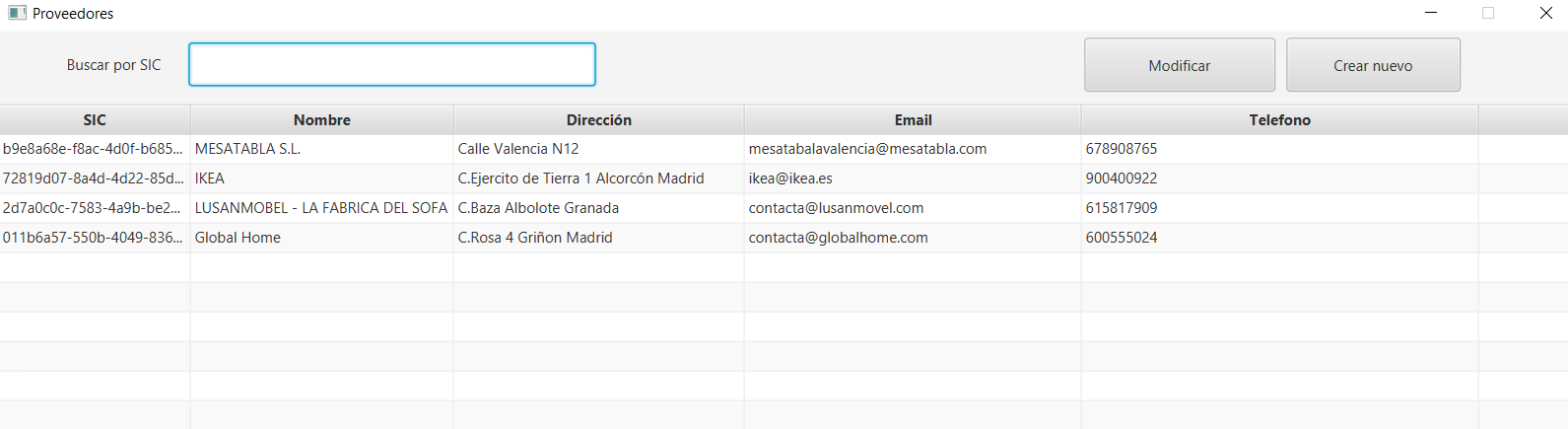
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar un cliente, seleccionamos el cliente que queremos modificar y pulsamos sobre “Modificar”. Ahora modificamos los campos que deseamos una vez hayamos realizado las modificaciones sobre el usuario hacemos click en “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones consultar, crear y modificar los proveedores. Para ello hacemos click sobre “Proveedores”. Ahora podemos ver una tabla con el registro de todos los proveedores y también los botones de operaciones para crear y modificar un proveedor.

Para agregar un nuevo proveedor, pulsamos sobre “Crear nuevo”, aquí se nos abre una nueva ventana donde debemos introducir los datos del nuevo proveedor. Una vez estén rellenos todos los campos, y cumplan con las condiciones, hacemos click en “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar un proveedor, seleccionamos el proveedor que queremos modificar y pulsamos sobre “Modificar”. Ahora modificamos los campos que deseamos una vez hayamos realizado las modificaciones sobre el proveedor hacemos click en “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones de consultar los pedidos realizados al almacén. Para ello hacemos click sobre “Pedidos”. Ahora podemos ver una tabla con el registro de todos los pedidos.

Aplicación

Descripción generada automáticamente con confianza baja

Sí queremos consultar el detalle de un pedido para ello seleccionamos el pedido que deseemos y a continuación pulsamos sobre “Ver Detalle”.

Tabla

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones consultar y crear las recepciones del almacén. Para ello hacemos click sobre “Recepciones”. Ahora podemos ver una tabla con el registro de todas las recepciones y también botón para crear una nueva recepción.

Tabla

Descripción generada automáticamente

Para crear una nueva recepción pulsamos sobre “Crear nuevo” y aquí se nos abre una nueva ventana donde le indicamos los datos que nos solicita y por último hacemos click en “Aceptar”.

Interfaz de usuario gráfica

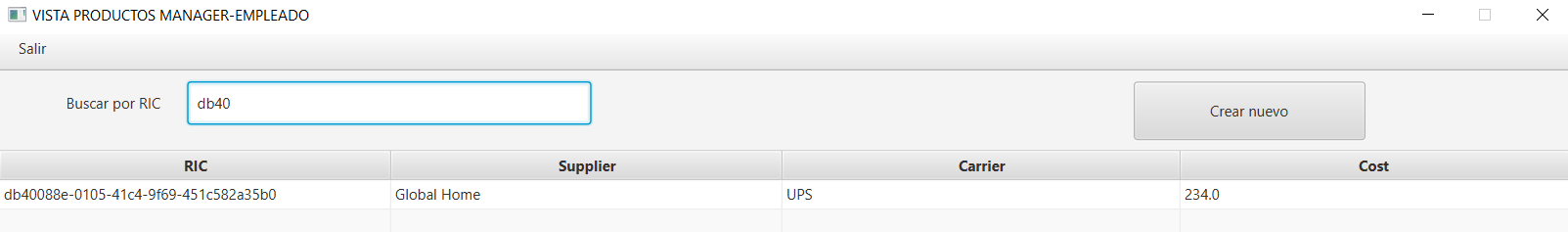
Descripción generada automáticamente

Sí queremos consultar los detalles de una Recepción, hacemos click sobre esa recepción y después en “Ver Detalles”.

Tabla

Descripción generada automáticamente

Si queremos consultar una recepción en el cuadro de búsqueda *RIC*, introducimos el número de RIC que deseemos buscar.



También tenemos un menú superior con las principales utilidades del programa, desde la pestaña de salir podremos cerrar el programa.

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

Desde la pestaña de herramientas podemos exportar e importar los datos del programa.

Texto

Descripción generada automáticamente

Desde la pestaña de “Acerca de” podemos consultar los datos más importantes del programa, versión, licencia y desarrolladores.

Diagrama

Descripción generada automáticamente con confianza media

Aquí podemos ver el contenido de la pestaña, una vez hayamos consultado la información hacemos click sobre “Salir”.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

## Interfaz de Empleado

Para identificarnos como Empleado en el programa, nuestro Manager superior nos debe haber dado de alta en el sistema.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez estemos identificados en el programa, podremos acceder a diferentes funcionalidades desde el menú de operaciones del Empleado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Operaciones sobre artículos, hacemos click sobre artículos, nos mostrará una tabla con todos los artículos que tenemos en nuestro almacén, desde esta ventana podremos crear artículos, modificarlos, consultar artículos por PIC y mostrar los detalles de un artículo.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Al pulsar sobre el cuadro de búsqueda sobre PIC de un artículo, nos mostrará los datos filtrados, según el PIC introducido.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Crear un nuevo producto para ello hacemos click sobre “Crear Nuevo”, nos abre una nueva ventana donde introducimos los datos del artículo. Para terminar de registrarlo pulsamos en “Aceptar”.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Para consultar los detalles de un producto, seleccionamos el producto en tabla de datos y después hacemos click sobre “Ver detalles”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar un producto, seleccionamos un producto y a continuación pulsamos en modificar producto. Ahora introducimos los nuevos datos del producto y hacemos click sobre “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones consultar, crear y modificar los empleados. Para ello hacemos click sobre “Clientes”. Ahora podemos ver una tabla con el registro de todos los clientes y también los botones de operaciones para crear y modificar un cliente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para consultar un cliente, el cuadro de texto podemos introducir su número de CIC o su Nick de usuario, una vez introducido el dato pulsamos en intro.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Para agregar un nuevo cliente, pulsamos sobre “Crear nuevo”, aquí se nos abre una nueva ventana donde debemos introducir los datos del nuevo usuario. Una vez estén rellenos todos los campos, indicar sí tiene perfil de manager y por último, hacemos click en “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para modificar un cliente, seleccionamos el cliente que queremos modificar y pulsamos sobre “Modificar”. Ahora modificamos los campos que deseamos una vez hayamos realizado las modificaciones sobre el usuario hacemos click en “Aceptar”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

También tenemos la posibilidad desde el menú de operaciones de consultar los pedidos realizados al almacén. Para ello hacemos click sobre “Pedidos”. Ahora podemos ver una tabla con el registro de todos los pedidos.

Aplicación

Descripción generada automáticamente con confianza baja

Sí queremos consultar el detalle de un pedido para ello seleccionamos el pedido que deseemos y a continuación pulsamos sobre “Ver Detalle”.

Tabla

Descripción generada automáticamente

También tenemos un menú superior con las principales utilidades del programa, desde la pestaña de salir podremos cerrar el programa.

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

Desde la pestaña de “Acerca de” podemos consultar los datos más importantes del programa, versión, licencia y desarrolladores.

Diagrama

Descripción generada automáticamente con confianza media

Aquí podemos ver el contenido de la pestaña, una vez hayamos consultado la información hacemos click sobre “Salir”.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

También tenemos la posibilidad desde el menú de operaciones consultar y crear las recepciones del almacén. Para ello hacemos click sobre “Recepciones”. Ahora podemos ver una tabla con el registro de todas las recepciones y también botón para crear una nueva recepción.

Tabla

Descripción generada automáticamente

Para crear una nueva recepción pulsamos sobre “Crear nuevo” y aquí se nos abre una nueva ventana donde le indicamos los datos que nos solicita y por último hacemos click en “Aceptar”.

Interfaz de usuario gráfica

Descripción generada automáticamente

Sí queremos consultar los detalles de una Recepción, hacemos click sobre esa recepción y después en “Ver Detalles”.

Tabla

Descripción generada automáticamente

Si queremos consultar una recepción en el cuadro de búsqueda *RIC*, introducimos el número de RIC que deseemos buscar.

Aplicación

Descripción generada automáticamente con confianza media

# Evaluación y Análisis

## CLASES POJO

En todas las clases POJO hemos utilizado la notación @Data, esta notación es utilizada para que cree de forma automática @toString, @Getter and @Setter, @EqualsAndHashCode and @RequiredArgsConstructor.

**CLASE Proveedor(Supplier),** hemos definido todos los atributos y métodos que contiene está clase para poder realizar las operaciones necesarias con estos datos.

El SIC es un código de identificación único generado aleatoriamente del proveedor.

Hacemos uso de dos constructores uno de ellos contiene el SIC, este constructor lo utilizamos cuando realizamos operaciones de insert, update, search … Hacemos uso de un Segundo constructor donde lo utilizamos en los test, en esta parte no es necesario conocer el SIC ya que cambia en cada test que realizamos.

@Data  
public class Supplier {  
 private String SIC = UUID.*randomUUID*().toString();  
 private StringProperty nameSupplier;  
 private StringProperty direction;  
 private StringProperty telephoneNumber;  
 private StringProperty email;  
  
  
 public Supplier(String nameSupplier, String direction, String telephonNumber, String email) {  
 this.SIC = SIC;  
 this.nameSupplier = new SimpleStringProperty(nameSupplier);  
 this.direction = new SimpleStringProperty(direction);  
 this.telephoneNumber = new SimpleStringProperty(telephonNumber);  
 this.email = new SimpleStringProperty(email);  
 }  
  
 *//En este constructor se agrega el SIC*  
public Supplier(String SIC, String nameSupplier, String direction, String telephonNumber, String email) {  
 this.SIC = SIC;  
 this.nameSupplier = new SimpleStringProperty(nameSupplier);  
 this.direction = new SimpleStringProperty(direction);  
 this.telephoneNumber = new SimpleStringProperty(telephonNumber);  
 this.email = new SimpleStringProperty(email);  
 }

**CLASE Artículo/Producto (Article),** en esta clase hemos definido todos los atributos necesarios para gestionar los artículos que tenemos en el almacén, también incluye los métodos necesarios para realizar operaciones con ellos.

El PIC es un número de identificación única que se asigna a cada artículo.

Hemos creado dos constructores, el primero de ellos contiene el PIC, de esta forma podemos hacer un update, insert, delete … ya que se crean automáticamente. El otro constructor no contiene el PIC para poder realizar el test, cuando realizamos el test no es necesario utilizar el PIC ya que va cambiando en cada test que hacemos, ya que son independientes unos de otros.

También tenemos un método para comprobar sí un producto está activo o no es un método boolean, siempre que el stock sea superior a 0 el producto estará activo.

Hemos implementado un toString adicional para poder insertar el separador y leer/escribir de un archivo CSV los artículos del almacén.

@Data  
public class Article {  
 private String PIC=UUID.*randomUUID*().toString();  
 private StringProperty article;  
 private StringProperty description;  
 private StringProperty location;  
 private DoubleProperty price;  
 private IntegerProperty stock;  
 private BooleanProperty isActive;  
 private StringProperty photo;  
 public Article(){  
  
 }  
  
 public Article(String article, String description, String location,  
 Double price, Integer stock, Boolean isActive,String photo) {  
 this.article =new SimpleStringProperty(article);  
 this.description = new SimpleStringProperty(description);  
 this.location = new SimpleStringProperty(location);  
 this.price = new SimpleDoubleProperty(price) ;  
 this.stock = new SimpleIntegerProperty(stock);  
 this.isActive = new SimpleBooleanProperty(isActive);  
 this.photo=new SimpleStringProperty(photo);  
 isActive();  
 }  
 public Article(String PIC ,String article, String description, String location,  
 Double price, Integer stock, Boolean isActive, String photo) {  
 this.PIC = PIC;  
 this.article =new SimpleStringProperty(article);  
 this.description = new SimpleStringProperty(description);  
 this.location = new SimpleStringProperty(location);  
 this.price = new SimpleDoubleProperty(price) ;  
 this.stock = new SimpleIntegerProperty(stock);  
 this.isActive = new SimpleBooleanProperty(isActive);  
 this.photo=new SimpleStringProperty(photo);  
 isActive();  
 }

public boolean isActive() {  
 this.isActive = new SimpleBooleanProperty(stock.get()>0);  
 return stock.get()>0;  
}

public String toString(String separator){  
 return PIC + separator +article.get()+separator+description.get()+separator+location.get()  
 +separator+price.get()+separator+stock.get()+separator+isActive.get()+separator+photo.get();  
}

**CLASE Copia De Seguridad(Backup),** este clase la hemos utilizado para realizar la copia de seguridad completa del programa.

Creamos una lista por cada Clase que usamos en nuestro programa, son clases DTO para poder transformarlas a clases POJO planas, es decir que no contienen observadores y poder almacenar los datos desde esta clase. Es decir que es una clase intermediaria. Creamos un constructor con todas las listas y poder insertarlas en uno/varios controladores. Además de crear los getter and setter de cada lista.

public class Backup {

private List<ArticleDTO> productos;

private List<CustomerDTO> customers;

private List<EmployeeDTO> empleados;

private List<LineOrderDTO> lineaPedidos;

private List<LineReceptionDTO> lineaReceptions;

private List<OrderDTO> pedidos;

private List<ReceptionDTO> reception;

private List<SupplierDTO> proveedores;

public Backup(List<ArticleDTO> productos, List<CustomerDTO> customers,List<EmployeeDTO> empleados,List<LineOrderDTO> lineaPedidos,

List<LineReceptionDTO> lineaReceptions,List<OrderDTO> pedidos,List<ReceptionDTO> reception,List<SupplierDTO> proveedores) {

this.productos = productos;

this.customers = customers;

this.empleados = empleados;

this.lineaPedidos = lineaPedidos;

this.lineaReceptions = lineaReceptions;

this.pedidos = pedidos;

this.reception = reception;

this.proveedores = proveedores;

}

public List<ArticleDTO> getProductos() {

return productos;

}

public List<CustomerDTO> getCustomers() {

return customers;

}

public List<EmployeeDTO> getEmpleados() {

return empleados;

}

public List<LineOrderDTO> getLineaPedidos() {

return lineaPedidos;

}

public List<LineReceptionDTO> getLineaReceptions() {

return lineaReceptions;

}

public List<OrderDTO> getPedidos() {

return pedidos;

}

public List<ReceptionDTO> getReception() {

return reception;

}

public List<SupplierDTO> getProveedores() {

return proveedores;

}

**Clase Carrito Compra (Carrito Item),** hemos creado esta clase para usar desde el cliente y almacenar los productos que va agregando al carrito. Con los atributos y métodos necesarios para poder operar con esta clase.

Contiene el método setAmount, para modificar el precio y cantidad de una línea del carrito y el método getTotal para obtener el total de ese pedido.

@Data  
public class CarritoItem {  
 private final SimpleStringProperty name;  
 private final SimpleStringProperty photo;  
 private final SimpleDoubleProperty price;  
 private SimpleIntegerProperty amount;  
 private double total;  
  
 public CarritoItem(String nombre, String imagen, double precio, int cantidad){  
 this.name = new SimpleStringProperty(nombre);  
 this.photo = new SimpleStringProperty(imagen);  
 this.price = new SimpleDoubleProperty(precio);  
 this.amount = new SimpleIntegerProperty(cantidad);  
 this.total = precio \* cantidad;  
 }

public void setAmount(int amount) {  
 this.amount.set(amount);  
 this.total = price.get() \* amount;  
}  
  
public double getTotal() {  
 return total;  
}

**Clase Cliente (Customer),** esta clase esta destinada para registrar e identificar a cada cliente del sistema, contiene los atributos necesarios que debe tener un cliente que se registre en el sistema y los métodos necesarios para operar con ella. También contiene los métodos Getter and Setter de la clase.

Hemos creado dos constructores el primero de ellos no contiene el CIC que es el código de identificación único de cada cliente, este constructor lo hemos utilizado para poder testear ya que cuando insertamos un cliente de prueba en un test no usamos el CIC ya que en cada test va a cambiar.

@Data  
public class Customer {  
 private String CIC = UUID.*randomUUID*().toString();  
 private StringProperty name;  
 private StringProperty surname;  
 private StringProperty cif;  
 private StringProperty direction;  
 private StringProperty nickName;  
 private StringProperty password;  
 private StringProperty telephoneNumber;  
 private StringProperty email;  
 private StringProperty photo;  
 private ObjectProperty<LocalDateTime> createdAt;  
 private BooleanProperty isActive;  
  
 public Customer(String name, String surname, String cif, String direction, String nickName, String password, String telephoneNumber, String email, String photo, LocalDateTime createdAt, Boolean isActive) {  
 this.name = new SimpleStringProperty(name);  
 this.surname = new SimpleStringProperty(surname);  
 this.cif = new SimpleStringProperty(cif);  
 this.direction = new SimpleStringProperty(direction);  
 this.nickName = new SimpleStringProperty(nickName);  
 this.password = new SimpleStringProperty(password);  
 this.telephoneNumber = new SimpleStringProperty(telephoneNumber);  
 this.email = new SimpleStringProperty(email);  
 this.photo = new SimpleStringProperty(photo);  
 this.createdAt = new SimpleObjectProperty<LocalDateTime>(createdAt);  
 this.isActive= new SimpleBooleanProperty(isActive);  
  
 }  
  
 public Customer(String cic, String name, String surname, String cif, String direction, String nickName, String password, String telephoneNumber, String email, String photo, LocalDateTime createdAt, Boolean isActive) {  
 this.CIC = cic;  
 this.name = new SimpleStringProperty(name);  
 this.surname = new SimpleStringProperty(surname);  
 this.cif = new SimpleStringProperty(cif);  
 this.direction = new SimpleStringProperty(direction);  
 this.nickName = new SimpleStringProperty(nickName);  
 this.password = new SimpleStringProperty(password);  
 this.telephoneNumber = new SimpleStringProperty(telephoneNumber);  
 this.email = new SimpleStringProperty(email);  
 this.photo = new SimpleStringProperty(photo);  
 this.createdAt = new SimpleObjectProperty<LocalDateTime>(createdAt);  
 this.isActive= new SimpleBooleanProperty(isActive);  
 }

**Clase Empleado (Employee),** esta clase está destinada para realizar las operaciones necesarias con los atributos que representan los datos de los empleados. Los atributos creados corresponden a la información personal de cada empleado.

Contiene dos constructores en uno de ellos incluye el EIC, es el codigo de identificación del empleado, este constructor se usa para realizar las operaciones de insert, update, findAll. El segundo constructor no contiene el EIC, este ha sido utilizado para realizar los test ya que este atributo no es necesario porque su valor varía en cada test.

@Data  
public class Employee {  
 private String EIC = UUID.*randomUUID*().toString();  
 private StringProperty name;  
 private StringProperty surname;  
 private StringProperty nif;  
 private StringProperty email;  
 private StringProperty photo;  
  
 private StringProperty nickName;  
 private StringProperty password;  
 private BooleanProperty isManager;  
 private ObjectProperty<LocalDateTime> createdAt;  
 private BooleanProperty isActive;  
  
 public Employee(String name, String surname, String nif, String email, String photo, String nickName, String password, boolean isManager, LocalDateTime createdAt,Boolean isActive) {  
 this.name = new SimpleStringProperty(name);  
 this.surname = new SimpleStringProperty(surname);  
 this.nif = new SimpleStringProperty(nif);  
 this.email = new SimpleStringProperty(email);  
 this.nickName = new SimpleStringProperty(nickName);  
 this.password = new SimpleStringProperty(password);  
 this.photo = new SimpleStringProperty(photo);  
 this.isManager = new SimpleBooleanProperty(isManager);  
 this.createdAt = new SimpleObjectProperty<LocalDateTime>(createdAt);  
 this.isActive= new SimpleBooleanProperty(isActive);  
 }  
  
 public Employee(String eic, String name, String surname, String nif, String email, String photo, String nickName, String password, boolean isManager, LocalDateTime createdAt, Boolean isActive) {  
 this.EIC = eic;  
 this.name = new SimpleStringProperty(name);  
 this.surname = new SimpleStringProperty(surname);  
 this.nif = new SimpleStringProperty(nif);  
 this.email = new SimpleStringProperty(email);  
 this.photo = new SimpleStringProperty(photo);  
 this.nickName = new SimpleStringProperty(nickName);  
 this.password = new SimpleStringProperty(password);  
 this.isManager = new SimpleBooleanProperty(isManager);  
 this.createdAt = new SimpleObjectProperty<LocalDateTime>(createdAt);  
 this.isActive= new SimpleBooleanProperty(isActive);  
 }  
 public Employee(){};

**Clase Linea Pedido (LineOrder),** esta clase se utiliza para crear cada línea de pedido que está asociada a un pedido, cada línea de pedido contiene un producto y su cantidad.

Hemos utilizado la notación de Lombok @AllArgsConstructor para crear un constructor con todos los parámetros, a parte hemos creado esté constructor de forma manual, debido a que algunas notaciones de Lombok nos generaban otros problemas en el proyecto.

@Data  
@AllArgsConstructor  
public class LineOrder {  
 private String OLIC = UUID.*randomUUID*().toString();  
 private StringProperty article;  
 private IntegerProperty load;  
 private DoubleProperty unitPrice;  
 private DoubleProperty totalPrice;  
 private StringProperty belongsOrder;  
  
 public LineOrder(String OLIC, String article, Integer load, Double unitPrice, String belongsOrder) {  
 this.OLIC = OLIC;  
 this.article = new SimpleStringProperty(article);  
 this.load = new SimpleIntegerProperty(load);  
 this.unitPrice = new SimpleDoubleProperty(unitPrice);  
 this.totalPrice = new SimpleDoubleProperty(unitPrice \* load);  
 this.belongsOrder = new SimpleStringProperty(belongsOrder);  
 }  
  
 public void setLoad(int load) {  
 this.load.set(load);  
 }  
  
 public void setUnitPrice(double unitPrice) {  
 this.unitPrice.set(unitPrice);  
 this.totalPrice = new SimpleDoubleProperty((this.load.intValue()) \* (this.unitPrice.doubleValue()));  
 }  
  
 public DoubleProperty totalPriceProperty() {  
 return totalPrice;  
 }  
  
 @Override  
 public String toString() {  
 return "LineOrder{" +  
 "OLIC=" + OLIC +  
 ", article=" + article +  
 ", load=" + load +  
 ", unitPrice=" + unitPrice +  
 ", totalPrice=" + totalPrice +  
 ", belongsOrder=" + belongsOrder +  
 '}';  
 }  
}

**Clase Linea Recepción (LineReception),** esta clase esta creada para tener los datos que son necesarios para crear una línea de recpcion que están asociadas a una recepción, una línea de recepción contiene el producto y cantidad que se va a depositar en nuestro almacen.

Hemos creado dos constructores uno de ellos contiene el RLIC que lo usamos en el programa cuando realizamos un insert, update, findaAll. El otro constructor que hemos utilizado no contiene el RLIC es usado en los test de esta clase ya que en cada test el RLIC es variable. Hemos usado la notación de Lombok @AllArgsConstructor para crear un constructor completo, nos ha dado algún problema finalmente no lo hemos utilizado.

@Data  
@AllArgsConstructor  
public class LineReception {  
 private String RLIC = UUID.*randomUUID*().toString();  
 private StringProperty articlePIC;  
 private IntegerProperty load;  
 private DoubleProperty unitPrice;  
 private DoubleProperty totalPrice;  
 private StringProperty belongsRecepcion;  
  
 public LineReception() {  
 }  
  
 public LineReception(String articlePIC, Integer load, Double unitPrice, String belongsRecepcion) {  
 this.articlePIC = new SimpleStringProperty(articlePIC);  
 this.load = new SimpleIntegerProperty(load);  
 this.unitPrice = new SimpleDoubleProperty(unitPrice);  
 this.totalPrice = new SimpleDoubleProperty(unitPrice \* load);  
 this.belongsRecepcion = new SimpleStringProperty(belongsRecepcion);  
 }  
  
 public LineReception(String rlic, String articlePIC, Integer load, Double unitPrice, String belongsRecepcion) {  
 this.RLIC = rlic;  
 this.articlePIC = new SimpleStringProperty(articlePIC);  
 this.load = new SimpleIntegerProperty(load);  
 this.unitPrice = new SimpleDoubleProperty(unitPrice);  
 this.totalPrice = new SimpleDoubleProperty(unitPrice \* load);  
 this.belongsRecepcion = new SimpleStringProperty(belongsRecepcion);  
 }

**Clase Pedido (Order),** esta clase esta creada para almacenar las líneas de pedido que realiza un cliente, también contiene el precio total del pedido.

Hemos creado dos constructores uno de ellos no contiene el OIC que es el identificador de cada pedido que lo utilizamos en los test ya que cada vez que realizamos un test el OIC cambia y por lo tanto no se puede utilizar. El otro constructor contiene el OIC que lo utilizamos cada vez que hacemos un insert, update, search by OIC …

@Data  
public class Order {  
 private String OIC= UUID.*randomUUID*().toString(); ;  
 private StringProperty customer;  
 private DoubleProperty price;  
 private ObjectProperty<Pay> methodPay;  
  
  
 public Order(String customer, double price, Pay methodPay) {  
 this.customer = new SimpleStringProperty(customer);  
 this.price = new SimpleDoubleProperty(price);  
 this.methodPay = new SimpleObjectProperty(methodPay);  
 }  
  
 public Order(String OIC, String customer, double price, Pay methodPay) {  
 this.OIC=OIC;  
 this.customer = new SimpleStringProperty(customer);  
 this.price = new SimpleDoubleProperty(price);  
 this.methodPay = new SimpleObjectProperty(methodPay);  
 }

**Enum Pago (Pay),** hemos creado este enum para introducir las diferentes formas de pago que se pueden realizar en el programa.

public enum Pay {  
 PAYPAL, CARD  
}

**Clase Recepción (Reception),** esta clase la hemos creado para almacenar todas las líneas de recepción que contiene una misma recepción y también el coste total que tiene una recepción.

Hemos creado dos constructores uno que está vacío y otro con el que podemos realizar las operaciones de insert, update, findAll, findByRIC …

@Data  
public class Reception {  
 private String RIC = UUID.*randomUUID*().toString();  
 private StringProperty supplierName;  
 private StringProperty carrier;  
 private DoubleProperty cost;  
  
 public Reception() {  
 }  
  
 public Reception(String ric, String supplierName, String carrier, Double cost) {  
 this.RIC = ric;  
 this.supplierName = new SimpleStringProperty(supplierName);  
 this.carrier = new SimpleStringProperty(carrier);  
 this.cost = new SimpleDoubleProperty(cost);  
 }  
  
 public Reception(String supplierName, String carrier, Double cost) {  
 this.supplierName = new SimpleStringProperty(supplierName);  
 this.carrier = new SimpleStringProperty(carrier);  
 this.cost = new SimpleDoubleProperty(cost);  
 }

## REPOSITORY

Para crear los repositorios hemos aplicado los principios SOLID, el primero de ellos es el Principio de Responsabilidad única es decir que cada clase solo realice una cosa teniendo un bajo acoplamiento y una alta cohesión. El repositorio de Artículos solo va a realizar las operaciones CRUD sobre Artículos en ningún momento lo realizara sobre Empleados. El segundo principio SOLID con el que cumple el programa es el de Abierto/Cerrado, es decir abierto a extensión y cerrado a modificación, el repositorio de artículos podrá tener un elemento de búsqueda nuevo sin tener que modificar los anteriores. El tercer principio SOLID que cumple el programa es el principio de Segregación de Interfaces, cada interfaz tiene una única responsabilidad, por lo tanto cada repositorio de nuestro proyecto tiene su propia interfaz que extiende de una interfaz genérica con los métodos comunes a todos, en cada interfaz de cada repositorio tendrá métodos extra que otros no tienen.

La interfaz CRUDRepository contiene los métodos comunes que usan todos los repositorios que implementa

public interface *CRUDRepository*<T, ID> {  
  
 *ObservableList*<T> findAll() throws SQLException;  
  
 Optional<T> save(T entity) throws SQLException;  
  
 Optional<T> update(ID id, T entity) throws SQLException;  
}

**Repositorio Artículos (Articles),** tiene implementada la interfaz IArticleRepository que esta a su vez extiende de CRUDRepository. Creamos una instancia de ArticleRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de Articles.

Completamos los métodos de búsqueda por Nombre, UUID, save and update. En el caso de findByName y findByPIC hemos utilizado la API STREAM para realizar la búsqueda de datos.

public class ArticleRepository implements *ArticleInterface* {  
 private static ArticleRepository *instance*;  
 private final DataBaseManager dataBaseManager;  
 private final *ObservableList*<Article> repository = FXCollections.*observableArrayList*();  
  
 private ArticleRepository(DataBaseManager databaseManager) {  
 this.dataBaseManager = databaseManager;  
 }  
  
 public static ArticleRepository getInstance(DataBaseManager dataBaseManager) {  
 if (*instance* == null) {  
 *instance* = new ArticleRepository(dataBaseManager);  
 }  
 return *instance*;  
 }  
  
 public DataBaseManager getDb() {  
 return dataBaseManager;  
 }  
  
 @Override  
 public Article findByName(String name) throws SQLException {  
 var repo = findAll();  
 return repo.stream().filter(article -> article.getArticle().get().equals(name)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
  
 @Override  
 public Article findByUuid(String PIC) throws SQLException {  
 var repo = findAll();  
 return repo.stream().filter(article -> article.getPIC().equals(PIC)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
  
 @Override  
 public *ObservableList*<Article> findAll() throws SQLException {  
 dataBaseManager.open();  
 String querry = "Select \* from Article";  
 *ResultSet* result = dataBaseManager.select(querry).orElseThrow(() -> new SQLException("Error al obtener los articulos."));  
 repository.clear();  
 while (result.next()) {  
 repository.add(  
 new Article(  
 result.getString("PIC"),  
 result.getString("article"),  
 result.getString("description"),  
 result.getString("location"),  
 result.getDouble("price"),  
 result.getInt("stock"),  
 result.getBoolean("isActive"),  
 result.getString("photo")  
  
 )  
 );  
 }  
 dataBaseManager.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<Article> save(Article article) throws SQLException {  
 dataBaseManager.open();  
 String query = " insert into Article (PIC, article, description, location, stock, price, isActive, photo) " +  
 "values(?, ?, ?, ?, ?, ?,?, ?) ;";  
 dataBaseManager.insert(query, article.getPIC(),  
 article.getArticle().get(), article.getDescription().get(),  
 article.getLocation().get(), article.getStock().get(), article.getPrice().get(),  
 article.getIsActive().get(), article.getPhoto());  
 dataBaseManager.close();  
 return Optional.*of*(article);  
 }  
  
 @Override  
 public Optional<Article> update(String pic, Article article) throws SQLException {  
 var a = findByUuid(pic);  
 var index = repository.indexOf(a);  
  
 dataBaseManager.open();  
 String query = " update Article set article = ?, description= ?, location=?, stock=?, price=?," +  
 " isActive = ?, photo = ? where PIC = ? ;";  
 dataBaseManager.update(query, article.getArticle().get(), article.getDescription().get(),  
 article.getLocation().get(), article.getStock().get(), article.getPrice().get(), article.getIsActive().get()  
 , article.getPhoto(), pic);  
 dataBaseManager.close();  
 repository.set(index, article);  
 return Optional.*of*(article);  
 }  
}

**Repositorio Clientes (CustomerRepository),** tiene implementada la interfaz ICostumerRepository que esta a su vez extiende de CRUDRepository. Creamos una instancia de CostumerRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de Cpstumer.

Completamos los métodos de búsqueda por UUID, save and update. En el caso de findByCIC hemos utilizado la API STREAM para realizar la búsqueda de datos.

public class CustomerRepository implements *ICustomerRepository* {  
 private static CustomerRepository *instance*;  
 private final *ObservableList*<Customer> repository = FXCollections.*observableArrayList*();  
 private final DataBaseManager db;  
  
 private CustomerRepository(DataBaseManager db) {  
 this.db = db;  
 }  
  
 public static CustomerRepository getInstance(DataBaseManager db) {  
 if (*instance* == null) {  
 *instance* = new CustomerRepository(db);  
 }  
 return *instance*;  
 }  
  
 public DataBaseManager getDb() {  
 return db;  
 }  
  
 @Override  
 public *ObservableList*<Customer> findAll() throws SQLException {  
 String sql = "SELECT \* FROM Customer";  
 db.open();  
 *ResultSet* resultado = db.select(sql).orElseThrow(() -> new SQLException("Error al obtener todos los clientes."));  
 repository.clear();  
 while (resultado.next()) {  
 repository.add(  
 new Customer(  
 resultado.getString("CIC"),  
 resultado.getString("name"),  
 resultado.getString("surname"),  
 resultado.getString("cif"),  
 resultado.getString("direction"),  
 resultado.getString("nickname"),  
 resultado.getString("password"),  
 resultado.getString("telephoneNumber"),  
 resultado.getString("email"),  
 resultado.getString("photo"),  
 LocalDateTime.*parse*(resultado.getString("createdAt")),  
 resultado.getBoolean("isActive")  
  
 )  
 );  
 }  
 db.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<Customer> save(Customer customer) throws SQLException {  
 String sql = "INSERT INTO Customer (CIC,name,surname,cif,direction,nickname,password,telephoneNumber,email,photo,createdAt,isActive) VALUES (?,?,?,?,?,?,?,?,?,?,?,?)";  
 db.open();  
 db.insert(sql, customer.getCIC(), customer.getName(), customer.getSurname(), customer.getCif(), customer.getDirection(), customer.getNickName(), customer.getPassword(), customer.getTelephoneNumber(), customer.getEmail(), customer.getPhoto(), customer.getCreatedAt().toString(),customer.isActive());  
 db.close();  
 return Optional.*of*(customer);  
 }  
  
 @Override  
 public Optional<Customer> update(UUID uuid, Customer customer) throws SQLException {  
 var c = findByUUID(uuid.toString());  
  
 var index = repository.indexOf(c);  
  
 String sql = "UPDATE Customer SET name = ?, surname = ?, cif = ?, direction = ?, nickname= ?, password= ?, telephoneNumber= ?, email = ?, photo = ?, createdAt = ?, isActive= ? WHERE CIC = ?";  
 db.open();  
 db.update(sql, customer.getName(), customer.getSurname(), customer.getCif(), customer.getDirection(), customer.getNickName(), customer.getPassword(), customer.getTelephoneNumber(), customer.getEmail(), customer.getPhoto(), customer.getCreatedAt(),customer.isActive(), customer.getCIC());  
 db.close();  
 repository.set(index, customer);  
  
 return Optional.*of*(customer);  
 }  
   
 @Override  
 public Customer findByUUID(String uuid) throws SQLException {  
 var repo = findAll();  
 return repo.stream().filter(customer -> customer.getCIC().equals(uuid)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
}

**Repositorio Empleados (EmployeeRepository),** tiene implementada la interfaz IEmployeeRepository que esta a su vez extiende de CRUDRepository. Creamos una instancia de EmployeeRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de Employee.

Completamos los métodos de búsqueda por UUID, save and update. En el caso de findByEIC hemos utilizado la API STREAM para realizar la búsqueda de datos.

public class EmployeeRepository implements *IEmployeeRepository* {  
 private static EmployeeRepository *instance*;  
 private final *ObservableList*<Employee> repository = FXCollections.*observableArrayList*();  
 private final DataBaseManager db;  
  
 private EmployeeRepository(DataBaseManager db) {  
 this.db = db;  
 }  
  
 public static EmployeeRepository getInstance(DataBaseManager db) {  
 if (*instance* == null) {  
 *instance* = new EmployeeRepository(db);  
 }  
 return *instance*;  
 }  
  
 public DataBaseManager getDb() {  
 return db;  
 }  
  
 @Override  
 public *ObservableList*<Employee> findAll() throws SQLException {  
 String sql = "SELECT \* FROM Employee";  
 db.open();  
 *ResultSet* resultado = db.select(sql).orElseThrow(() -> new SQLException("Error al obtener todos los empleados."));  
 repository.clear();  
 while (resultado.next()) {  
 repository.add(  
 new Employee(  
 resultado.getString("EIC"),  
 resultado.getString("name"),  
 resultado.getString("surname"),  
 resultado.getString("nif"),  
 resultado.getString("email"),  
 resultado.getString("photo"),  
 resultado.getString("nickname"),  
 resultado.getString("password"),  
 resultado.getBoolean("isManager"),  
 LocalDateTime.*parse*(resultado.getString("createdAt")),  
 resultado.getBoolean("isActive")  
  
 )  
 );  
 }  
 db.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<Employee> save(Employee employee) throws SQLException {  
 String sql = "INSERT INTO Employee (EIC,name,surname,nif,email,photo,nickname,password,isManager,createdAt,isActive) VALUES (?,?,?,?,?,?,?,?,?,?,?)";  
 db.open();  
 db.insert(sql, employee.getEIC(), employee.getName(), employee.getSurname(), employee.getNif(), employee.getEmail(), employee.getPhoto(), employee.getNickName(), employee.getPassword(), employee.isManager(), employee.getCreatedAt().toString(),employee.isActive());  
 db.close();  
 return Optional.*of*(employee);  
 }  
  
 @Override  
 public Optional<Employee> update(UUID uuid, Employee employee) throws SQLException {  
 var c = findByUUID(uuid.toString());  
 var index = repository.indexOf(c);  
 String sql = "UPDATE Employee SET name = ?, surname = ?, nif = ?, email = ?, photo = ?, nickname= ?, password= ?, isManager= ?, createdAt = ?, isActive= ? WHERE EIC = ?";  
 db.open();  
 db.update(sql, employee.getName(), employee.getSurname(), employee.getNif(), employee.getEmail(), employee.getPhoto(), employee.getNickName(), employee.getPassword(), employee.isManager(), employee.getCreatedAt().toString(),employee.isActive() , employee.getEIC());  
 db.close();  
 repository.set(index, employee);  
  
 return Optional.*of*(employee);  
 }  
  
 @Override  
 public Employee findByUUID(String uuid) throws SQLException {  
 var repo = findAll();  
 return repo.stream().filter(employee -> employee.getEIC().equals(uuid)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
}

**Repositorio Linea Pedido (LineOrderRepository),** tiene implementada la interfaz LineOrderInterface que esta a su vez extiende de CRUDRepository. Creamos una instancia de LineOrderRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de LineOrder.

Completamos los métodos de búsqueda por UUID, save and update y el método de búsqueda del UUID de un Pedido.

public class LineOrderRepository implements *LineOrderInterface*<LineOrder,String> {  
 private static LineOrderRepository *instance*;  
 private final DataBaseManager dataBaseManager;  
 private final *ObservableList*<LineOrder> repository = FXCollections.*observableArrayList*();  
 private LineOrderRepository(DataBaseManager dataBaseManager) {  
 this.dataBaseManager = dataBaseManager;  
 }  
 public DataBaseManager getDb(){  
 return dataBaseManager;  
 }  
 public static LineOrderRepository getInstance(DataBaseManager dataBaseManager){  
 if(*instance* ==null){  
 *instance* = new LineOrderRepository(dataBaseManager);  
 }  
 return *instance*;  
 }  
  
 @Override  
 public *ObservableList*<LineOrder> findAll() throws SQLException {  
 dataBaseManager.open();  
 String query = "select \* from LineOrder";  
 *ResultSet* result = dataBaseManager.select(query).orElseThrow(SQLException::new);  
 while (result.next()){  
 String OLIC= (result.getString("OLIC"));  
 StringProperty articlePIC = new SimpleStringProperty(result.getString("article"));  
 IntegerProperty load = new SimpleIntegerProperty(result.getInt("load"));  
 DoubleProperty unitPrice = new SimpleDoubleProperty(result.getDouble("unitPrice"));;  
 DoubleProperty totalPrice = new SimpleDoubleProperty(result.getDouble("totalPrice"));;  
 StringProperty belongsOrder = new SimpleStringProperty(result.getString("belongsOrder"));  
 LineOrder lineOrder = new LineOrder(OLIC, articlePIC, load, unitPrice, totalPrice, belongsOrder);  
 repository.add(lineOrder);  
 }  
 dataBaseManager.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<LineOrder> save(LineOrder lineOrder) throws SQLException {  
 dataBaseManager.open();  
 String query = "Insert into LineOrder (OLIC, article, load, unitPrice, totalPrice, BelongsOrder) " +  
 "values (?, ?, ?, ?, ?, ?);";  
 *ResultSet* resultado = dataBaseManager.insert(query,  
 lineOrder.getOLIC(),  
 lineOrder.getArticle().get(),  
 lineOrder.getLoad().get(),  
 lineOrder.getUnitPrice().get(),  
 lineOrder.getTotalPrice().get(),  
 lineOrder.getBelongsOrder().get()  
 ).orElseThrow(()-> new SQLException("Error al obtener las lineas."));  
 dataBaseManager.close();  
 return Optional.*of*(lineOrder) ;  
 }  
  
 @Override  
 public Optional<LineOrder> update(String olic, LineOrder lineOrder) throws SQLException {  
 var line= findByUuid(olic);  
 int index= repository.indexOf(line);  
 dataBaseManager.open();  
 String query = "Update LineOrder set article = ? , load = ? " +  
 ", unitPrice = ?, totalPrice = ?, BelongsOrder = ? where OLIC = ? ;";  
 dataBaseManager.update(query,  
 lineOrder.getArticle().get(),  
 lineOrder.getLoad().get(),  
 lineOrder.getUnitPrice().get(),  
 lineOrder.getTotalPrice().get(),  
 lineOrder.getBelongsOrder().get(),  
 olic);  
 dataBaseManager.close();  
 repository.set(index,lineOrder);  
 return Optional.*of*(lineOrder) ;  
 }  
  
 @Override  
 public LineOrder findByUuid(String uuid) throws SQLException {  
 var repo= findAll();  
 return repo.stream().filter(lineOrder-> lineOrder.getOLIC().equals(uuid)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
  
 @Override  
 public *ObservableList*<LineOrder> searchByUuidOrder(String identifier) throws SQLException {  
 dataBaseManager.open();  
 String query = "select \* from LineOrder where belongsOrder = ?";  
 *ResultSet* result = dataBaseManager.select(query, identifier).orElseThrow(()-> new SQLException("No existe lineas con el identificador."));  
 repository.clear();  
 while (result.next()){  
 String OLIC = result.getString("OLIC");  
 StringProperty articlePIC = new SimpleStringProperty(result.getString("article"));  
 IntegerProperty load = new SimpleIntegerProperty(result.getInt("load"));  
 DoubleProperty unitPrice = new SimpleDoubleProperty(result.getDouble("unitPrice"));;  
 DoubleProperty totalPrice = new SimpleDoubleProperty(result.getDouble("totalPrice"));;  
 StringProperty belongsOrder = new SimpleStringProperty(result.getString("belongsOrder"));  
 LineOrder lineOrder = new LineOrder(OLIC, articlePIC, load, unitPrice, totalPrice, belongsOrder);  
 repository.add(lineOrder);  
 }  
 dataBaseManager.close();  
 return repository;  
 }  
}

En este caso en la Interfaz LineOrderInterface hemos implementado otra Interfaz llamada SearchByOrderBelongs, a través de esta interfaz implementamos el método de buscar UUID de un Pedido.

***SearchByOrderBelongs***

public interface *SearchByOrderBelongs*<LineOrder,String> {  
 *ObservableList*<LineOrder> searchByUuidOrder(String identifier) throws SQLException;  
}

***LineOrderInterface***

public interface *LineOrderInterface*<LineOrder,String> extends *CRUDRepository*<LineOrder, java.lang.String>, *SearchByOrderBelongs*<LineOrder, java.lang.String>, *SearchByUuid*<LineOrder, java.lang.String> {  
}

**Repositorio Linea Recepcion (LineReceptionRepository),** tiene implementada la interfaz LineReceptionInterface que esta a su vez extiende de CRUDRepository. Creamos una instancia de LineReceptionRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de LineRepository.

Completamos los métodos de búsqueda por UUID, save and update y el método de búsqueda del UUID de una Recepcion.

public class LineReceptionRepository implements *LineReceptionInterface*{  
 private static LineReceptionRepository *instance*;  
 private final DataBaseManager dataBaseManager;  
 private *ObservableList*<LineReception> repository = FXCollections.*observableArrayList*();  
 private LineReceptionRepository(DataBaseManager dataBaseManager) {  
 this.dataBaseManager = dataBaseManager;  
 }  
 public DataBaseManager getDb(){  
 return dataBaseManager;  
 }  
 public static LineReceptionRepository getInstance(DataBaseManager dataBaseManager){  
 if(*instance* == null){  
 *instance* = new LineReceptionRepository(dataBaseManager);  
 }  
 return *instance*;  
 }  
  
 @Override  
 public *ObservableList*<LineReception> findAll() throws SQLException {  
 dataBaseManager.open();  
 String query = "select \* from LineReception";  
 *ResultSet* result = dataBaseManager.select(query).orElseThrow(SQLException::new);  
 repository.clear();  
 while (result.next()){  
 String RLIC = (result.getString("RLIC"));  
 StringProperty articlePIC = new SimpleStringProperty(result.getString("article"));  
 IntegerProperty load = new SimpleIntegerProperty(result.getInt("load"));  
 DoubleProperty unitPrice = new SimpleDoubleProperty(result.getDouble("unitPrice"));;  
 DoubleProperty totalPrice = new SimpleDoubleProperty(result.getDouble("totalPrice"));;  
 StringProperty belongsRecepcion = new SimpleStringProperty(result.getString("belongsReception"));  
  
 LineReception lineReception = new LineReception(RLIC, articlePIC, load, unitPrice, totalPrice, belongsRecepcion);  
 repository.add(lineReception);  
 }  
 dataBaseManager.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<LineReception> save(LineReception lineReception) throws SQLException {  
 dataBaseManager.open();  
 String query = "Insert into LineReception values (?, ?, ?, ?, ?, ?);";  
 *ResultSet* resultado = dataBaseManager.insert(query,  
 lineReception.getRLIC(),  
 lineReception.getArticlePIC().get(),  
 lineReception.getLoad().get(),  
 lineReception.getUnitPrice().get(),  
 lineReception.getTotalPrice().get(),  
 lineReception.getBelongsRecepcion().get()  
 ).orElseThrow(SQLException::new);  
 dataBaseManager.close();  
 return Optional.*of*(lineReception) ;  
 }  
  
 @Override  
 public Optional<LineReception> update(String rlic, LineReception lineReception) throws SQLException {  
 var o= findByUUID(rlic);  
 int index= repository.indexOf(o);  
 dataBaseManager.open();  
 String query = "Update LineReception set article = ? , load = ? " +  
 ", unitPrice = ?, totalPrice = ?, BelongsReception = ? where RLIC = ? ;";  
 dataBaseManager.update(query,  
 lineReception.getArticlePIC().get(),  
 lineReception.getLoad().get(),  
 lineReception.getUnitPrice().get(),  
 lineReception.getTotalPrice().get(),  
 lineReception.getBelongsRecepcion().get(),  
 rlic);  
 dataBaseManager.close();  
 repository.set(index,lineReception);  
  
 return Optional.*of*(lineReception);  
 }  
  
 @Override  
 public *ObservableList*<LineReception> SerachByReceptionsBelong(String identifier) throws SQLException {  
 repository.clear();  
 dataBaseManager.open();  
 String query = "select \* from LineReception where belongsReception = ?";  
 *ResultSet* result = dataBaseManager.select(query, identifier).orElseThrow(SQLException::new);  
 while (result.next()){  
 String RLIC = result.getString("RLIC");  
 StringProperty articlePIC = new SimpleStringProperty(result.getString("article"));  
 IntegerProperty load = new SimpleIntegerProperty(result.getInt("load"));  
 DoubleProperty unitPrice = new SimpleDoubleProperty(result.getDouble("unitPrice"));;  
 DoubleProperty totalPrice = new SimpleDoubleProperty(result.getDouble("totalPrice"));;  
 StringProperty belongsRecepcion = new SimpleStringProperty(result.getString("belongsReception"));  
  
 LineReception lineReception = new LineReception(RLIC, articlePIC, load, unitPrice, totalPrice, belongsRecepcion);  
 repository.add(lineReception);  
 }  
 dataBaseManager.close();  
 return repository;  
 }  
   
 public LineReception findByUUID(String rlic) throws SQLException {  
 var repo= findAll();  
 return repo.stream().filter(lineReception->lineReception.getRLIC().equals(rlic)).findFirst().orElseThrow(() -> new SQLException("No existe"));  
 }  
}

En este caso en la Interfaz LineReceptionInterface hemos implementado otra Interfaz llamada SearchByReceptionBelongs, a través de esta interfaz implementamos el método de buscar UUID de una Recepción.

***SearchByReceptionBelongs***

interface *SerachByReceptionsBelong*<T,ID> {  
 *ObservableList*<T> SerachByReceptionsBelong(ID identifier) throws SQLException;  
}

***LineReceptionInterface***

public interface *LineReceptionInterface* extends *CRUDRepository*<LineReception,String>, *SerachByReceptionsBelong*<LineReception,String> {  
}

**Repositorio Supplier (SupplierRepository),** tiene implementada la interfaz ICRUDSupplier que esta a su vez extiende de CRUDRepository. Creamos una instancia de SupplierRepository y llamamos a DataBaseManager, también creamos la lista de observadores del repositorio de Supplier.

Completamos los métodos de búsqueda por UUID, save and update. En el caso de findBySIC hemos utilizado la API STREAM para realizar la búsqueda de datos.

public class SupplierRepository implements *ICRUDSupplier* {  
 private static SupplierRepository *instance*;  
 private final *ObservableList*<Supplier> repository = FXCollections.*observableArrayList*();  
 private final DataBaseManager bbdd;  
  
 private SupplierRepository(DataBaseManager bbdd) {  
 this.bbdd = bbdd;  
 }  
  
 public static SupplierRepository getInstance(DataBaseManager bbdd) {  
 if (*instance* == null) {  
 *instance* = new SupplierRepository(bbdd);  
 }  
 return *instance*;  
 }  
  
 public DataBaseManager getBbdd() {  
 return bbdd;  
 }  
  
 @Override  
 public *ObservableList*<Supplier> findAll() throws SQLException {  
 String sql = "SELECT \* FROM Supplier";  
 bbdd.open();  
 *ResultSet* resultado = bbdd.select(sql).orElseThrow(() -> new SQLException("Se ha producido un error obteniendo" +  
 " los datos"));  
 repository.clear();  
 while (resultado.next()) {  
 repository.add(  
 new Supplier(  
 resultado.getString("SIC"),  
 resultado.getString("nameSupplier"),  
 resultado.getString("direction"),  
 resultado.getString("telephoneNumber"),  
 resultado.getString("email")  
 )  
 );  
 }  
 bbdd.close();  
 return repository;  
 }  
  
 @Override  
 public Optional<Supplier> save(Supplier supplier) throws SQLException {  
 String sql = "INSERT INTO Supplier (SIC, nameSupplier, direction, telephoneNumber, email) VALUES (?, ?, ?, ?, ?)";  
 bbdd.open();  
 System.out.println("bbdd abierta");  
 bbdd.insert(sql, supplier.getSIC(), supplier.getNameSupplier(), supplier.getDirection(), supplier.getTelephoneNumber(), supplier.getEmail());  
 bbdd.close();  
 return Optional.*of*(supplier);  
 }  
  
 @Override  
 public Optional<Supplier> update(String uuid, Supplier supplier) throws SQLException {  
 var c = findByUUID(uuid);  
 int index = repository.indexOf(c);  
 String sql = "UPDATE Supplier SET nameSupplier = ?, direction = ?, telephoneNumber = ?, email = ? WHERE SIC = ?";  
 bbdd.open();  
 bbdd.update(sql, supplier.getNameSupplier(), supplier.getDirection(), supplier.getTelephoneNumber(), supplier.getEmail(), uuid);  
 bbdd.close();  
 repository.set(index, supplier);  
 return Optional.*of*(supplier);  
 }  
  
 @Override  
 public Supplier findByUUID(String uuid) throws SQLException {  
 var resultado = findAll();  
 return resultado.stream().filter(supplier -> supplier.getSIC().equals(uuid)).findFirst().orElseThrow(  
 () -> new SQLException("No existe ningún proveedor con ese SIC"));  
 }  
}

**Repositorio Carrito (CarritoRepository),** este repositorio es usado en la interfaz de cliente, contiene los métodos necesarios para añadir, borrar, limpiar el carrito y obtener el total de una compra que realice un cliente, para ello implementa una lista de observadores y una instancia de CarritoRepository, en el caso de que sea null creará un nuevo CarritoRepository.

public class CarritoRepository {  
 private static CarritoRepository *instance*;  
  
 private final *ObservableList*<CarritoItem> items = FXCollections.*observableArrayList*();  
  
 private CarritoRepository() {  
 }  
  
 public static CarritoRepository getInstance() {  
 if (*instance* == null) {  
 *instance* = new CarritoRepository();  
 }  
 return *instance*;  
 }  
  
 public *ObservableList*<CarritoItem> getItems() {  
 return items;  
 }  
  
 public void addItem(CarritoItem item) {  
 items.add(item);  
 }  
  
 public void removeItem(CarritoItem item) {  
 items.remove(item);  
 }  
  
 public double getTotal() {  
 return items.stream().mapToDouble(CarritoItem::getTotal).sum();  
 }  
  
 public void clear() {  
 items.clear();  
 }  
}

## CONTROLADORES

Para un correcto funcionamiento de nuestras interfaces y una cohesión con los repositorios tenemos los Controladores. Estos implementan cada uno, uno o varios repositorios necesarios para el correcto funcionamiento de la interfaz de usuario, explicaremos a continuación la escena que controlan, los repositorios que implementan y un resumen de los métodos que contienen.

**Controlador: AcercaDeController controla la escena de: (AcercaDe.fx),** su función es abrir y cargar los datos presentados en la opción de IU Acerca de , no trabaja con ningún repositorio, se compone de un método para inicializarlo al implementar “ Initializable ” y otro para cerrarlo al seleccionar el botón correspondiente.

public class AcercaDeController {  
 private Stage dialogStage;  
 @FXML  
 private ImageView logoGadam;  
 @FXML  
 private Label titulo;  
 @FXML  
 private Label version;  
 @FXML  
 private Label licencia;  
 @FXML  
 private Label integrantes;  
 @FXML  
 private Label integrante1;  
 @FXML  
 private Label integrante2;  
 @FXML  
 private Label integrante3;  
 @FXML  
 private Label integrante4;  
 @FXML  
 private Label integrante5;  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
  
 @FXML  
 private void initialize() {  
 titulo.setText(Properties.APLICACION\_NAME);  
 version.setText("Versión: " + Properties.APLICACION\_VERSION);  
 licencia.setText(Properties.APLICACION\_LICENCIA);  
 }  
  
 @FXML  
 private void salirAction() {  
 dialogStage.close();  
 }  
}

**Controlador: SplashControler controla la escena de: (splash.fxml) ,** Realiza la función presentación de una pantalla de inicio a través del método Initialize, gracias a que implementa Initializable.

public class SplashController implements *Initializable* {  
 private Stage stage;  
 @FXML  
 private ImageView fondoSplash;  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
 FadeTransition transition = new FadeTransition(Duration.*millis*(3000), fondoSplash);  
 transition.setFromValue(1.0);  
 transition.setToValue(1.0);  
 transition.play();  
  
 transition.setOnFinished(event -> {  
 SceneManager sceneManager = SceneManager.*get*();  
 try {  
 sceneManager.initLogin(stage);  
  
 } catch (IOException e) {  
 System.out.println(e);  
 }  
 });  
 }  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
}

**Controlador: RegisterController controla la escena de: (register.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository),** tiene como opción registrar a los clientes comprobar los datos pasados a través de la interfaz y en el caso de ser correctos guardarlos en el repositorio.

public class RegisterController {  
 CustomerRepository repository = CustomerRepository.*getInstance*(DataBaseManager.*getInstance*());  
 EmployeeRepository employeeRepo = EmployeeRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 TextField txtName;  
 @FXML  
 TextField txtSurname;  
 @FXML  
 TextField txtCIF;  
 @FXML  
 TextField txtNick;  
 @FXML  
 TextField txtPassword;  
 @FXML  
 TextField txtPasswordConfir;  
 @FXML  
 TextField txtDirection;  
 @FXML  
 TextField txtPhone;  
 @FXML  
 TextField txtEmail;  
 @FXML  
 ImageView photoView;  
  
 @FXML  
 CheckBox checkAcept;  
 private Stage stage;  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 public void onPhotoClick(MouseEvent event) {  
 FileChooser filechooser = new FileChooser();  
 filechooser.setTitle("Indica la foto que desea.");  
 filechooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Imagenes", "\*.jpg", "\*.png"));  
 File file = filechooser.showOpenDialog(photoView.getScene().getWindow());  
  
 if (file != null) {  
 photoView.setImage(new Image(file.toURI().toString()));  
 }  
 }  
  
 public void btnSaveAction(ActionEvent event) {  
 if (checkAcept.isSelected()) {  
 if (isValid()) {  
 if (resumen()) {  
 System.out.println("Guardando nuevo usuario...");  
 saveCustomer();  
 stage.close();  
 }  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("");  
 alert.setHeaderText("No han sido aceptado los términos.");  
 alert.setContentText("Seleccione Aceptar los Términos.");  
 alert.showAndWait();  
  
 }  
  
 }  
  
 public boolean isValid() {  
 String errorMessage = "";  
  
 if (txtName.getText() == null || !Patterns.*patternName*(txtName.getText())) {  
 errorMessage += "El nombre no puede estar vacio o es incorrecto. Ejemplo: Jaime\n";  
 txtName.setText("");  
 }  
 if (txtSurname.getText() == null || !Patterns.*patternSurnames*(txtSurname.getText())) {  
 errorMessage += "El apellido no puede estar vacio o es incorrecto. Ejemplo: Santoyo Salazar\n";  
 txtSurname.setText("");  
 }  
 if (txtNick.getText() == null || txtNick.getText().length() == 0) {  
 errorMessage += "El nick no puede estar vacio o es incorrecto. Ejemplo: Darkness07\n";  
 txtNick.setText("");  
 }  
 if (txtCIF.getText() == null || !Patterns.*patternCif*(txtCIF.getText())) {  
 errorMessage += "El CIF no puede estar vacio o es incorrecto. Ejemplo: 12345678A\n";  
 txtCIF.setText("");  
 }  
 if (txtPassword.getText() == null || !Patterns.*patternPassword*(txtPassword.getText())) {  
 errorMessage += "La contraseña debe contener letras,números y símbolos. Además que debe incluir una letra en mayúscula.\n";  
 txtPassword.setText("");  
 }  
 if (!txtPasswordConfir.getText().equals(txtPassword.getText()) || txtPasswordConfir.getText() == null || txtPasswordConfir.getText().length() == 0) {  
 errorMessage += "La contraseña no coinciden.\n";  
 txtPasswordConfir.setText("");  
 }  
 if (txtDirection.getText() == null || txtDirection.getText().length() == 0) {  
 errorMessage += "La dirección no puede estar vacia.\n";  
 txtDirection.setText("");  
 }  
 if (txtPhone.getText() == null || !Patterns.*patterPhone*(txtPhone.getText())) {  
 errorMessage += "El campo de teléfono esta vacio o es incorrecto: Ejemplo: 610245879\n";  
 txtPhone.setText("");  
 }  
 if (txtEmail.getText() == null || !Patterns.*patternEmail*(txtEmail.getText())) {  
 errorMessage += "El correo no puede estar vacio o es incorrecto. Ejemplo: ejemplo1@gmail.com\n";  
 txtEmail.setText("");  
 }  
 if (isExist(txtNick.getText())) {  
 errorMessage += "Ya existe un usuario con ese nick. Pruebe otro.\n";  
 txtNick.setText("");  
 }  
 if (isExistCif(txtCIF.getText())) {  
 errorMessage += "Ya existe un usuario con ese CIF.\n";  
 txtCIF.setText("");  
 }  
  
  
 if (errorMessage.length() > 0) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error en los datos");  
 alert.setHeaderText("Datos introducidos incorrectos");  
 alert.setContentText("Hay datos incorrectos");  
 Label label = new Label("Los errores son:");  
  
 TextArea textArea = new TextArea(errorMessage);  
 textArea.setEditable(false);  
 textArea.setWrapText(true);  
  
 textArea.setMaxWidth(Double.MAX\_VALUE);  
 textArea.setMaxHeight(Double.MAX\_VALUE);  
 GridPane.*setVgrow*(textArea, Priority.ALWAYS);  
 GridPane.*setHgrow*(textArea, Priority.ALWAYS);  
  
 GridPane expContent = new GridPane();  
 expContent.setMaxWidth(Double.MAX\_VALUE);  
 expContent.add(label, 0, 0);  
 expContent.add(textArea, 0, 1);  
  
 alert.getDialogPane().setExpandableContent(expContent);  
 alert.showAndWait();  
 return false;  
 } else {  
 return true;  
 }  
  
 }  
  
 private void saveCustomer() {  
 try {  
 repository.save(  
 new Customer(  
 txtName.getText(),  
 txtSurname.getText(),  
 txtCIF.getText(),  
 txtDirection.getText(),  
 txtNick.getText(),  
 txtPassword.getText(),  
 txtPhone.getText(),  
 txtEmail.getText(),  
 photoView.getImage().getUrl().replaceFirst("file:/", ""),  
 LocalDateTime.*now*(),  
 true  
 )  
 );  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public boolean resumen() {  
 var message = "Nombre: " + txtName.getText() + "\n" +  
 "Apellidos: " + txtSurname.getText() + "\n" +  
 "CIF: " + txtCIF.getText() + "\n" +  
 "Dirección: " + txtDirection.getText() + "\n" +  
 "NickUser: " + txtNick.getText() + "\n" +  
 "Teléfono: " + txtPhone.getText() + "\n" +  
 "Email: " + txtEmail.getText();  
 Alert alert = new Alert(Alert.AlertType.CONFIRMATION);  
 alert.setTitle("Datos de Usuario Nuevo.");  
 alert.setHeaderText("Compruebe si todos los datos son correctos.");  
 alert.setContentText("Si desea modificar algo, pulse cancelar.");  
 Label label = new Label("Datos:");  
  
 TextArea textArea = new TextArea(message);  
 textArea.setEditable(false);  
 textArea.setWrapText(true);  
  
 textArea.setMaxWidth(Double.MAX\_VALUE);  
 textArea.setMaxHeight(Double.MAX\_VALUE);  
 GridPane.*setVgrow*(textArea, Priority.ALWAYS);  
 GridPane.*setHgrow*(textArea, Priority.ALWAYS);  
  
 GridPane expContent = new GridPane();  
 expContent.setMaxWidth(Double.MAX\_VALUE);  
 expContent.add(label, 0, 0);  
 expContent.add(textArea, 0, 1);  
  
 alert.getDialogPane().setExpandableContent(expContent);  
 var res = alert.showAndWait();  
 return res.get() == ButtonType.OK;  
 }  
  
 public boolean isExist(String text) {  
 Optional<Customer> customer = null;  
 Optional<Employee> employee = null;  
 boolean ok = true;  
 try {  
 customer = repository.findAll().stream().filter(c -> c.getNickName().equals(text)).findFirst();  
 employee = employeeRepo.findAll().stream().filter(c -> c.getNickName().equals(text)).findFirst();  
 if (customer.isEmpty() && employee.isEmpty()) {  
 ok = false;  
 }  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
  
 return ok;  
 }  
 public boolean isExistCif(String text) {  
 Optional<Customer> customer = null;  
 boolean ok = true;  
 try {  
 customer = repository.findAll().stream().filter(c -> c.getCif().equals(text)).findFirst();  
 if (customer.isEmpty()) {  
 ok = false;  
 }  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
  
 return ok;  
 }  
}

**Controlador: LoingController controla la escena de: (Loing.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository y EmployeeRepository),** tiene como función confirmar el correcto logado de los clientes, Managers o Empleados y redireccionarlos a la pantalla correspondiente.

public class LoginController {  
 CustomerRepository customerRepository = CustomerRepository.*getInstance*(DataBaseManager.*getInstance*());  
 EmployeeRepository employeeRepository = EmployeeRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Stage stage;  
  
 @FXML  
 private TextField txtNick;  
 @FXML  
 private TextField txtPassword;  
 @FXML  
 private void btnLoginClick(ActionEvent event) {  
 System.out.println("Login");  
 accionLogin();  
 }  
  
 @FXML  
 private void btnNewUserClick(ActionEvent event) throws IOException {  
 System.out.println("Nuevo Usuario");  
 var scene = SceneManager.*get*();  
 scene.initRegister();  
 }  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 private void accionLogin() {  
 String nick = txtNick.getText();  
 String password = txtPassword.getText();  
  
 System.out.println("Nick: " + nick);  
 System.out.println("Password: " + password);  
  
 if (nick.isEmpty() || password.isEmpty()) {  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.setTitle("Faltan datos...");  
 alert.setContentText("Debe introducir el nick y la contraseña.");  
 txtNick.requestFocus();  
 alert.showAndWait();  
 } else {  
 loggear(nick, password);  
 }  
 }  
  
 private void loggear(String nick, String password) {  
 try {  
 var listCustomer = customerRepository.findAll();  
 var customer = listCustomer.stream().filter(c -> c.getNickName().equals(nick) && c.getPassword().equals(password)).findFirst();  
  
 var listEmployee = employeeRepository.findAll();  
 var employee = listEmployee.stream().filter(e -> e.getNickName().equals(nick) && e.getPassword().equals(password)).findFirst();  
  
  
 if (customer.isEmpty() && employee.isEmpty()) {  
 Alert error = new Alert(Alert.AlertType.ERROR);  
 error.setTitle("Error de usuario");  
 error.setContentText("El usuario no existe");  
 error.showAndWait();  
 } else {  
 if (!customer.isEmpty()) {  
 System.out.println("Usuario cliente logueado.");  
 var scene = SceneManager.*get*();  
 scene.initMainCustomer(stage, customer.get());  
 } else {  
 System.out.println("Usuario empleado logueado.");  
 var scene = SceneManager.*get*();  
 if (employee.get().isManager()) {  
 scene.initAPPManager(stage, employee.get());  
 } else {  
 scene.initAPPEmployee(stage, employee.get());  
 }  
 }  
 }  
  
 } catch (SQLException e) {  
 Alert error = new Alert(Alert.AlertType.ERROR);  
 error.setTitle("Error no hay usuarios");  
 error.setContentText("No se ha podido encontrar ningún usuario.");  
 error.showAndWait();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

**Controlador: MenuEmployerController controla la escena de: (MenuEmployee.fxml),** tiene como función redireccionar a la pantalla correspondiente a el Empleado ya logado dependiendo de la opción que elija en la IU.

public class MenuEmployeeController {  
 private Stage dialogStage;  
 private Employee employee;  
 @FXML  
 private Label nickname;  
  
 public void setEmployee(Employee employee) {  
 this.employee = employee;  
 nickname.setText(employee.getNickName());  
 }  
  
 @FXML  
 public void onSalirAction() {  
 Alert alert = new Alert(Alert.AlertType.CONFIRMATION);  
 alert.setTitle("Salir");  
 alert.setContentText("¿Está seguro que desea salir?");  
  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 Platform.*exit*();  
 } else {  
 alert.close();  
 }  
 }  
  
 @FXML  
 public void onAcercaDe() {  
 try {  
 SceneManager.*get*().initAcercaDe();  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationPedidos() {  
 try {  
 SceneManager.*get*().initOrderView(dialogStage);  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
  
 public void openOperationArticles() {  
 try {  
 SceneManager.*get*().initArticleView(dialogStage);  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationClientes() {  
 try {  
 SceneManager.*get*().initViewCustomer(dialogStage);  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationRecepciones() {  
 try {  
 System.out.println("Entra");  
 SceneManager.*get*().initReception(dialogStage);  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void onImportarAction() {  
 try {  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void onExportarAction() {  
 try {  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void setStage(Stage stage) {  
 this.dialogStage = stage;  
 }  
}

**Controlador: MenuManagerController controla la escena de: (MenuManager.fxml),** tiene como función redireccionar a la pantalla correspondiente a el Manager ya logado dependiendo de la opción que elija en la IU.

public class MenuManagerController {  
 private Stage dialogStage;  
 private Employee employee;  
 @FXML  
 private Label nickname;  
  
 public void setEmployee(Employee employee) {  
 this.employee = employee;  
 nickname.setText(employee.getNickName());  
 }  
  
 @FXML  
 public void onSalirAction() {  
 Alert alert = new Alert(Alert.AlertType.CONFIRMATION);  
 alert.setTitle("Salir");  
 alert.setContentText("¿Está seguro que desea salir?");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 Platform.*exit*();  
 } else {  
 alert.close();  
 }  
 }  
  
 @FXML  
 public void onAcercaDe() {  
 try {  
 SceneManager.*get*().initAcercaDe();  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 @FXML  
 public void openOperationEmployee() {  
 try {  
 SceneManager.*get*().initEmployee(dialogStage);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationSupplier() {  
 try {  
 SceneManager.*get*().initSupplierView(dialogStage);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationOrder() {  
 try {  
 SceneManager.*get*().initOrderView(dialogStage);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationArticle() {  
 try {  
 SceneManager.*get*().initArticleView(dialogStage);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationCustumer() {  
 try {  
 SceneManager.*get*().initViewCustomer(dialogStage);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void openOperationReception() {  
 try {  
 SceneManager.*get*().initReception(dialogStage);  
 } catch (IOException e) {  
 System.out.println();  
 }  
 }  
  
 public void setStage(Stage stage) {  
 this.dialogStage = stage;  
 }  
}

**Controlador: MainCustomerController controla la escena de: (Main-customer.fxml) y realiza operaciones sobre el repositorio: (ArticleRepository),** tiene como función mostrar el menú inicial del cliente donde podrá seleccionar hacer un pedido ver sus pedidos, modificar su perfil y ver su perfil.

public class MenuCustomerController {  
 private final ArticleRepository articleRepository = ArticleRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Stage stage;  
 @FXML  
 private TextField txtSearch;  
 @FXML  
 private GridPane grid;  
 private Customer customer;  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 @FXML  
 private void btnCarritoAction(ActionEvent actionEvent) {  
 try {  
 SceneManager.*get*().initCarrito(customer, stage);  
 initArticles(null);  
 } catch (IOException | SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void btnMyOrderClick(MouseEvent mouseEvent) {  
 try {  
 SceneManager.*get*().initViewOrderCustomer(customer);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 @FXML  
 private void btnMyUserClick(MouseEvent mouseEvent) {  
 try {  
 SceneManager.*get*().initViewDataCustomer(customer);  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 private void initArticles(String name) throws SQLException {  
 System.out.println("Iniciando los articulos.");  
 *List*<Article> li;  
  
 if (name == null) {  
 li = articleRepository.findAll();  
 } else {  
 li = articleRepository.findAll().stream().filter(a -> a.getArticle().get().toLowerCase().contains(name.toLowerCase().trim())).toList();  
 }  
 grid.getChildren().clear();  
 var k = 0;  
 for (int i = 0; i < grid.getRowCount(); i++) {  
 for (int j = 0; j < grid.getColumnCount(); j++) {  
 if (k < li.size()) {  
 VBox vbox = getItemCell(li, k);  
 grid.add(vbox, j, i);  
*// grid.setGridLinesVisible(true);*  
k++;  
 }  
 }  
 }  
  
 }  
  
 private VBox getItemCell(*List*<Article> list, int pos) throws SQLException {  
 var article = list.get(pos);  
 VBox vbox = new VBox();  
 vbox.setSpacing(10);  
 vbox.setAlignment(Pos.TOP\_CENTER);  
 vbox.setStyle("-fx-border-color: transparent gray gray transparent ");  
 ImageView iv = new ImageView();  
 iv.setPreserveRatio(true);  
 iv.setFitHeight(130);  
  
 if (!article.getPhoto().isBlank() && Files.*exists*(Paths.*get*(article.getPhoto()))) {  
 Image image = new Image(new File(article.getPhoto()).toURI().toString());  
 iv.setImage(image);  
 } else {  
 iv.setImage(new Image(Resources.*get*(HelloApplication.class, "images/article\_default.png")));  
 list.get(pos).setPhoto(new SimpleStringProperty(Resources.*getPath*(HelloApplication.class, "images/article\_default.png").replaceFirst("/", "")));  
 articleRepository.update(list.get(pos).getPIC(), list.get(pos));  
 }  
  
 HBox hbox = new HBox();  
 hbox.setSpacing(15);  
 hbox.setAlignment(Pos.TOP\_CENTER);  
 Label nombre = new Label(list.get(pos).getArticle().get());  
 nombre.setStyle("-fx-font-weight: bold");  
 nombre.setStyle("-fx-font-size: 16");  
 Label precio = new Label(list.get(pos).getPrice().get() + " €");  
 precio.setStyle("-fx-font-size: 16");  
 hbox.getChildren().addAll(nombre, precio);  
 vbox.getChildren().addAll(iv, hbox);  
  
 vbox.setOnMouseClicked(event -> {  
 if (event.getButton() == MouseButton.PRIMARY && event.getClickCount() == 2) {  
 SceneManager scene = SceneManager.*get*();  
 try {  
 scene.initViewArticle(list.get(pos));  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 });  
  
 return vbox;  
 }  
  
 public void findByName(ActionEvent actionEvent) {  
 try {  
 initArticles(txtSearch.getText());  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 public void setCustomer(Customer customer) {  
 this.customer = customer;  
 try {  
 initArticles(null);  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
}

**El controlador: ViewCarritoController controla la escena de: (view-carrito.fxml) y realiza operaciones sobre el repositorio: (CarritoRepository,),** tiene como función mostrar los productos seleccionados en el carrito del cliente para poder confirmarlo.

public class ViewCarritoController {  
 private final *ObservableList*<Integer> cantidadList = FXCollections.*observableArrayList*();  
 private final CarritoRepository carritoRepository = CarritoRepository.*getInstance*();  
 private final OrderRepository orderRepository = OrderRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private final LineOrderRepository lineOrderRepository = LineOrderRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private final ArticleRepository articleRepository = ArticleRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 private Button btnRealizarPedido;  
 private Customer customer;  
 private Stage main;  
 private Stage viewCarrito;  
 @FXML  
 private ListView<CarritoItem> listProducts;  
 @FXML  
 private ListView<CarritoItem> listPrice;  
 @FXML  
 private TextField txtTotal;  
  
 public void setData() {  
 System.out.println("Cargando los productos de la cesta.");  
 listProducts.setItems(carritoRepository.getItems());  
 listPrice.setItems(carritoRepository.getItems());  
 calcularTotal();  
 if (listProducts.getItems().size()==0){  
 btnRealizarPedido.setDisable(true);  
 }  
 }  
  
 public void onPedidoAction(ActionEvent actionEvent) {  
 Order order = new Order(customer.getName(), Pay.CARD);  
 var list = carritoRepository.getItems();  
 *List*<LineOrder> lista = new ArrayList<>();  
 int cost = 0;  
 boolean error = false;  
  
 try {  
 for (int i = 0; i < list.size(); i++) {  
 var aux = articleRepository.findByName(list.get(i).getName());  
 if (aux.getStock().get() - list.get(i).getAmount() >= 0) {  
 addItem(order, list, lista, i);  
 cost += list.get(i).getTotal();  
 } else {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error");  
 alert.setContentText("No hay suficiente stock del producto " + aux.getArticle().get());  
 alert.showAndWait();  
 error = true;  
 }  
 }  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 if (!error) {  
 saveData(order, lista, cost);  
  
 }  
   
 }  
  
 private void addItem(Order order, *ObservableList*<CarritoItem> list, *List*<LineOrder> lista, int i) {  
 lista.add(  
 new LineOrder(  
 list.get(i).getName(),  
 list.get(i).getAmount(),  
 list.get(i).getPrice(),  
 order.getOIC()  
 )  
 );  
 }  
  
 private void saveData(Order order, *List*<LineOrder> lista, int cost) {  
 order.setPrice(new SimpleDoubleProperty(cost));  
 try {  
 orderRepository.save(order);  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
  
 lista.forEach(item -> {  
 try {  
 var aux = articleRepository.findByName(item.getArticle().get());  
 aux.setStock(new SimpleIntegerProperty(aux.getStock().get() - item.getLoad().get()));  
 articleRepository.update(aux.getPIC(), aux);  
 lineOrderRepository.save(item);  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 });  
 imprimirTicket(order);  
 carritoRepository.clear();  
 viewCarrito.close();  
 main.show();  
 }  
  
 public void setDialogStage(Stage stage) {  
 this.main = stage;  
 }  
  
 public void setCustomer(Customer customer) {  
 this.customer = customer;  
 }  
  
 public void setStage(Stage stage) {  
 cantidadList.addAll(1, 2, 3, 4, 5);  
 this.viewCarrito = stage;  
 main.close();  
 initList();  
 initPrice();  
 setData();  
 }  
  
 private void initPrice() {  
 listPrice.setCellFactory(param -> new ListCell<>() {  
 @Override  
 public void updateItem(CarritoItem item, boolean empty) {  
 super.updateItem(item, empty);  
 if (item != null) {  
 VBox vbox = new VBox();  
 GridPane grid = new GridPane();  
 Label name= new Label(item.getName());  
 name.setPrefWidth(130);  
 Label cant= new Label(item.getAmount() + " x " + item.getPrice()+" = ");  
 cant.setPrefWidth(160);  
 cant.setWrapText(true);  
 Label price = new Label(item.getTotal() + " €");  
 grid.add(name,0,0);  
 grid.add(cant,1,0);  
 grid.add(price,2,0);  
 name.setStyle("-fx-font-weight: bold ;-fx-font-size: 24px");  
 cant.setStyle("-fx-font-weight: bold ;-fx-font-size: 24px");  
 price.setStyle("-fx-font-weight: bold ;-fx-font-size: 24px");  
 grid.setAlignment(Pos.CENTER\_LEFT);  
 grid.setHgap(10);  
 vbox.getChildren().add(grid);  
 setGraphic(vbox);  
 }  
 }  
 });  
 }  
  
 public void initList() {  
 listProducts.setCellFactory(param -> new ListCell<>() {  
 @Override  
 public void updateItem(CarritoItem item, boolean empty) {  
 super.updateItem(item, empty);  
 if (item != null) {  
 HBox hbox = new HBox();  
 hbox.setSpacing(15);  
 VBox vbox = new VBox();  
 vbox.setSpacing(30);  
 ImageView img = new ImageView();  
 Image image = new Image(item.getPhoto());  
 img.setImage(image);  
 img.setFitHeight(100);  
 img.setFitWidth(150);  
  
 HBox hbox2 = new HBox();  
 hbox2.setSpacing(10);  
 Label label = new Label(item.getName());  
 Label cant = new Label("Cantidad: ");  
 Label price = new Label("Price: " + item.getPrice());  
 cant.setStyle("-fx-font-weight: bold ;-fx-font-size: 16px");  
 label.setStyle("-fx-font-weight: bold ;-fx-font-size: 21px");  
 label.setPadding(new Insets(10,0,0,0));  
 label.setPrefWidth(170);  
 label.setWrapText(true);  
 price.setStyle("-fx-font-weight: bold ;-fx-font-size: 20px");  
 price.setTextFill(Color.*valueOf*("#f79329"));  
 price.setAlignment(Pos.BOTTOM\_LEFT);  
 price.setPadding(new Insets(0, 0, 0, 90));  
 price.setPrefHeight(95);  
 ChoiceBox choice = new ChoiceBox(cantidadList);  
 choice.getSelectionModel().select(cantidadList.indexOf(item.getAmount()));  
  
 choice.setOnAction(event -> {  
 var cantidad = (int) choice.getSelectionModel().getSelectedItem();  
 item.setAmount(cantidad);  
 calcularTotal();  
 initPrice();  
 });  
  
 hbox2.getChildren().addAll(cant, choice);  
 vbox.getChildren().addAll(label, hbox2);  
 hbox.getChildren().addAll(img, vbox, price);  
 setGraphic(hbox);  
 }  
 }  
 });  
 }  
  
 private void calcularTotal() {  
 txtTotal.setEditable(false);  
 txtTotal.setText(carritoRepository.getTotal() + " €");  
 }  
 private void imprimirTicket(Order o){  
 HtmlPrinterOrder printer = new HtmlPrinterOrder(o);  
  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Ticket Realizado");  
 alert.setContentText("Puede encontrar su ticket en : GestionAlmacenDAM\\order\\Pedido."+o.getOIC()+"\\html");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 }  
}

**El controlador: ArticuloViewController controla la escena de: (view-Ariculo.fxml) y realiza operaciones sobre el repositorio: (ArticleRepository),** tiene como función mostrar los productos existentes, y muestras las opciones para crear artículo, buscar uno o todos.

public class ArticleViewController {  
 ArticleRepository productosrepository = ArticleRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 TableView<Article> productosTable;  
 @FXML  
 TableColumn<Article, String> pic;  
 @FXML  
 TableColumn<Article, String> producto;  
 @FXML  
 TableColumn<Article, String> descripcion;  
 @FXML  
 TableColumn<Article, String> ubicacion;  
 @FXML  
 TableColumn<Article, Double> precio;  
 @FXML  
 TableColumn<Article, Integer> stock;  
 @FXML  
 TableColumn<Article, Boolean> activo;  
 @FXML  
 TextField busqueda;  
 @FXML  
 private void initialize() {  
 try {  
 loadProd();  
 productosTable.refresh();  
 } catch (SQLException e) {  
 System.err.println("hey no he encontrado nada");  
 }  
 initColumns();  
 }  
  
 @FXML  
 private void loadProd() throws SQLException {  
 System.out.println("Cargando Productos");  
 productosTable.setItems(productosrepository.findAll());  
 }  
  
 private void initColumns() {  
 pic.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getPIC()));  
 producto.setCellValueFactory(data -> data.getValue().getArticle());  
 descripcion.setCellValueFactory(data -> data.getValue().getDescription());  
 ubicacion.setCellValueFactory(data -> data.getValue().getLocation());  
 precio.setCellValueFactory(data -> data.getValue().getPrice().asObject());  
 activo.setCellValueFactory(data -> data.getValue().getIsActive());  
 stock.setCellValueFactory(data -> data.getValue().getStock().asObject());  
 }  
  
 @FXML  
 private void onCreateAction() throws IOException {  
 System.out.println("Integrando el Producto...");  
 Article producto = new Article();  
 boolean aceptarClicked = SceneManager.*get*().initProducto(false, producto);  
 if (aceptarClicked) {  
 try {  
 productosrepository.save(producto);  
 loadProd();  
 } catch (SQLException e) {  
 System.err.println(("Error al crear el producto: " + e.getMessage()));  
 }  
 }  
 }  
  
 @FXML  
 private void onModAction() throws IOException {  
 System.out.println(("Se ha pulsado accion Editar"));  
 Article producto = productosTable.getSelectionModel().getSelectedItem();  
 boolean aceptarClicked = SceneManager.*get*().initProducto(true, producto);  
 if (aceptarClicked) {  
 try {  
 productosrepository.update(producto.getPIC(), producto);  
 } catch (SQLException e) {  
 System.err.println(("Error al actualizar dicha persona: " + e.getMessage()));  
 }  
 }  
 }  
  
 @FXML  
 private void onVerDetalleAction() throws IOException {  
 System.out.println(("Se ha pulsado ver Detalle"));  
 Article producto = productosTable.getSelectionModel().getSelectedItem();  
 SceneManager.*get*().initResumeArticle(producto);  
 }  
  
 @FXML  
 private void findByName() throws SQLException {  
 String name = busqueda.getText();  
 if (name.isEmpty()) {  
 loadProd();  
 } else {  
 productosTable.setItems(productosrepository.findAll().filtered(x -> x.getArticle().get().contains(name) || x.getPIC().contains(name)));  
 }  
 productosTable.refresh();  
 }  
}

**El controlador: EmpleadoVistaManagerController controla la escena de: (empleado-vista.fxml) y realiza operaciones sobre el repositorio: (EmployeeRepository,),** tiene como función mostrar los empleados registrados en el sistema, buscar uno, y da opción de crear uno nuevo o modificar uno ya creado.

public class EmpleadosVistaManagerController {  
 @FXML  
 public TextField buscar;  
 EmployeeRepository employeeRepository = EmployeeRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 private TableView<Employee> employeeTable;  
 @FXML  
 private TableColumn<Employee, String> photo;  
 @FXML  
 private TableColumn<Employee, String> EIC;  
 @FXML  
 private TableColumn<Employee, String> nick;  
 @FXML  
 private TableColumn<Employee, String> name;  
 @FXML  
 private TableColumn<Employee, String> surname;  
 @FXML  
 private TableColumn<Employee, String> nif;  
 @FXML  
 private TableColumn<Employee, String> email;  
 @FXML  
 private TableColumn<Employee, Employee> manager;  
 @FXML  
 private TableColumn<Employee, Employee> createdAt;  
 @FXML  
 private TableColumn<Employee, Employee> active;  
  
 public static String dateParser(LocalDateTime date, Locale locale) {  
 *// private String pattern = "dd/MM/yyyy";* return date.format(DateTimeFormatter  
 .*ofLocalizedDate*(FormatStyle.FULL).withLocale(locale));  
 }  
  
  
 @FXML  
 private void initialize() {  
 try {  
 loadData();  
 } catch (SQLException e) {  
 }  
 photo.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getPhoto()));  
 nick.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getNickName()));  
 name.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getName()));  
 EIC.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getEIC()));  
 surname.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getSurname()));  
 nif.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getNif()));  
 email.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getEmail()));  
 manager.setCellValueFactory(cellData -> new SimpleObjectProperty<>(cellData.getValue()));  
 setManegerValueFactory();  
 createdAt.setCellValueFactory(cellData -> new SimpleObjectProperty<>(cellData.getValue()));  
 setCellCreateAt();  
 active.setCellValueFactory(cellData -> new SimpleObjectProperty<>(cellData.getValue()));  
 setActiveValueFactory();  
 }  
  
 private void setActiveValueFactory() {  
 active.setCellFactory(param -> new TableCell<>() {  
 @Override  
 public void updateItem(Employee item, boolean empty) {  
 if (item != null) {  
 CheckBox check = new CheckBox();  
 check.setDisable(true);  
 check.setSelected(item.isActive());  
 setGraphic(check);  
 }  
 }  
 });  
 }  
  
 private void setManegerValueFactory() {  
 manager.setCellFactory(param -> new TableCell<>() {  
 @Override  
 public void updateItem(Employee item, boolean empty) {  
 if (item != null) {  
 CheckBox check = new CheckBox();  
 check.setDisable(true);  
 check.setSelected(item.isManager());  
 setGraphic(check);  
 }  
 }  
 });  
 }  
  
 private void setCellCreateAt() {  
 createdAt.setCellFactory(param -> new TableCell<>() {  
 @Override  
 public void updateItem(Employee item, boolean empty) {  
 if (item != null) {  
 Label label = new Label(*dateParser*(item.getCreatedAt(), new Locale("es", "ES")));  
 setGraphic(label);  
 }  
 }  
 });  
 }  
  
 @FXML  
 private void onModificarAction() {  
 Employee employee = employeeTable.getFocusModel().getFocusedItem();  
 System.out.println(employee);  
 try {  
 SceneManager.*get*().initModifyEmployee(employee);  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 @FXML  
 private void loadData() throws SQLException {  
 employeeTable.setItems(employeeRepository.findAll());  
 }  
  
 public void onNewEmployee() throws IOException {  
 System.out.println("Insertando Empleado");  
 Employee employee = new Employee();  
 boolean aceptarClicked = SceneManager.*get*().initNewEmployee(false, employee);  
 System.out.println("ESTAS??");  
 if (aceptarClicked) {  
 try {  
 loadData();  
 } catch (SQLException e) {  
 System.err.println(("Error al cargar los empleados " + e.getMessage()));  
 }  
 }  
 }  
  
 public void findByUUID() throws SQLException {  
 String uuid = buscar.getText();  
 if (uuid.isEmpty()) {  
 loadData();  
 } else {  
 employeeTable.setItems(employeeRepository.findAll().filtered(x -> x.getNickName()  
 .contains(uuid) || x.getEIC().contains(uuid)));  
 }  
 employeeTable.refresh();  
 }  
}

**El controlador: EditarEmployeeController controla la escena de: (EditarEmpleadoVistaManager.fxml) y realiza operaciones sobre el repositorio: (EmployeeRepository),** tiene como función confirmar los datos de un empleado ya seleccionado y creado, comprobarlos y en el caso de que sea necesario modificarlos.

public class EditarEmployeeController {  
 private static EditarEmployeeController *instance*;  
 private static Employee *employee*;  
 private final boolean aceptarClicked = false;  
 private final boolean editMode = false;  
 EmployeeRepository repository = EmployeeRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 TextField nombre;  
 @FXML  
 TextField surname;  
 @FXML  
 TextField nick;  
 @FXML  
 PasswordField password;  
 @FXML  
 TextField nif;  
 @FXML  
 TextField email;  
 @FXML  
 CheckBox isManager;  
 @FXML  
 ImageView imagePerfil;  
 @FXML  
 CheckBox isActive;  
 private Stage dialogStage;  
  
 public static EditarEmployeeController get() {  
 return *instance*;  
 }  
  
 public void setEmployee(Employee emplo) {  
 *employee* = emplo;  
 try {  
 setDataInfo();  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
  
 public boolean isAceptarClicked() {  
 return aceptarClicked;  
 }  
  
 private void setDataInfo() throws SQLException {  
 System.out.println("Esto es el SET DATA INFO");  
 nombre.setText(*employee*.getName());  
 surname.setText(*employee*.getSurname());  
 nick.setText(*employee*.getNickName());  
 password.setText(*employee*.getPassword());  
 email.setText(*employee*.getEmail());  
 nif.setText(*employee*.getNif());  
 isManager.setSelected(*employee*.getIsActive().get());  
 isActive.setSelected(*employee*.getIsActive().get());  
  
 if (!*employee*.getPhoto().isBlank() && Files.*exists*(Paths.*get*(*employee*.getPhoto().replace("file:/", "")))) {  
 System.out.println(("Buscando la imagen: " + *employee*.getPhoto()));  
 Image image = new Image(new File(*employee*.getPhoto()).toURI().toString());  
 System.out.println(("Imagen Encontrada en: " + image.getUrl()));  
 imagePerfil.setImage(image);  
 } else {  
 System.out.println(("No existe la imagen. Usando imagen por defecto"));  
 imagePerfil.setImage(new Image(Resources.*get*(HelloApplication.class, "images/user.png")));  
 System.out.println();  
 *employee*.setPhoto(new SimpleStringProperty(Resources.*getPath*(HelloApplication.class, "images/user.png").replaceFirst("/", "")));  
 repository.update(UUID.*fromString*(*employee*.getEIC()), *employee*);  
 }  
 }  
  
 public void onAceptarAction() {  
 if (isDataValid()) {  
 saveEmployee();  
 dialogStage.close();  
 } else {  
 System.out.println("Error");  
 }  
 }  
  
 private void saveEmployee() {  
 try {  
 *employee*.setName(new SimpleStringProperty(nombre.getText()));  
 *employee*.setSurname(new SimpleStringProperty(surname.getText()));  
 *employee*.setNickName(new SimpleStringProperty(nick.getText()));  
 *employee*.setPassword(new SimpleStringProperty(password.getText()));  
 *employee*.setNif(new SimpleStringProperty(nif.getText()));  
 *employee*.setEmail(new SimpleStringProperty(email.getText()));  
 *employee*.setIsManager(new SimpleBooleanProperty(isManager.isSelected()));  
 *employee*.setIsActive(new SimpleBooleanProperty(isActive.isSelected()));  
 *employee*.setPhoto(new SimpleStringProperty(imagePerfil.getImage().getUrl().replaceFirst("file:/", "")));  
 repository.update(  
 UUID.*fromString*(*employee*.getEIC()), *employee*  
);  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 private boolean isDataValid() {  
 String errorMessage = "";  
 if (nombre.getText() == null || nombre.getText().isBlank() || !Patterns.*patternName*(nombre.getText())) {  
 errorMessage += "Debes introducir el nombre";  
 nombre.setText("");  
 }  
 if (surname.getText() == null || surname.getText().isBlank() || !Patterns.*patternSurnames*(surname.getText())) {  
 errorMessage += "Debes introducir apellidos";  
 surname.setText("");  
 }  
 if (nick.getText() == null || nick.getText().isBlank()) {  
 errorMessage += "Debes introducir nick de usuario";  
 nick.setText("");  
 }  
 if (password.getText() == null || password.getText().isBlank() || !Patterns.*patternPassword*(password.getText())) {  
 errorMessage += "Debes introducir una contraseña";  
 password.setText("");  
 }  
 if (email.getText() == null || email.getText().isBlank() || !Patterns.*patternEmail*(email.getText())) {  
 errorMessage += "Debes introducir un email con un formato correcto";  
 email.setText("");  
 }  
 if (nif.getText() == null || nif.getText().isBlank() || !Patterns.*patternCif*(nif.getText())) {  
 errorMessage += "Debes introducir NIF valido";  
 nif.setText("");  
 }  
 return errorMessage.length() == 0;  
 }  
  
 @FXML  
 private void onClickImg() {  
 FileChooser buscadorImg = new FileChooser();  
 buscadorImg.setTitle("Selecciona la imagen del employee: ");  
 buscadorImg.getExtensionFilters().add(new FileChooser.ExtensionFilter("Imagenes", "\*.png"));  
 File file = buscadorImg.showOpenDialog(imagePerfil.getScene().getWindow());  
  
 if (file != null) {  
 System.out.println(("Seleccion del archivo: " + file.getAbsolutePath()));  
 imagePerfil.setImage(new Image(file.toURI().toString()));  
 *employee*.setPhoto(new SimpleStringProperty(file.getAbsolutePath()));  
 }  
 }  
}

**El controlador: CustomerViewController controla la escena de: (customerView.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository,),** tiene como función mostrar los Customer registrados en el sistema, buscar uno, y da opción de crear uno nuevo o modificar uno ya creado.

public class CustomerViewController {  
 private final CustomerRepository customerRepository = CustomerRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Stage stage;  
 @FXML  
 private TextField busqueda;  
 @FXML  
 private TableView<Customer> customerTable;  
 @FXML  
 private TableColumn<Customer, String> tbCic;  
 @FXML  
 private TableColumn<Customer, String> tbName;  
 @FXML  
 private TableColumn<Customer, String> tbSurname;  
 @FXML  
 private TableColumn<Customer, String> tbCif;  
 @FXML  
 private TableColumn<Customer, String> tbDirection;  
 @FXML  
 private TableColumn<Customer, String> tbNick;  
 @FXML  
 private TableColumn<Customer, String> tbPhone;  
 @FXML  
 private TableColumn<Customer, String> tbEmail;  
 @FXML  
 private TableColumn<Customer, Customer> tbCreateAt;  
 @FXML  
 private TableColumn<Customer, Customer> tbActive;  
  
 public static String dateParser(LocalDateTime date, Locale locale) {  
 *// private String pattern = "dd/MM/yyyy";* return date.format(DateTimeFormatter  
 .*ofLocalizedDate*(FormatStyle.FULL).withLocale(locale));  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 init();  
 }  
  
 public void findByCIC(ActionEvent actionEvent) {  
 String name = busqueda.getText().trim().toLowerCase();  
 if (name.isEmpty()) {  
 try {  
 loadData();  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 } else {  
 try {  
 customerTable.setItems(customerRepository.findAll()  
 .filtered(x -> x.getName().toLowerCase().contains(name) || x.getCIC().contains(name)));  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
 customerTable.refresh();  
 }  
  
 private void init() {  
 try {  
 loadData();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 initColum();  
  
 }  
  
 private void initColum() {  
 tbName.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getName()));  
 tbSurname.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getSurname()));  
 tbCic.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getCIC()));  
 tbCif.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getCif()));  
 tbDirection.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getDirection()));  
 tbNick.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getNickName()));  
 tbPhone.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getTelephoneNumber()));  
 tbEmail.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getEmail()));  
 tbCreateAt.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue()));  
 setCellCreateAt();  
 tbActive.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue()));  
 setCellActive();  
 }  
  
 private void setCellActive() {  
 tbActive.setCellFactory(param -> new TableCell<>() {  
 @Override  
 public void updateItem(Customer item, boolean empty) {  
 if (item != null) {  
 CheckBox check = new CheckBox();  
 check.setDisable(true);  
 check.setSelected(item.isActive());  
 setGraphic(check);  
 }  
 }  
 });  
 }  
  
 private void setCellCreateAt() {  
 tbCreateAt.setCellFactory(param -> new TableCell<>() {  
 @Override  
 public void updateItem(Customer item, boolean empty) {  
 if (item != null) {  
 Label label = new Label(*dateParser*(item.getCreatedAt(), new Locale("es", "ES")));  
 setGraphic(label);  
 }  
 }  
 });  
 }  
  
 private void loadData() throws SQLException {  
 customerTable.setItems(customerRepository.findAll());  
 }  
  
 public void onModifyAction(ActionEvent actionEvent) {  
 Customer customer = customerTable.getFocusModel().getFocusedItem();  
 System.out.println(customer);  
 try {  
 SceneManager.*get*().initModifyCustomer(customer);  
 customerTable.refresh();  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void onAddAction(ActionEvent actionEvent) {  
 System.out.println("Creando Usuario");  
 try {  
 SceneManager.*get*().initNewCustomer();  
 init();  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
}

**El controlador: EditarSupplierController controla la escena de: (supplierModifyView.fxml) y realiza operaciones sobre el repositorio: (SupplierRepository,),** editar Supplier registrados en el sistema, comprobando los datos nuevos si son correctos

public class EditarSuplierController {  
 SupplierRepository repository = SupplierRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Supplier supplier;  
 @FXML  
 private TextField areaD;  
 @FXML  
 private TextField areaTelefono;  
 @FXML  
 private TextField areaemail;  
 @FXML  
 private TextField areanombre;  
  
 public void setSupplier(Supplier supplier) {  
 this.supplier = supplier;  
 initData();  
 }  
  
 void initData() {  
 areaD.setText(supplier.getDirection());  
 areaTelefono.setText(supplier.getTelephoneNumber());  
 areanombre.setText(supplier.getNameSupplier());  
 areaemail.setText(supplier.getEmail());  
 }  
  
 @FXML  
 void onAceptarAction(ActionEvent event) {  
 boolean datosOk = comprobarDatos();  
 if (datosOk) {  
 try {  
 guardarSuplier();  
  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Confirmacion");  
 alert.setContentText("El suplier se ha modificado");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 } catch (Exception e) {  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Confirmacion");  
 alert.setContentText("El suplier NO se ha modificado");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
   
 System.out.println("no se ha podido aptualizar");  
 }  
 }  
 }  
  
 private void guardarSuplier() throws SQLException {  
 supplier.setTelephoneNumber(areaTelefono.getText());  
 supplier.setEmail(areaemail.getText());  
 supplier.setNameSupplier(areanombre.getText());  
 supplier.setDirection(areaD.getText());  
 repository.update(supplier.getSIC(), supplier);  
 }  
  
 private boolean comprobarDatos() {  
 String errorMessage = "";  
  
 if (areanombre.getText() == null || areanombre.getText().isBlank()) {  
 errorMessage += "Debes introducir nombre correcto. ";  
 }  
 if (areaD.getText() == null || areaD.getText().isBlank()) {  
 errorMessage += "Debes introducir la dirección correcta. ";  
 }  
 if (areaTelefono.getText() == null || areaTelefono.getText().isBlank() || !Patterns.*patterPhone*(areaTelefono.getText())) {  
 errorMessage += "Debes introducir un teléfono correcto. ";  
 }  
 if (areaemail.getText() == null || areaemail.getText().isBlank() || !Patterns.*patternEmail*(areaemail.getText())) {  
 errorMessage += "Debes introducir email correcto. ";  
 }  
  
 if (errorMessage.length() != 0) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error en los datos");  
 alert.setHeaderText("Datos introducidos incorrectos");  
 alert.setContentText("Hay datos incorrectos");  
 alert.setContentText(errorMessage);  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 }  
 return errorMessage.length() == 0;  
 }  
}

**El controlador: modifyCustomerDataController controla la escena de: (CustomerModifyView.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository,),** tiene como función modificar un Customer ya creado confirmando sus datos y los ya creados.

public class ModifyCustomerDataController {  
 private final CustomerRepository repository = CustomerRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 private ImageView img;  
 @FXML  
 private TextField txtName;  
 @FXML  
 private TextField txtSurname;  
 @FXML  
 private TextField txtCif;  
 @FXML  
 private TextField txtDirection;  
 @FXML  
 private TextField txtPhone;  
 @FXML  
 private TextField txtEmail;  
 private Customer customer;  
 private Stage stage;  
  
 public void btnSaveAction(ActionEvent actionEvent) {  
 if (isValid()) {  
 System.out.println("Guardando nuevo usuario...");  
 saveCustomer();  
 stage.close();  
 }  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 public void setCustomer(Customer customer) {  
 this.customer = customer;  
 initData();  
 }  
  
 private void initData() {  
 Image image = new Image(new File(customer.getPhoto()).toURI().toString());  
 img.setImage(image);  
 txtName.setText(customer.getName());  
 txtSurname.setText(customer.getSurname());  
 txtCif.setText(customer.getCif());  
 txtDirection.setText(customer.getDirection());  
 txtPhone.setText(customer.getTelephoneNumber());  
 txtEmail.setText(customer.getEmail());  
 }  
  
 public void onPhotoClick(MouseEvent event) {  
 System.out.println("Seleccionando la foto...");  
 FileChooser filechooser = new FileChooser();  
 filechooser.setTitle("Indica la foto que desea.");  
 filechooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Imagenes", "\*.jpg", "\*.png"));  
 File file = filechooser.showOpenDialog(img.getScene().getWindow());  
  
 if (file != null) {  
 img.setImage(new Image(file.toURI().toString()));  
 }  
 }  
  
 public boolean isValid() {  
 String errorMessage = "";  
  
 if (txtName.getText() == null || !Patterns.*patternName*(txtName.getText())) {  
 errorMessage += "El nombre no puede estar vacio o es incorrecto. Ejemplo: Jaime\n";  
 }  
 if (txtSurname.getText() == null || !Patterns.*patternSurnames*(txtSurname.getText())) {  
 errorMessage += "El apellido no puede estar vacio o es incorrecto. Ejemplo: Santoyo Salazar\n";  
 }  
 if (txtCif.getText() == null || !Patterns.*patternCif*(txtCif.getText())) {  
 errorMessage += "El CIF no puede estar vacio o es incorrecto. Ejemplo: 12345678A\n";  
 }  
 if (txtDirection.getText() == null || txtDirection.getText().length() == 0) {  
 errorMessage += "La dirección no puede estar vacia.\n";  
 }  
 if (txtPhone.getText() == null || !Patterns.*patterPhone*(txtPhone.getText())) {  
 errorMessage += "El campo de teléfono esta vacio o es incorrecto: Ejemplo: 610245879\n";  
 }  
 if (txtEmail.getText() == null || !Patterns.*patternEmail*(txtEmail.getText())) {  
 errorMessage += "El correo no puede estar vacio o es incorrecto. Ejemplo: ejemplo1@gmail.com\n";  
 }  
 if (isExistCif(txtCif.getText())) {  
 errorMessage += "Ya existe un usuario con ese CIF.\n";  
 txtCif.setText("");  
 }  
  
 if (errorMessage.length() > 0) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error en los datos");  
 alert.setHeaderText("Datos introducidos incorrectos");  
 alert.setContentText("Hay datos incorrectos");  
 Label label = new Label("Los errores son:");  
  
 TextArea textArea = new TextArea(errorMessage);  
 textArea.setEditable(false);  
 textArea.setWrapText(true);  
  
 textArea.setMaxWidth(Double.MAX\_VALUE);  
 textArea.setMaxHeight(Double.MAX\_VALUE);  
 GridPane.*setVgrow*(textArea, Priority.ALWAYS);  
 GridPane.*setHgrow*(textArea, Priority.ALWAYS);  
  
 GridPane expContent = new GridPane();  
 expContent.setMaxWidth(Double.MAX\_VALUE);  
 expContent.add(label, 0, 0);  
 expContent.add(textArea, 0, 1);  
  
 alert.getDialogPane().setExpandableContent(expContent);  
 alert.showAndWait();  
 return false;  
 } else {  
 return true;  
 }  
 }  
  
 private void saveCustomer() {  
 System.out.println(customer);  
 try {  
 System.out.println("custo");  
 customer.setName(new SimpleStringProperty(txtName.getText()));  
 customer.setSurname(new SimpleStringProperty(txtSurname.getText()));  
 customer.setCif(new SimpleStringProperty(txtCif.getText()));  
 customer.setDirection(new SimpleStringProperty(txtDirection.getText()));  
 customer.setTelephoneNumber(new SimpleStringProperty(txtPhone.getText()));  
 customer.setEmail(new SimpleStringProperty(txtEmail.getText()));  
 customer.setPhoto(new SimpleStringProperty(img.getImage().getUrl().replaceFirst("file:/", "")));  
 repository.update(  
 UUID.*fromString*(customer.getCIC()),  
 customer  
 );  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
 public boolean isExistCif(String text) {  
 Optional<Customer> customer = null;  
 boolean ok = true;  
 try {  
 customer = repository.findAll().stream().filter(c -> c.getCif().equals(text)).findFirst();  
 if (customer.isEmpty()) {  
 ok = false;  
 }  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 return ok;  
 }  
}

**El controlador: NewSupplierController controla la escena de: (NewSupplierView.fxml) y realiza operaciones sobre el repositorio: (SupplierRepository),** tiene como función de crear y confirmar los datos de un nuevo Supplier.

public class NewSupplierController {  
 SupplierRepository repository = SupplierRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 private TextField areaDirection;  
 @FXML  
 private TextField areaPhone;  
 @FXML  
 private TextField areaEmail;  
 @FXML  
 private TextField areaNombre;  
 @FXML  
 public void onAceptarAction(ActionEvent event) {  
 boolean datosValidos = coprobarDatos();  
 if (datosValidos) {  
 saveSuplier();  
  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Confirmación");  
 alert.setContentText("El supplier se ha creado");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 } else {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("");  
 alert.setHeaderText("Datos no validos.");  
 alert.showAndWait();  
 }  
 }  
  
  
 @FXML  
 private boolean coprobarDatos() {  
 String errorMessage = "";  
  
 if (areaNombre.getText() == null || areaNombre.getText().isBlank()) {  
 errorMessage += "Debes introducir nombre correcto. ";  
 }  
 if (areaDirection.getText() == null || areaDirection.getText().isBlank()) {  
 errorMessage += "Debes introducir la dirección correcta. ";  
 }  
 if (areaPhone.getText() == null || areaPhone.getText().isBlank() || !Patterns.*patterPhone*(areaPhone.getText())) {  
 errorMessage += "Debes introducir un teléfono correcto. ";  
  
 }  
 if (areaEmail.getText() == null || areaEmail.getText().isBlank() || !Patterns.*patternEmail*(areaEmail.getText())) {  
 errorMessage += "Debes introducir email correcto. ";  
 }  
  
 if (errorMessage.length() != 0) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error en los datos");  
 alert.setHeaderText("Datos introducidos incorrectos");  
 alert.setContentText("Hay datos incorrectos");  
 alert.setContentText(errorMessage);  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
  
 }  
 return errorMessage.length() == 0;  
 }  
  
 @FXML  
 private void saveSuplier() {  
 try {  
 repository.save(  
 new Supplier(  
 areaNombre.getText(),  
 areaDirection.getText(),  
 areaPhone.getText(),  
 areaEmail.getText()));  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
}

**El controlador: NewCustomerController controla la escena de: (NewCostumerView.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository,),** tiene como función de crear y confirmar los datos de un nuevo Customer.

public class NewCustomerController {  
 private final CustomerRepository repo = CustomerRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Stage stage;  
 @FXML  
 private TextField txtName;  
 @FXML  
 private TextField txtSurname;  
 @FXML  
 private TextField txtCif;  
 @FXML  
 private TextField txtNick;  
 @FXML  
 private PasswordField txtPassword;  
 @FXML  
 private TextField txtDirection;  
 @FXML  
 private TextField txtEmail;  
 @FXML  
 private TextField txtPhone;  
 @FXML  
 private ImageView imgView;  
 @FXML  
 private CheckBox isActive;  
  
 public void onAceptarAction(ActionEvent actionEvent) {  
 if (isDataValid()) {  
 saveCustomer();  
 stage.close();  
 }  
 }  
  
 public void onClickImg(MouseEvent mouseEvent) {  
 FileChooser buscadorImg = new FileChooser();  
 buscadorImg.setTitle("Selecciona la imagen del employee: ");  
 buscadorImg.getExtensionFilters().add(new FileChooser.ExtensionFilter("Imagenes", "\*.png"));  
 File file = buscadorImg.showOpenDialog(imgView.getScene().getWindow());  
  
 if (file != null) {  
 System.out.println(("Seleccion del archivo: " + file.getAbsolutePath()));  
 imgView.setImage(new Image(file.toURI().toString()));  
 }  
 }  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 private boolean isDataValid() {  
 String errorMessage = "";  
 if (txtName.getText() == null || txtName.getText().isBlank() || !Patterns.*patternName*(txtName.getText())) {  
 errorMessage += "Debes introducir el nombre\n";  
 txtName.setText("");  
 }  
 if (txtSurname.getText() == null || txtSurname.getText().isBlank() || !Patterns.*patternSurnames*(txtSurname.getText())) {  
 errorMessage += "Debes introducir apellidos\n";  
 txtSurname.setText("");  
 }  
 if (txtNick.getText() == null || txtNick.getText().isBlank()) {  
 errorMessage += "Debes introducir nick de usuario\n";  
 txtNick.setText("");  
 }  
 if (txtEmail.getText() == null || txtEmail.getText().isBlank() || !Patterns.*patternEmail*(txtEmail.getText())) {  
 errorMessage += "Debes introducir un email con un formato correcto\n";  
 txtEmail.setText("");  
 }  
 if (txtCif.getText() == null || txtCif.getText().isBlank() || !Patterns.*patternCif*(txtCif.getText())) {  
 errorMessage += "Debes introducir CIF valido\n";  
 txtCif.setText("");  
 }  
 if (txtDirection.getText() == null || txtDirection.getText().isBlank()) {  
 errorMessage += "Debes introducir una dirección.\n";  
 txtDirection.setText("");  
 }  
 if (txtPassword.getText() == null || txtPassword.getText().isBlank() || !Patterns.*patternPassword*(txtPassword.getText())) {  
 errorMessage += "Debes introducir una contraseña correcta.\n";  
 txtPassword.setText("");  
 }  
 if (txtPhone.getText() == null || txtPhone.getText().isBlank() || !Patterns.*patterPhone*(txtPhone.getText())) {  
 errorMessage += "Debes introducir una teléfono válido.\n";  
 txtPhone.setText("");  
 }  
 if (errorMessage.length() > 0) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Error en los datos");  
 alert.setHeaderText("Datos introducidos incorrectos");  
 alert.setContentText("Hay datos incorrectos");  
 Label label = new Label("Los errores son:");  
  
 TextArea textArea = new TextArea(errorMessage);  
 textArea.setEditable(false);  
 textArea.setWrapText(true);  
  
 textArea.setMaxWidth(Double.MAX\_VALUE);  
 textArea.setMaxHeight(Double.MAX\_VALUE);  
 GridPane.*setVgrow*(textArea, Priority.ALWAYS);  
 GridPane.*setHgrow*(textArea, Priority.ALWAYS);  
  
 GridPane expContent = new GridPane();  
 expContent.setMaxWidth(Double.MAX\_VALUE);  
 expContent.add(label, 0, 0);  
 expContent.add(textArea, 0, 1);  
  
 alert.getDialogPane().setExpandableContent(expContent);  
 alert.showAndWait();  
 return false;  
 } else {  
 return true;  
 }  
 }  
  
 private void saveCustomer() {  
 try {  
 repo.save(  
 new Customer(  
 txtName.getText(),  
 txtSurname.getText(),  
 txtCif.getText(),  
 txtDirection.getText(),  
 txtNick.getText(),  
 txtPassword.getText(),  
 txtPhone.getText(),  
 txtEmail.getText(),  
 imgView.getImage().getUrl().replaceFirst("file:/", ""),  
 LocalDateTime.*now*(),  
 isActive.isSelected()  
 )  
 );  
 } catch (SQLException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
}

**El controlador: OrderController controla la escena de: (OrderView.fxml) y realiza operaciones sobre el repositorio: (OrderRepository)**, tiene como función presentar las order que existen, y tener las opciones de ver en detalle y ver una.

public class OrderController {  
 OrderRepository repository = OrderRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 private TextField textAreaBuscarPorOic;  
 @FXML  
 private TableView<Order> tablaPedidos;  
 @FXML  
 private TableColumn<Order, String> columnaCliente;  
 @FXML  
 private TableColumn<Order, Pay> columnaMetodoDePago;  
 @FXML  
 private TableColumn<Order, String> columnaOIC;  
 @FXML  
 private TableColumn<Order, Double> columnaPrecio;  
 @FXML  
 void onButonVerDetalle(MouseEvent event) throws SQLException {  
  
 String s = textAreaBuscarPorOic.getText();  
 var order = repository.findAll().stream().filter(x -> x.getOIC().contains(s)).findFirst();  
  
 if (order.isEmpty()) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Pedido no selecionado");  
 alert.setContentText("No ha selecionado ningun Pedido para ver su detalle");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 } else {  
 try {  
 SceneManager.*get*().initLineOrderView(order.get());  
 } catch (Exception e) {  
 System.out.println("no se ha podido cargar el line order view");  
 }  
 }  
 }  
  
 @FXML  
 private void initialize() {  
  
 try {  
 loadData();  
 } catch (SQLException e) {  
 System.out.println("No se ha podido cargar la lista de personas");  
 }  
  
 columnaOIC.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getOIC()));  
 columnaCliente.setCellValueFactory(cellData -> cellData.getValue().getCustomer());  
 columnaMetodoDePago.setCellValueFactory(cellData -> cellData.getValue().getMethodPay());  
 columnaPrecio.setCellValueFactory(cellData -> cellData.getValue().getPrice().asObject());  
  
 tablaPedidos.getSelectionModel().selectFirst();  
 }  
  
 @FXML  
 private void loadData() throws SQLException {  
 System.out.println("Cargando datos...");  
 if (repository.findAll().size() == 0) {  
 System.out.println("Repositorio vacio a 0");  
  
 } else {  
 System.out.println("Repositorio NO vacio");  
 System.out.println(repository.findAll());  
 }  
 tablaPedidos.setItems(repository.findAll());  
 }  
  
 public void selecionarAcion(MouseEvent mouseEvent) {  
 Order o = tablaPedidos.getSelectionModel().getSelectedItem();  
 textAreaBuscarPorOic.setText(o.getOIC());  
 }  
  
  
 public void vertodos(ActionEvent actionEvent) throws SQLException {  
 tablaPedidos.setItems(repository.findAll());  
 }  
  
 public void findByName(ActionEvent actionEvent) throws SQLException {  
 String name = textAreaBuscarPorOic.getText();  
 if (name.isEmpty()) {  
 loadData();  
 } else {  
 tablaPedidos.setItems(repository.findAll().filtered(x -> x.getCustomer().get()  
 .toLowerCase().contains(name) || x.getOIC().toUpperCase().contains(name)));  
 }  
 tablaPedidos.refresh();  
 }  
}

**El controlador: ProductoController controla la escena de: (NewProducto.fxml) y realiza operaciones sobre el repositorio: (ArticleRepository,),** tiene como función de crear y confirmar los datos de un nuevo Supplier.

public class ProductoController {  
 @FXML  
 TextField productName;  
 @FXML  
 TextField productLocat;  
 @FXML  
 TextField productDesc;  
 @FXML  
 TextField productPrice;  
 @FXML  
 TextField productStock;  
 @FXML  
 CheckBox isActive;  
 @FXML  
 ImageView imageProd;  
   
 private Stage dialogStage;  
  
 private Article producto;  
  
 private boolean aceptarClicked = false;  
 private boolean editMode = false;  
  
 *// GETTERS AND SETTERS*  
public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
  
 public void setProducto(Article producto) {  
 this.producto = producto;  
 System.out.println("Producto asociado: " + producto);  
 if (editMode) {  
 setDataInfo();  
 }  
 productName.requestFocus();  
 }  
  
 public void setEditarModo(boolean editarModo) {  
 this.editMode = editarModo;  
 }  
  
 public boolean isAceptarClicked() {  
 return aceptarClicked;  
 }  
  
  
 private void setDataInfo() {  
 productName.setText(producto.getArticle().get());  
 productLocat.setText(producto.getLocation().get());  
 productDesc.setText(producto.getDescription().get());  
 productPrice.setText(producto.getPrice().asObject().get().toString());  
 productStock.setText(producto.getStock().asString().getValue());  
 isActive.setSelected(producto.getIsActive().get());  
  
 if (!producto.getPhoto().isBlank() && Files.*exists*(Paths.*get*(producto.getPhoto().replace("file:/", "")))) {  
 System.out.println(("Buscando la imagen: " + producto.getPhoto()));  
 Image image = new Image(new File(producto.getPhoto()).toURI().toString());  
 System.out.println(("Imagen Encontrada en: " + image.getUrl()));  
 imageProd.setImage(image);  
 } else {  
 System.out.println(("No existe la imagen. Usando imagen por defecto"));  
 imageProd.setImage(new Image(Resources.*get*(HelloApplication.class, "img/cajita.png")));  
 producto.setPhoto(new SimpleStringProperty(Resources.*getPath*(HelloApplication.class, "img/cajita.png")));  
 }  
 }  
  
 private boolean isDataOk() {  
 String errorMessage = "";  
  
 if (productName.getText() == null || productName.getText().isBlank()) {  
 errorMessage += "El nombre no puede estar en blanco\n";  
 }  
 if (productStock.getText() == null || productStock.getText().isBlank() || Patterns.*isNumberInt*(productStock.getText())) {  
 errorMessage += "Cantidad Inválida\n";  
 }  
 if (productLocat.getText() == null || productLocat.getText().isBlank()) {  
 errorMessage += "Ubicación no puede estar vacía\n";  
 }  
  
 if (productDesc.getText() == null || productDesc.getText().isBlank()) {  
 errorMessage += "Descripción no puede estar vacía\n";  
 }  
  
 if (productPrice.getText() == null || productPrice.getText().isBlank() || Patterns.*isNumerDouble*(productPrice.getText())) {  
 errorMessage += "Precio no puede estar vacío\n";  
 }  
  
 if (errorMessage.length() == 0) {  
 return true;  
 } else {  
 Alert alert = AlertInfo.*getAlertErrorDetails*("Error en datos", "Datos del producto", "Existen problemas al intentar Aceptar.", errorMessage);  
 alert.showAndWait();  
 return false;  
 }  
 }  
  
 @FXML  
 private void onAcept() {  
 System.out.println(("Se ha pulsado en Aceptar"));  
 if (isDataOk()) {  
 producto.setArticle(new SimpleStringProperty(productName.getText()));  
 producto.setLocation(new SimpleStringProperty(productLocat.getText()));  
 producto.setDescription(new SimpleStringProperty(productDesc.getText()));  
 producto.setStock(new SimpleIntegerProperty(Integer.*parseInt*(productStock.getText())));  
 producto.setPrice(new SimpleDoubleProperty(Double.*parseDouble*(productPrice.getText())));  
 producto.setPhoto(new SimpleStringProperty(imageProd.getImage().getUrl().replaceFirst("file:/", "")));  
 producto.setIsActive(new SimpleBooleanProperty(isActive.isSelected()));  
 aceptarClicked = true;  
 dialogStage.close();  
 } else {  
 System.out.println(("Datos no validos"));  
 }  
 }  
   
 @FXML  
 private void onCancel() {  
 System.out.println(("Has pulsado Cancelar"));  
 dialogStage.close();  
 }  
  
 @FXML  
 private void onClickImg() {  
 FileChooser buscadorImg = new FileChooser();  
 buscadorImg.setTitle("Selecciona la imagen del producto: ");  
 buscadorImg.getExtensionFilters().add(new FileChooser.ExtensionFilter("Imagenes", "\*.png"));  
 File file = buscadorImg.showOpenDialog(imageProd.getScene().getWindow());  
  
 if (file != null) {  
 System.out.println(("Seleccion del archivo: " + file.getAbsolutePath()));  
 imageProd.setImage(new Image(file.toURI().toString()));  
 producto.setPhoto(new SimpleStringProperty(file.getAbsolutePath()));  
  
 }  
 }  
}

**El controlador: ReceptionViewController controla la escena de: (RecepcionView.fxml) y realiza operaciones sobre el repositorio: (ReceptionRepository),** tiene como función de ver las recepciones creadas y sus líneas de venta.

public class ReceptionViewController {  
 ReceptionRepository repo = ReceptionRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 TableView<Reception> tableRecept;  
 @FXML  
 TableColumn<Reception, String> ric;  
 @FXML  
 TableColumn<Reception, String> supplier;  
 @FXML  
 TableColumn<Reception, String> carrier;  
 @FXML  
 TableColumn<Reception, Double> cost;  
 @FXML  
 TextField find;  
   
 @FXML  
 private void initialize() {  
 try {  
 loadProd();  
 tableRecept.refresh();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 initColumns();  
 }  
  
 private void initColumns() {  
 ric.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getRIC()));  
 supplier.setCellValueFactory(data -> data.getValue().getSupplierName());  
 carrier.setCellValueFactory(data -> data.getValue().getCarrier());  
 cost.setCellValueFactory(data -> data.getValue().getCost().asObject());  
 }  
  
 private void loadProd() throws SQLException {  
 tableRecept.setItems(repo.findAll());  
 }  
  
  
 @FXML  
 private void findByName() throws SQLException {  
 String name = find.getText().trim().toLowerCase();  
 if (name.isEmpty()) {  
 loadProd();  
 } else {  
 tableRecept.setItems(repo.findAll()  
 .filtered(x -> x.getSupplierName().get().toLowerCase().contains(name) || x.getRIC().contains(name)));  
 }  
 tableRecept.refresh();  
 }  
  
  
 @FXML  
 private void onCreateAction() throws IOException {  
 System.out.println("Insertando la Recepcion...");  
 Reception recepcion = new Reception();  
 LineReception lineReception = new LineReception();  
 boolean aceptarClicked = SceneManager.*get*().initNewReception(recepcion, lineReception);  
 if (aceptarClicked) {  
 try {  
 repo.save(recepcion);  
 loadProd();  
 } catch (SQLException e) {  
 System.err.println(("Error al crear la recepción: " + e.getMessage()));  
 }  
 }  
 }  
  
 @FXML  
 private void onClickAction() throws IOException, SQLException {  
 System.out.println("Mostrando Resumen.");  
 Reception reception = tableRecept.getSelectionModel().getSelectedItem();  
 SceneManager.*get*().initResumeReception(reception);  
 }  
}

**El controlador: ResumenReceptionController controla la escena de: (VerDetalleReception.fxml) y realiza operaciones sobre el repositorio: (ReceptionRepository),** tiene como función ver las líneas de una Recepcion y poder imprimirlas en formato HTML.

public class ResumenReceptionController {  
 LineReceptionRepository repo = LineReceptionRepository.*getInstance*(DataBaseManager.*getInstance*());  
 @FXML  
 TableColumn<LineReception, String> articleCol;  
 @FXML  
 TableColumn<LineReception, Integer> cantCol;  
 @FXML  
 TableColumn<LineReception, Double> unitCol;  
 @FXML  
 TableColumn<LineReception, Double> totalCol;  
 @FXML  
 TableColumn<LineReception, String> recepCol;  
 @FXML  
 TableColumn<LineReception, String> rlic;  
 @FXML  
 TableView<LineReception> recepcionView;  
 @FXML  
 TextField suplierName;  
 @FXML  
 TextField carrier;  
 @FXML  
 TextField cost;  
  
 private Reception reception;  
  
 public void setRecepcion(Reception recepcion) throws SQLException {  
 this.reception = recepcion;  
 setDataInfo();  
 loadLineReception();  
 initColumns();  
 }  
  
 private void initColumns() {  
 articleCol.setCellValueFactory(data -> data.getValue().getArticlePIC());  
 cantCol.setCellValueFactory(data -> data.getValue().getLoad().asObject());  
 unitCol.setCellValueFactory(data -> data.getValue().getUnitPrice().asObject());  
 totalCol.setCellValueFactory(data -> data.getValue().getTotalPrice().asObject());  
 recepCol.setCellValueFactory(data -> data.getValue().getBelongsRecepcion());  
 rlic.setCellValueFactory(data -> new SimpleStringProperty(data.getValue().getRLIC()));  
  
 }  
  
 private void loadLineReception() throws SQLException {  
 recepcionView.setItems(repo.findAll().filtered(z -> z.getBelongsRecepcion().get().contains(reception.getRIC())));  
 }  
  
  
 private void setDataInfo() {  
 suplierName.setText(reception.getSupplierName().get());  
 carrier.setText(reception.getCarrier().get());  
 cost.setText(reception.getCost().asObject().get().toString());  
  
  
 }  
  
}

**El controlador: ResumenController controla la escena de: (VerDetalle.fxml) y realiza operaciones sobre el repositorio: (OrderRepository,),** tiene como función ver las líneas de un Pedido y poder imprimirlas en formato HTML.

public class ResumenController {  
 @FXML  
 ImageView imagen;  
 @FXML  
 CheckBox productActive;  
 @FXML  
 Label productStock;  
 @FXML  
 Label producPrice;  
 @FXML  
 Label productLocat;  
 @FXML  
 Label productDesc;  
 @FXML  
 Label productoTitle;  
 @FXML  
 Label productoName;  
 private Article producto;  
  
 public void setProducto(Article producto) {  
 this.producto = producto;  
 System.out.println(producto);  
 setDataInfo();  
 }  
  
 private void setDataInfo() {  
 productoTitle.setText(producto.getArticle().get());  
 productoName.setText(producto.getArticle().get());  
 productLocat.setText(producto.getLocation().get());  
 productDesc.setText(producto.getDescription().get());  
 producPrice.setText(producto.getPrice().asObject().get().toString());  
 productStock.setText(producto.getStock().asString().getValue());  
 productActive.setSelected(producto.getIsActive().get());  
  
 if (!producto.getPhoto().isBlank() && Files.*exists*(Paths.*get*(producto.getPhoto().replace("file:/", "")))) {  
 System.out.println(("Buscando la imagen: " + producto.getPhoto()));  
 Image image = new Image(new File(producto.getPhoto()).toURI().toString());  
 System.out.println(("Imagen Encontrada en: " + image.getUrl()));  
 imagen.setImage(image);  
 } else {  
 System.out.println(("No existe la imagen. Usando imagen por defecto"));  
 imagen.setImage(new Image(Resources.*get*(HelloApplication.class, "images/cajita.png")));  
 producto.setPhoto(new SimpleStringProperty(Resources.*getPath*(HelloApplication.class, "images/cajita.png")));  
 }  
 }  
}

**El controlador: ViewArticleController controla la escena de: (ViewArticle.fxml) y realiza operaciones sobre el repositorio: (ArticleRepository,),** tiene como función ver los artículos y sus detalles.

public class ViewArticleController {  
 private final CarritoRepository carrito = CarritoRepository.*getInstance*();  
 private Article article;  
 private Stage stage;  
 @FXML  
 private ImageView viewArticle;  
 @FXML  
 private Label txtNameArticle;  
 @FXML  
 private Label txtStock;  
 @FXML  
 private Label txtPrice;  
 @FXML  
 private Label txtDescription;  
 @FXML  
 private Button btnAddArticle;  
  
 @FXML  
 public void btnAddArticleAction(ActionEvent actionEvent) {  
 var aux = findItem();  
 if (aux.isEmpty()) {  
 carrito.addItem(setItem());  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Añadiendo");  
 alert.setHeaderText("Añadiendo el producto seleccionado a la cesta...");  
 alert.show();  
  
 } else {  
 aux.get().setAmount(aux.get().getAmount() + 1);  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle("Añadiendo");  
 alert.setHeaderText("Aumentando la cantidad del producto seleccionado en la cesta...");  
 alert.show();  
 }  
  
 }  
  
 public void setDialogStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 public Article getArticle() {  
 return this.article;  
 }  
  
 public void setArticle(Article article) {  
 this.article = article;  
 initArticle();  
 }  
  
 private void initArticle() {  
 Image image = new Image(new File(article.getPhoto()).toURI().toString());  
 viewArticle.setFitHeight(250);  
 viewArticle.setFitWidth(320);  
 viewArticle.setPreserveRatio(false);  
 viewArticle.setImage(image);  
  
 txtNameArticle.setText(article.getArticle().get());  
 txtStock.setText("Stock: " + article.getStock().get());  
 txtPrice.setText("Precio: " + article.getPrice().get() + "€");  
 txtDescription.setText(article.getDescription().get());  
 txtDescription.setWrapText(true);  
 if (!article.isActive()) {  
 btnAddArticle.setDisable(true);  
 }  
 }  
  
 private CarritoItem setItem() {  
 System.out.println("Metiendo articulo a la cesta.");  
 return new CarritoItem(  
 article.getArticle().get(),  
 article.imagenProperty().get(),  
 article.getPrice().get(),  
 1  
 );  
 }  
  
 private Optional<CarritoItem> findItem() {  
 return carrito.getItems().stream().filter(item -> item.getName().equals(article.getArticle().get())).findFirst();  
 }  
}

**El controlador: ShowCustomerDataControler controla la escena de: (CustomerData.fxml) y realiza operaciones sobre el repositorio: (CustomerRepository),** tiene como función ver las características de un customer.

public class ShowCustomerDataController {  
 @FXML  
 private ImageView img;  
 @FXML  
 private Label txtName;  
 @FXML  
 private Label txtSurname;  
 @FXML  
 private Label txtCif;  
 @FXML  
 private Label txtDirection;  
 @FXML  
 private Label txtNick;  
 @FXML  
 private Label txtPhone;  
 @FXML  
 private Label txtEmail;  
 private Customer customer;  
 private Stage stage;  
  
 public void btnModAction(ActionEvent actionEvent) {  
 try {  
 SceneManager.*get*().initViewModifyDataCustomer(customer, stage);  
 initData();  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 public void setCustomer(Customer customer) {  
 this.customer = customer;  
 initData();  
 }  
  
 public void initData() {  
 Image image = new Image(new File(customer.getPhoto()).toURI().toString());  
 img.setImage(image);  
 txtName.setText(customer.getName());  
 txtSurname.setText(customer.getSurname());  
 txtCif.setText(customer.getCif());  
 txtDirection.setText(customer.getDirection());  
 txtNick.setText(customer.getNickName());  
 txtPhone.setText(customer.getTelephoneNumber());  
 txtEmail.setText(customer.getEmail());  
 }  
}

**El controlador: SupplierVistaController controla la escena de: (SupplierVistaManager.fxml) y realiza operaciones sobre el repositorio: (SupplierRepository,),** tiene como función presentar las distintas opciones del manager sobre los suppliers.

public class SupplierVistaController {  
 SupplierRepository repository = SupplierRepository.*getInstance*(DataBaseManager.*getInstance*());  
 private Stage stage;  
 @FXML  
 private TableColumn<Supplier, String> colDireci;  
 @FXML  
 private TableColumn<Supplier, String> colEmail;  
 @FXML  
 private TableColumn<Supplier, String> colName;  
 @FXML  
 private TableColumn<Supplier, String> colSIC;  
 @FXML  
 private TableColumn<Supplier, String> colTelefo;  
 @FXML  
 private TableView<Supplier> suplierTable;  
 @FXML  
 private TextField textAreaSIC;  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 initView();  
 }  
  
 @FXML  
 void findByUUID(ActionEvent event) throws SQLException {  
  
 String name = textAreaSIC.getText();  
 if (name.isEmpty()) {  
 initView();  
 } else {  
 suplierTable.setItems(repository.findAll().filtered(x -> x.getNameSupplier()  
 .toLowerCase().contains(name) || x.getSIC().toUpperCase().contains(name)));  
 }  
 suplierTable.refresh();  
 }  
  
 @FXML  
 void onModificarAction(ActionEvent event) {  
 var supplier = suplierTable.getFocusModel().getFocusedItem();  
 System.out.println(supplier);  
 try {  
 SceneManager.*get*().initModificarSuplier(supplier);  
 } catch (Exception e) {  
 System.out.println("No se ha seleccionado el empleado");  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 void onNewSuplier(ActionEvent event) {  
 System.out.println("intentando cargar nuevo supplier ");  
 try {  
 SceneManager.*get*().initNewSuplier();  
 suplierTable.refresh();  
 } catch (Exception e) {  
 System.out.println("no se ha podido cargar ");  
 }  
 }  
  
 private void initView() {  
 try {  
 loadData();  
 } catch (SQLException e) {  
 System.out.println("error al cargar datos");  
 }  
 colDireci.setCellValueFactory(cellData -> cellData.getValue().directionProperty());  
 colEmail.setCellValueFactory(cellData -> cellData.getValue().emailProperty());  
 colName.setCellValueFactory(cellData -> cellData.getValue().nameSupplierProperty());  
 colSIC.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getSIC()));  
 colTelefo.setCellValueFactory(cellData -> cellData.getValue().telephoneNumberProperty());  
 }  
  
 @FXML  
 private void loadData() throws SQLException {  
 suplierTable.setItems(repository.findAll());  
 }  
  
 @FXML  
 void errorDeBusqueda() {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Suplier selecionado no existe o incorrecto");  
 alert.setContentText("No ha selecionado ningun suplier o suplier selecionado no existe o incorrecto");  
 Optional<ButtonType> result = alert.showAndWait();  
 if (result.get() == ButtonType.OK) {  
 alert.close();  
 } else {  
 alert.close();  
 }  
 }  
  
 public void seleccionarAction(MouseEvent mouseEvent) {  
 Supplier o = suplierTable.getSelectionModel().getSelectedItem();  
 textAreaSIC.setText(o.getSIC());  
 }  
  
 public void findByUUID() throws SQLException {  
 String name = textAreaSIC.getText();  
 if (name.isEmpty()) {  
 errorDeBusqueda();  
 } else {  
 suplierTable.setItems(repository.findAll().filtered(x -> x.getSIC().contains(name)));  
 }  
 suplierTable.refresh();  
 }  
}

**El controlador: ResumenOrderController controla la escena de: (LineOrderNewView.fxml) y realiza operaciones sobre el repositorio: (LineOrderRepository),** tiene como función ver las líneas de una Order y poder imprimirlas en formato HTML.

public class LineaOrderController {

private Order o;

LineOrderRepository repo = LineOrderRepository.getInstance(DataBaseManager.getInstance());

@FXML

TableColumn<LineOrder, String> articleCol;

@FXML

TableColumn<LineOrder, Integer> cantCol;

@FXML

TableColumn<LineOrder, Double> unitCol;

@FXML

TableColumn<LineOrder, Double> totalCol;

@FXML

TableView<LineOrder> OrderView;

@FXML

private TextField order;

@FXML

TextField cost;

@FXML

private Button crearTiketButon;

@FXML

void onCrearTiket(ActionEvent event) {

HtmlPrinterOrder printer = new HtmlPrinterOrder(o);

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Tiket Realizado");

alert.setContentText("Puede encontrar su tiket en : \n GestionAlmacenDAM\order\ \nPedido.uuid\_del\_pedido\html");

Optional<ButtonType> result = alert.showAndWait();

if (result.get() == ButtonType.OK) {

alert.close();

} else {

alert.close();

}

}

private void initColumns() {

articleCol.setCellValueFactory(data -> data.getValue().getArticle());

cantCol.setCellValueFactory(data -> data.getValue().getLoad().asObject());

unitCol.setCellValueFactory(data -> data.getValue().getUnitPrice().asObject());

totalCol.setCellValueFactory(data -> data.getValue().getTotalPrice().asObject());

}

private void loadLineOrder() throws SQLException {

OrderView.setItems(repo.findAll().filtered(z -> z.getBelongsOrder().get().equals(o.getOIC())));

}

private void setDataInfo() {

order.setText(o.getOIC());

}

public void setOrder(Order order) throws SQLException {

System.out.println("entra en resumen");

this.o = order;

setDataInfo();

loadLineOrder();

initColumns();

}

}

# Conclusiones

El proyecto ha sido desarrollado en un tiempo de 5 semanas un total de 35 días. Se han implementado metodologías agiles como forma de trabajo.

## Fases:

* FASE 1: Análisis
* FASE 2: Estructura del programa y base de datos.
* FASE 3: Desarrollo y Testing
* FASE 4: Desarrollo de Interfaces
* FASE 5: Documentación.

## Tareas asignadas por persona:

CARMONA RODRÍGUEZ, DANIEL

Captura de pantalla de computadora

Descripción generada automáticamente

RODRÍGUEZ FERNÁNDEZ, DANIEL

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

RAMOS SEGURA, JEREMY

Texto

Descripción generada automáticamente con confianza media

BLANCO RODRÍGUEZ, AZAHARA

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

RAMOS MARTINEZ, JAN CARLOS

Imagen que contiene computadora

Descripción generada automáticamente

## Costes

**COSTE RECURSOS ENERGETICOS/MATERIALES**

Tabla

Descripción generada automáticamente

**COSTES DE PERSONAL**











**COSTE TOTAL DEL PROYECTO**

* COSTE DE RECURSOS MATERIALES/ENERGETICOS 🡪 271,88€
* COSTE PERSONAL 🡪 2273,33€
* **COSTE FINAL = 2273,33€**

# Futuras Implementaciones

Este programa en un futuro podría ser implementado en un empresa de sector logístico y de esta forma se podría hacer una gestión completa del almacén, donde los clientes hagan pedidos desde el programa y llegan al almacén donde se procesa su solicitud, también se podrán crear recepciones de productos para cargar el stock.

Una futura mejora para este programa sería implementar el lector de código de barras para la gestión de productos e imprimir código de barras asociado a un producto.

Otra futura implementación sería que implementar una aplicación móvil para realizar la gestión de pedidos y recepciones desde la aplicación Android.

# Bibliografía

Proyectos GitHub:

<https://github.com/joseluisgs/Programacion-09-2021-2022>

<https://github.com/joseluisgs/Programacion-08-2021-2022>

<https://github.com/joseluisgs/EntornosDesarrollo-05-2021-2022>

<https://github.com/joseluisgs/EntornosDesarrollo-06-2021-2022>

<https://github.com/idanirf/ProgramacionUD8>

<https://github.com/idanirf/BaseDeDatosUD4>

Referencias:

<https://projectlombok.org/api/lombok/AllArgsConstructor.html>

Entornos de Desarrollo y Tool:

<https://www.jetbrains.com/es-es/idea/>

<https://www.sqlite.org/index.html>

Enlace a video de presentación del programa.

<https://youtu.be/R6vK3x2md40>

Enlace al Repositorio del Programa

<https://github.com/idanirf/GADAM>