



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

CAMPUS D'ALCOI

*Aplicación para control de
producción y etiquetado de conos
de helado, barquillos y galletas*

PROYECTO FINAL DE CARRERA POR:

Ismael Hernández Bernabeu

GRADO DE INGENIERÍA INFORMÁTICA

DIRECTOR:

Javier Esparza Peidro

Convocatoria de defensa: Septiembre 2016

Realizado con la colaboración de SIPE Informática S.L. y Dicarcono S.L.

Índice

1.	Introducción.....	5
1.1.	Motivación	5
1.2.	Objetivos del proyecto	6
1.3.	Estructura de la memoria.....	6
2.	Análisis del problema	7
2.1.	Entorno, agentes y casos de uso.....	7
2.2.	Mockups	9
2.3.	Etiquetado	12
2.4.	Trazabilidad y control de presencia	15
2.5.	Estadísticas de producción	15
2.6.	Interfaz y categorización.....	16
2.7.	Avisos e información para el operario.....	16
3.	Diseño de la aplicación	18
3.1.	Tecnologías	18
3.2.	Arquitectura	19
3.3.	Capa base	21
3.3.1.	'Core'	21
3.3.2.	DAOs	22
3.3.3.	Utils	23
3.3.4.	Entities	23
3.3.5.	Logic	25
3.4.	Cliente administración	28
3.5.	Cliente impresión	29
3.6.	Infraestructura.....	31
4.	Resultado	32
5.	Conclusiones y trabajos futuros	40
6.	Bibliografía.	41
7.	Anexos	42

Agradecimientos

Me gustaría agradecer a todos los profesores del grado de Ingeniería Informática de la Escuela Politécnica Superior de Alcoy. Sin ellos no estaría haciendo este proyecto. Por su paciencia y sus horas de esfuerzo, que no son en vano.

En especial a los profesores Javier Esparza, por tener la paciencia de lidiar conmigo como tutor para este proyecto final, cosa que no es fácil con mis cambios de idea.

A Keith Stuart, por sus grandes consejos en el momento en que los necesitaba.

A mis compañeros de trabajo, Jose Ángel Martínez y Raúl Guerrero. A los que debo el trabajo que me permite hacer este proyecto y otras muchas cosas más.

A Dicarcono, en especial a Jonatan Hernández y Azahara Sellés. Por permitirme y facilitar este proyecto.

Y como no, a mis amigos Imanol, Barreto, Alejandro, Javi, Paula, Pedro y Joan. Por su apoyo, aún en momentos realmente difíciles.

1. Introducción

1.1. Motivación



Dicarcono S.L. es una empresa de fabricación de productos alimenticios para el **helado**. Situada en Ibi, inicia sus actividades en 1955 y continúa hasta la actualidad. Fabrica conos, galletas, obleas, tubos de chocolate, por cuyo lado no puedes pasar sin comer alguno. Su volumen de producción se cifra en millones de unidades diarias y da empleo a más de 100 empleados. Su infraestructura cuenta con maquinaria muy potente que supone una gran inversión. Tiene como **clientes** grandes superficies de supermercados, heladerías locales y varios clientes internacionales. Sin duda, no habrás podido pasar un verano sin haberte comido alguno de sus productos.

Dicarcono está buscando automatizar una parte de su producción. Para ello se ha puesto en contacto con **SIPE Informática S.L.** con el propósito de crear un programa personalizado, cuyo fin es bajar los costes y facilitar y mejorar la gestión, reduciendo tareas para los empleados.

Cada uno de los paquetes de producto embalados por Dicarcono lleva una etiqueta que identifica el producto e informa de algunos datos sobre el mismo. Actualmente estas **etiquetas** se imprimen en una oficina. Cuando se necesitan, un empleado imprime etiquetas editándolas manualmente para toda la fábrica con un software tercero. Además, el etiquetado proporciona a su vez información sobre la producción real que se está teniendo.

Este proyecto trata acerca del **software** y los sistemas que harán que Dicarcono alcance estos objetivos, ayudándoles a cocinar barquillos como estos:



1.2. Objetivos del proyecto

A grandes rasgos, el proyecto persigue crear una aplicación para el control de la producción de Dicarcono, que debe realizar las siguientes funciones:

- Controlar la presencia de los operarios en las máquinas
- Realizar la impresión de todas las etiquetas necesarias
- Contar la velocidad de producción
- Proporcionar trazabilidad de la fabricación
- Contar los tiempos de parada
- Proporcionar estadísticas y gráficas de la información recogida
- Mostrar información al empleado sobre el embalaje final
- Permitir la administración de todos los datos para las etiquetas y control de acceso a la administración

Y en general, analizar las funciones que Dicarcono espera que el programa cumpla, proporcionarles un diseño realista, desarrollarlo y desplegarlo. Todo esto teniendo en cuenta que la duración del proyecto debe ser limitada y que el coste del proyecto debe ser el menor posible.

1.3. Estructura de la memoria

La memoria consta de un apartado de introducción en que se introduce el propósito y entorno del proyecto y proporciona una motivación al lector para leer la memoria.

El segundo apartado es un análisis detallado de los requisitos de la aplicación, así como de las funciones que debe cumplir y el entorno en que debe hacerlo.

El tercer apartado documenta el diseño de la aplicación, sus diversos componentes, apartados y las tecnologías elegidas. Además en él se justifican las decisiones de diseño basadas en los requerimientos.

El cuarto apartado muestra el resultado conseguido.

El quinto es una reflexión acerca de lo conseguido y qué trabajos futuros son posibles.

El sexto referencias bibliográficas referenciadas durante este documento.

El séptimo son anexos diversos.

2. Análisis del problema

A continuación se discuten los **requisitos** que debe cumplir el programa. Estos en parte se recogen en un **contrato** con una serie de puntos que el programa debe cumplir. Sin embargo, estos no son comprensivos hasta el nivel que realmente es necesario para poder crear el programa. Y más aún, muchas veces estos cambian porque, bien aparecen necesidades nuevas o distintas partes quieren cosas distintas del programa, o porque la realidad o la tecnología nos impiden de alguna forma cumplir con ellas.

Por este motivo debemos examinar más de cerca y conocer todos los detalles acerca del uso que pretende darse al programa y las necesidades que esto implica.

2.1. Entorno, agentes y casos de uso

En primer lugar vamos a describir un poco el entorno físico y humano en que el programa debe funcionar. Empecemos con el humano:

Dicarcono es una empresa grande, dividida en varios **departamentos**, con distintos cargos con intereses diferentes de esta aplicación. Entre ellos, sin duda cabe destacar el departamento de **calidad**. El departamento de calidad es quien pretende crear e implantar el programa y propone la idea básica del mismo. Dado el volumen de producción, se ha convertido para ellos en una necesidad el automatizar ciertas tareas repetitivas que un programa puede suplir sin problema. El departamento de calidad cuenta con unos cuatro empleados. Su misión es la de garantizar la producción, comprobar que el producto está dentro de los parámetros y que cumple unos requisitos. Su interés en el programa es que ilite sus tareas y cubra sus necesidades, de las que luego hablaremos. Son el agente principal, con el que más requisitos trataremos.

Por otro lado, tenemos a los **operarios**. Los operarios recogen el producto producido por las máquinas, alimentan las máquinas, cocinan las mezclas y masas, embalan, empaican, ponen las etiquetas en los embalajes y guardan el producto en el almacén. Los empleos de los operarios son en muchos casos estacionales, por lo que vienen y van y es frecuente que no conozcan el proceso más allá de sus tareas concretas. También es normal que estos operarios no tengan muchos conocimientos de informática como usuarios. Por esto debe ser sencillo. El programa debe estar pensado para que estos empleados puedan utilizarlo con facilidad, sin necesidad de estar muy informados.

Dirección quiere que el programa les dé beneficios y no sólo sea un gasto económico. Realmente también piensan que el programa puede ser un punto de marketing interesante, ya que demuestra una inversión en ello y, por otro lado, a los clientes les interesa bastante la trazabilidad. Hablaremos más

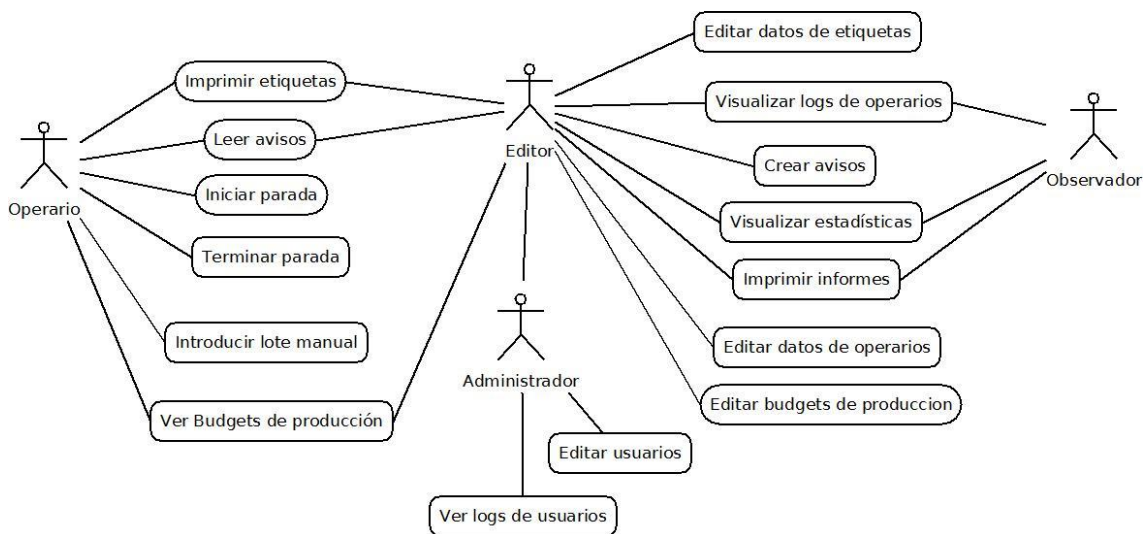
adelante de qué entienden por trazabilidad y qué debe cumplir el programa respecto a ello.

El departamento de **administración** realmente no requiere nada del programa, pero les beneficiará en el sentido de obtención de estadísticas e informes de producción.

Entendido en el programa, cada uno de los agentes según el papel que cumplirán en el programa y las capacidades que tendrán sobre él, serán:

- **Operarios.** Los operarios de la fábrica tal como los acabo de describir.
- **Editores.** Empleados de calidad. Pueden editar e introducir los datos del programa. Sin embargo no pueden ver logs de usuarios ni editar usuarios.
- **Administradores.** Todas las capacidades de editores, más gestión de usuarios. Este papel lo cumplirá el jefe de departamento de calidad.
- **Observadores.** Solo pueden visualizar ciertos datos. Para administración o terceros como auditores.

Y más detalladamente en el siguiente diagrama:



Será necesario que un sistema de **autenticación** de usuarios haga cumplir estas restricciones. No debemos permitir que un operario modifique datos o que acceda a datos privados sin más.

Con esto hemos detallado las partes humanas involucradas y sus intereses. En cuanto a lo físico y espacial, Dicarcono posee una **nave** inmensa dividida en varias secciones. De ellas nos interesan únicamente la de fabricación y las oficinas de calidad.

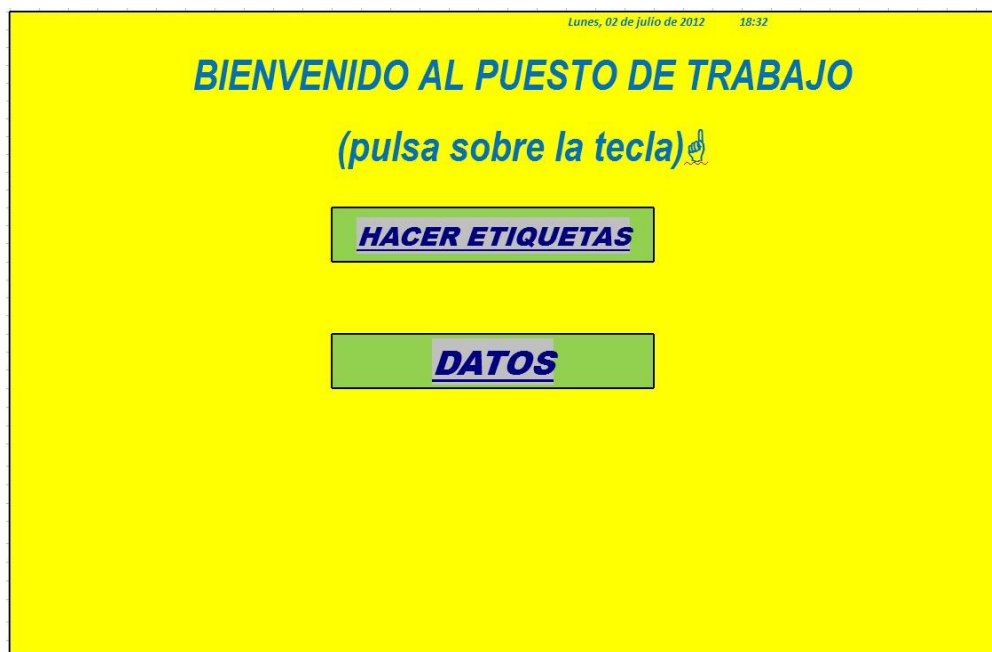
El espacio para fabricación de la nave es amplio y bien ventilado, con máquinas potentes y cableadas. El programa debe funcionar aquí para que los

operarios puedan imprimir en el sitio etiquetas. Ya hay algunos equipos instalados, sin periféricos, que van conectados a unas balanzas. Hay una **red** WiFi disponible y además los equipos están cableados a la red interna por Ethernet. Es un **entorno** en que hay mucho polvo y ruido eléctrico. Sin embargo, esto no nos concierne salvo por las condiciones en que esto pueda afectar al funcionamiento del programa.

Con frecuencia, los operarios son **negligentes** con los equipos y les cortan la corriente. Además la conexión con la red puede perderse en cualquier momento. Debemos proporcionar alguna forma de tolerancia a fallos.

2.2. Mockups

El departamento de calidad nos proporciona algunos mockups de cómo les gustaría que fuese el programa, también a modo de especificación. Los mockups son elementos de prototipado que sirven a modo de imaginación, dibujos del programa sin funcionalidad. Los que nos han enviado son algunos ficheros de hoja de cálculo. Veámoslos y hablemos de su contenido:




Mockup 1

Aquí ya nos queda una cosa muy clara. Obviamente la interfaz del programa debe estar pensada para funcionar en una pantalla táctil.

Para empezar se debe saber todos los que vais a trabajar en esta linea ahora

Cada persona debe introducir su tarjeta en la terminal y el que no la tenga. que escriba su numero de DNI o NIE : solo numeros. sin letras pulsando sobre cada uno de los botones

(si meten tarjeta esta pantalla es [nuti] y pasa directamente a la siguiente)



DNI / NIE	ENVASADOR 1
DNI / NIE	ENVASADOR 2
DNI / NIE	ENVASADOR 3
DNI / NIE	ENVASADOR 4

Mockup 2

ESCRIBE LOS NUMEROS DE TU DNI O NIE

0	1	2	3	4	5	6	7	8	9	0
---	---	---	---	---	---	---	---	---	---	---

3 9 9 0 😊 😊 😊 😊

(si meten la tarjeta esta pantalla es [nuti] y pasa directamente a la siguiente)

JUAN COLGADO DE LA PARRA

ES CORRECTO TU NOMBRE?	Te has equivocado ¿quieres cambiarlo?
-------------------------------	--

(aparece la misma en blanco para rellenar)

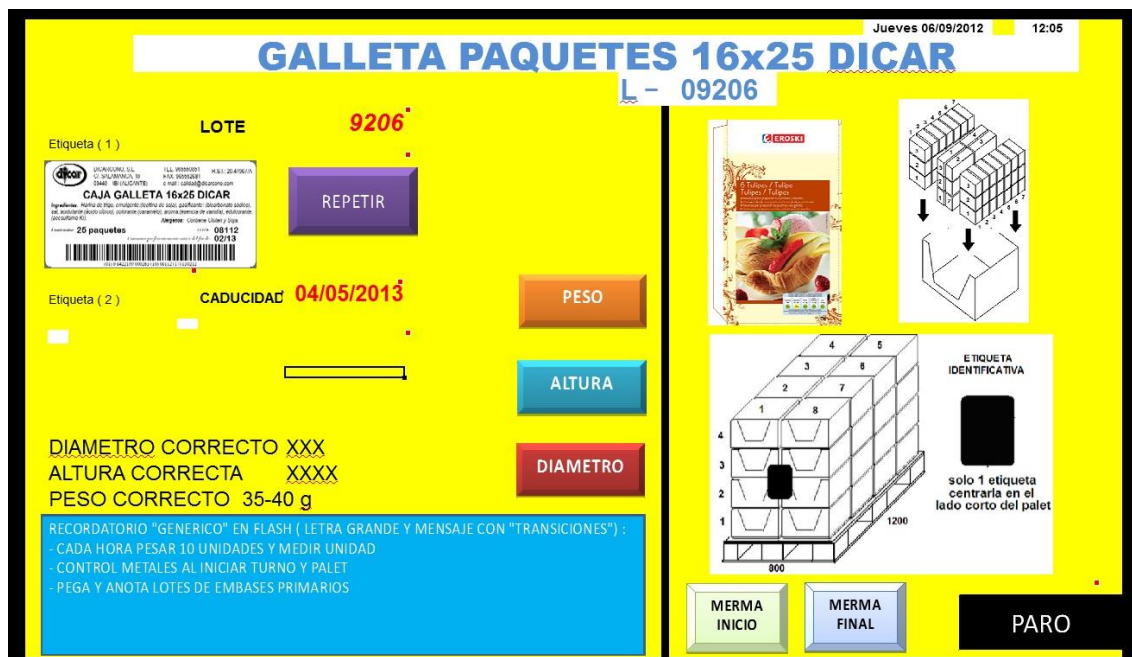
Pulsa aqui cuando acabes para ver tu nombre

Mockup 3



Mockup 4

Se pide al usuario que se autentifique, opcionalmente con un lector de tarjetas o con su DNI, y luego que elija su máquina.



Mockup 5

Como hemos podido observar, son una serie de pantallas en las que seleccionamos información y nos identificamos como empleados para terminar en una pantalla en la que se imprimen etiquetas y varias otras cosas. Además habremos percibido una cierta sensibilidad artística en la persona que creó estos diseños. Sin embargo, estos mockups son algo antiguos y Dicarcono no

quiere que el programa sea así literalmente, sino que los usemos de inspiración. Hay muchos cambios en la funcionalidad que esperan respecto de estos dibujos.

Sin embargo, aquí ya sabemos algo importante y que condicionará la **arquitectura** del programa. La aplicación debe tener dos interfaces de usuario distintas. Una preparada para pantallas **táctiles** para ser usada por los operarios, y otra para administrar los datos que aparecerán en ella. El código del programa deberá estar construido en cierta forma en que podamos **compartir lógica** entre una interfaz y otra.

El **flujo** para los operarios debe ser del tipo:

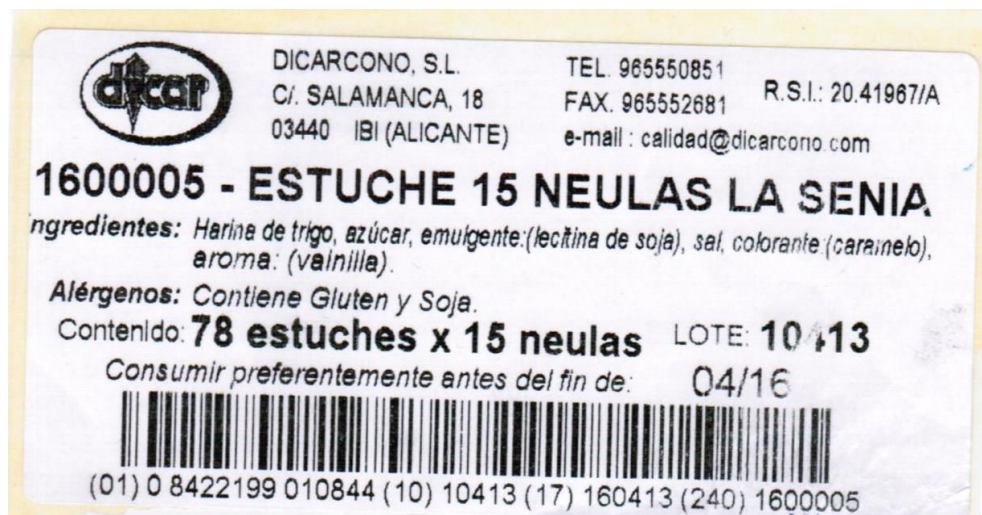
autenticación -> selección -> impresión

2.3. Etiquetado

La funcionalidad más importante del programa es sin duda la **impresión de etiquetas**. Sin duda porque, controlando la impresión controlamos el volumen de producción. Además, es crítico que sean correctas. Las etiquetas deben incluir la siguiente información:

- Logo, dirección y datos de contacto, no necesariamente de Dicarcono.
- Código de producto.
- Nombre del producto.
- Ingredientes y alérgenos, posiblemente en varios idiomas.
- Cantidad del contenido.
- Lote.
- Fecha de caducidad.
- Información de los envasadores.
- Código de barras en el que se incluye parte de toda esta información más el EAN14 del producto.

Aquí tenemos un ejemplo de las etiquetas que actualmente colocan:



Los **motivos** por los que es tan **importante** que sean correctas son varios:

- La identificación del producto vale **dinero**. Esto es, una incorrecta identificación puede llevar a mal almacenaje, pérdidas de tiempo, envíos incorrectos, producto caducado.
- La información sobre **alérgenos** es muy importante, puede conllevar riesgos para la salud de personas.
- El lote es muy importante a la hora de **trazabilidad** y buscar errores en la cadena en caso de defectos y así poder retirar producto de un lote que se encuentre en malas condiciones.
- El código de barras permite que se **informaticen** todos los datos rápidamente al recibirlos en otro lado.

Además, hay diversos factores de riesgo que ponen en peligro la corrección de la etiqueta. Los datos pueden haberse introducido erróneamente, puede haber algún fallo de programación, la impresora puede estar incorrectamente configurada. Por este motivo, deberemos forzar a los usuarios editores a **verificar** los datos de la etiqueta, imprimiendo una, para que comprueben que es correcta. Sólo entonces los dejaremos proceder.

Otra cosa que deberá preocuparnos es que no existe un formato de etiqueta único. El programa deberá ser capaz de alguna forma de imprimir una **etiqueta** genérica, que será la más usada comúnmente. Sin embargo hay casos especiales de etiqueta en que pueden **variar** los siguientes elementos:

- Colocación, tamaño y fuente de los textos de la etiqueta
- Logos e imágenes diversas
- Distintos formatos de código de barras
- Algunos elementos pueden no estar presentes

Y más aún, en cuanto a la etiqueta **genérica**, esta también puede cambiar de la siguiente forma:

- Etiqueta anónima: no aparecerán ni el logo ni los datos de contacto y dirección
- Hasta 4 idiomas. Bien 3 idiomas en una etiqueta imprimiendo una sola etiqueta, 2 idiomas en cada etiqueta imprimiendo dos etiquetas, o un idioma en cada etiqueta imprimiendo 4 etiquetas una para cada idioma

Respecto al **formato** del **código de barras**, debe seguir la especificación GS1-128 / EAN-128 (1), que es una especificación para uso industrial subconjunto del formato de etiqueta Code 128 (2) en la etiqueta genérica.

El contenido del código de barras debe seguir el siguiente formato:

01 <GTIN-14> 10 <lote> <terminador de campo variable> 17
<fecha de caducidad> 240 <código de producto>

Los números fijos son códigos definidos en la especificación que separan e identifican el contenido de cada uno de los campos.

- 01: Global Trade Item Number. Identificador del producto regulado por un organismo supraestatal.
- 10: Lote. Tamaño variable hasta 20 dígitos.
- 17: Fecha de caducidad. 6 dígitos.
- 240: Identificador de producto adicional.

Los campos variables han de estar terminados por un carácter **separador** especial.

En nuestro caso, Dicarcono posee códigos de producto en formato EAN13, lo que sin más se transforma en EAN14 con un 0 de padding a la izquierda. Como son códigos repartidos por una organización y asignados a un producto único, se llaman GTIN-14.

La obtención de los requerimientos del formato de la etiqueta conllevó un proceso de ensayo-error bastante costoso. Dicarcono desconocía los nombres de los estándares y los detalles del funcionamiento del código de barras.

Sin embargo, como hemos mencionado ya, este no será el único formato posible que el programa debe manejar. De hecho, debemos soportar que el formato de código de barras sea cualquiera y puedan añadirse nuevos.

En cuanto al resto de **campos**:

El **lote** es un campo que identifica el grupo de fabricación del producto. Es decir, la tirada o conjunto de artículos que se fabrican en un periodo y comparten los mismos materiales. En nuestro caso, se identificarán típicamente con el formato mmyydd. Es decir, dos dígitos del número de mes, dos dígitos del número del año y dos dígitos del número de día, sin separaciones.

Además también Dicarcono necesita que en el lote aparezcan 'T1' o bien 'T2' para identificar el período temporal en que se han fabricado. De 00:00 a 11:59 debe aparecer 'T1' y de 12:00 a 23:59 debe aparecer 'T2'. Sin embargo, esto solo debe aparecer para las etiquetas de los productos producidos en unas máquinas concretas. Así mismo, algunas máquinas en concreto deben añadir un sufijo al lote que identifique a la **máquina**. Por ejemplo, 'M' para la Sprematec M.

Y para complicar la **lógica** del lote aún más, deben permitirse tanto la introducción manual de un lote como el mantener un lote anterior al cambiar la fecha. Es decir, cuando pasa el tiempo y pasamos del día 1 a las 23:59 al día 2

a las 00:00, el programa debe permitir seguir con el lote anterior y solo cambiar el número de lote cuando el operario desee hacerlo.

Es **crucial** que esto sea correcto porque la trazabilidad depende del lote.

La fecha de **caducidad** debe seguir el formato yymmdd. Dos dígitos de año, dos dígitos del mes, dos dígitos del día sin separadores.

2.4. Trazabilidad y control de presencia

La **trazabilidad** de un producto consiste en la habilidad de poder conseguir el histórico por el que un determinado producto ha pasado. Sus lugares de fabricación, operarios, ingredientes. Como dice la **AECOC**¹:

Se entiende como trazabilidad aquellos procedimientos preestablecidos y autosuficientes que permiten conocer el histórico, la ubicación y la trayectoria de un producto o lote de productos a lo largo de la cadena de suministros en un momento dado, a través de unas herramientas determinadas. (3)

Nuestro programa debe permitir que dado uno de nuestros lotes, se pueda conocer la circunstancia de su fabricación. Fecha, hora, máquina en que se fabricó y los operarios activos en ese momento en la máquina.

Por tanto deberemos almacenar toda esta información y debemos ofrecer las facilidades para su consulta.

Adicionalmente Dicarcono quiere que los operarios se identifiquen antes de poder producir etiquetas, y de esta forma también aparezcan los mismos en las etiquetas y en la base de datos.

La **identificación** debe ser fácil. Como hemos visto en los mockups, la idea es que de alguna forma introduzcan una llave, tarjeta u otro elemento similar que **automáticamente** les identifique y puedan portar consigo. En caso de no portarla, también se ha de permitir una identificación **manual**.

2.5. Estadísticas de producción

Otra de las cosas necesarias es que la aplicación recoja **datos** para generar estadísticas de producción. Puesto que al imprimir la etiqueta de un producto conocemos la **cantidad** que se ha embalado, la cantidad de impresiones sobre el tiempo nos darán los números de producción y velocidad. También podremos conocer si esto fluctúa en el tiempo y cómo, y relacionar estos datos con los operarios presentes en las máquinas.

A partir de esto, nuestro programa debe producir **gráficas e informes** para poder visualizar la información. Sin embargo, los detalles de qué informes y gráficas todavía está por determinar. Aun así, queda claro que deberemos

¹ Asociación Española de Codificación Comercial

recoger tanta información sobre la impresión como nos sea posible para utilizarla en el futuro.

2.6. Interfaz y categorización

En cuanto a la interfaz, debemos tener en cuenta las **características** de los operarios que hemos descrito antes. Pueden ser gente sin conocimientos de informática como usuarios. Además no suelen preocuparse demasiado por informarse más allá de lo que hacen y una aplicación que les requiriese aprender cosas complejas sería un fracaso que terminarían no utilizando o daría pie a muchos errores.

Por esto, la **interfaz** debe ser sencilla y rápida de entender. Los artículos se irán filtrando a partir de hasta 3 pantallas de **categorías** en las que estarán clasificados. Finalmente se elegirá el cliente al que van destinados los artículos. De forma que quedaría así el flujo de la interfaz:

```
Seleccionar máquina    -> Seleccionar categoría 1 ->
Seleccionar categoría 2 -> Seleccionar categoría 3 ->
Seleccionar cliente    -> Seleccionar Artículo    ->
Panel de impresión.
```

De esta forma aligeramos la carga de memorización y lo reducimos a unas pocas palabras de categoría. Además dispondremos de un botón de “No sé” que permitirá avanzar de categoría en caso de **desconocer** alguna de ellas. En ese caso, no se filtrará por esa categoría y se mostrarán todos los artículos.

Además realizaremos todo esto para que funcione en una pantalla **táctil** y funcione pulsando un botón directamente.

Sin embargo para poder **administrar** todos estos datos y poder realizar el alta y modificación y diversas tareas, que corresponden todas con las capacidades de los administradores, editores y observadores, crearemos una segunda interfaz.

Esta segunda interfaz estará pensada para funcionar en equipos de **escritorio**, con teclado y ratón.

2.7. Avisos e información para el operario

Es necesario también que se muestren algunos detalles informativos en pantalla a modo de **comunicación**.

Avisos por máquina:

Para cada **máquina**, aparecerán algunos **avisos**. Esto solo son mensajes que envía administración con información sobre eventos de interés, cosas como notas técnicas de las máquinas o particularidades de la línea de producción.

Avisos por artículo:

Por cada **artículo** también pueden aparecer mensajes con información sobre el artículo en concreto. Así pues deberemos tener dos paneles de avisos distintos con información de ambos tipos.

Imágenes informativas:

Por otro lado mostraremos algunas **imágenes** con detalles del artículo. Esto serán imágenes con diagramas que informarán sobre la colocación de las cajas en los palets, información del embalado de los productos o cualquier otra imagen que se desee añadir para ilustrar el proceso.

Deberemos almacenar, servir y permitir la modificación de estas imágenes de alguna forma.

Producción y Budget:

Se mostrarán unos indicadores de velocidad llamados Budgets. Esto son datos de **velocidad** y **rendimiento** que el operario debe estar cumpliendo. Además se mostrará la velocidad de producción que se está teniendo en ese momento, calculada a partir de las impresiones de etiqueta.

3. Diseño de la aplicación

3.1. Tecnologías

La elección de las tecnologías a emplear es importante porque determinará y condicionará la arquitectura y el funcionamiento de la aplicación. Así que hablemos de ellas primero.

Escribiremos en **Java** el programa. Primero porque Java es un lenguaje que ya domino y el no tener que aprenderlo hace más corto el tiempo de desarrollo. Segundo porque Java ofrece una **librería** estándar que es bastante completa, aún con las limitaciones que implican los lenguajes independientes del sistema operativo. Aunque controlaremos la decisión de en qué sistema operativo se ejecutará el programa y el que sea multiplataforma no nos ofrece realmente un beneficio. También hay otras librerías gratuitas disponibles para Java que lo hacen muy interesante, y haremos uso de ellas. Además, las herramientas de desarrollo son buenas, en mi caso usaré Netbeans prácticamente todo el tiempo para programar.

Si bien es cierto que los lenguajes **multiplataforma** encima de máquinas virtuales como Java conllevan un coste de recursos mayor que el nativo, las máquinas virtuales de Java modernas hacen uso de un compilador **JIT** (just-in-time) que realiza algunas optimizaciones (4) que van más allá incluso de las que puede hacer un compilador tradicional. Cosas como inlining de funciones dinámicamente según el tamaño de estas y la cantidad de veces que se llaman. Por lo que, a cambio de un tiempo de inicio mayor incluso podremos obtener ventajas.

Las **herramientas** son importantes también en esta decisión. Netbeans es un editor que nos facilita muchas cosas, además de compilación y despliegue, sus capacidades de refactoring², la integración con GIT y demás lo hacen una buena herramienta. Además Netbeans nos ofrece un editor de interfaces visual aceptable, que nos ayudará con este proyecto. Además, tanto el lenguaje como las herramientas son gratuitos para uso comercial.

Tendremos algún problema por utilizar Java con la lectura de **tarjetas** o dispositivo de identificación para el control de presencia de los operarios, puesto que Java no nos permite acceder al hardware directamente sin programar código nativo. Sin embargo, le daremos una solución a esto de otra forma, luego veremos cómo.

Por lo demás, también necesitaremos utilizar un servidor de **base de datos**. Elegiremos MySQL porque es gratuito y ya conozco su funcionamiento y es más que suficiente, sin embargo como también usaremos un ORM para

² El refactoring son una serie de técnicas de reestructuración de código, tales como mover función, introducir función, extraer interfaz, etc.

comunicarnos con la base de datos, podremos cambiarla posteriormente si es necesario.

ORM, *Object Relational Mapper* es una librería que nos permite gestionar la base de datos como objetos Java. Esto es bueno primero porque no escribiremos sentencias MySQL que harían que nuestro programa dependiese de MySQL y necesitaríamos modificarlo si quisiéramos cambiar a otro sistema gestor de bases de datos. Y segundo porque es mucho más fácil de mantener. Si escribiésemos queries SQL directamente, al cambiar columnas en una tabla por ejemplo, también necesitaríamos actualizar las queries para que el programa funcionase.

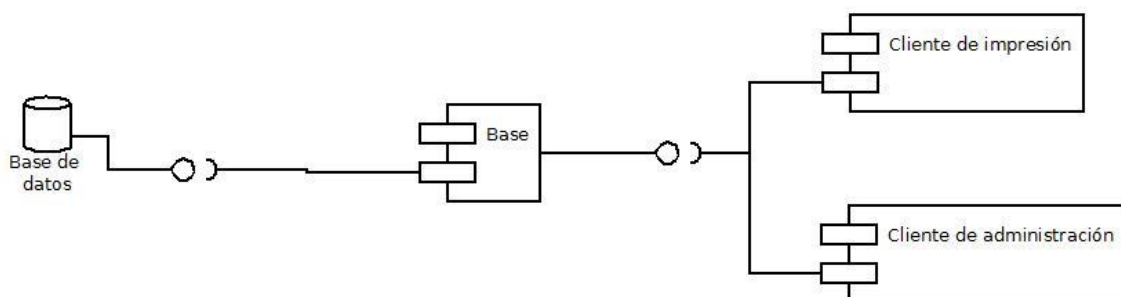
En concreto, usaré Hibernate como librería de ORM, sin embargo usándolo a través de la API de ORM de Java EE: **JPA**. Esto es porque, aunque en el tiempo en que empecé a hacer este programa no sabía JPA, uno de mis compañeros sí lo conoce y podremos preguntarle en caso de tener dudas o problemas.

Para crear los **informes** y etiquetas utilizaremos una librería llamada Jasperreports, con la que generaremos documentos imprimibles, gráficas y demás elementos de reports. Junto con esta utilizaremos Barbecue, una librería para generar códigos de barras. Con ambas librerías crearemos todo lo necesario para las etiquetas.

Y por supuesto se usará GIT durante el desarrollo, aunque solo sea un programador. Primero porque así mantendremos backups del código en otro lugar y segundo porque podremos revertir o **versionar** el código rápidamente si lo necesitamos.

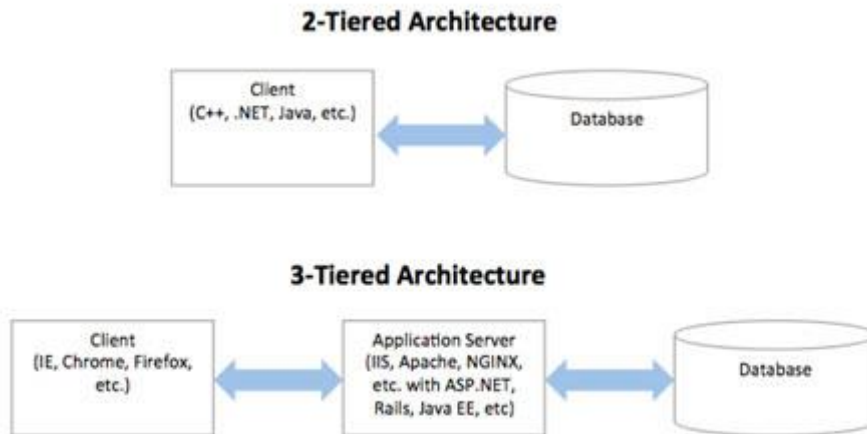
3.2. Arquitectura

Lo siguiente es un diagrama a grandes rasgos de la arquitectura de la aplicación:



Tenemos código para cada uno de los clientes **separado**, que comunica con un código base que define la llamada 'Lógica de negocio'. Realmente, esto no es una típica arquitectura 3-tier. La diferencia está en que la capa de código base de la aplicación la ejecutan los clientes por sí mismos y conectan con la

capa de persistencia, la base de datos, directamente. No existe una capa con un servidor de aplicaciones intermedia como sería el caso típico. Esto es más una arquitectura **2-tier** en la que hemos separado parte de la lógica para poder reutilizarla en dos clientes distintos.

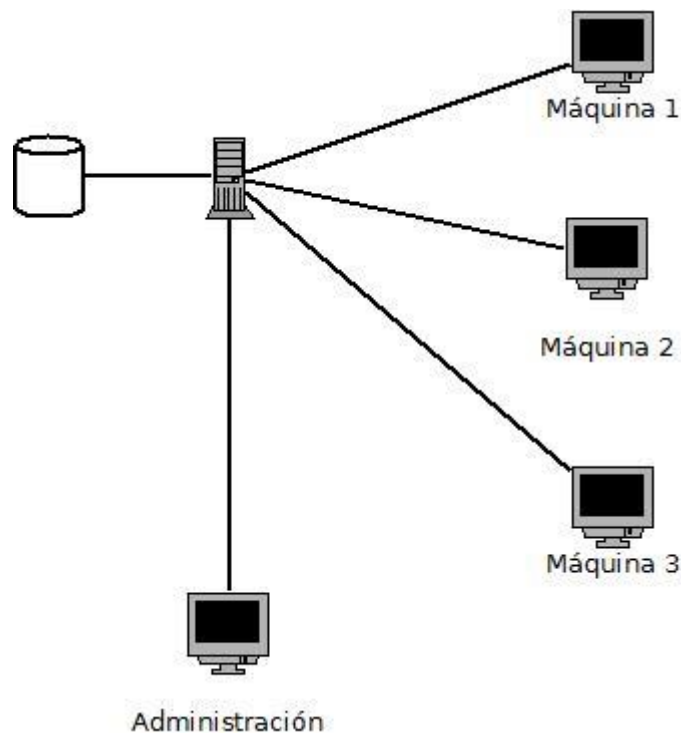


Los motivos por los que utilizar este tipo de arquitectura en vez de una arquitectura clásica 3-tier de Java son dos. El primero la existencia de un código anterior utilizado en otras aplicaciones, previsto para ser **reutilizado**, que ya utiliza esta arquitectura. Esto nos ahorra tiempo y acelera el desarrollo porque algunas cosas ya están programadas.

El segundo motivo es que, puesto que es una aplicación pensada para funcionar únicamente de forma interna y nunca por un tercero, nos es más barato en tiempo no tener que publicar una API que defina unos límites sobre los datos y realice comprobaciones sobre ellos. La aplicación está pensada para funcionar dentro de una intranet cerrada.

Sin embargo, sí es cierto que tiene algunas desventajas esta arquitectura frente a la 3-tier. Puesto que no poseemos un servidor de aplicaciones de Java EE, no podremos utilizar características como la inyección de dependencias. Debemos controlar nosotros la lógica de instanciación.

La aplicación funcionará en una red como esta:



Tendremos un servidor con un servicio de base de datos. Para cada máquina en que deba existir un puesto de impresión, se colocará un equipo con el cliente de impresión. Y aparte, en las oficinas, tenemos los equipos de administración en que se instalará el cliente de administración.

3.3. Capa base

La capa base es la capa que aglutina la lógica de datos, la conexión con la base de datos, define los objetos de persistencia, crea las transacciones, y genera los informes de las etiquetas. Todo esto se usa desde los dos clientes que compondrán la aplicación. Veamos los **componentes** concretos en más detalle.

3.3.1. 'Core'

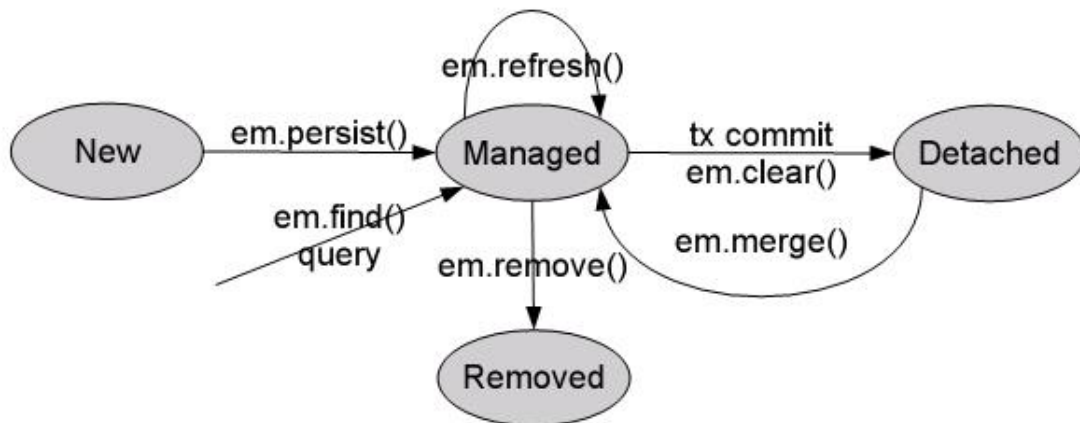
El 'core' es una clase del tipo *singleton*³. Este componente mantiene las referencias e instancia los distintos componentes de la aplicación que basta con que se inicialicen una vez, pero a los que debemos acceder desde distintas partes de la aplicación. La responsabilidad de esta clase es ser el *pegamento* del resto de componentes. Instancia y mantiene referencias y accesores a los DAOs (Data Access Objects), de los que luego hablaremos. Además mantiene referencias a objetos constantes, como son los distintos formatos de conversión entre tipo Date y String, los formatos numéricos, e inicializa la persistencia al ser creado.

³ Singleton es un pattern de arquitectura de software en que solo se permite una instancia de un objeto.

3.3.2. DAOs

Los Data Access Objects son objetos cuya responsabilidad es abrir y cerrar las **transacciones** y definir qué **manipulaciones** se permiten sobre los **datos**. Son accesores sin estado. Por cada Entity que puede existir por sí mismo, existe un Data Access Object que se encarga de guardar, borrar, actualizar información, obtener entidades relacionadas y otros métodos que requieran estar encerrados dentro de una transacción y necesiten de un contexto de persistencia abierto. Hay otros Entities que no tienen un DAO, pero son Entities anidadas que dependen de otros Entities.

Los DAOs tratan con el ciclo de vida de las **Entities** y su **estado**. Típicamente aceptan entidades fuera del contexto de persistencia, y ahí ya bien se vuelven a unir al contexto con una operación *merge* o bien se borran o se crean de nuevas según se necesite con las operaciones *delete* y *persist* respectivamente.



Esto es un sistema complejo y muy técnico que trata con el funcionamiento en concreto de la **Java Persistence API**. Se puede encontrar más información acerca de esto en los tutoriales de Oracle (5).

Puesto que la aplicación necesita cierta **tolerancia** a fallos, todas las llamadas a métodos de los DAOs están rodeados del siguiente tipo de bucle en el código del cliente de impresión de etiquetas. Es lo que yo llamo un **retry-catch**:

```
boolean retry = true;
while (retry) {
    try {
        ...
        retry = false;
    } catch (PersistenceException ex) {
        Logger.getLogger(LoginForm.class.getName())
            .log(Level.SEVERE, null, ex);
        Thread.sleep(1000);
    }
}
```

El retry-catch permite que la red falle. El cliente entonces se **bloqueará** tal como estaba en ese momento hasta que el problema sea solucionado. En cuanto la conexión vuelva a estar disponible, el cliente continuará como estaba sin provocar inconsistencias.

3.3.3. Utils

Utils son clases de utilidad que no contienen ninguna dependencia a ninguna otra parte del proyecto ni poseen estado. Son clases cuyos métodos son estáticos y **puramente funcionales**, es decir, son funciones con un valor de retorno que únicamente depende de los parámetros que se les pasan.

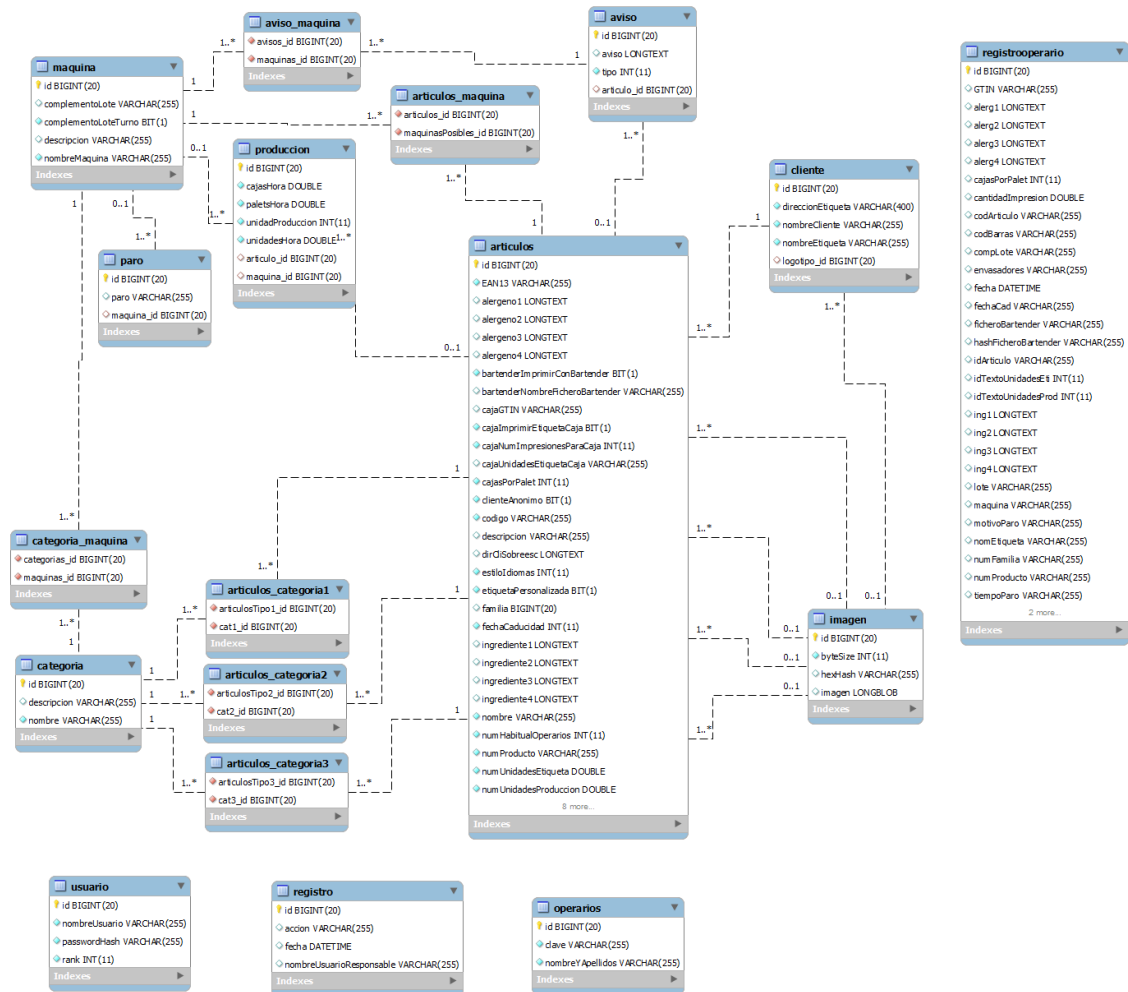
Encontramos aquí las clases HexUtils (herramientas con operaciones en base hexadecimal), ImageUtils (conversiones entre tipos de imágenes, operaciones de recortado de imágenes, etc), ConfUtils (lectura del fichero .properties que sirve como configuración estática para el programa).

3.3.4. Entities

Aquí se encuentran los objetos que definen la persistencia. Esto son *POJOs*⁴ con **anotaciones** que posteriormente serán traducidos a tablas relacionales por el ORM. Estos definen los **datos** que se almacenarán junto con sus tipos de datos y sus relaciones con otras entidades. Los usaremos por toda la aplicación.

⁴ Plain Old Java Objects

Aquí un diagrama de las entidades ya traducidas a **tablas relacionales** que utiliza la aplicación con sus campos:



Estos son todos los datos que necesitamos para generar las etiquetas, datos de producción, parte del sistema de usuarios y control de presencia como hemos recogido en el análisis de requerimientos.

Hay una **particularidad** y es que algunas tablas de registro no están relacionadas con ninguna entidad, sin embargo **copian** sus campos directamente. Esto es necesario. Hemos de mantener los datos tal como eran en el momento en que se crearon las entradas de registro, a modo de log. Si almacenásemos referencias a otras entidades, por ejemplo, el id del artículo en ellas, y alguien edita el artículo, estaríamos perdiendo esa información. Para garantizar la trazabilidad, debemos mantenerlos intactos.

3.3.5. Logic

Aquí están comprendidas varias clases que aunque son lógicas, no tratan directamente con la persistencia. En estas damos **solución** a algunos de los **problemas** más **importantes** del programa. Estas clases son las de impresión de reportes, etiquetas, manipulación de USBs para control de presencia de operarios e integración con *Bartender*. Veamos estos uno por uno también.

Para **imprimir** las etiquetas genéricas tenemos una clase en la que mediante setters rellenamos los campos. Algunos de ellos, opcionales. A partir de esto y unos ficheros en formato *.jasper*, formato propietario de la librería *Jasper Reports*, nos definen el formato de las **etiquetas**. Aquí controlamos cosas como los múltiples idiomas, la creación del código de barras, y los ingredientes y alérgenos.

Esto posteriormente se imprime o visualiza mediante el widget que Jasper Reports nos proporciona para *Swing*⁵.

Sin embargo, esto no es suficiente como hemos visto en los requerimientos de la aplicación. Para poder imprimir el resto de etiquetas, he utilizado el software **Bartender**. Es un software muy completo de impresión de etiquetas, con su propio editor *WYSIWYG*⁶, que permite toda la flexibilidad necesaria. Se da el conveniente caso de que además Dicarcono ya posee una licencia de *Bartender*, por lo que solo debemos preocuparnos por cómo integrar este programa con el nuestro.

La mejor solución al final consiste en hacer que el sistema ejecute por línea de **comandos** *Bartender*, via *Runtime::exec()* y generar un fichero *.csv* temporal con los campos variables que van a pasarse a la etiqueta de *Bartender*.

Desgraciadamente el uso de *Bartender* nos obliga a utilizar **Windows** en todos los equipos para la impresión, elemento que encarece el despliegue del programa.

En esta parte tenemos una clase que se encarga de gestionar toda esta lógica de integración y de generar el fichero de datos temporal que pide *Bartender*.

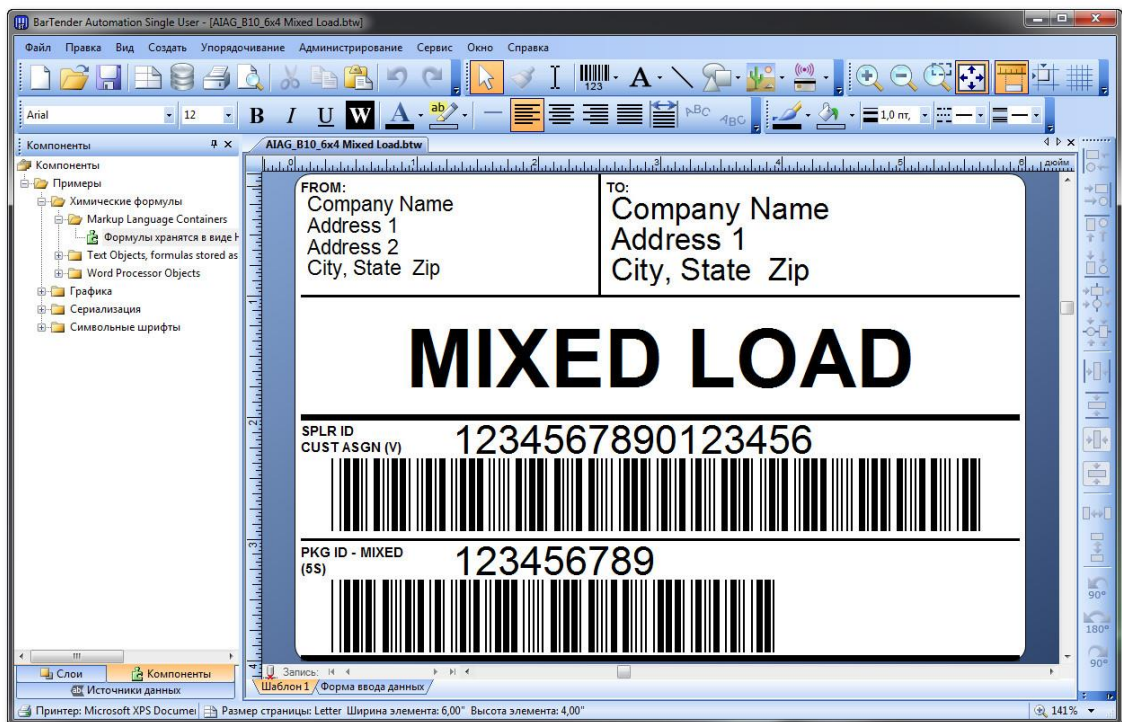
Los equipos de las máquinas deben acceder a los ficheros de *Bartender* que definen el formato de las etiquetas. Podríamos copiarlos a sus discos duros, sin embargo al realizar una modificación habría que copiarlos uno a uno y complicaría el mantenimiento. Es por eso que aquí nuestro programa también se encargará de leer estos ficheros desde una **unidad de red** que se pueda

⁵ Swing es la librería para interfaces de usuario incluida en Java.

⁶ What You See Is What You Get

controlar desde administración, para así tener un único lugar en que realizar las modificaciones.

Aquí tenemos una foto de la pinta que tiene *Bartender*.



Respecto al **control de presencia**, el componente restante, finalmente la solución consiste en utilizar memorias **USB tipo flash**, *pen drives* para los amigos. Esto es mucho más sencillo que utilizar tarjetas de algún tipo como era la idea primitiva de Dicarcono. Estas memorias son baratas y fáciles tanto de disponer como de acceder programáticamente: sin más son sistemas de **ficheros**.

Hay algunos problemas con esta solución. En la API de Java, no somos capaces de distinguir si una ruta es una unidad de almacenamiento extraíble o no, ni tenemos la capacidad de escuchar un posible evento de inserción o extracción de estas unidades. Necesitamos ser capaces de detectar estos **eventos**. Si un operario pincha su llave USB, el programa debe sin más agregarlo como un envasador activo en esa máquina y permitirle imprimir etiquetas.

La solución a esto es ejecutar algunos **comandos** del sistema a través de `Runtime::exec()`, puesto que estamos en Windows y conocemos la plataforma. Ejecutando un cierto comando, en Windows:

```
>wmic logicaldisk where drivetype=2 get deviceid
```

obtenemos un listado de los USBs conectados. Parseando la salida, con un sistema de **polling**, somos capaces de crear un sistema de eventos en el que

podemos detectar la entrada y salida de dispositivos, así como algunas de sus características. En otras plataformas podríamos utilizar el mismo principio para detectar la entrada y salida de USB drives en Java. Si bien es cierto que en las otras plataformas los comandos son **distintos**, pero podemos detectar la plataforma y cambiar el comportamiento.

No es la más eficiente de las soluciones, sin embargo funciona correctamente. Una **alternativa** sería utilizar *JNI* (6) Java Native Interface, que nos permitiría ejecutar código nativo escrito en C, C++ o ASM bajo Java. Es un sistema **complejo** y no nos vale la pena el esfuerzo de aprenderlo únicamente para esto.

Así que en los USBs escribiremos un par de ficheros que identificarán al usuario para el sistema. Esto serán un fichero .id y un fichero .key. El primero contendrá un identificador único para el operario y el segundo el valor hashado⁷ de la clave del operario concatenada al UUID⁸ de la memoria USB. Esto tiene como efecto que si los ficheros son copiados a un USB distinto, no funcionarán, puesto que el UUID será distinto.

Por supuesto nuestro programa debe de ofrecer la capacidad para leer estos USBs llave como para escribirlos.

⁷ Los hashes son funciones matemáticas de un único sentido, que permiten obtener un hash a partir de un valor, de forma que conociendo el valor puede volverse a calcular su hash y compararlo con el hash anterior. Los hashes coincidirán para el mismo valor, pero no puede obtenerse este valor a partir del hash.

⁸ Identificador Único Universal. Las memorias extraíbles exponen este valor que sirve a modo de identificador para el dispositivo.

3.4. Cliente administración

El cliente de administración es sin más un **GUI** para usuario de **escritorio** programado en **Swing**, en ocasiones con ayuda del editor visual de Netbeans, en otras se añaden **elementos programáticamente** para poder reutilizarlos.

Tiene algunos **bloques** de construcción comunes como tablas de ítems, tablas paginadas, selectores desde tabla, diálogos de confirmación, mensajes de error y búsqueda.

Los componentes básicos de la aplicación se disponen en un menú superior a modo de **toolbar** en que se abren los necesarios pinchando sobre ellos.

Dentro de cada panel disponemos de un menú de **acciones** que actúan sobre la selección y comúnmente una tabla.

Dentro de esto, tenemos la ficha de ítem, que se encarga de editar los campos de las entidades. Con controles adecuados según el tipo de campo. Selectores si son entidades relacionadas, campos de texto, selectores de archivo, checkboxes, y demás.

Es un **flujo complejo** con muchos paneles, ventanas y lógica de interfaz.

Tiene algunas características interesantes como **copiar y pegar** de las tablas a un Excel, lo que permite al usuario una flexibilidad para crear sus propias gráficas y demás a partir de los datos que se muestran.

Aquí tenemos una captura de la interfaz de administración:

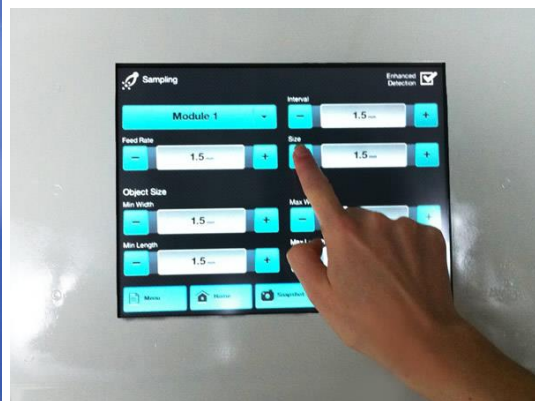
Identificador	Nombre	Descripcion	Híjamas	Número de artículos
1	1NELLA		NELAS(4) CHOCO TULPA(7)	16
2	25LUPINELLA		NELAS(4) CHOCO TULPA(7)	11
3	26CUBIERTO		NELAS(4) CHOCO TULPA(7)	3
4	4TULULAR		NELAS(4)	4
5	5CUBIERTO		NELAS(4)	8
6	6RELLENO		NELAS(4)	4
7	7ABANCO		CHOCO TULPA(7) ABANCO(TULPA(14)	1
8	8GALLETAS		CHOCO TULPA(7) GALLETA(1)	1
9	9TULIPA		CHOCO TULPA(7) TULPA(9) ABANCO(TULPA(14)	13
10	10TAPALETA		CHOCO TULPA(7)	3
11	11BARCAS		CHOCO TULPA(7) TULPA(9) ABANCO(TULPA(14)	2
12	12CONOS		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
13	13BARRAS		SPREIMATEC C(2) CHOCO TULPA(7) CHOCO CONO(8)	0
14	14ESTUQUE		SPREIMATEC N(2) SPREIMATEC N(2) SPREIMATEC R(2) NELAS(4)	28
15	15RTE		NELAS(4) ABANCO(TULPA(14)	3
16	16CAJA		NELAS(4) CHOCO TULPA(7) SPREIMATEC C(2) CHOCO CONO...	17
17	17PAPA BARRA		SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC R(2) NELAS(4)	4
18	18INTERMEDIO		CHOCO TULPA(7)	1
19	19FRECUEN		CHOCO TULPA(7)	4
20	20REGIANA		CHOCO TULPA(7)	1
21	21GRANDE		CHOCO TULPA(7)	1
22	22BARRO TOTAL		CHOCO TULPA(7)	2
23	23INTERIOR CORONA		CHOCO TULPA(7)	5
24	24TURBON		NELAS(4)	0
25	25CHOCOLATE		NELAS(4)	4
26	26BICOLOR		NELAS(4)	1
27	27HONDA		NELAS(4)	22
28	28MDE		CHOCO TULPA(7) TULPA(9) ABANCO(TULPA(14)	2
29	29INTERIOR		CHOCO TULPA(7)	1
30	30MDE EXTERIOR		SPREIMATEC N(2) CHOCO CONO(8)	1
31	31COP LUCHE		SPREIMATEC N(2) CHOCO CONO(8)	1
32	32NEGRO		SPREIMATEC N(2) CHOCO TULPA(7) CHOCO CONO(8)	1
33	33STD CORONA		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC R(2) CHOCO C...	1
34	34LAKATOS		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
35	35REDIANA MD		ABANCO(TULPA(14)	0
36	36FLOR GRANDE		TULPA(9) ABANCO(TULPA(14)	0
37	37BARRAS		ABANCO(TULPA(14)	0
38	38CON BLISTER		NELAS(4) CHOCO TULPA(7) CHOCO CONO(8) TULPA(9) A...	0
39	39SUSI BITE		ABANCO(TULPA(14)	0
40	4065 SENCILLO		SPREIMATEC N(2) CHOCO CONO(8)	0
41	4165 DOBLE		SPREIMATEC N(2) CHOCO CONO(8)	2
42	4275 SENCILLO		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
43	4375 DOBLE		SPREIMATEC N(2) CHOCO CONO(8)	0
44	44NORVAL		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
45	455N ALICAR		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
46	46CACAO		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	0
47	47SPO		SPREIMATEC N(2)	0
48	485N SANDO		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	4
49	49CON ALMORON		SPREIMATEC N(2) SPREIMATEC S(2) SPREIMATEC N(2) SPREIMATEC...	4

3.5. Cliente impresión

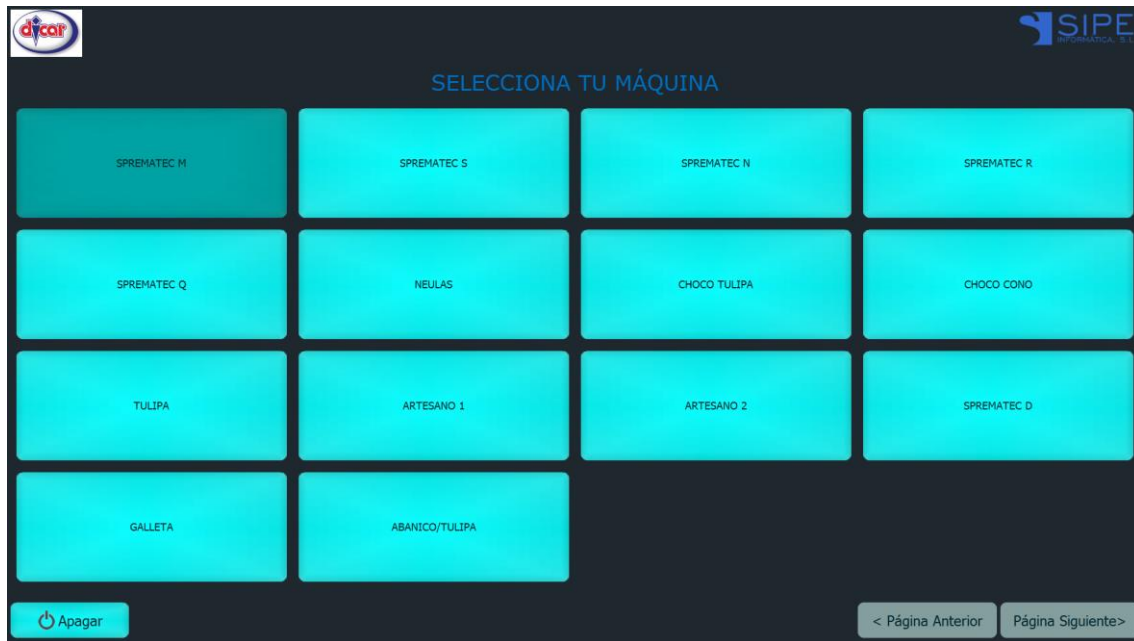
El cliente de impresión debe ser una interfaz pensada para manipularse fácilmente con los **dedos**. Sin embargo, Swing, la librería para programar interfaces que hemos usado hasta ahora, ofrece componentes de escritorio únicamente, ¿verdad? Así que deberíamos cambiarnos y usar Java FX u otra librería similar. O quizá no.

Si ahondamos un poco en la librería Swing, descubriremos que hay uno de sus *look-and-feels* que permite cambiar prácticamente todo el aspecto de toda la interfaz. Absolutamente todo lo que queramos. En un estilo parecido a como lo hace Java FX, que es con CSS, **Synth** (7) utiliza unos ficheros **.xml** que especifican la apariencia de los componentes de nuestra aplicación. Algo que nos da la potencia que necesitamos sin tener que utilizar una nueva tecnología distinta, aunque Java FX no habría sido en absoluto una mala solución.

Al principio busqué algunos **diseños** de interfaces para inspirarme en qué pinta debía tener y usé estas imágenes para el diseño:



Empece a estilar botones redondeados, gradientes y colores neon. Tenía esta pinta, estando a medias:



Sin embargo, no gustó mucho este diseño y despues de algunas discusiones, al final terminé usando los colores tal cual nos habían enviado en los **mockups**. Con lo que el programa al final tiene esta apariencia:



Interpretaciones artísticas aparte, el cambio entre estos dos estilos se hace simplemente **cambiando** el fichero synth.xml.

En lo que respecta al diseño de la interfaz y su programación, esta también hace uso de **paginación** y **reutiliza** varios componentes, como el panel selector de botones. El flujo es más sencillo y es tal cual he presentado en el análisis de requerimientos. Tiene algunas cosas interesantes como el teclado en pantalla, que emula la pulsación de las teclas. El programa está pensado para **cubrir** por completo y **eclipsar** la interfaz del sistema operativo y se incluyó un botón de apagado.

3.6. Infraestructura

Además del programa, deben tenerse en cuenta una serie de cosas en la **infraestructura** para que funcione correctamente.

El cliente de impresión debe **iniciarse** nada más arranca el equipo. En los equipos de impresión no debe haber un **teclado** físico conectado, excepto por razones de mantenimiento.

La red debe ser preferiblemente **cableada**. Los equipos deben tener instalado *Java Runtime Environment* versión ≥ 1.7 . El sistema operativo debe ser *Windows 7* o superior. Los equipos deben tener instalado *Bartender*. Se instalarán los scripts de inicio automático y limitado del sistema operativo.

Es necesaria una pantalla táctil que emule el comportamiento de un ratón y unos hubs USB con suficientes puertos para que los operarios introduzcan sus llaves USB. Debe configurarse la unidad de red en que los ficheros de formatos de etiqueta de *Bartender* residen.

El servidor debe tener MySQL versión ≥ 5.6 .

4. Resultado

El resultado es el siguiente programa que muestro ahora en capturas. En total el tiempo de desarrollo fue de aproximadamente 320 horas.

Cliente de impresión

Selección de máquina:



Selección de categorías:





SPREMATEC N - NORMAL



SELECCIONA LA CATEGORÍA 2

STANDARD

Volver a empezar selección

No lo sé/Ninguna

< Página Anterior

Página Siguiente>



SPREMATEC N - NORMAL - STANDARD



SELECCIONA LA CATEGORÍA 3

CAJA 720

Volver a empezar selección

No lo sé/Ninguna

< Página Anterior

Página Siguiente>

Selección de cliente:

dicor

SPREMATEC N - NORMAL - STANDARD - CAJA 720

SIPE INFORMATICA, S.L.

SELECCIONA EL CLIENTE

Volver a empezar selección

No lo sé/Ninguno < Página Anterior Página Siguiente>

Selección de artículo:

dicor

SPREMATEC N - NORMAL - STANDARD - CAJA 720

SIPE INFORMATICA, S.L.

SELECCIONA EL ARTÍCULO

(CONO STD PLATA (252922))

Volver a empezar selección

< Página Anterior Página Siguiente>

Panel de impresión:

Teclado en pantalla:

Ficha de artículo, información básica:

Nombre Etiqueta*: ESTUJO 10 NEULAS CACAO SELECCION C/33 Est.
Descripción Operario: ESTUJO 10 NEULAS CACAO SELECCION C/33 Est.
Código*: 140001
Núm. Producto*: 30
Núm. de familia: 140
EAN 13 (GTIN/EAN13)*: 8422199020034
Cajas por paquete*: 50 Unidades (Etiqueta)*: 33 Estuches* Unidades (Producción)*: 33 Estuches*
Cliente*: DICAR
Cajabilidad (en días)*: 140 Número habitual de operaciones: 0

Guardar Cancelar (Los campos con * son obligatorios)

Ficha de artículo, pestaña de etiquetas:

Personalizar Etiqueta
Nombre Cliente
Dirección Logo
Etiqueta de caja Imprimir una etiqueta de caja cada impresiones de etiqueta de artículo
EAN13 Caja* Uds (Etiqueta Caja)*
Imprimir una etiqueta de 2 idiomas
Imprimir dos etiquetas, dos idiomas en cada una
Imprimir una etiqueta por idioma
Imprimir etiqueta con Bar-tender
Ver Etiqueta Ver Etiqueta Caja La etiqueta es correcta
Guardar Cancelar (Los campos con * son obligatorios)

Ficha de artículo, previsualización de etiqueta:

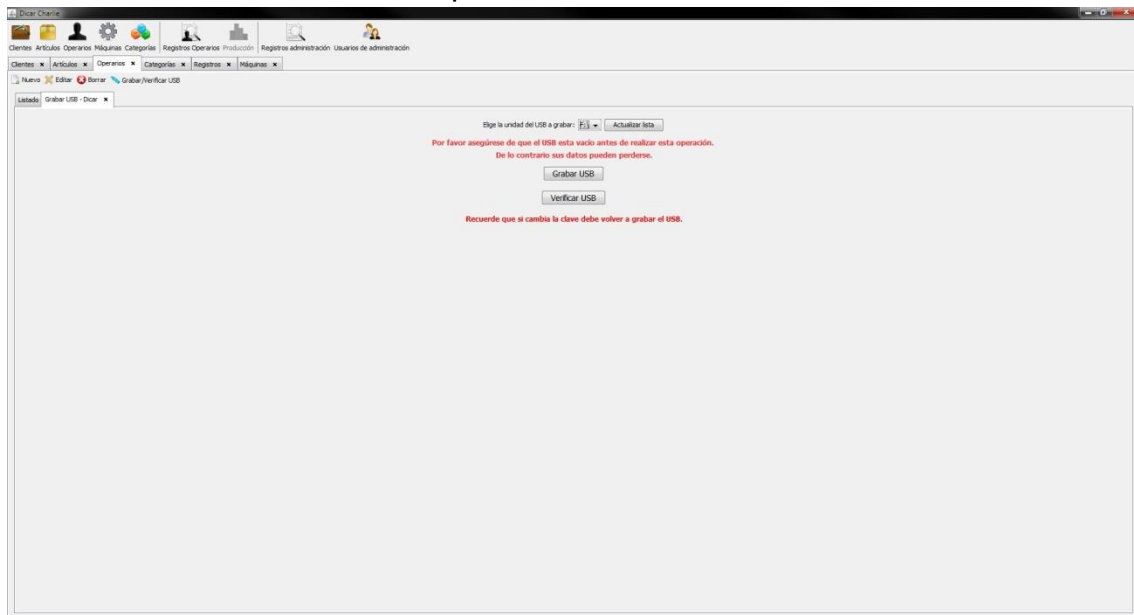
Listado de registros de operario:

Registro	Tipo	Emisadores	Fecha	Maquina	Lote	Tiempo para	Motivo para	ID Artículo	Cantidad Producción	Código de barras
6	Impresión	Prueba (2)	05/09/2016 09:40:36	SPRENATEC N	091605			1	720.0	0108421990023061009160591...
5	Impresión	Prueba (2)	05/09/2016 09:40:34	SPRENATEC N	091605			1	720.0	0108421990023061009160591...
4	Impresión	Prueba (2)	05/09/2016 09:40:33	SPRENATEC N	091605			1	720.0	0108421990023061009160591...
3	Impresión	Prueba (2)	05/09/2016 09:40:32	SPRENATEC N	091605			1	720.0	0108421990023061009160591...
2	Falso	Prueba (2)	05/09/2016 09:41:34	SPRENATEC N		00:00:42	Falso mecánico			
1	Alta operario	Prueba	05/09/2016 09:40:08	SPRENATEC N						

Listado de registros de administración:

Fecha	Usuario	Acción
05/09/2016 18:07:46	Sara	Modificar articulo [ID]=42, Nombre=TRAYALETA 42 mm TODA CHOCOLATE
05/09/2016 18:07:50	Sara	Modificar articulo [ID]=41, Nombre=TRAYALETA 42 mm TODA CHOCOLATE
05/09/2016 18:06:55	Sara	Modificar articulo [ID]=60, Nombre=SUPER NELA CHOCOLATE GRAVEL C/200 UNDS
05/09/2016 18:04:54	Sara	Modificar articulo [ID]=89, Nombre=TULIPA FLORES GRANDE CHOCO TOTAL ANONIMA
05/09/2016 18:04:56	Sara	Modificar articulo [ID]=87, Nombre=TULIPA MEDIANA CHOCO INTERIOR CORONA ANONIMA
05/09/2016 18:04:15	Sara	Modificar articulo [ID]=88, Nombre=TULIPA MEDIANA TODA CHOCOLATE OSCAR
05/09/2016 18:03:49	Sara	Modificar articulo [ID]=87, Nombre=TULIPA MEDIANA CHOCO INTERIOR CORONA ANONIMA
05/09/2016 17:44:27	Sara	Modificar articulo [ID]=86, Nombre=TULIPA BARCA CHOCOLATE INTERIOR Y CORONA
05/09/2016 17:44:02	Sara	Modificar articulo [ID]=85, Nombre=TULIPA FREQUENA FLORES CHOCO TOTAL OSCAR C/225 UNDS
05/09/2016 17:43:14	Sara	Modificar articulo [ID]=84, Nombre=MOLSA 1 LOGI NELLON CHOCO 94G
05/09/2016 17:43:42	Sara	Modificar articulo [ID]=81, Nombre=TULIPA FREQUENA CHOCO INTERIOR CORONA C/ANONIMA
05/09/2016 17:43:12	Sara	Modificar articulo [ID]=80, Nombre=TULIPA FREQUENA CHOCO INTERIOR CORONA C/ OSCAR
05/09/2016 17:41:27	Sara	Modificar articulo [ID]=79, Nombre=ESTUQUE ORLEAS CHOCOLATE
05/09/2016 17:40:50	Sara	Modificar articulo [ID]=78, Nombre=CUBANITO 5 CM CHOCOLATE C/175G
05/09/2016 17:40:04	Sara	Modificar articulo [ID]=75, Nombre=ESTUQUE 15 NELLAS C/CAJAS
05/09/2016 17:38:35	Sara	Modificar articulo [ID]=72, Nombre=CAJA TULIPA 144 UNDS BLISTER INT COB. BK
05/09/2016 17:38:14	Sara	Modificar articulo [ID]=72, Nombre=CAJA TULIPA 144 UNDS BLISTER INT COB. BK
05/09/2016 17:36:39	Sara	Modificar articulo [ID]=72, Nombre=CAJA TULIPA 144 UNDS BLISTER INT COB. BK
05/09/2016 17:34:46	Sara	Modificar articulo [ID]=71, Nombre=CHOCO CHOCOLATE BK CORONA C/2 ANONIMA
05/09/2016 17:34:02	Sara	Modificar articulo [ID]=70, Nombre=CHOCO NEGRO Easy Oper (5402372)
05/09/2016 17:33:13	Sara	Modificar articulo [ID]=69, Nombre=CHOCO NEGRO 65 EXT LEONIE Easy Oper (5402363)
05/09/2016 17:33:17	Sara	Modificar articulo [ID]=68, Nombre=ESTUQUE 18 BARROTES CHOCOLATE CALA TB (BK)
05/09/2016 17:31:00	Sara	Modificar articulo [ID]=47, Nombre=ESTUQUE 6 TULIPAS BARCA CHOCO OSCAR
05/09/2016 17:30:23	Sara	Modificar articulo [ID]=46, Nombre=ESTUQUE 6 TULIPAS CHOCO SAN RAFAEL 900/300G
05/09/2016 17:29:31	Sara	Modificar articulo [ID]=45, Nombre=TULIPA MEDIANA TODA CHOCOLATE ANONIMA
05/09/2016 17:28:01	Sara	Modificar articulo [ID]=43, Nombre=TULIPA MEDIANA CHOCO INTERIOR CORONA
05/09/2016 17:03:39	Sara	Modificar articulo [ID]=41, Nombre=MOLSA 2 OREANAS
05/09/2016 17:03:15	Sara	Modificar articulo [ID]=38, Nombre=TULIPA MEDIANA FLORES CHOCO TOTAL OSCAR C/200 UNDS
05/09/2016 17:03:17	Sara	Modificar articulo [ID]=37, Nombre=TULIPA FREQUENA FLORES CHOCO TOTAL OSCAR C/200 UNDS
05/09/2016 17:04:40	Sara	Modificar articulo [ID]=56, Nombre=ESTUQUE CUBANITOS SUR
05/09/2016 17:04:08	Sara	Modificar articulo [ID]=51, Nombre=ESTUQUE SUPER NELLAS CHOCOLATE C/14 EST
05/09/2016 17:03:32	Sara	Modificar articulo [ID]=50, Nombre=ESTUQUE SUPER NELLAS CHOCOLATE C/14 EST
05/09/2016 17:03:09	Sara	Modificar articulo [ID]=49, Nombre=ESTUQUE NELLAS BLACKSWAN
05/09/2016 17:02:45	Sara	Modificar articulo [ID]=48, Nombre=ESTUQUE NELLAS BLACKSWAN
05/09/2016 17:01:58	Sara	Modificar articulo [ID]=47, Nombre=ESTUQUE 12 NELLAS CALCAO SEL SECCION C/14 EST
05/09/2016 17:01:51	Sara	Modificar articulo [ID]=46, Nombre=ESTUQUE CUBANITOS SUR
05/09/2016 17:00:40	Sara	Modificar articulo [ID]=45, Nombre=ESTUQUE 12 NELLAS CALCAO SELECCION C/14 EST
05/09/2016 17:00:05	Sara	Modificar articulo [ID]=44, Nombre=CAJA 200 G TULIPAS NEGROTERIA VANILLA
05/09/2016 16:58:43	Sara	Borrar articulo [ID]=40, Nombre=TULIPA MEDIANA CHOCO INTERIOR CORONA 144 CAJAS BK
05/09/2016 16:57:17	Sara	Modificar articulo [ID]=39, Nombre=BOITE BARO ARTESANO 175 G
05/09/2016 16:55:30	Sara	Modificar articulo [ID]=38, Nombre=ESTUQUE NELLAS OSCAR SELECCION C/14 EST (BK)
05/09/2016 16:54:21	Sara	Modificar articulo [ID]=35, Nombre=ESTUQUE 6 SUPER NELLAS 50 Gramos C/14 EST (BK)
05/09/2016 16:53:54	Sara	Modificar articulo [ID]=35, Nombre=ESTUQUE 6 SUPER NELLAS 50 Gramos C/14 EST (BK)
05/09/2016 16:53:46	Sara	Modificar articulo [ID]=35, Nombre=ESTUQUE 6 SUPER NELLAS 50 Gramos C/14 EST (BK)
05/09/2016 16:53:01	Sara	Modificar articulo [ID]=17, Nombre=ESTUQUE 12 SUPER NELLAS LA SENIA C/12 EST (BK)
05/09/2016 16:52:11	Sara	Modificar articulo [ID]=35, Nombre=EST 12 SUPER NELLAS LA SENIA C/12 EST (BK)
05/09/2016 16:50:57	Sara	Modificar articulo [ID]=15, Nombre=ESTUQUE 12 NELLAS VADSL
05/09/2016 16:50:19	Sara	Modificar articulo [ID]=14, Nombre=ESTUQUE NELLAS RIZO 96
05/09/2016 16:49:11	Sara	Modificar articulo [ID]=13, Nombre=ESTUQUE NELLAS OSCAR SELECCION 78 EST (BK)
05/09/2016 16:48:48	Sara	Modificar articulo [ID]=13, Nombre=ESTUQUE NELLAS OSCAR SELECCION 78 EST (BK)
05/09/2016 16:48:42	Sara	Modificar articulo [ID]=13, Nombre=ESTUQUE NELLAS OSCAR SELECCION 78 EST (BK)
05/09/2016 16:47:55	Sara	Modificar articulo [ID]=2, Nombre=ESTUQUE 15 NELLAS VADSL
05/09/2016 16:46:06	Sara	Modificar articulo [ID]=1, Nombre=CHOCO STD PLATA (250X22) C/750

Grabado de llaves USB de operarios:



5. Conclusiones y trabajos futuros

Esto ha sido un proyecto ambicioso con muchos factores, del que se esperan muchas cosas y no solo quedarse aquí. Una de las cosas más importantes que me ha enseñado es que a veces no es mejor un programa con un diseño muy elaborado y con mucha reusabilidad de componentes. A veces es mejor escribir un programa con piezas que sean fáciles de borrar. Hay cambios, siempre hay cambios, a mitad del desarrollo. No hay nada que se pueda hacer para combatirlo, lo único que queda es adaptarse a ello. Bien con metodologías de desarrollo ágil, que también hay que decirlo, suponen un coste temporal mayor, o simplemente previendo el cambio y haciendo el mejor diseño que podamos. Y aun así, siempre hay modificaciones aun cuando has terminado. Así que sin más, a borrar y que así sea, mientras nos paguen por nuestro tiempo.

Es un proyecto muy interesante porque ha sido uno de mis primeros proyectos grandes. Hay algunas cosas que podrían haber sido mejores, pero el proyecto debe crecer más, así que habrá espacio para mejorarlas. Cosas como la apariencia visual de la interfaz, la habilidad para deshacer cambios, la falta de paginación en algunas tablas. Dicarcono tiene al menos otras dos aplicaciones separadas a esta, una de control de pesos de balanzas y una aplicación de control de entrada y salida de palets en el almacén, que piden a gritos ser integradas con esta en un gran programa que informaticice y permita el control de todo lo que sucede en la fábrica.

Es un programa personalizado y completamente hecho a medida, hasta el más mínimo detalle, encajado a las necesidades y situación de la empresa. Mi esperanza es que le den buen uso y les suponga un beneficio importante. Al menos tanto como lo ha sido para mí profesionalmente.

6. Bibliografía.

1. **GS1**. GS1 General Specifications Document. [En línea] Enero de 2016. http://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf.
2. **Raj, A S Bhaskar**. *Bar Codes Technology and Implementation*. s.l. : Tata McGraw-Hill, 2001. 0-07-463849-1.
3. **AECOC**. Guía de Trazabilidad de Productos Envasados. [En línea] 2010. http://www.aecoc.es/admin/web/documento_socio/Gu%C3%ADa%20trazabilidad%20productos%20envasados%202010.pdf.
4. **Evans, Ben**. Understanding Java JIT Compilation with JITWatch. [En línea] Julio de 2014. <http://www.oracle.com/technetwork/articles/java/architect-evans-pt1-2266278.html>.
5. **Oracle**. Java Platform, Enterprise Edition: The Java EE Tutorial. Persistence. [En línea] 2014. <https://docs.oracle.com/javaee/7/tutorial/partpersist.htm>.
6. —. Java Native Interface Specification. [En línea] <https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html#wp9502>.
7. —. The Synth Look and Feel. [En línea] <https://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/synth.html>.

7. Anexos

synth.xml

```
<synth>
  <style id="panelStyle">
    <opaque value="true"/>
    <font name="Verdana" size="32"/>
    <state>
      <color value="#1F282F" type="BACKGROUND"/>
      <color value="#F6FAFD" type="FOREGROUND"/>
    </state>
  </style>
  <style id="labelStyle">
    <state>
      <color value="#00272E" type="FOREGROUND"/>
    </state>
  </style>
  <style id="buttonStyle">
    <state value="PRESSED">
      <imagePainter method="buttonBackground"
path="img/bot2.png"
      sourceInsets="13 13 13 13"/>
    </state>
    <state value="ENABLED">
      <color value="#00272E" type="FOREGROUND"/>
      <imagePainter method="buttonBackground"
path="img/bot1.png"
      sourceInsets="13 13 13 13"/>
    </state>
    <state value="DISABLED">
      <imagePainter method="buttonBackground"
path="img/bot3.png"
      sourceInsets="13 13 13 13"/>
    </state>
    <state>
      <font name="Verdana" size="24"/>
    </state>
    <opaque value="true"/>
  </style>
  <bind style="panelStyle" type="region" key=".*"/>
  <bind style="labelStyle" type="region" key="Label"/>
  <bind style="buttonStyle" type="region" key="Button"/>
</synth>
```

run.bat

```
@echo off
:loop
taskkill /IM explorer.exe /F
C:
cd C:\Charlie
"C:\Program Files (x86)\Java\jre1.8.0_40\bin\javaw.exe" -jar
CharlieCliente.jar
goto loop
```