UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

APLICACIÓN PARA EL CONTROL DE UNA PLANTA DE ELABORACIÓN DE CERVEZA

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Paula Berlanga López

TUTORIZADO POR

Juan Carlos Martínez González

CURSO ACADÉMICO: 2019/2020

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Resumen

A partir de una planta de fabricación de cerveza ya automatizada se pretende realizar una aplicación móvil de monitorización y control para el proceso de fabricación de la cerveza. Este proyecto se va a centrar en el desarrollo en una versión prototipo de la aplicación en el que la planta de fabricación se simulará, en base a una planta de fabricación real.

Con ella los empleados tendrán mayor flexibilidad a la hora de conocer y gestionar la información más relevante de la planta, es decir, les permitirá tratar la información de forma remota en cualquier momento. La aplicación recogerá y controlará las distintas variables del proceso en tiempo real de forma fácil y sencilla.

Algunos de los parámetros más importantes a controlar son la acidez, cantidad de agua necesaria (ya que es el ingrediente más importante en la elaboración de los distintos tipos de cerveza), la temperatura de maderación, cocción y el resto de los procesos, así como los tiempos necesarios en cada proceso anterior y su posterior fermentación.

Además, se podrá almacenar los datos de los procesos y crear historiales e informes útiles para la mejora del proceso de fabricación y el rendimiento de esta.

Resum

A partir d'una planta de fabricació de cervesa ja automatitzada es pretén realitzar una aplicació mòbil de monitorització i control per al procés de fabricació de la cervesa. Este projecte es va a centrar en el desenrotllament en una versió prototip de l'aplicació en què la planta de fabricació se simularà, basant-se en una planta de fabricació real.

Amb ella els empleats tindran més flexibilitat a l'hora de conéixer i gestionar la informació més rellevant de la planta, és a dir, els permetrà tractar la informació de forma remota en qualsevol moment. L'aplicació arreplegarà i controlarà les distintes variables del procés en temps real de forma fàcil i senzilla.

Alguns dels paràmetres més importants a controlar són l'acidesa, quantitat d'aigua necessària (ja que és l'ingredient més important en l'elaboració dels distints tipus de cervesa) , la temperatura de fusteria, cocció i la resta dels processos, així com els temps necessaris en cada procés anterior i la seua posterior fermentació.

A més, es podrà emmagatzemar les dades dels processos i crear historials i informes útils per a la millora del procés de fabricació i el rendiment d'esta.

Abstract

From an already automated beer manufacturing plant, a mobile monitoring and control application is intended for the beer manufacturing process. This project will be focused on the development of a prototype version of the application in which the manufacturing plant will be simulated, based on a real manufacturing plant.

With it, employees will have greater flexibility in knowing and managing the most relevant information of the plant: it will allow them to process the information remotely at any time. The application will collect and control the different process variables in real time in an easy and simple way.

Some of the most important parameters to control are the acidity, quantity of water necessary (since it is the most important ingredient in the elaboration of the different types of beer), the temperature of brewing and cooking, as well as the necessary times in each previous process and its subsequent fermentation.

In addition, we can store the process data and create useful records and reports for the improvement of the manufacturing process and its performance.

1. Introducción

1.1. Motivación

La elección del tema para este TFG, nace de una motivación personal que tiene dos funciones.

Por un lado, la redacción de este TFG me permite poner en práctica los conocimientos en programación adquiridos durante el tercer año del grado gracias a distintas asignaturas que cursé durante la especialidad de informática. La programación también despierta gran interés en mi al igual que los dispositivos móviles, además se debe tener en cuenta que los dispositivos móviles están teniendo un gran crecimiento de consumo para todo tipo de ámbitos.

Otra de las menciones o especialidades que me gusta del grado que he cursado es la parte de automática por ello que este tema tenga que ver con un proceso industrial automatizado.

Por otro lado, me permite profundizar en el aprendizaje sobre de la elaboración de la cerveza. Gracias a un familiar que trabaja en la elaboración de cerveza he podido llevar a cabo mis conocimientos previos sobre esta y ampliarlos con su ayuda para poder desarrollar con mayor exactitud la aplicación.

Es por todo esto por lo que he decidió realizar la primera fase del desarrollo de una aplicación para el control y supervisión de un proceso industrial como es la elaboración de la cerveza para hacer más simple y fácil la forma de controlar el proceso desde cualquier lugar. Por lo que he podido investigar este tipo de aplicaciones no están para dispositivos Android y es por ello por lo que he decidido utilizar esta plataforma ya que es el sistema operativo más usado actualmente.

1.2. Objetivos y alcance del sistema

El objetivo del proyecto es el desarrollo de un prototipo de aplicación móvil en la plataforma Android. Esta podría ser distribuida a través de Play Store, aunque en este proyecto no tiene mucho sentido ya que se trata de un proyecto desarrollado para una empresa privada. Se podría instalar a través de un archivo APK (Android Application Package), es decir, no es necesario tenerla en una tienda virtual mientras que si se trata de una aplicación para IOS sí que es necesario. Estos archivos pueden compartirse entre móviles Android lo que facilita la distribución entre los trabajadores de la empresa. Son archivos ejecutables para Android.

Esta aplicación tiene como objetivo supervisar el estado de cada proceso de una planta de fabricación de cerveza, así como el control de variables importantes de cada proceso en tiempo real. Esta aplicación servirá exclusivamente para aquellos que tengan acceso permitido a la misma, en este caso, trabajadores o gerentes de la empresa.

Los objetivos específicos de este proyecto son los siguientes:

- Permitir al usuario identificarse de forma segura con sus credenciales de la empresa.

- Proporcionar información sobre las variables más importantes de cada etapa del proceso y el control de alguna de ellas.
- Mostrar las temperaturas del agua y del mosto durante las etapas de elaboración, así como los tiempos necesarios para cada etapa del proceso.
- Controlar la cantidad de materias primas existentes en la fábrica y notificar si es insuficiente alguna de ellas.
- Crear un historial con los lotes de cerveza que se han producido.
- Notificar cuando exista algún incidente en alguno de los sensores o procesos de la fábrica.
- Proporcionar la información sobre las recetas disponibles en la fábrica.
- Mantener un rendimiento y un tiempo de respuesta de la aplicación decente.
- Hacer que el uso de la aplicación sea cómodo en todo momento.
- Poder visualizar el estado del proceso de elaboración desde cualquier lugar.
- Almacenar datos de forma fácil y sencilla.

2. Elaboración de la cerveza

2.1. Origen e historia de la cerveza

En base a la publicación *"Breve historia del origen de la cerveza"* [1] la existencia de la cerveza es probable que sea tan antigua como el origen del hombre.

El cultivo de la cebada se data aproximadamente en el año 9000 a.C. Se han encontrado evidencias históricas de la cerveza en antiguas civilizaciones de los egipcios y los sumerios.

Los sumerios elaboraban una especie de cerveza que llamaban "bebida fuerte" con extracto de cebada. Se sabe de su existencia gracias a unas pequeñas tablas de arcilla donde se mencionaba tal licor. Se cree que los sumerios fueron quienes por casualidad dejaron germinar algunos granos de cebada y al combinarlos con agua y levaduras salvajes dieron como resultado una incipiente cerveza.

Los conocimientos de la elaboración de la cerveza se fueron ampliando. Hasta el punto de que elaboraban diferentes tipos de cerveza.

La primera cerveza conocida se encuentra en el Código de Hamurabi. En el Museo Británico se encuentran dos piedras grabadas con más de 5000 años de antigüedad donde se puede ver una ofrenda de cerveza a la diosa Nin-Harra. Se trata del Monumento azul.

Figura 1: *Monumento Azul.* Fuente: [1]

Los egipcios también bebían un licor a base cebada que llamaban “zytum”. Los trabajadores de las pirámides cobraban parte de sus salarios en cerveza. Los egipcios exportaron la cerveza a los griegos y estos a su vez la exportaron a los romanos, estos a los galos y germánicos.

Los galos y germánicos fueron quienes empezaron a elaborar la cerveza con malta de cebada y avena. Para dar aroma usaban comino. Los galos dan al licor que elaboraban el nombre de “cerevisa” en honor al Dios Ceres. De ahí el nombre de origen latino de “cerveza” que se conoce ahora.

Durante la Edad Media las Abadías las guardianas de la cerveza fueron mejorando la calidad de las recetas. Los monjes lograron mejorar el aspecto, el aroma y el sabor de las cervezas que elaboraban.

Hacia el siglo XV se empieza a utilizar el lúpulo como aromatizante. Observaron que la cerveza a la que se le añadía lúpulo se conservaba durante mucho más tiempo. Habían encontrado un conservante natural debido a las propiedades antisépticas que posee.

El 23 de abril de 1516 el Emperador Guillermo IV de Babiera decreta la Ley de Pureza donde decreta que la cerveza solamente se debía elaborar a partir de tres ingredientes: agua, malta de cebada y lúpulo. Todavía no se conocía el ingrediente fundamental, la levadura.

Es en el siglo XVIII cuando la cerveza consigue su gran expansión. Cerveceros bárbaros almacenaban sus cervezas en sótanos y cuevas a los que llamaban Lagern (almacén) donde dejaban que la cerveza madurara lentamente con temperaturas constantes, normalmente 9 grados. Utilizaban levaduras de baja fermentación.

A finales del siglo XIX se comenzó a industrializar la cerveza. Los principales centros productores en Europa eran Burton-onTrent en Inglaterra, München en Alemania y Pilsen en Bohemia. Cada uno tenía un estilo propio.

En el siglo XX la cerveza nace con una vocación global de ser industrializada. Aparecen las grandes compañías cerveceras que fagocitan a las pequeñas industrias artesanales.

En el siglo XXI parece que esta tendencia se está invirtiendo, y asistimos al florecimiento de la pequeña industria cervecera con carácter artesano que da a la cerveza un toque de calidad y exclusividad que había perdido.

2.2. Materias primas

Las cantidades varían en función del tipo de cerveza que se ha de elaborar, pero en general se considera que, por cada litro de cerveza, se necesitan:

- 175 gramos de cebada.
- entre 1,5 y 3 gramos de lúpulo.
- un gramo de levadura.
- 6 gramos de azúcar común (hacia el final del proceso).

2.2.1. Agua

La materia prima utilizada en mayor proporción en la elaboración de la cerveza es el agua por lo que es necesario que las industrias cerveceras dispongan de un suministro abundante y contante de este líquido. Es por ello por lo que las fábricas históricamente se situaban cerca de manantiales o a las orillas de los ríos.

Alrededor del 90% del contenido de la cerveza es agua y su composición química influye de forma directa en las propiedades sensoriales de la cerveza, se emplea a lo largo de todo el proceso de fabricación. Aunque solamente una parte de la cantidad de agua es usada directamente en la cerveza, mientras que la otra parte se utiliza en el remojado de la germinación, en el macerado, en la limpieza del equipo, etc.

El tipo de agua utilizado en la elaboración es determinante en la calidad de la cerveza y las condiciones del agua empleada condicionaran por ejemplo los tiempos de operación en las diferentes etapas. El agua que se utiliza para la elaboración de cerveza tiene que ser un agua pura, potable, libre de sabores y olores, sin exceso de sales y exenta de materia orgánica [2]

Para la elaboración de cervezas más ligeras como puede ser la Pilsen se utilizan aguas blandas que se caracterizan por tener una baja concentración mineral, comúnmente bajos niveles de socio, calcio y magnesio. Este tipo de aguas retrasan el proceso de crecimiento de la levadura y por lo tanto ralentiza el proceso. Estas cervezas son de baja fermentación.

Las cervezas oscuras como Porters, en cambio, se elaboran con aguas más duras. Estas aguas son las preferidas para la elaboración de la cerveza, sobre todo si son ricas en calcio, magnesio y bicarbonato, ya que producen un pH más ácido que potencia la acción enzimática y no disuelve los polifenoles que contribuyen a dar sabor a la cerveza. Los altos niveles de bicarbonato ayudan a que las levaduras puedan llevar a cabo el proceso de fermentación en mejores condiciones.

El pH mide la acidez o alcalinidad de una disolución, lo que influye directamente en el proceso de fermentación y, por lo tanto, en el sabor y características finales de la cerveza.

La dureza se mide en base a la concentración de compuestos minerales presentes en una cierta cantidad de agua, en particular sales de magnesio y calcio. La cantidad de estas también influye en el sabor y propiedades del producto final.

Se estima que para producir 1 litro de cerveza en una fábrica moderna se consumen entre 3 y 5 litros de agua. En los últimos años este consumo se ha reducido drásticamente y en la actualidad la industria cervecera tiene como objetivo reducir el consumo de este preciado recurso.

A continuación, se muestra una tabla que muestra los valores máximos y mínimos de las concentraciones de los iones químicamente activos, adecuados para la elaboración de cerveza:

Ión químicamente activo	Valor mínimo (ppm)	Valor máximo (ppm)
Calcio (Ca^{+2})	50	150
Magnesio (Mg^{+2})	10	30
Potasio (K^{+})	5	10
Sodio (Na^{+})	5	150
Bicarbonato/Carbonato	0	250
Cloruro (Cl^{-})	0	250
Sulfato (SO_4^{-2})	10	250

Tabla 1: Valores máximos y mínimos adecuados de los iones del agua cervecera. **Fuente:** John Palmer [29]

2.2.2. Cebada

De los ingredientes de la cerveza, la cebada es sin duda el principal. Es el más rico en almidón y posee proteínas suficientes para proporcionar el alimento necesario para el desarrollo de la levadura[3].

La cebada ha tenido distintos usos a lo largo de la historia, como la fabricación del pan, el tostado para hacer infusiones e incluso para la preparación de sopas y yogures. Es una planta que pertenece a la familia de las gramíneas “Hordeum vulgare” que desciende de la cebada silvestre “Hordeum spontaneum”, la cual crece en el Oriente Medio. Se cosecha a finales de primavera y crece bien en suelos drenados y fértiles.

Para que la cebada sea óptima para la elaboración de la cerveza debe tener una serie de características físicas y bioquímicas adecuadas para la cerveza. Se utilizan las denominadas cebadas cerveceras aptas para ser malteadas.

Físicas:

- Un grano grueso y de tamaño uniforme
- Color amarillo claro
- Cascarilla fina y rizada
- Libre de infecciones de microorganismos

Bioquímicas:

- Bajo periodo de letargo
- Buena absorción de agua
- Capaz de germinar uniformemente y en un tiempo mínimo.

A continuación, se pueden observar las diferentes partes de un grano de cebada:

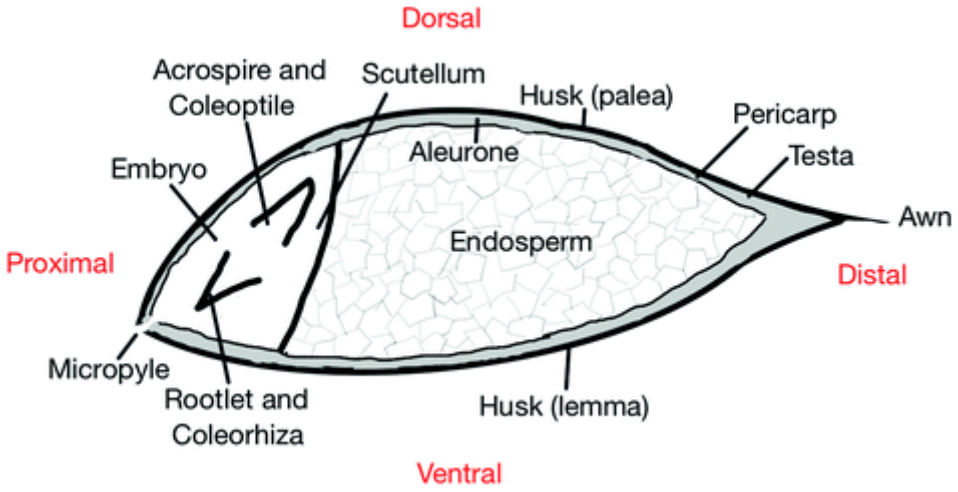

Figura 2: *Estructura del grano de cebada* [4]

Las reservas de almidón se encuentran en el endospermo del grano, estas son utilizadas para obtener los azucares aptos para ser fermentados. En la maceración se utilizan las enzimas alfa y beta amilasa que se producen en el micrópilo gracias a las raíces que se producen en este. Además, con la molturación o molienda se pretende romper la cascarilla la cual va desde la barba (Awn) hasta el micrópilo (Micropyle).

2.2.3. Lúpulo

El lúpulo es un ingrediente esencial para la elaboración de la cerveza. Este posee un leve efecto antibiótico contra bacterias[5].

El lúpulo (Humulus lupulus) es una planta trepadora y perenne. Siendo una especie dioica, las flores femeninas y masculinas surgen en plantas separadas. El lúpulo hace que la espuma de la cerveza sea más estable, conserva su frescor, es la causa del apetito que produce la cerveza y le confiere otras propiedades

El cultivo de lúpulo se realiza en zonas con condiciones para ello. Despues de la cosecha se realiza el secado y preparado para poder almacenarlos y para evitar pérdidas de valor.

Dentro del proceso de elaboración la función principal del lúpulo es dar un tenue sabor amargo para contrarrestar el sabor dulce de los azúcares de los cereales. También contribuye a eliminar los microorganismos del proceso de elaboración de la cerveza

Existen numerosas clases de lúpulo (más de 200 tipos), unos se concentran en otorgar el amargor a la cerveza, otros en dotarla de aromas y sabores y otros que asumen una función mixta de las dos primeras.

Figura 3: *Lúpulo. Fuente:*[5]

En resumen, el lúpulo es sustancial para determinar las propiedades organolépticas de la cerveza (sabor, olor, color, textura…).

2.2.4. Levadura

Son organismos unicelulares que transforman mediante fermentación los glúcidos y los aminoácidos de los cereales en alcohol etílico y dióxido de carbono.

En la industria cervecera se diferencian dos tipos de levadura, ambos tipos tienen diferentes características que afectan al sabor, el aroma y la sensación en boca de la cerveza terminada [6]:

- Levadura ale (Saccharomyces cerevisiae). Es una levadura de fermentación alta y trabaja a una temperatura de fermentación entre 15°C y 26°C. Esta cálida fermentación promueve la creación de subproductos que afectan al sabor y aroma de la cerveza de forma positiva.
  Corresponden con levaduras flotantes, es decir la fermentación ocurre en la superficie del mosto, las levaduras suben a la superficie en el transcurso de la fermentación. Esto implica que es más susceptible a especies invasoras o a sufrir alteraciones que pueden provocar mal sabor en la cerveza.

- Levadura lager (Saccharomyces pastorianus) también conocida como Saccharomyces calsbergensis. Se trata de una levadura de baja fermentación.
  Las levaduras de baja fermentación fermentan a bajas temperaturas, entre 7°C y 14°C.
  Son capaces de fermentar ciertas cadenas largas de azúcares que las levaduras ales no pueden fermentar.
  Las temperaturas bajas de fermentación inhiben la producción de ésteres y fenoles, dando a las cervezas un perfil limpio, sin notas especiadas o afrutadas derivadas de la levadura. El proceso de fermentación en este caso es más lento, por lo que requiere un condicionamiento mucho más largo.

Este tipo de levadura se deposita en el fondo y fermenta los azucares presentes en el mosto.

2.3. Proceso de fabricación de cerveza

El proceso cervecero siempre consta de una serie de pasos, por lo general estos pasos no varían, aunque las pequeñas variaciones aportan los diferentes matices de la cerveza. Las etapas de elaboración de la cerveza son las siguientes:

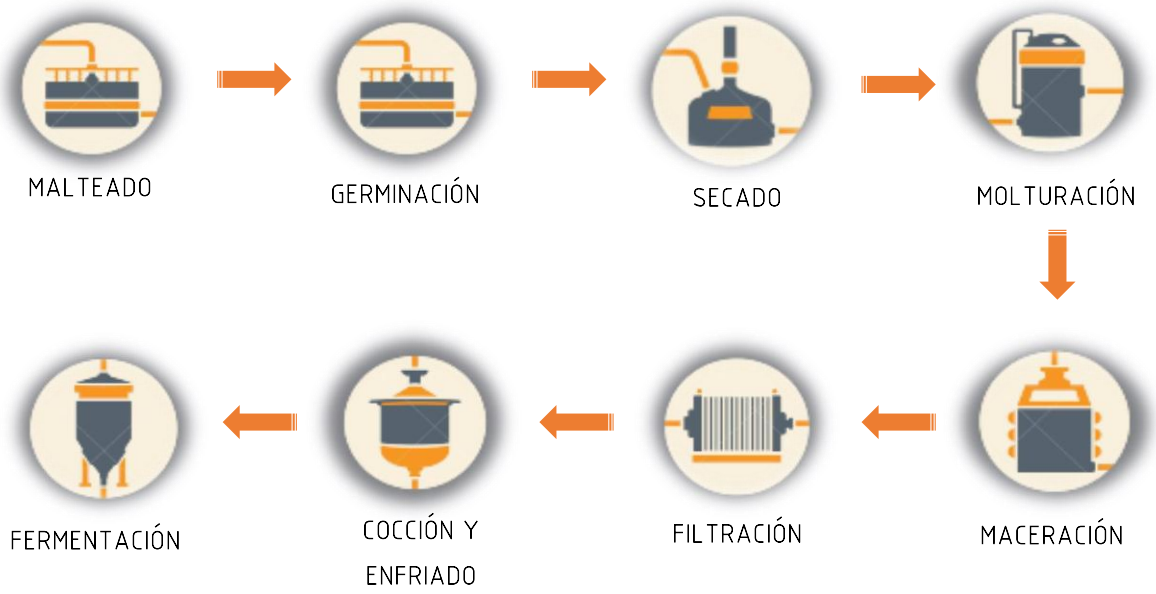

Figura 4: *Esquema elaboración cerveza*

2.3.1. Germinación y malteado

La primera fase del malteado se denomina remojado, es un proceso similar al de introducir té en agua caliente, pero en este caso el objetivo es conseguir que las semillas de cebada empiecen a crecer o germinen. Existen dos objetivos fundamentales en el remojado, lavar la cebada y aumentar su humedad para que comience el proceso de germinación. Se considera que la humedad final teórica que debe de adquirir el grano es de 45%, aunque debido a la lentitud de la etapa final de absorción de agua, esta generalmente queda por encima del 40%.

La humedad inicial del grano debe estar entre 11% y 13% para que pueda empezar el proceso. De no ser así, deberán ser secados antes de almacenarse.

La temperatura del agua con la cual se remoja la semilla no debe facilitar el desarrollo microrganismos. Suele hacerse con agua cuya temperatura este comprendida entre 15 y 20 °C.

Con el depósito de remojo lleno se inyecta aire para evitar el desarrollo de fermentaciones y la asfixia del grano.

A continuación, se realiza el vaciado del agua para una aspiración del anhidrido carbónico que va formándose y se pulveriza la superficie del lecho del grano con agua. Finalmente se vuelve a llenar el tanque hasta que se detecta que la humedad final es la adecuada.

La cebada permanece en remojo durante más de 24 horas y es transferido tras la fase de descanso a los enormes tanques de germinación, donde se reparte el grano de forma uniforme y extendida, con unas condiciones de humedad y ventilación adecuadas, para que el embrión comience a germinar y el grano empiece a sufrir su modificación.

Un grano de cebada con suficiente humedad, con oxígeno disponible y a la temperatura adecuada inicia un período de vida activa con numerosas modificaciones morfológicas, químicas y biológicas, denominado germinación. Las semillas producen las enzimas necesarias para convertir el almidón en azúcar durante el proceso de elaboración de la cerveza.

Esta fase tiene una duración aproximada de 5 días.

2.3.2. Secado o tostado

Las semillas se trasladan a un horno o estufa de malteado, allí el calor detiene bruscamente la germinación. Se remueve para que se forma una capa totalmente lisa para que circule el aire de manera homogénea.

El objetivo del secado es detener la germinación y modificar el color, el aroma y el sabor de la malta dependiendo de la cerveza que se va a elaborar.

Se retira la humedad del grano y se acaba toda la actividad enzimática. La humedad debe tener un nivel suficiente bajo. Si esto no ocurre, se quedará almidón sin transformar o el exceso de azúcar hará que el brote siga creciendo. Para retirar esta humedad se aplicará aire caliente durante un tiempo estimado de unos dos o tres días.

Este proceso tiene dos fases. La primera de ellas es la desecación a temperatura moderada en la cual la temperatura del aire tiene que estar entre unos 50 y 70 °C y la temperatura de la malta se encuentra entre 25 y 30 °C. La temperatura de la malta se aprovecha para la evaporación del agua.

La siguiente fase es la fase de calentamiento, cuando la humedad del grano es inferior al 10% la difusión del agua es cada vez más difícil y parte del calor aportado a la malta se emplea en aumentar la temperatura de esta.

La temperatura del aire se va aumentando y su valor depende del tipo de malta que se esté fabricando. Cuando se tratan de maltas pálidas la temperatura del aire debe ser de 80 °C durante 5 horas y para maltas negras debe estar entre 100 y 105 °C, también durante 5 horas. La temperatura de la malta es de unos 60-65 °C.

Las maltas negras se corresponden con el uso de levadura lager, mientras que para las maltas pálidas se usa levadura ale.

Este proceso se llevará a cabo hasta alcanzar una humedad de un 5%.

Una vez secado el grano, se procede a retirar las raíces y tallos que se pudieron desarrollar durante la germinación, y la malta ya se encuentra lista para su uso, aunque deben realizarse los

ensayos pertinentes que garanticen su calidad, y tras estos ensayos la malta ya es apta para la elaboración de cerveza.

2.3.3. Molturación de la malta

La molturación es un proceso de trituración mecánica, en el que las cascaras del grano deben ser separadas de su endospermo. Es importante llevar a cabo este proceso justo antes de mezclarla con agua en la maceración para evitar la oxidación de ácidos grasos.

Se puede realizar en seco o húmeda. Y por ello existen distintos tipos de molinos. En caso de fábricas pequeñas se suele realizar en seco, ya que son molinos económicos.

El tiempo de molturado será de aproximadamente 1 hora para unos 200 kg de malta. Por cada kilo de malta se obtienen aproximadamente 2 litros de cerveza para un empaste denso.

2.3.4. Maceración

Se trata del proceso más importante en la fabricación del mosto. La molienda se mezcla con agua (maceración).

No todos los componentes de la molienda son solubles. Pero a la cerveza solo pueden pasar sustancias solubles. Pero ello, es necesario que las sustancias insolubles sean convertidas en sustancias solubles durante este proceso. Por ejemplo, el almidón es una sustancia insoluble.

La maceración es el proceso mediante el cual, el grano se mezcla con agua a ciertas temperaturas para activar diversas enzimas de la malta, convirtiendo así los almidones en azucares fermentables que producirán alcohol. Estos azucares en un proceso posterior se verán metabolizados por la levadura.

Por otro lado, las enzimas son proteínas complejas que hacen de catalizadores, induciendo reacciones entre distintas sustancias. Estas enzimas se activan y desactivan en ciertas condiciones. Estas condiciones se manipulan en el proceso de macerado.

A continuación, se puede ver una tabla con los rangos óptimos de temperatura que activan las diferentes encimas.

Enzima	Rango Optimo de Temperatura	Rango Optimo de PH	Función
Fitasa	30 – 52°C	4.4 – 5.5	Baja el PH del Mosto. Actualmente no es utilizado.
Beta Glucanasa	36 – 45°C	4.5 – 5.0	Reduce la viscocidad del mosto, y mejora la clarificación.
Peptidasa	46 – 57°C	4.6 – 5.2	Produce Amino Nitrógeno Libre (FAN), que es esencial para la levadura y la fermentación.
Proteasa	46 – 57°C	4.6 – 5.2	Rompe proteínas grandes y reduce la turbiedad.
Beta Amilasa	54 – 65°C	5.0 – 5.6	Produce azucares cortas, altamente fermentables.
Alpha Amilasa	68 – 75°C	5.3 – 5.8	Produce azucares de larga cadena, poco fermentables, que agregan cuerpo a la cerveza.

Tabla 2: Rangos óptimos de temperatura que activan las diferentes enzimas. Fuente: [7]

Siempre habrá algo de actividad de las enzimas en los rangos de temperaturas inferiores y superiores de estas. Por lo que muchos cerveceros optan por macerar a una sola temperatura. A unos 67°C, durante 60 minutos, lo que permite que se obtengan azucares fermentables y dextrinas.

Como se puede observar en la siguiente gráfica, en la cual se muestra la actividad de las enzimas, su punto de traslape máximo es a 67°C.

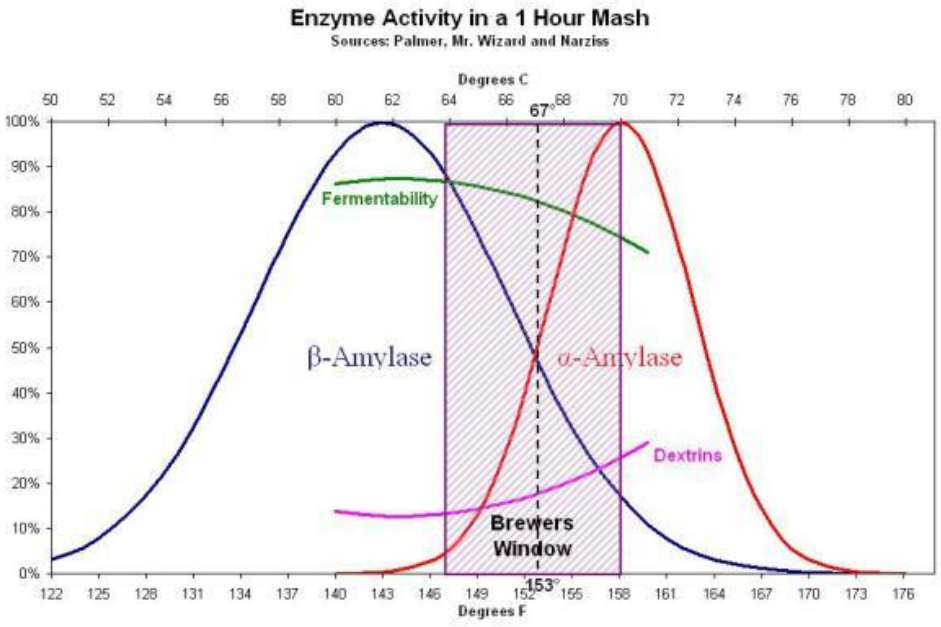

Figura 5: *Actividad de las enzimas.* Fuente: [7]

Si se macera a temperaturas entre 60 y 65°C, se obtienen cervezas más alcohólicas y secas con mayor contenido de maltosa y la mayor atenuación límite. Los mostos ricos en maltosa fermentan más rápidamente (levadura ale). En este caso el tiempo de maceración varían entre 10 y 20 minutos. Si la temperatura de maceración se encuentra entre 70 y 75°C se obtienen cervezas ricas en dextrinas con baja atenuación límite, es decir cervezas con menor contenido en alcohol, más dulces y de mayor cuerpo (levadura lager). El tiempo es menor que en el caso anterior ya que no se alcanza el máximo de actividad enzimática.

A continuación, se muestra una gráfica con las distintas enzimas, sus temperaturas y el tiempo necesario para activarlas.

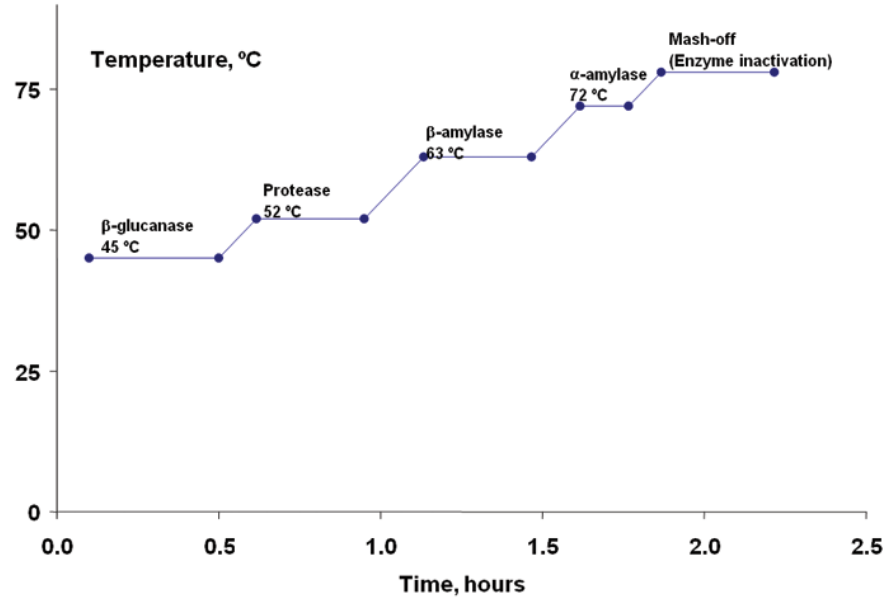

Figura 6: *Temperatura y tiempo de activación de las enzimas*. Fuente:[7]

Después de 40 a 60 minutos, la actividad enzimática va disminuyendo, primero de forma rápida y despues va decreciendo esta reducción de actividad.

Finalmente, el proceso de maceración dura entre 60 y 90 minutos. Nunca más de 120 minutos.

Para el proceso de remojado se añade agua a unos 9°C superior a la temperatura que queremos macerar, ya que al juntarla con la malta esta temperatura disminuirá.

Otra de las variables importantes en la maceración es el valor de pH. De este valor depende la actividad de la encimas.

El rango óptimo del pH es de 5,4 a 5,7 en la maceración. Este rango es óptimo para ambas amilasas [Buscar referencia], esto hace que se formen más azúcares fermentables y aumenta la atenuación límite. El valor de pH no deberá ser en ningún caso inferior a 5,2.

Para acidificar la mezcla se añadirá ácido fosfórico en cantidades muy pequeñas, chequeando el pH hasta llegar al pH objetivo.

Por último, otra variable importante es el valor del empaste (relación kg agua : kg malta). Los valores más habituales de empaste son de 2,5:1 a 3:1.

2.3.5. Filtrado del mosto

Tras la maceración, se separa el mosto líquido de los restos de malta. Para ello filtramos el mosto a través de una cuba filtro o de un filtro prensa, en ambos casos se separa el líquido del sólido, a este último le llamamos bagazo y normalmente es reaprovechado para alimentación animal.

En primer lugar, se inicia con una primera filtración del mosto. Algunas partículas pueden haber pasado el filtro, para eliminarlas, se recircula a una velocidad lenta hasta que se observa que el mosto empieza a ser claro.

Una vez realizada la primera filtración y antes de acabar de filtrar todo el mosto, se añade agua a temperatura de unos 78°C y con un pH adecuado, con el fin de recuperar todos los azucares atrapados en el filtro. Esta fase se denomina lavado del bagazo. Es importante mantener un nivel de líquido adecuado para que el agua no percole por las paredes o el filtro ceda.

La densidad del mosto disminuirá debido a su menor concentración de azucares, por lo que será importante medir la densidad antes de comenzar la etapa de cocción.

Esta fase finaliza cuando la concentración del mosto empieza a disminuir, ya que indica que los azucares han sido extraídos. La duración de la etapa suele ser de unos 45-60 minutos.

2.3.6. Cocción y enfriado

Al inicio de la cocción se añadirá el lúpulo necesario.

A continuación, el mosto se lleva a ebullición con una temperatura de unos 100°C con el objetivo de aportar amargor y aroma presentes en el lúpulo.

Además, durante esta etapa se esteriliza el mosto, se coagulan proteínas y se evaporan aromas indeseables. Normalmente este proceso dura en torno a una hora o más, dependiendo del estilo de cerveza que se esté elaborando. Posteriormente el mosto final es sometido a un centrifugado o whirpool.

La duración de la cocción es de unos 75-90 minutos.

Tras la cocción llega el momento de enfriar y airear el mosto rápidamente, ya que si se produce de forma lenta el mosto es susceptible de ser colonizado por microorganismos no deseados.

Para ello se debe disminuir la temperatura del mosto. Para cervezas Ale se debe alcanzar una temperatura entre 16-20°C, en el caso de cervezas Lager en torno a los 10-15°C.

Se utiliza un intercambiador de placas que es enfriado a 10°C previamente y hará que el mosto llegue a unos 20°C aproximadamente. El agua que saldrá del proceso tendrá una temperatura de 88°C y se recogerá en un tanque de agua caliente para posteriores procesos de limpieza y desinfección de equipos.

Si es necesario se realiza un segundo enfriamiento con agua glicolada enfriada a una temperatura de 0°C que llevará finalmente al mosto a la temperatura óptima para su fermentación.

Por último, el mosto debe ser aireado para favorecer la fermentación ya que ayudara a que las levaduras se activen con mayor facilidad gracias al oxígeno. Los valores de oxigenación deben estar comprendidos entre 8-10 mg/l de oxígeno disuelto.

La etapa de enfriado y aireado tiene una duración de unos 60-75 minutos aproximadamente.

2.3.7. Fermentación

Finalmente llegó el momento de enfriar y airear el mosto para luego sembrar la levadura. Durante la fermentación se transforman los azúcares fermentables en alcohol y CO2, al tiempo que se generan una gran variedad de compuestos, muchos de los cuales contribuyen a darle los aromas característicos tan populares de la cerveza. Usualmente en el proceso cervecero se utilizan dos grandes familias de levaduras: lager y ale. Este proceso se desarrolla en tanques de fermentación que en ocasiones son conocidos como fermentadores.

Inicialmente la levadura se activará y reproducirá. Posteriormente comenzará una respiración anaeróbica de esta levadura llamada fermentación donde se producirá la siguiente transformación:

$$C_6H_{12}O_6 \rightarrow 2C_2H_5OH + 2CO_2 + Energía$$

(Glucosa) → (Etanol) + (Dióxido de carbono) + Energía

En esta etapa se producen dos tipos de reacciones, la fermentación alcohólica la cual es la más importante y la reacción de maduración. Para que la fermentación sea correcta debe ser rápida y potente, con una temperatura adecuada. Si esto no ocurre podría suceder el caso de que el mosto se contaminase con otros microorganismos.

La temperatura de fermentación hace la diferencia entre una cerveza Ale y una Lager. La temperatura debe ser controlada en todo momento ya que una temperatura elevada al inicio puede producir aromas no deseados. La temperatura inicial del mosto será de unos 7-14°C para las cervezas Lager y de unos 15-26°C para las Ale. Esta temperatura irá aumentando y será controlada gracias a la refrigeración del fermentador.

La fermentación durara unos 4-5 días para las cervezas Ale y unos 7-8 días para la cerveza Lager.

Este proceso se desarrolla en tanques de fermentación conocidos como fermentadores. Una vez finalizada la fermentación alcohólica comenzaría la maduración. Este último proceso no lo abarcamos en este proyecto.

En este proceso se distinguen cuatro fases:

- Lag o de retardo: se trata de las primeras 15 horas. Una vez incorporada la levadura al mosto se necesita un tiempo de aclimatación. Esta fase es necesaria para que las células de la levadura absorban oxígeno, vitaminas, minerales y aminoácidos presentes en el mosto.
- Crecimiento: en esta fase el crecimiento de la levadura es exponencial y se crea una superficie de espuma en la superficie.

- Fermentación o estacionaria: en esta fase las levaduras existentes ya no pueden crecer por lo que su actividad decae y la velocidad se vuelve constante. Esto se produce debido al agotamiento de los nutrientes.
- Sedimentación: en esta fase la levadura ha producido prácticamente todos los sabores y aromas.

3. Planteamiento de soluciones alternativas y justificación de la solución adoptada

3.1. ¿Qué es Android?

Android es un sistema operativo móvil desarrollado por Google, basado en Kernel de Linux y otros softwares de código abierto y permite el desarrollo de aplicaciones por terceros gracias a un conjunto de APIs y herramientas. Está diseñado para dispositivos móviles con pantalla táctil.

El código fuente principal de Android se conoce como Android Open Source Project (AOSP), que se licencia principalmente bajo la Licencia Apache.

3.2. Historia Android

En el año 2003 se desarrolló Android por la empresa Android Inc. Esta empresa fue respaldada económicamente por Google y hasta el momento este sistema operativo era prácticamente desconocido.

En 2005 Google compró Android y el 5 de noviembre de 2007 fue presentado junto a la fundación Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) liderada por Google. Al mismo tiempo se anuncia la primera versión beta pública de Android y se dio a conocer lo que hoy conocemos por Android, una plataforma de código abierto para móviles que se presentaba con la garantía de estar basada en el sistema operativo Linux.

Google ha publicado la mayor parte del código fuente del sistema operativo, gracias a la Licencia Apache. Además, se lanzó un conjunto de herramientas de desarrollo, depuración, bibliotecas, simulación y documentación denominado Android SDK (Software Development Kit).

En octubre de 2008 por primera vez estaba en funcionamiento Android en un HTC Dream creado en USA, con la primera versión final de Android, la 1.0.

Durante todos estos años Android ha evolucionado hasta llegar a ser el sistema operativo más usado. Las aplicaciones Android se desarrollan en lenguaje java con Android SDK.

3.3. Estudio de alternativas. ¿Por qué Android?

En la actualidad, la tecnología de los teléfonos inteligentes nos ofrece un sinfín de posibilidades para elegir entre una gran variedad de ofertas distintas.

Para la creación de aplicaciones y sistemas operativos se realiza a través de un código de programación. El programador o programadores escriben el código fuente. Este código se compila para obtener un archivo o código ejecutable para finalmente obtener el producto que finalmente se ejecuta en el dispositivo. Existen muchos sistemas operativos, principalmente de dos tipos unos son de código abierto y otros de código cerrado. Estos sistemas operativos pueden ser para uso personal, para negocios o para industrias.

La diferencia entre los sistemas de código abierto y código cerrado es que un sistema de código abierto es aquel que el código es visible para el usuario, mientras que los sistemas de código cerrado el código fuente no es visible. Aunque esta no es la única diferencia entre estos dos sistemas.

Otra de las diferencias entre estos dos sistemas operativos o sistemas es que los sistemas de código cerrado son desarrollados y modificados por corporaciones por lo que solo ellos tienen acceso a modificaciones. Mientras que los de código abierto son desarrollados por grandes comunidades. Gracias a que el código es visible, los usuarios y consumidores de este programa pueden identificar y arreglar errores. Los sistemas de código abierto son mejorados con mayor frecuencia que los sistemas de código cerrado por este motivo.

Los sistemas de código cerrado más conocidos y en uso son los sistemas operativos Microsoft Windows y Apple OS. Estos sistemas proveen programas para muchos contextos de computación, incluyendo personal, servidores y móvil. La mayoría de PCs funcionan con este tipo de sistemas, aunque una porción sustancia de los servidores Web funcionan con plataformas de código abierto, en particular Linux.

Linux durante el año 2011 se convirtió en el sistema de código abierto más común.

En cuanto a los sistemas operativos de código abierto existe una gran variedad. El incremento de computadoras móviles también ha causado un aumento en el uso de sistemas operativos de código abierto, incluyendo Ubuntu Linux y los sistemas Android y Chrome de Google.

En cuanto a sistemas operativos de código abierto para dispositivos móviles también existen otras alternativas como pueden ser las siguientes:

- **LineageOS:** se trata de una herencia directa de CyanogenMod. Se basa en los lanzamientos de Google para la plataforma de Android. Este sistema incluye código adicional. Fue lanzado el 24 de diciembre de 2016.

- **Ubuntu Touch:** es un sistema operativo móvil basado en Linux. Este sistema está desarrollado por Canonical. El primer teléfono inteligente con este sistema fue lanzado en febrero de 2015.

- **KaisOs:** al igual que Ubuntu Touch es un sistema operativo móvil basado en Linux. Fusiona el poder de un smartphone con la accesibilidad de un teléfono básico.

- **Tizen:** está basado en Linux y se construye a partir de la plataforma Linux de Samsung (SLP). Su versión 2.2. fue lanzada el 22 de julio de 2013. Su SDK completo no es de código abierto ya que se ha publicado a través de una licencia de Samsung.

- **PostmarketOS:** está basado en Alpine Linux. Este proyecto tiene como objetivo darle un ciclo de vida de 10 años a los teléfonos inteligentes.

- **Sailfish OS:** es compatible con el entorno de Android, principalmente con sus aplicaciones, aunque no con todas. Esta desarrollado por la compañía finlandesa Jolla Ltd.

Estos sistemas están dirigidos a investigadores ya que estos sistemas no se encuentran instalados en casi ningún dispositivo móvil.

Por lo tanto, a la hora de elegir un sistema operativo para nuestro dispositivo, las opciones siempre se resumen al Sistema de Google o al de Apple.

A continuación, se muestra una gráfica en la cual se pueden observar los porcentajes de uso de los distintos sistemas operativos móviles.

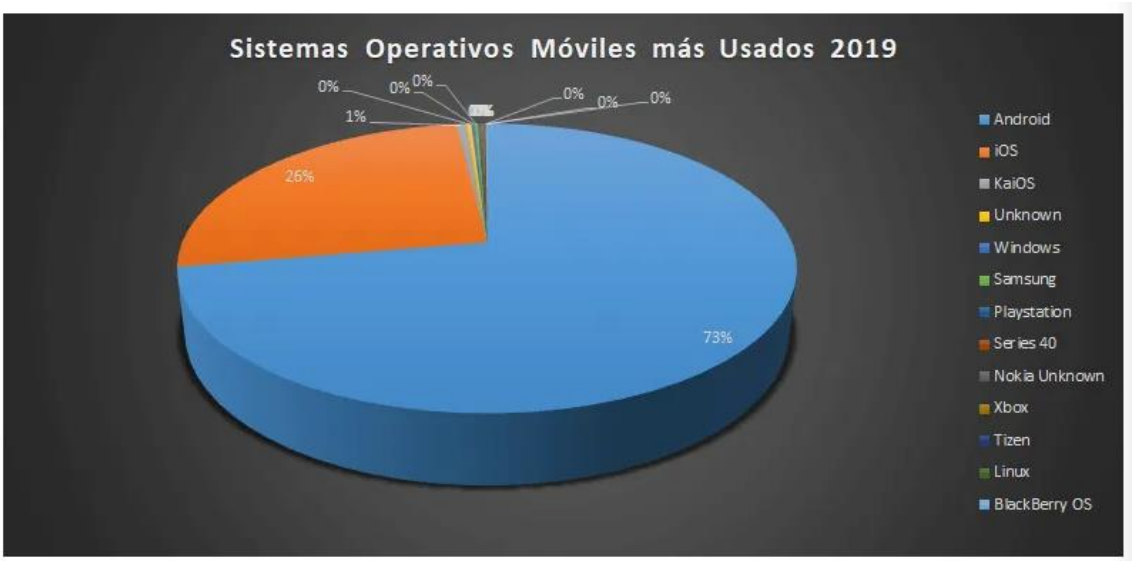

Figura 7: *Sistemas operativos móviles más usados en 2019.* Fuente: [8]

En este proyecto se necesita un sistema operativo difundido que todo el mundo utilice y de código abierto. Es decir, con el cual se pueda descargar la aplicación sin necesidad de borrarle al usuario su sistema y poner otro sistema para poder instalar la aplicación. Por ello vamos a elegir la plataforma de Android para el desarrollo de nuestro prototipo.

3.3.1. Ventajas Android

Algunas de las ventajas que tenemos al usar Android son las siguientes:

1. **Personalización:** permite personalizar los dispositivos de manera casi infinita, se pueden instalar fondos de pantalla, animaciones, widgets o skins. Además, nos permite escoger aplicaciones preferentes para nuestras tareas.

2. **Diversidad de dispositivos:** Android es el sistema operativo más utilizado en el mundo. Funciona en la mayoría de los dispositivos móviles como: Samsung, HTC, Motorola, ZTE, Huawei, LG, etc. Esto hace que se pueda contar con dispositivos de todas las gamas y diferentes costes.

3. **Comunidad Android:** nos da muchas posibilidades de mejora y actualización, contando con versiones alternativas a los Sistemas Oficiales.

4. **Código abierto:** Android está liberado con licencia Apache y código abierto, lo que hace que sea un sistema completamente libre para que el código pueda ser mejorado o modificado sin pedir autorización a su cambio. Además, también facilita el control del dispositivo.

5. **Asequibilidad**: la libertad del código de Android ha conseguido que en poco tiempo este sistema se utilice en todo tipo de dispositivos electrónicos, desde móviles, portátiles, navegadores GPS, relojes, lavadoras incluso marcos digitales. Esto hace que este sea totalmente asequible y alcanzable.

6. **Integración con Google:** permite el uso libre de aplicaciones y servicios de Google como pueden ser: Gmail, Youtube, Google Talk, Google Chrome, Google Maps, Google Calendar, etc.

7. **Libertad**: permite al usuario o propietario de un terminal instalar lo que quiera ya sea desde una tienda virtual como Play Store o Android Market o como un ejecutable aparte (APK). De esta misma forma da libertad a los desarrolladores o empresas pudiendo realizar aplicaciones o complementos sin necesidad de pedir permiso a nadie para ofrecerlo e instalarlo.

8. **Multitarea:** es capaz de gestionar varias aplicaciones a la vez, usarlas, dejarlas en segundo plano y cerrarlas por inactividad. Gracias a esto evitamos un consumo de memoria y batería innecesario.

¿Por qué Android Studio?

Una vez elegido el sistema operativo Android, veremos las distintas alternativas que se presentan para desarrollar la aplicación.

Las herramientas de desarrollo más comunes y utilizadas para el desarrollo de aplicaciones para Android son las siguientes:

- **Android Studio:** es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android.

- **Eclipse IDE:** es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido"

También existen dos alternativas que se tratan de herramientas de desarrollo Móvil Multi-plataforma e híbridas, y funcionarían para distintas plataformas.

- **Xamarin:** es una plataforma de código abierto para compilar aplicaciones modernas y con mejor rendimiento para iOS, Android y Windows con .NET.
  Xamarin tiene bibliotecas de código abierto. Pero a veces, estas tienen limitado el acceso. El desarrollo nativo hace uso de tecnologías de código abierto. Los componentes Xamarin ofrecen controles de interfaz personalizada. Se pueden usar estos controles con unos pocos clics, pero conllevan un coste adicional. Es posible añadir gráficos, diagramas, temas y otras características como pasarelas de pago, notificaciones push, etc.

- **PhoneGap:** es un framework para el desarrollo de aplicaciones móviles producido por Nitobi, y comprado posteriormente por Adobe Systems.34 Principalmente, PhoneGap permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo. Esta alternativa, también es de pago.

Por todo lo anterior la opción más adecuada para nuestro proyecto es Android Studio ya que es la mejor opción y la más intuitiva que se encuentra disponible para desarrollo de aplicaciones Android y además su licencia es gratuita.

Tipos de aplicaciones

En cuanto al tipo de aplicaciones existentes podrían clasificarse en tres tipos:

- Apps Nativas: son las aplicaciones desarrolladas en el lenguaje de programación correspondiente a cada sistema operativo. Permiten gestionar fácilmente los datos de forma local y el diseño harán que estas Apps ofrecen una experiencia de usuario mucho más consolidada que cualquier otra opción. Estas son las mas caras.

- WebApps: en realidad no son un tipo de App como tal, sino que se tratan de webs diseñadas para el entorno móvil al 100%. Estas aplicaciones pierden bastante fuerza ya que no son "instalables". No pueden instalarse a través de una tienda virtual como Apple o Google.

- Apps Híbridas: este tipo de apps son una mezcla entre las dos anteriores intentando coger lo mejor de cada una de ellas. Se tratan de Apps nativas con una sola pantalla que en realidad es una WebApp.

En nuestro caso se tratará de una App nativa ya que será desarrollada a través del lenguaje de Java, y necesitamos gestionar datos de forma local. Además, la principal ventaja de estas aplicaciones es que se puede utilizar el hardware del terminal, teniendo acceso a los recursos del dispositivo.

3.4. Arquitectura

Android es una pila de software y está construido con una arquitectura de 4 capas o niveles relacionados entre sí. A continuación, observamos un diagrama donde se observan estas agrupaciones o capas y sus módulos principales:

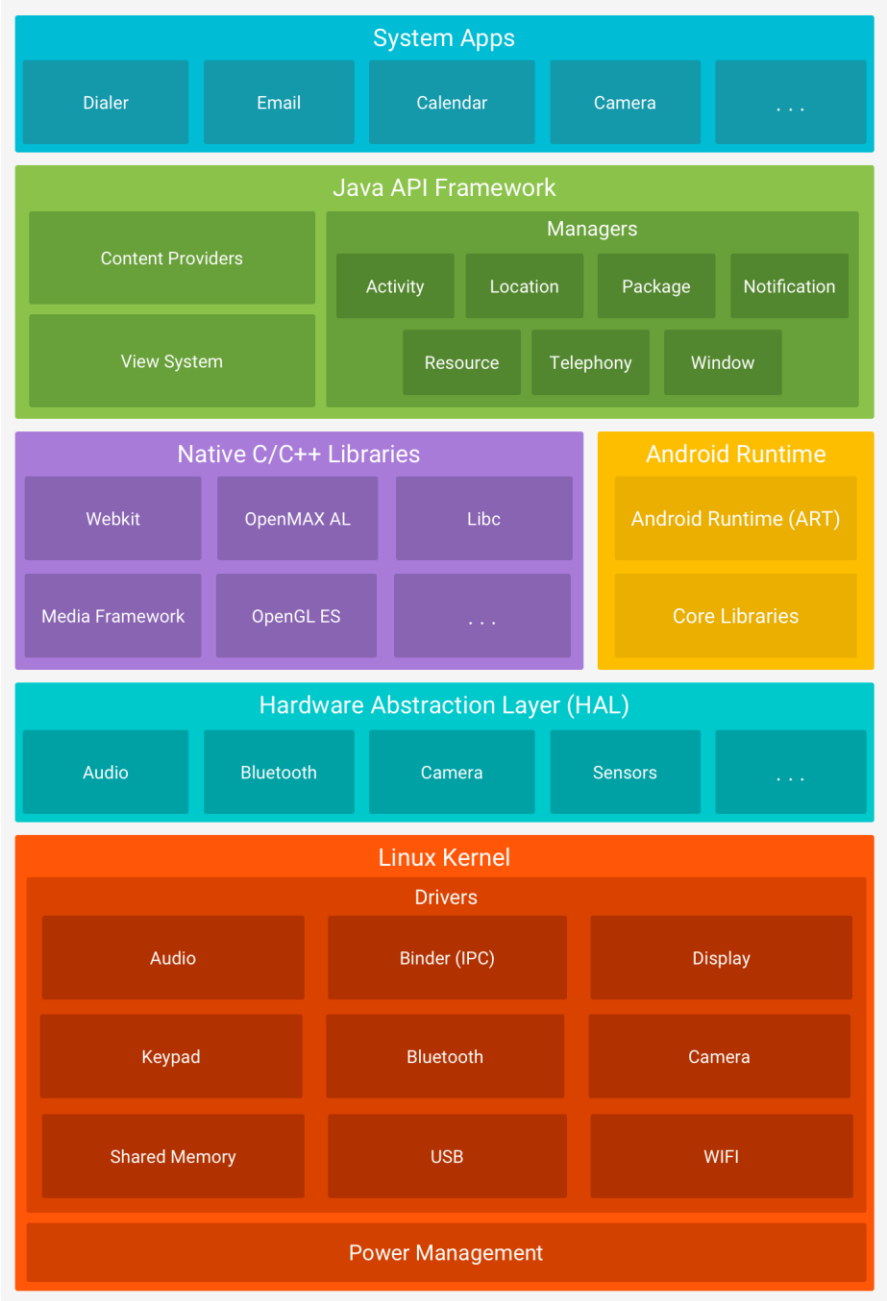

Figura 8: *Arquitectura Android.* ***Fuente:*** [9]

En el diagrama se observa que la estructura Android se encuentra construida sobre el Kernel de Linux. Luego se encuentra la capa de Librerías relacionadas con una estructura administradora en Tiempo de ejecución. En el siguiente nivel se encuentra un Framework de apoyo para construcción de aplicaciones y por encima la capa de Aplicaciones.

3.4.1. Kernel de Linux

Android está construido sobre el núcleo de Linux, pero se ha modificado dramáticamente para adaptarse a dispositivos móviles.

El tiempo de ejecución de Android (ART) se basa en el Kernel de Linux para funcionalidades subyacentes, como la generación de subprocesos y la administración de memoria a bajo nivel. Se encarga de la seguridad, gestión de memoria, gestión de procesos, modelo de controladores y pila de red.

El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

3.4.2. Capa de abstracción de hardware (HAL)

La capa de abstracción de hardware (HAL) brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la cámara o de Bluetooth. Cuando el marco de trabajo de una API realiza una llamada para acceder a hardware del dispositivo, el sistema Android carga el módulo de biblioteca para el componente de hardware en cuestión.

3.4.3. Tiempo de ejecución (Runtime)

Para los dispositivos con Android 5.0 (API 21) o versiones posteriores, cada aplicación ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de Android (ART).

El ART ejecuta varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo.

Antes de Android 5.0, Dalvik era el entorno de ejecución del sistema operativo.

3.4.4. Librerías nativas C/C++

Muchos componentes y servicios centrales del sistema Android, se basan en el código nativo que requiere bibliotecas nativas escritas en C y C++. Son junto con el núcleo los pilares que proporciona Android.

Entre ellas se destacan:

- **System C Library:** adaptada para dispositivos embebidos en Linux.

- **WebKit:** Soporta el navegador web de Android y su vista webview. Misma librería que Google Chrome y Safari de Apple.
- **SQLite:** Ligero pero potente motor de bases de datos relacionales.
- **SSL:** Proporciona servicios de encriptación (Secure Socket Layer)

3.4.5. Marco de trabajo de la API en Java (Framework)

Conjunto de herramientas de desarrollo y funciones del Sistema Operativo Android que está disponible mediante API escritas en el lenguaje Java.

Estas API son los cimientos que se necesitan para implementar Aplicaciones simplificando la reutilización de componentes del sistema y servicios centrales.

Los servicios más importantes que incluye son:

- **Sistema de vista:** conjuntos de vistas, es decir, la parte visual de los componentes (listas, cuadrículas, botones, cuadros de texto, etc…).
- **Administrador de recursos:** brinda acceso a recursos que no se encuentran en código como strings localizadas, gráficos, imágenes.
- **Administrador de notificaciones:** permite que las aplicaciones muestren alertas personalizadas en la barra de estado.
- **Administrador de actividad:** administra el ciclo de vida de las aplicaciones y proporciona una pila de retroceso de navegación común.
- **Proveedores de contenido:** permiten que las aplicaciones accedan a datos de otras aplicaciones, como la app de Contactos, o compartan sus propios datos.

3.4.6. Aplicaciones

Formada por el conjunto de aplicaciones del dispositivo, ya sean nativas o instaladas por el usuario.

Es la última instancia de funcionamiento de Android. Esta instancia se centra en la comunicación en la ejecución y en la estabilidad de las aplicaciones tanto las incluidas por Android como aquellas que el usuario haya añadido a posteriori, ya sean de terceras empresas o de desarrollo propio.

3.5. Las versiones de Android y niveles de API

Android está en constante evolución desde el lanzamiento de su primera versión. Desde que Google lanzase la versión beta de Android en noviembre de 2007, tras comprar la compañía con el mismo nombre en 2003, muchos han sido los cambios que ha experimentado en las 15 versiones que el gigante de Mountain View ha lanzado al mercado de los smartphones.

- **Android 1.0 Apple Pie (API 1):** es septiembre de 2008 sus creadores destacaban de este sistema operativo móvil que era totalmente gratuito y open source, a diferencia de su principal competidor, iOS.

- **Android 1.1 Petit Four (API 2):** con ella, los desarrolladores del sistema operativo móvil de Google querían resolver ciertos fallos, cambiar la API y agregar algunas prestaciones nuevas. Fue lanzada en febrero de 2009
- **Android 1.5 Cupcake (API 3):** también en 2009, concretamente el 27 de abril, lanzaron la actualización Android 1.5 Cupcake, la primera que utilizó de forma oficial un nombre basado en un postre.
- **Android 1.6 Donut (API 4):** apenas cinco meses más tarde, en septiembre, llegó el lanzamiento de Android 1.6 Donut.
- **Android 2.0 – 2.1 Eclair (API 5):** y todavía faltaba una actualización más en 2009. Una versión que supuso un antes y un después en el desarrollo de este sistema operativo móvil, puesto que incluía importantes novedades tanto en el diseño como en la arquitectura del propio software.
- **Android 2.2 – 2.3 Froyo (API 7 y 8):** llegó en enero de 2010. Permite introducir un campo de texto dictando sin necesidad de usar el teclado. Fecha de lanzamiento 20 de mayo de 2010.
- **Android 2.3 – 2.7 Gingerbread (API 9):** antes de que 2010 tocara a su fin, llegó una nueva versión de Android, en este caso la 2.3, también conocida como Gingerbread, que se convertiría en la versión más extendida del sistema operativo de Google durante los siguientes años.
- **Android 3.0 – 3.2 Honeycomb (API 11, 12 y 13):** en febrero de 2011, Google lanzó esta nueva versión de Android. Es la primera y única versión desarrollada exclusivamente para televisores y tabletas.
- **Android 4.0 – 4.0.5 Ice Cream Sandwich (API 14 y 15):** en octubre de 2011, esta nueva versión de Android estaba basada en la anterior, en Honeycomb, pero optimizada para smartphones.
- **Android 4.1 – 4.3.1 Jelly Bean (API 16, 17 y 18):** esta nueva versión vio la luz en julio de 2012, durante la I/O de ese mismo año.
- **Android 4.4 KitKat (API 19):** el 31 de octubre de 2013 llegó esta nueva actualización del sistema operativo móvil de Google.
- **Android 5.0 – 5.1.1 Lollipop (API 21 y 22):** esta nueva versión vio la luz en noviembre de 2014.
- **Android 6.0 Marshmallow (API 23):** el lanzamiento de esta versión se produjo el 5 de octubre de 2015.
- **Android 7.0 – 7.1.2 Nougat (API 24 y 25):** su lanzamiento se produjo el mes de julio de 2016.
- **Android 8.0 – 8.1 Oreo (API 26)**: su lanzamiento se produjo en agosto de 2017. Se implementó con este una arquitectura modular cuyo cometido era facilitar y agilizar a los fabricantes de hardware la entrega de actualizaciones de Android.
- **Android 9.0 Pie (API 28):** en agosto de 2018 se produjo el lanzamiento de esta novena versión de Android.
- **Android 10 (API 29):** en el segundo semestre de 2019 se lanzó al mercado.
- **Android 11 (API 30):** en junio de 2020 Google lanzó la primera beta de Android 11.

A continuación, se proporcionan datos sobre la cantidad relativa de dispositivos que usan una versión determinada de la plataforma Android.

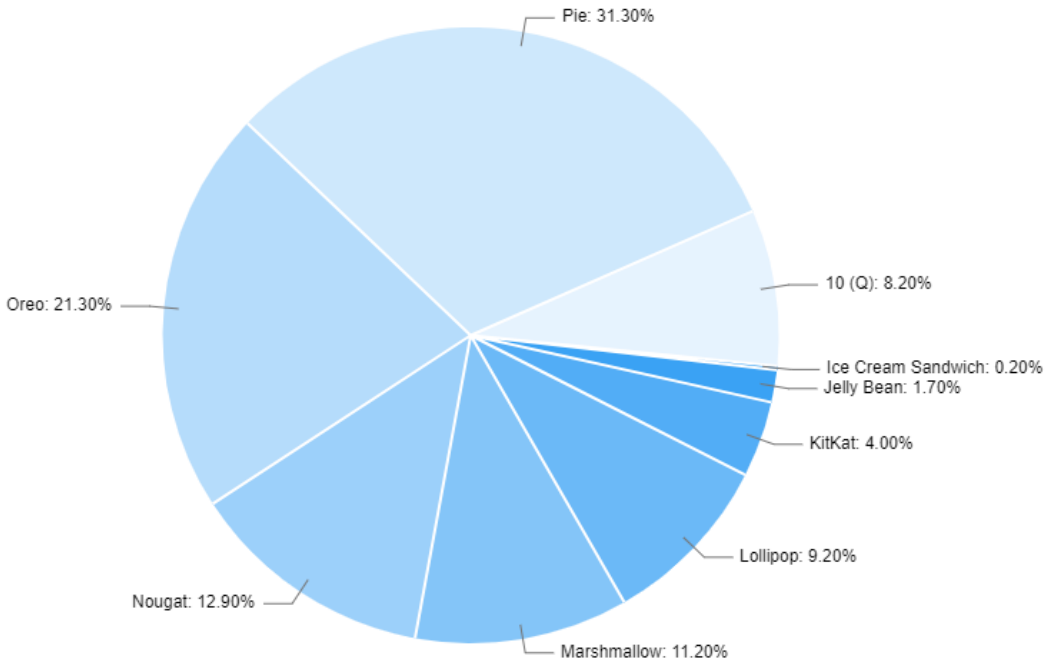

Figura 9: *Porcentaje uso de versiones Android.* Fuente: [10]

VERSIÓN DE ANDROID	PORCENTAJE
ICE CREAM SANDWICH (4.0)	0,2%
JELLY BEAN (4.1 - 4.3)	1,7%
KITKAT (4.4)	4%
LOLLIPOP (5.0 - 5.1)	9,2%
MARSHMALLOW (6.0)	11,2%
NOUGAT (7.0 - 7.1)	12,9%
OREO (8.0 - 8.1)	21,3%
PIE (9.0)	31,3%
ANDROID 10 (10.0)	8,2%

Figura 10: *Versiones Android 2020.* Fuente:[10]

4. Estudio de necesidades, factores a considerar: limitaciones y condicionantes.

En este apartado hablaremos de los requisitos necesarios que se han tenido en cuenta para desarrollar la aplicación.
Esta fase es la más importante para poder desarrollar el software ya que se realiza un estudio y captura de requisitos necesarios, mediante reuniones con nuestro cliente para cubrir todas las necesidades que tenga nuestro cliente.

La principal necesidad es la posibilidad de los trabajadores de poder supervisar que todo funcione de forma correcta desde cualquier lugar, incluso desde casa.

También existen limitaciones como son la obtención de datos reales ya que las empresas no nos proporcionan datos de las lecturas de sus sensores, por ejemplo, o la disposición de la planta. Por ello, este proyecto se llevará a cabo mediante datos basados en una empresa real y la disposición de la fábrica se diseñará también en base a la disposición de una fábrica de cerveza real.

Una vez recogida todas las necesidades se ha procedido al estudio de las tecnologías para poder implementar la aplicación.

4.1. Requisitos funcionales

Los requisitos funcionales describen todas las interacciones que tendrán los usuarios con el software.

4.1.1. Gestión de usuarios

Registro de usuario

• Entradas: la aplicación debe permitir al usuario introducir sus datos en el formulario de registro.

• Comportamiento: en caso de que los datos sean correctos, el sistema los guardara en la base de datos. En esta parte de debe comprobar que los campos introducidos son correctos, que no existe ese usuario en nuestra base de datos, que no se dejen campos obligatorios en blanco.

• Salidas: si todo es correcto el usuario se almacena en la base de datos, en caso de error se le informara con una notificación de los campos que no son correcto y no se introduce en la base de datos.

Solicitud de baja de usuario

• Entradas: la aplicación debe permitir dar de baja a un usuario.

• Comportamiento: se introducirá el nombre de usuario y el sistema lo buscará, si el usuario es correcto, rellenará el resto de los campos automáticamente y se podrá eliminar el usuario.

• Salidas: si el nombre usuario introducido no existe el sistema mostrará un mensaje comunicándolo.

Modificación cuenta de usuario

• Entradas: la aplicación debe permitir modificar los datos de un usuario existente.

• Comportamiento: se introducirá el nombre de usuario y el sistema lo buscará, si el usuario es correcto, rellenará el resto de los campos automáticamente, se podrá modificar el campo que se desee y actualizar.

• Salidas: si el nombre usuario introducido no existe el sistema mostrará un mensaje comunicándolo.

Listar usuarios locales

• Entradas: la aplicación debe permitir ver todos los usuarios que se encuentran en la base de datos.

• Comportamiento: el sistema generará una lista con los usuarios existentes.

• Salidas: la aplicación mostrará la lista anterior por pantalla.

Identificación

• Entradas: Al usuario se le pide el nombre como identificador y la contraseña con la que se registró.

• Comportamiento: el sistema consulta la base de datos para comprobar los datos metidos por el usuario.

• Salidas: si son correctos podrá ingresar a nuestra aplicación, si falla alguno de los datos se le comunica mediante una notificación en la aplicación que el usuario o la contraseña son incorrectas.

Cerrar sesión

• Entradas: cualquier usuario de la aplicación debe poder finalizar la sesión en la aplicación mediante un botón que aparece en la *ActionBar* y que indique “Cierre de sesión”.

• Comportamiento: la aplicación ejecutará el evento de este botón y el usuario será redirigido a la pantalla de inicio de sesión.

• Salidas: cuando el usuario pulse el botón de cierre de sesión, el sistema mostrará un mensaje, para asegurarse de que el usuario quiere cerrar sesión, permitiendo aceptar o cancelar.

4.1.2. Proceso de fabricación

Producción

• Entradas: la aplicación debe permitir introducir la cantidad de cerveza que se quiere fabricar por cada lote nuevo, así como la receta que se desea producir. La producción mínima debe de ser de 200 litros. La producción máxima será de 2000 litros.

• Comportamiento: cuando se introducen los litros que se desean producir y la receta deseada, la aplicación calcula las cantidades necesarias para esa cantidad de producción.

• Salidas: la aplicación muestra las cantidades necesarias de las materias primas y si están disponibles en los silos. Estas cantidades se restarán a las materias primas disponibles para llevar un control de estas. Si por algún motivo alguna de ellas no está disponible se muestra una alerta indicándolo.

Si la cantidad de producción está por debajo de la producción mínima o por encima de la máxima, el sistema nos mostrará un mensaje indicándolo.

Si las materias primas están disponibles y una vez seleccionada la receta, automáticamente las variables objetivo que varían dependiendo de la receta elegida se deberán guardar en variables globales. También creará

Malteado

• Entradas: el proceso de malteado tiene como entradas dos variables, la humedad del grano y la temperatura del agua que se necesita para remojar el grano y que la humedad del grano aumente, el sistema debe permitir introducir estas dos variables de forma manual, de esta forma simularemos la lectura de los sensores de la fábrica correspondientes a estas variables.

Además, contaremos con 3 botones para casos extraordinarios. Uno para forzar el inicio del malteado, otro para pausar el proceso en caso de alguna emergencia y otro para finalizar el proceso.

• Comportamiento: el sistema comprobará si la humedad inicial se encuentra entre el 11 y 13%, si esto ocurre mostrará una ventana emergente que te permitirá iniciar el proceso de malteado. Si la humedad no es la correcta habrá que esperar a que la humedad sea la correcta cambiándola de forma manual.

Una vez iniciado el malteado la temperatura del agua irá aumentando, simulando que está calentándose hasta alcanzar los 20°C, momento en el cuál empezará el remojado del grano y por lo tanto el aumento de la humedad de este.

En cuanto a los botones, con el botón de iniciar proceso, el malteado se iniciará sin tener en cuenta la humedad inicial del grano, es decir con este botón podremos forzar su inicio. Además, una vez iniciado el proceso no se podrán cambiar las variables de entrada. Con el botón de pausar, se pausará el proceso, lo cual permitirá cambiar alguna de las variables. Y, por último, con el botón de finalizar proceso, se simulará que el proceso ha terminado, es decir que la humedad del grano es de 45% y por lo tanto se pasará a la siguiente fase, la de germinación.

• Salidas: la aplicación mostrará lo que está ocurriendo en el proceso, en primer lugar, mostrará que el grano se introduce en el depósito de remojo cuando su humedad inicial es la correcta. Seguidamente mostrará que se introduce el agua con la temperatura deseada, y a continuación mostrará que ambas materias se están mezclando hasta alcanzar la humedad del grano deseada (45%).

También se dispondrá de una ventana que mostrará el valor de las dos variables en tiempo real en una gráfica.

Germinación

• Entradas: la única entrada para este proceso es que la caja de germinación disponible no esté ocupada y que además el proceso de malteado haya finalizado.

En este caso contamos con 4 botones: iniciar, pausar, simular tiempo y terminar germinación.

• Comportamiento: el sistema iniciará un cronómetro de 5 días, tiempo estimado de la duración de la germinación.

Con el botón de iniciar, se inicia el proceso de germinación si el malteado ya ha acabado. Con el botón de pausar, simplemente se pausa el cronómetro. Y con el botón de terminar germinación lo que se hace es simular que el tiempo de germinación ya ha pasado y así avanzar al siguiente proceso de fabricación. Con el botón de terminar, simulamos que el proceso ha acabado.

• Salidas: una vez alcanzado el tiempo estimado, el sistema nos mostrará una ventana emergente que dará la opción de avanzar e iniciar a la siguiente fase que es el secado o tostado. Cuando se utiliza el botón de finalizar malteado también nos muestra esta ventana emergente.

Secado o tostado

• Entradas: la aplicación debe permitir introducir de forma manual la temperatura del aire, la temperatura de la malta y la humedad de la malta. De esta forma simulamos la lectura de los sensores correspondientes.

Contamos con 2 botones uno para iniciar el proceso de forma manual y otro para simular que el proceso ha finalizado o para casos de emergencia que sea necesario parar el proceso.

La temperatura del aire mínima y máxima para la fase 2 variará en función de la recete que hayamos elegido.

• Comportamiento: una vez introducidas estas variables, se guardan en las variables globales correspondientes. Se comprueba la humedad de la malta y dependiendo de esta humedad se inicia una fase u otra del proceso. Dependiendo de la fase que se inicie se ejecuta una tarea u otra simulando el aumento o disminución tanto de las temperaturas como de la humedad de la malta.

• Salidas: La aplicación nos irá mostrando con los *EditText* los valores de las temperaturas y la humedad y el estado en el que se encuentra dependiendo de la imagen que este activa en cada momento.

Una vez finalizado el proceso nos lo indicará con una ventana emergente.

Molturación

• Entradas: este proceso tiene como entradas la condición de si el proceso de secado ha finalizado y los litros que se están produciendo ya que de los litros dependerá el tiempo del proceso. El tiempo de molturado será de aproximadamente 1 hora para unos 400 litros de cerveza.

Al igual que en la germinación contaremos con 4 botones: iniciar, pausar, simular tiempo y terminar el proceso.

• Comportamiento: se leerá la producción elegida con anterioridad y calculará el tiempo necesario para la molturación. Si el proceso anterior ha acabado se iniciará la molturación, si no, no se podrá iniciar.

Con el botón de iniciar, se inicia el proceso de molturación si el secado ya ha acabado. Con el botón de pausar, simplemente se pausa el cronómetro. Y con el botón de terminar molturación lo que se hace es simular que el tiempo de molturación está a punto de acabar y así avanzar al siguiente proceso de fabricación. Con el botón de terminar, simulamos que el proceso ha acabado.

• Salidas: la aplicación nos mostrará el tiempo que lleva la molturación en marcha y además también nos mostrará el progreso que lleva con un *progressbar*.

Una vez se alcance el tiempo necesario, la aplicación nos lo indicará mediante una ventana emergente. Y nos permitirá comenzar el siguiente proceso.

Maceración

• Entradas: la aplicación nos debe permitir introducir la temperatura inicial del agua con la que vamos a realizar la maceración simulando la lectura del sensor correspondiente en la fábrica.

En este proceso también se incluirán tres botones: iniciar, pausar y terminar maceración para controlar el proceso de forma manual.

• Comportamiento: el sistema calentará simulará el calentamiento del agua hasta alcanzar la temperatura objetivo del agua de este proceso que dependerá de la receta objetivo. Es por ello por lo que la temperatura objetivo se leerá de una variable global. Cuando el agua alcance este valor, se iniciará el cronómetro de la maceración que dura 120 minutos, nunca podrá ser superior el tiempo.

• Salidas: la aplicación nos mostrará el estado del proceso en cada momento mediante un *TextView* y el tiempo que lleva en marcha el proceso mediante un cronómetro y su *progressbar*.

Filtrado

• Entradas: la aplicación nos debe permitir introducir la temperatura inicial del agua con la que vamos a realizar el lavado del bagazo.

• Comportamiento: el sistema pondrá en marcha la filtración del mosto, la cual dura unos 60 minutos. Los primero 45 minutos se tratan de la primera filtración del mosto, al alcanzar este tiempo el sistema calentará o enfriará el agua hasta que este entre 75°C y 82°C y con esta agua se iniciará el proceso de lavado del bagazo. Que finalizará cuando el proceso llegue a los 60 minutos de duración.

• Salidas: la aplicación nos mostrará el porcentaje de proceso y el proceso activo de cada momento. Pudiendo ser estos procesos el primer filtrado del mosto o el lavado del bagazo. También nos mostrará la temperatura del agua de forma numérica y gráfica. Y una alerta cuando el proceso haya finalizado.

Cocción y enfriamiento

• Entradas: la aplicación nos debe permitir introducir la temperatura inicial del mosto con la que va a comenzar el proceso simulando la lectura del sensor correspondiente en la fábrica.

En este proceso también se incluirán tres botones: iniciar, pausar y terminar maceración para controlar el proceso de forma manual.

• Comportamiento: el sistema pondrá en marcha el proceso, calentando el mosto hasta 100°C para llevarlo a ebullición, en este momento comenzará la cocción durante 90 minutos. Una vez finalizada la cocción comenzará el enfriado, haciendo que el mosto alcance los 20°C aproximadamente en el caso de cervezas Ale. Y unos 15°C para las cervezas Lager.

• Salidas: la aplicación nos mostrará el porcentaje de proceso y el proceso activo de cada momento. Pudiendo ser estos procesos la cocción o el enfriado. También nos mostrará la temperatura del agua de forma numérica y gráfica. Y una alerta cuando el proceso haya finalizado.

Fermentación

• Entradas: la aplicación nos debe permitir introducir la temperatura inicial del mosto con la que va a comenzar el proceso simulando la lectura del sensor correspondiente en la fábrica.

En este proceso también se incluirán tres botones: iniciar, pausar y terminar maceración para controlar el proceso de forma manual.

• Comportamiento: el sistema pondrá en marcha el proceso, el cual debe mantener la temperatura del mosto controlada y evitar que esta aumente debido a las reacciones que se producen en la fermentación. Para las cervezas Ale la temperatura deberá estar entre 15-26°C durante unos 4-5 días y para las cervezas Lager la temperatura deberá estar entre 7-14°C durante unos 7-8 días.

• Salidas: la aplicación nos mostrará el porcentaje de proceso y la temperatura del mosto. También nos mostrará una alerta cuando el proceso haya finalizado.

4.1.3. Materias primas

Agua

• Entradas: se introducirá la cantidad de agua disponible en el silo de forma manual, de esta forma se simulará la lectura de los sensores de las correspondientes variables.

• Comportamiento: el sistema guardará este valor para comprobar en el apartado de producción si la cantidad disponible es suficiente. También se comprobará que su valor es de mínimo 800 litros que son los necesarios para una producción mínima de 200 litros de cerveza.

• Salidas: en caso de que el nivel del silo sea inferior a 800 litros, el sistema mostrará una alerta.

Cebada

• Entradas: se introducirá la cantidad de agua disponible en el silo de forma manual, de esta forma se simulará la lectura del sensor encargado la cantidad de cebada disponible en el silo.

• Comportamiento: el sistema guardará este valor para comprobar en el apartado de producción si la cantidad disponible es suficiente. También se comprobará que su valor es de mínimo 35 kg que son los necesarios para una producción mínima de 200 litros de cerveza.

• Salidas: en caso de que el nivel del silo sea inferior a 35 kg, el sistema mostrará una alerta.

Lúpulo

• Entradas: se introducirá la cantidad de lúpulo disponible en el silo de forma manual, de esta forma se simulará la lectura del sensor encargado la cantidad de lúpulo disponible en el silo.

• Comportamiento: el sistema guardará este valor para comprobar en el apartado de producción si la cantidad disponible es suficiente. También se comprobará que su valor es

de mínimo 0.4 kg que son los necesarios para una producción mínima de 200 litros de cerveza.

• Salidas: en caso de que el nivel del silo sea inferior a 0.4 kg, el sistema mostrará una alerta.

Levadura Ale y levadura Lager

• Entradas: se introducirá la cantidad de levadura disponible en los silos de forma manual, de esta forma se simulará la lectura del sensor encargado la cantidad de levadura disponible en los silos correspondientes.

• Comportamiento: el sistema guardará este valor para comprobar en el apartado de producción si la cantidad disponible es suficiente. También se comprobará que su valor es de mínimo 0.2 kg que son los necesarios para una producción mínima de 200 litros de cerveza.

• Salidas: en caso de que el nivel del silo sea inferior a 0.2 kg, el sistema mostrará una alerta.

4.1.4. Recetas

• Comportamiento: el sistema cargará los datos de las recetas de la aplicación

• Salidas: se trata de una pantalla informativa la cual nos muestra las recetas que tenemos en la aplicación y los datos sobre ellas. Como son las temperaturas objetivas de los procesos.

4.1.5. Historial

• Entradas: esta pantalla no necesita datos de entrada.

• Comportamiento: el sistema consultara los datos de la tabla de lotes de la base de datos. Realizará la conexión correspondiente.

• Salidas: mostrará los lotes producidos.

5. Funcionamiento aplicación

5.1. Identificación e inicio de sesión

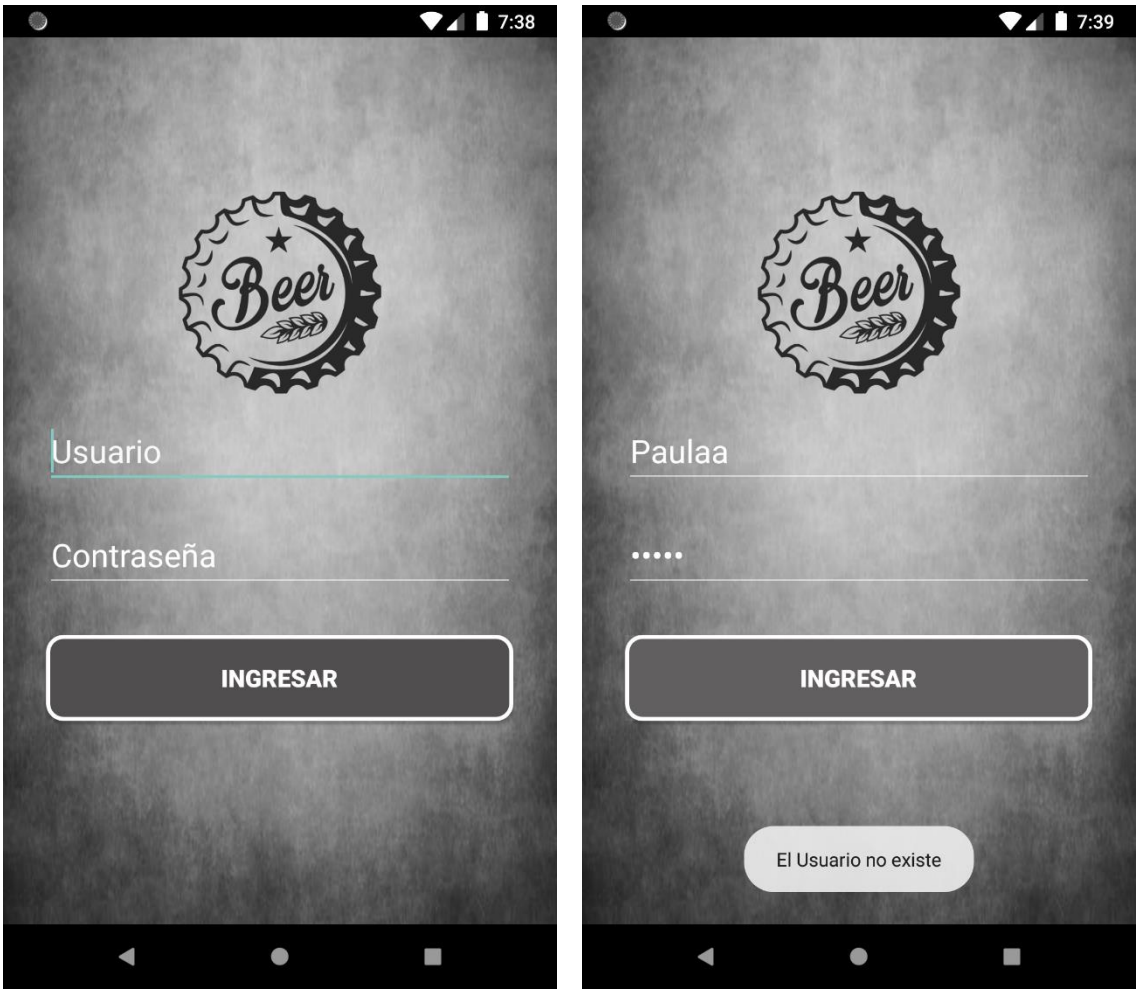

Figura 11: *Pantalla de identificación*

Como se puede comprobar en la figura 11, tenemos la pantalla de identificación de usuario. En ella se pide el nombre de usuario y la contraseña para poder iniciar sesión en la aplicación.

También se observa como si el usuario es erróneo, en este caso el usuario correcto sería Paula, la aplicación nos muestra un mensaje indicándonos que el usuario introducido no existe.

5.2. Gestión de usuarios

Figura 12: *Pantalla administrador de Usuarios*

En la figura 12, se muestran 3 botones para las diferentes opciones de gestión de usuarios

Registro de usuario

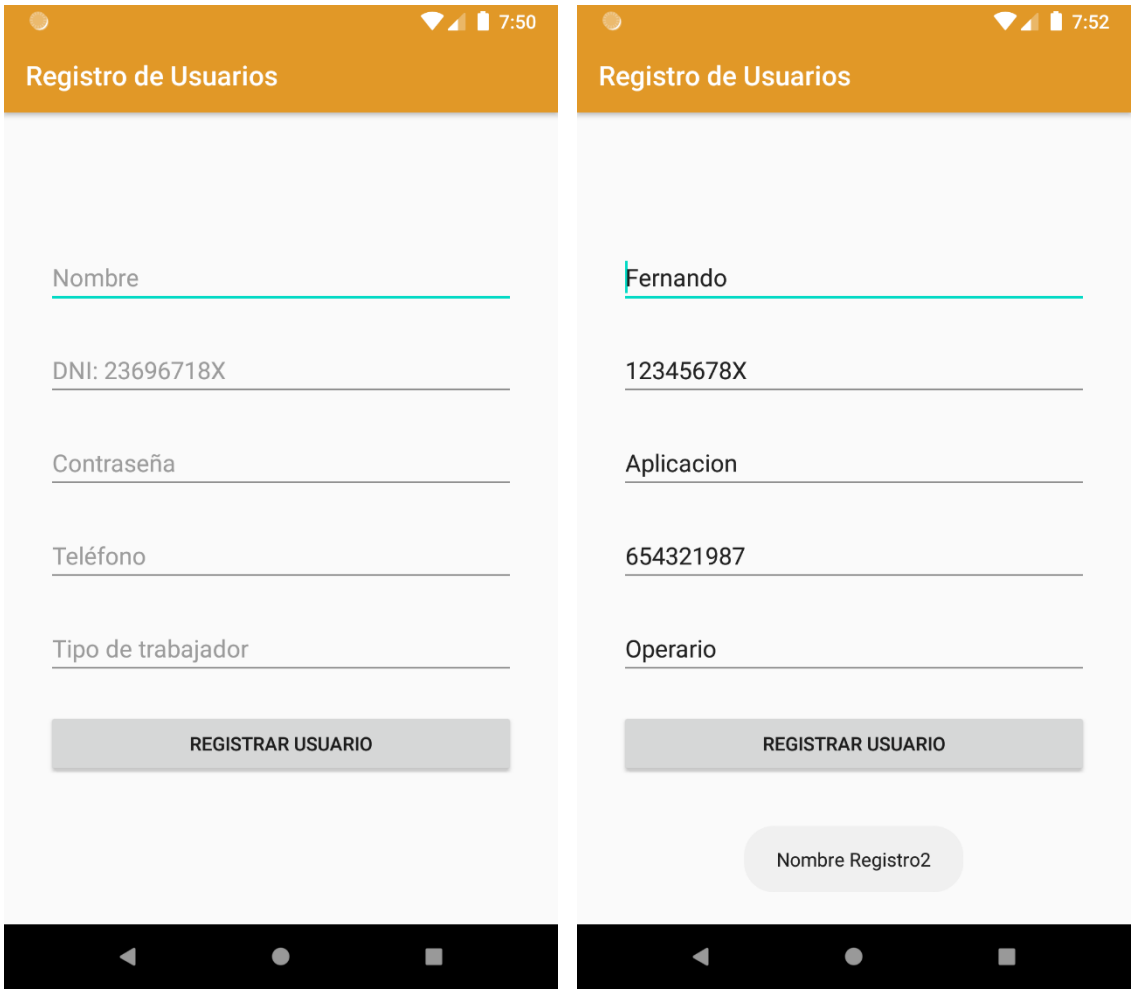

Figura 13: *Pantalla Registro de usuario*

En la figura 13 se observa cómo se piden por pantalla todos los datos requeridos para registrar un usuario y para registrarlo es necesario pulsar el botón registrar Usuario. A continuación, aparece un mensaje indicando que es el Registro número 2.

Solicitud de baja de usuario

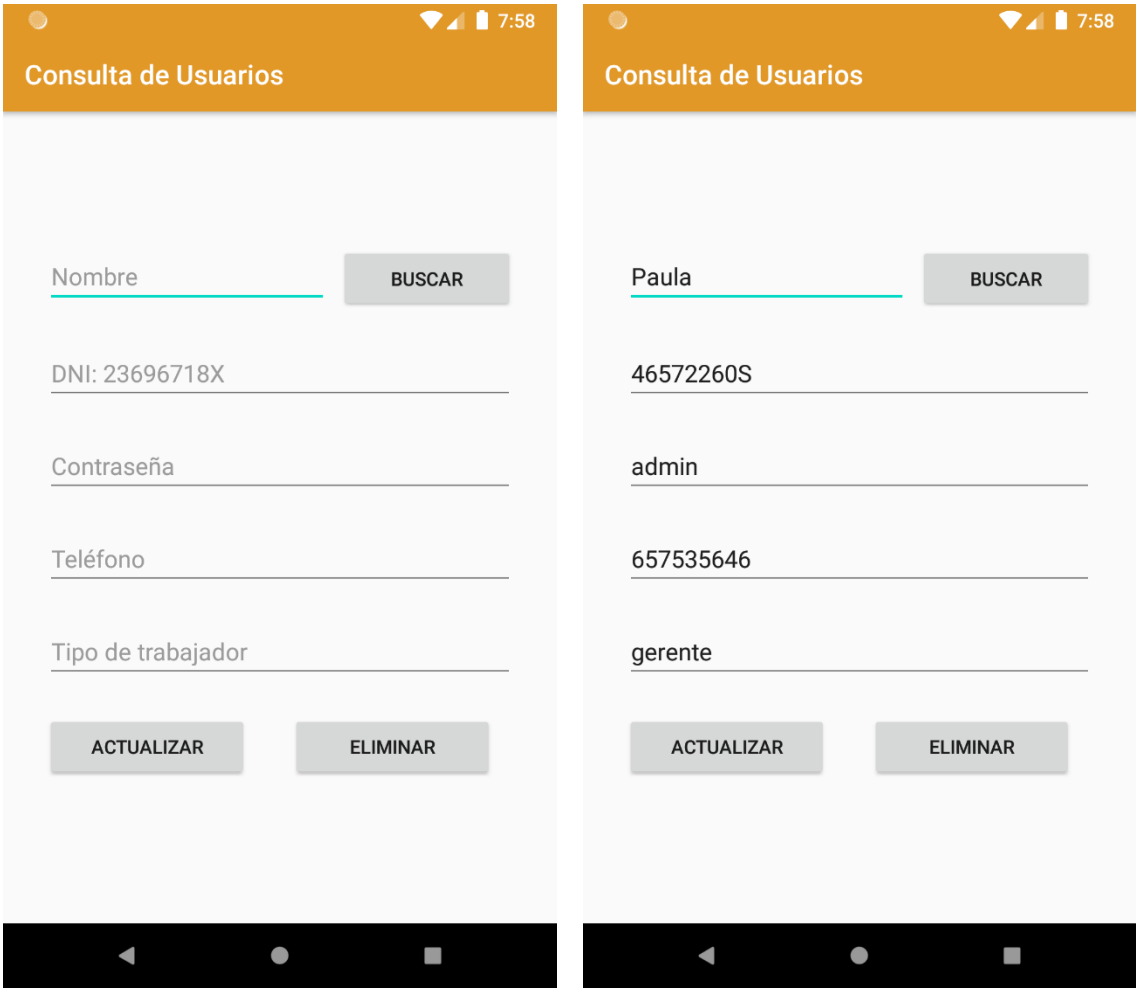

Figura 14: *Pantalla Eliminar usuario*

Para poder eliminar un usuario se debe buscar por su nombre de usuario, el resto de los campos se autor rellenarán y a continuación se deberá pulsas el botón de eliminar.

Cuando se pulsa el botón aparece un mensaje que indica que el usuario ha sido eliminado (Figura 15).

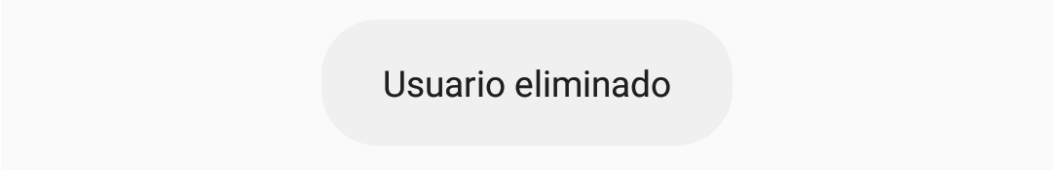

Figura 15: *Mensaje eliminar usuario*

Modificación cuenta de usuario

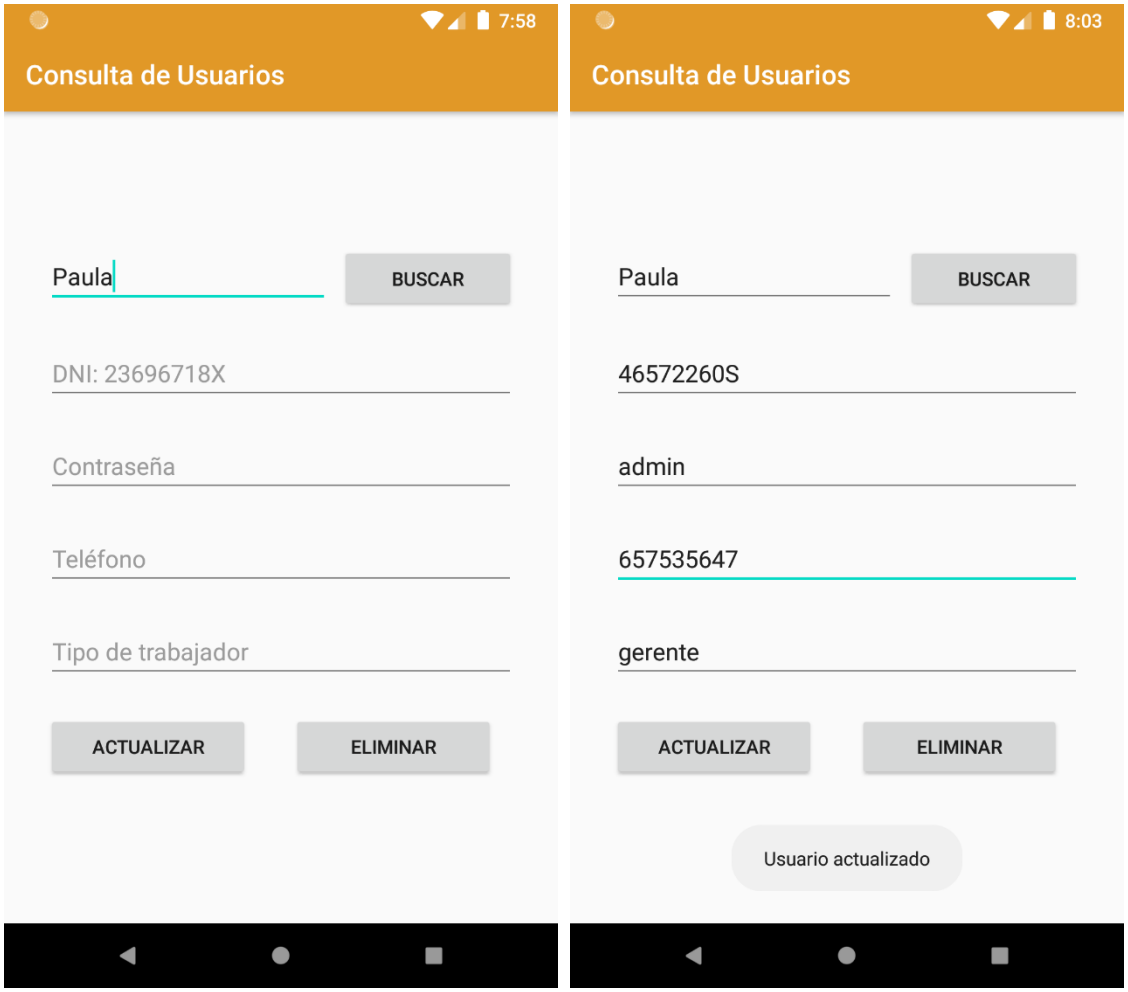

Figura 16: *Pantalla Modificación de usuario*

Para poder modificar un usuario se debe buscar por su nombre de usuario, el resto de los campos se autor rellenarán. Se modificará el campo que se desee y a continuación se deberá pulsas el botón de actualizar.

Cuando se pulsa el botón aparece un mensaje que indica que el usuario ha sido actualizado.

Listar usuarios

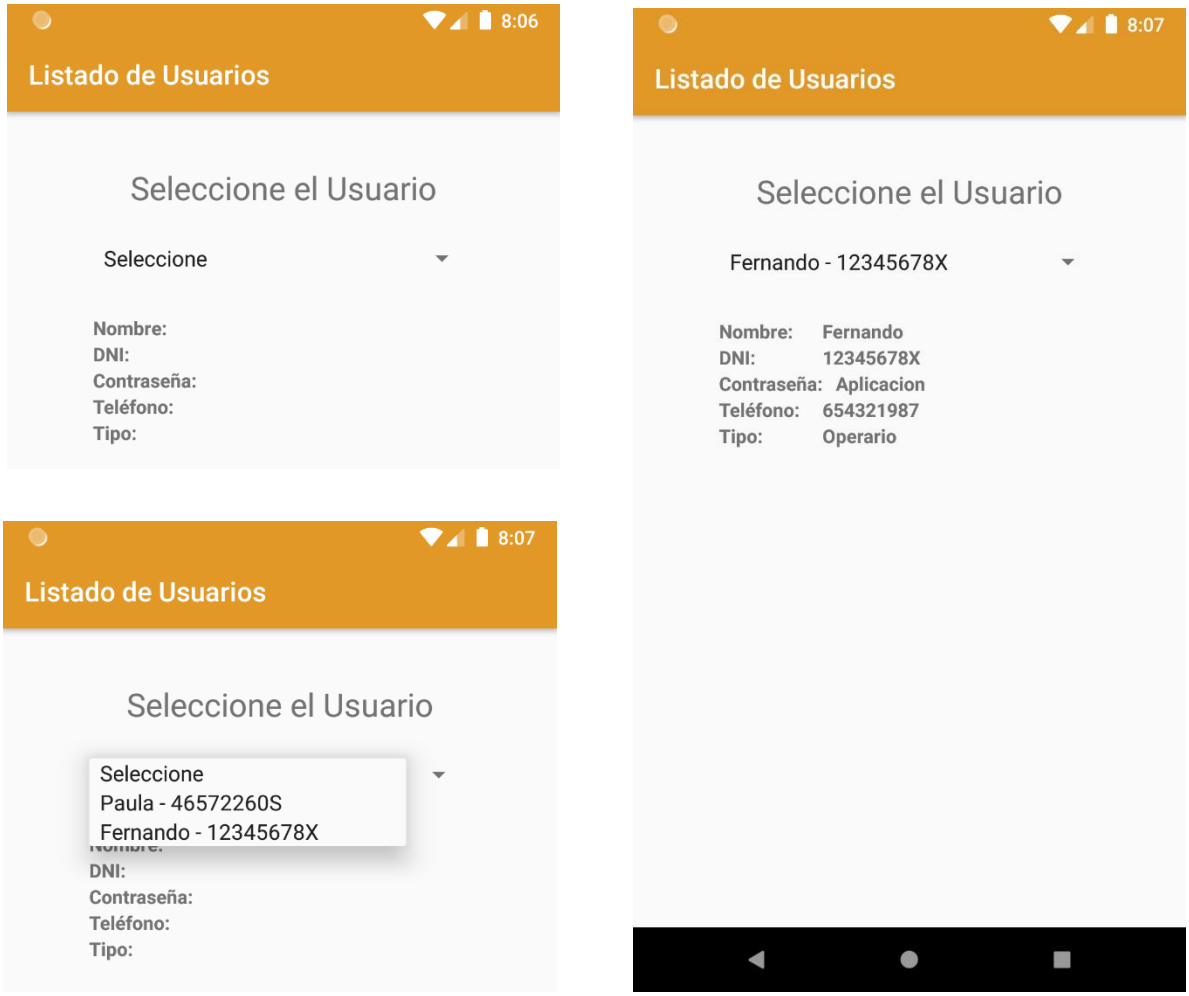

Figura 17: *Pantalla Listar usuarios*

Para consultar todos los usuarios registrados y sus datos en la pantalla Listado de Usuarios se debe abrir el desplegable en el cual nos aparecerán los usuarios registrados con su nombre y DNI, se selecciona el que se desee y a continuación, se auto rellenan los campos de abajo que se encontraban vacíos con todos los datos correspondientes.

Cerrar sesión

Figura 18: *Botón Cerrar Sesión*

En todas las pantallas de la aplicación, en la barra horizontal, aparecerá un botón que al pulsar sobre él se cerrará la sesión del usuario, nos mostrará un mensaje y volverá a la pantalla de identificación (Figura 18).

Figura 19: *Mensaje Sesión Cerrada*

5.3. Menú Principal

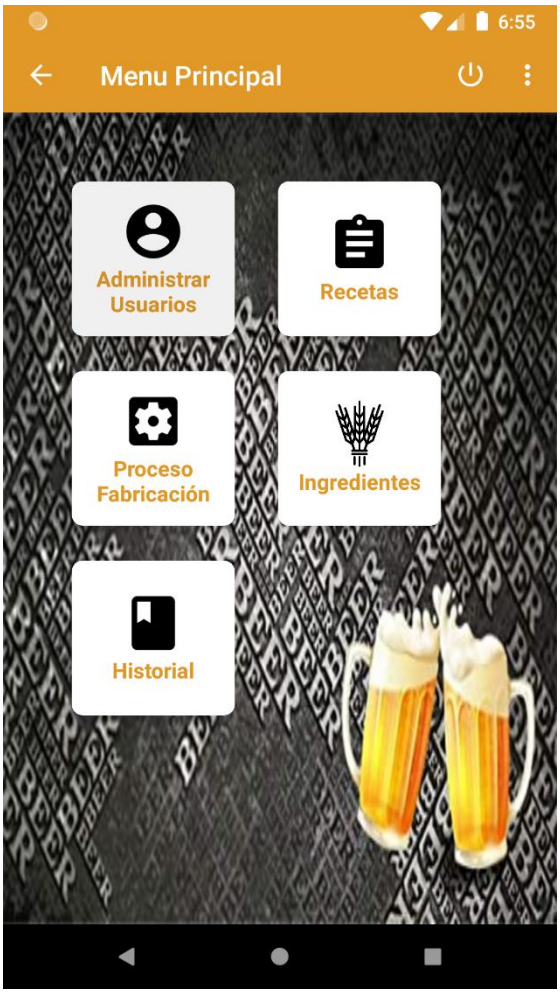

Figura 20: *Pantalla Menú Principal*

Como se observa en el menú principal tenemos las distintas funciones de la aplicación, y mediante *CardView* podemos acceder a ellas. Además, tenemos la barra que hemos creado para el menú principal en la cual se ha habilitado el botón de volver a la ventana anterior que en este caso sería el inicio de sesión, también aparece el botón de cerrar sesión explicado anteriormente, y otro botón que son tres puntos que serían opciones.

Figura 21: *Botón volver atrás*

5.4. Ingredientes

En la siguiente ventana se muestran los silos con todas las materias primas necesarias para la fabricación de la cerveza y la cantidad de la que se dispone de cada una de ellas.

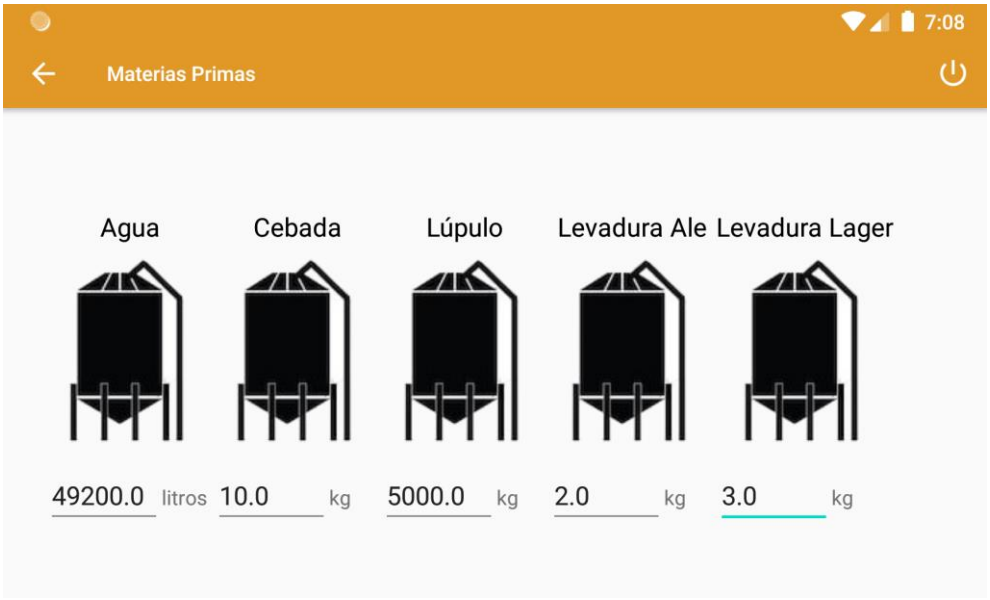

Figura 22: *Pantalla Ingredientes*

Sabiendo las cantidades necesarias de cada ingrediente hemos calculado la cantidad mínima de cada uno de ellos para que cuando esto ocurra nos lo indique mediante un *AlertDialog*.

En este *AlertDialog* nos muestra que ingrediente es el que deberemos de comprar o añadir.

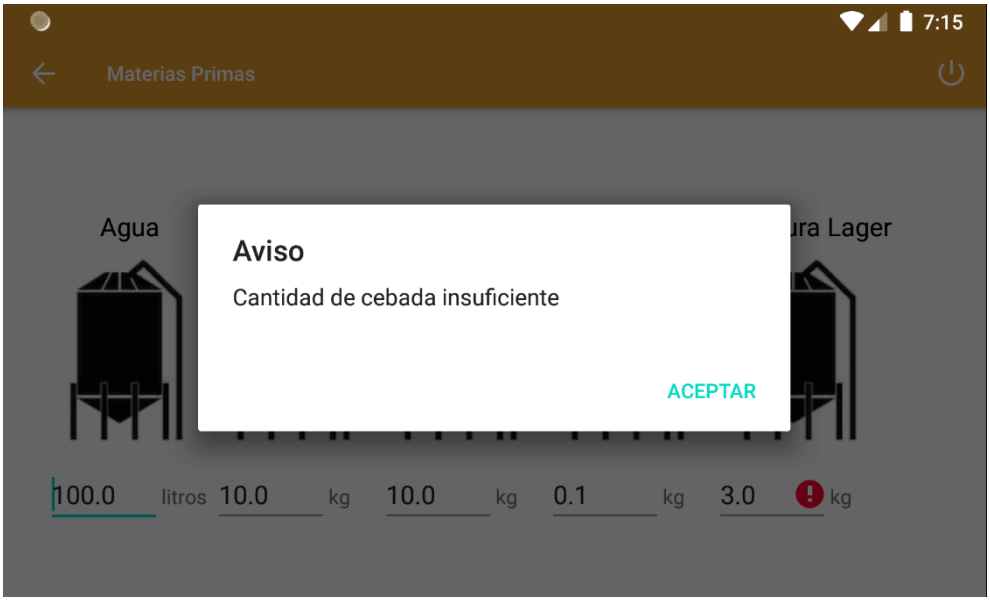

Figura 23: *Alerta cebada insuficiente*

En caso de que corresponda nos mostrará la misma alerta pero indicando el ingrediente insuficiente. Puede ser el agua, la cebada, el lúpulo, la levadura Ale o la levadura Lager.

Por último, la aplicación también nos notifica si alguno de los EditText esta vacío o es un dato erróneo cuando lo introducimos.

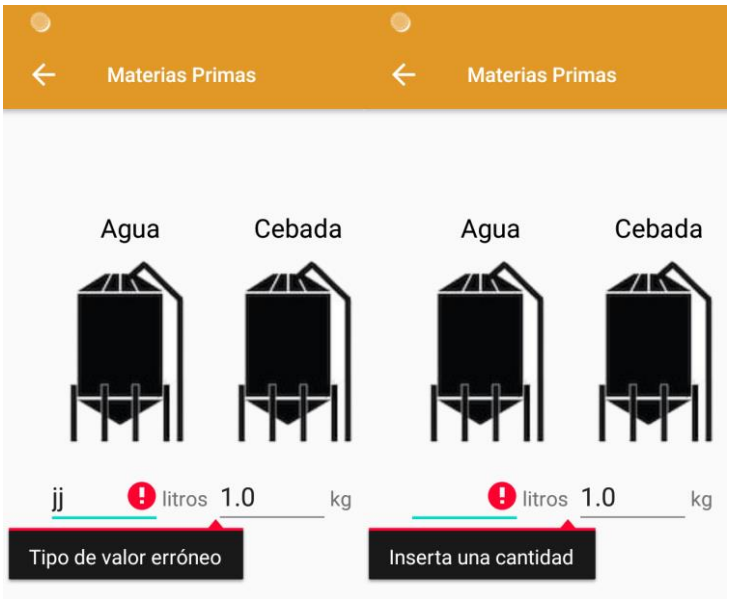

Figura 24: *Campo erróneo o vacío*

5.5. Proceso de fabricación

Figura 25: *Menú proceso fabricación*

En esta ventana mostramos todos los procesos que se deben llevar a cabo para la fabricación de la cerveza, en primer lugar, tenemos el botón de Producción, cuando pulsemos este botón se nos abrirá la ventana de producción (Figura 25). Y este botón se deshabilitará cuando le demos al botón Comenzar Producción que se encuentra en la ventana Producción.

Si no hemos pulsado el botón Comenzar Producción los demás botones de esta ventana estarán deshabilitados ya que es necesario elegir la receta y los litros de cerveza que se desean fabricar.

Como se observa también tenemos habilitado el botón de volver atrás y el de cerrar sesión en la barra superior.

Producción

Figura 26: *Pantalla Producción*

Como se ha comentado en el apartado anterior, en la ventana de producción se deberá introducir los litros de cerveza que se desean producir y la receta.

Dependiendo de la receta que se elija, algunas temperaturas durante el proceso de fabricación variarán, así como el tiempo de algunos procesos. También variará la levadura que se utilice para la fabricación.

Cuando se introducen los litros automáticamente se actualizan las cantidades necesarias para la fabricación.

Una vez elegida la receta y los litros se pulsa el botón *Comenzar Producción*. En caso de que la producción sea menor que la mínima (200 litros) nos muestra un mensaje indicándonos que si deseamos producir 200 litros.

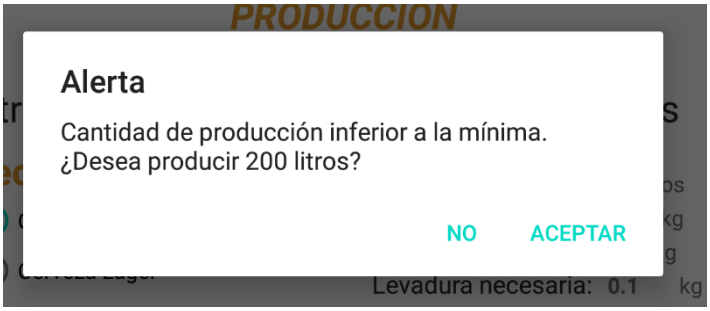

Figura 27: *Alerta producción inferior a 200 litros*

Si la producción es igual o superior a 200 litros, se comprueba si las materias primas disponibles son suficientes para la producción elegida. Si no son suficientes nos lo muestra mediante *AlertDialog*.

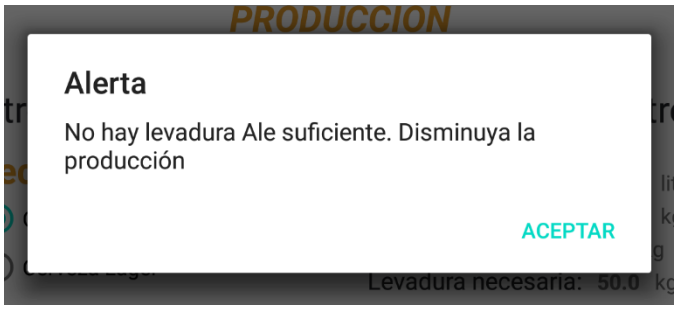

Figura 28: *Alerta levadura Ale. Disminuir producción.*

Como se observa nos salta la alerta de levadura Ale ya que es la receta que habíamos escogido con anterioridad. En caso de que la cantidad de cebada, agua, lúpulo o levadura Lager sea insuficiente también nos lo mostrará de la misma forma.

Si las cantidades disponibles son suficientes esta ventana se cierra.

Por último, como hemos comentado antes en la ventana de los procesos el botón de producción se deshabilitará y los demás se habilitarán.

Malteado

Si ya hemos elegido la producción que deseamos, se activa el botón malteado en el Menú Principal (Figura 24), lo pulsamos y se ejecuta la *activity Malteado*. Lo primero que hace la aplicación es comprobar que la humedad del grano este entre 11% y 13%. Si no es así, nos muestra las siguientes alertas ademas de indicarlo en los TextView de la parte superior de la ventana.

Además se deshabilitan los botones de Pausar y Finalizar malteado y se habilita el botón de Iniciar Malteado

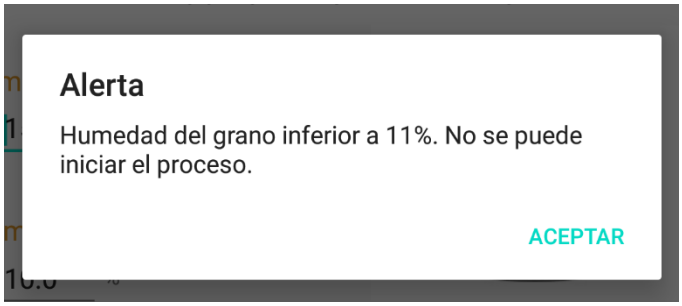

Figura 29: *Alerta humedad malteado inferior a 11%*

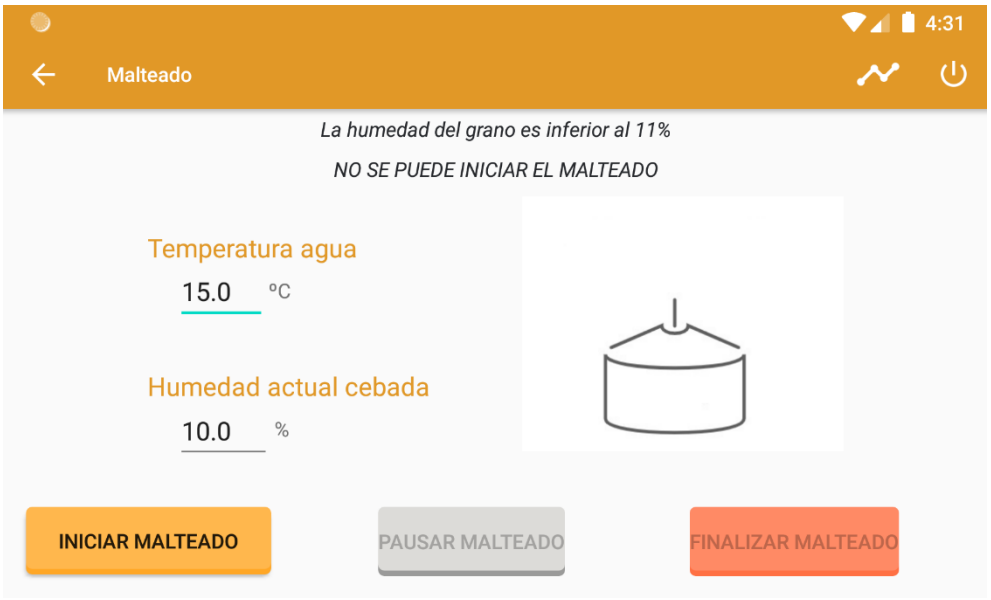

Figura 30: *Pantalla Malteado humedad inferior 11%*

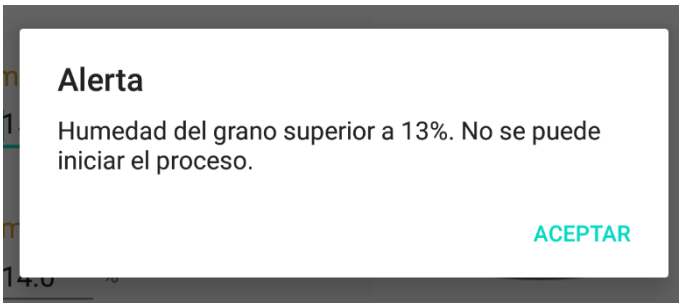

Figura 31: *Alerta humedad malteado superior a 13%*

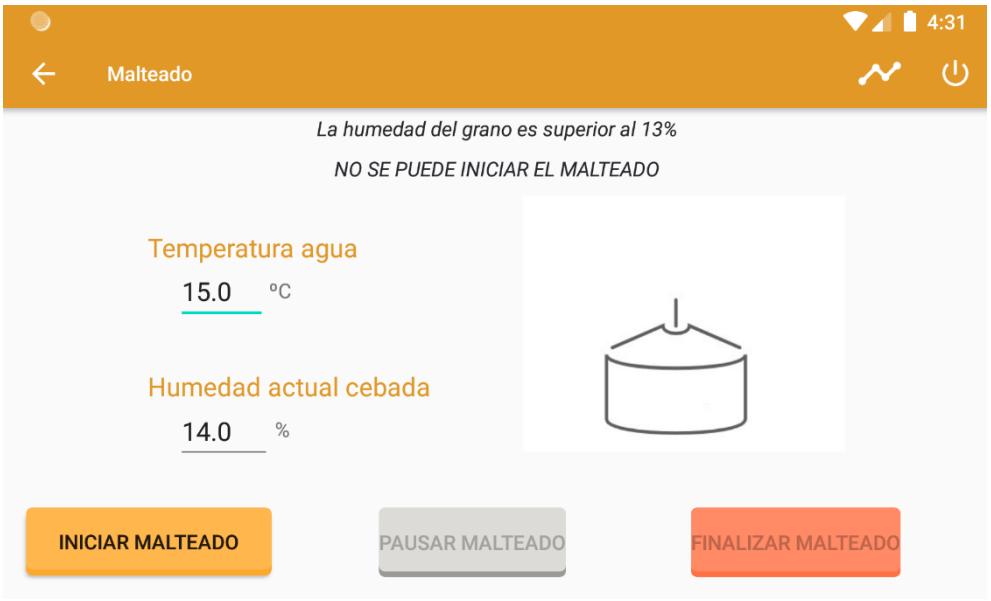

Figura 32: *Pantalla Malteado humedad superior 13%*

En caso de que la humedad estuviera entre 11% y 13% nos mostraría lo siguiente:

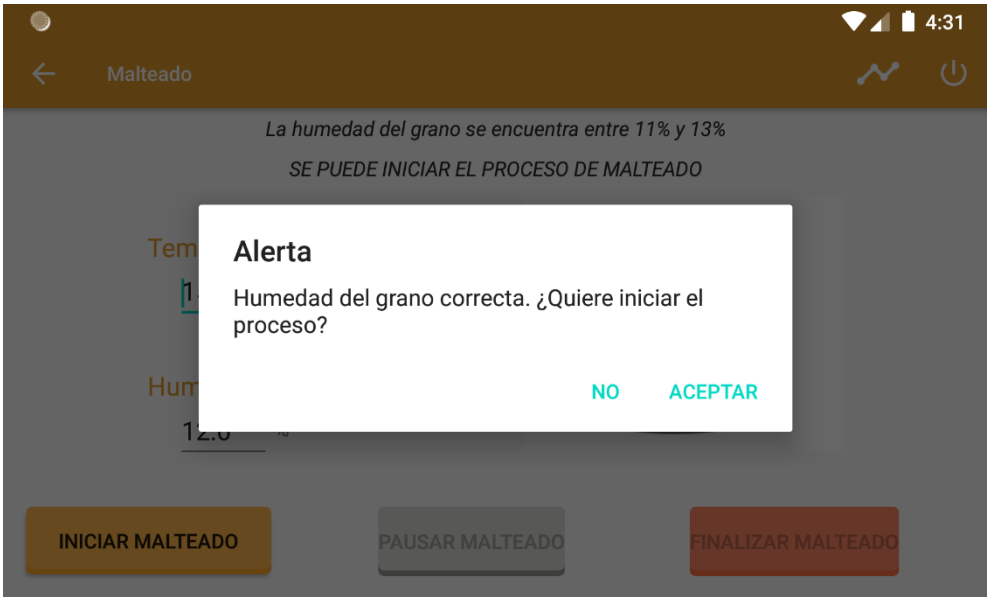

Figura 33: *Pantalla Malteado humedad inicial correcta*

Como se observa en los *TextView* de la pantalla también nos indica que la humedad inicial del grano es correcta y que por lo tanto podemos iniciar el malteado.
Si pulsamos el botón No, el malteado no se iniciará, lo cual podremos iniciarlo más adelante con el botón iniciar malteado. Si se pulsa el botón Aceptar el malteado comenzará y por lo tanto el botón de iniciar malteado se habilitará y se habilitarán los botones de pausar y finalizar.

Una vez iniciado el malteado se comprobará que la temperatura del agua se encuentre entre 15°C y 20°C. Si esto no es así la temperatura del agua para el remojado aumentará o disminuirá simulando que la estamos calentando o enfriando respectivamente.

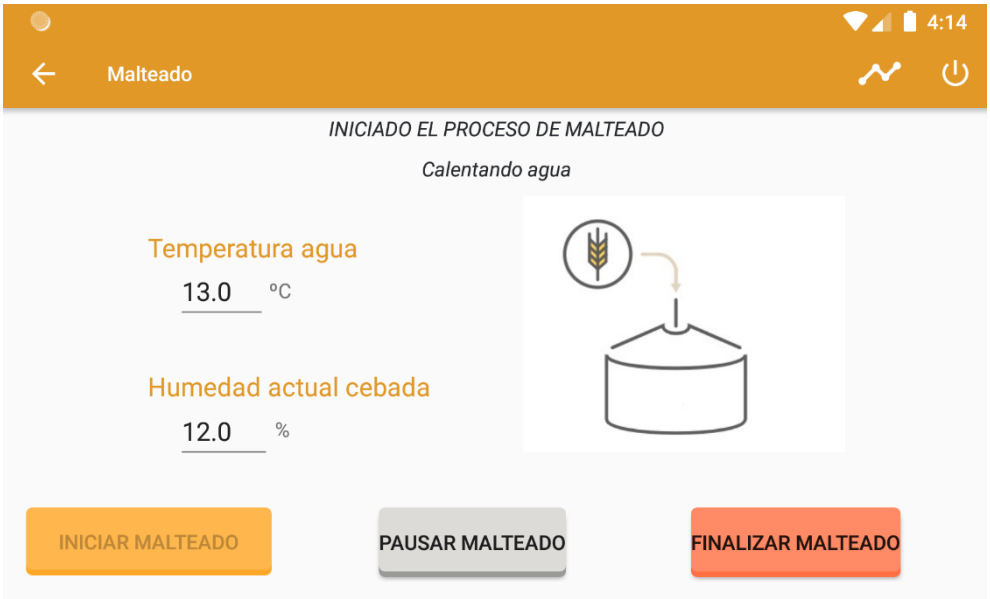

Figura 34: *Pantalla Malteado "Calentando agua"*

Como observamos, hemos introducido una temperatura de agua inferior a la necesaria para iniciar el remojado por lo tanto Nos indica que el agua se está calentando y vemos en la imagen que se está introduciendo la cebada en el depósito de malteado.

Una vez el agua ha alcanzado la temperatura entre 15 y 20°C, se introduce el agua en el depósito. La aplicación nos lo indica mediante *TextView* y mediante la imagen.

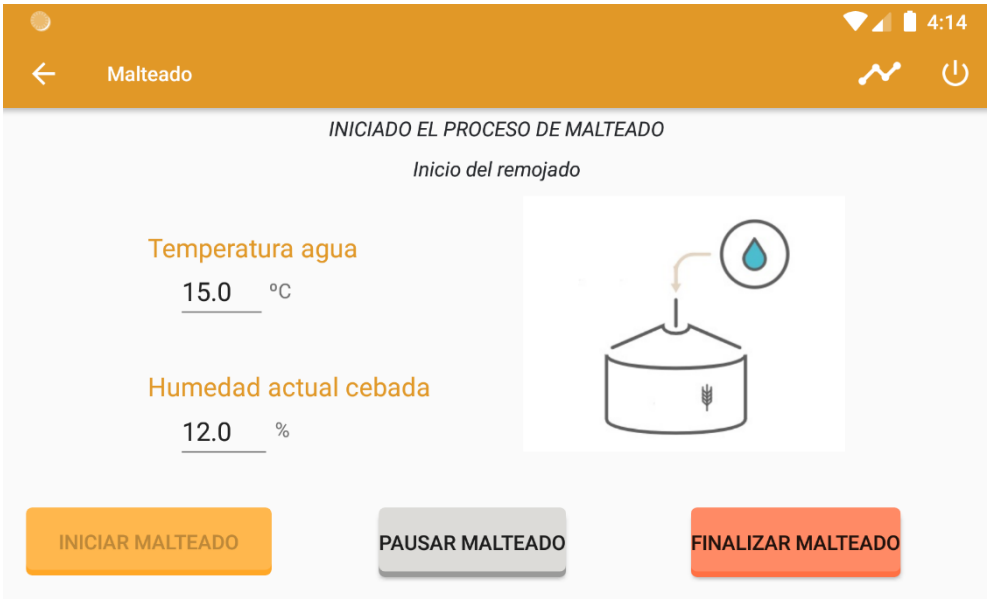

Figura 35: *Pantalla Malteado "Inicio del remojado"*

Tras unos segundos, simulando el tiempo que se tarda en introducir el agua, comienza a mezclarse el agua con el grano y la humedad del grano va a aumentando de forma lineal simulando la lectura del sensor de humedad del proceso.

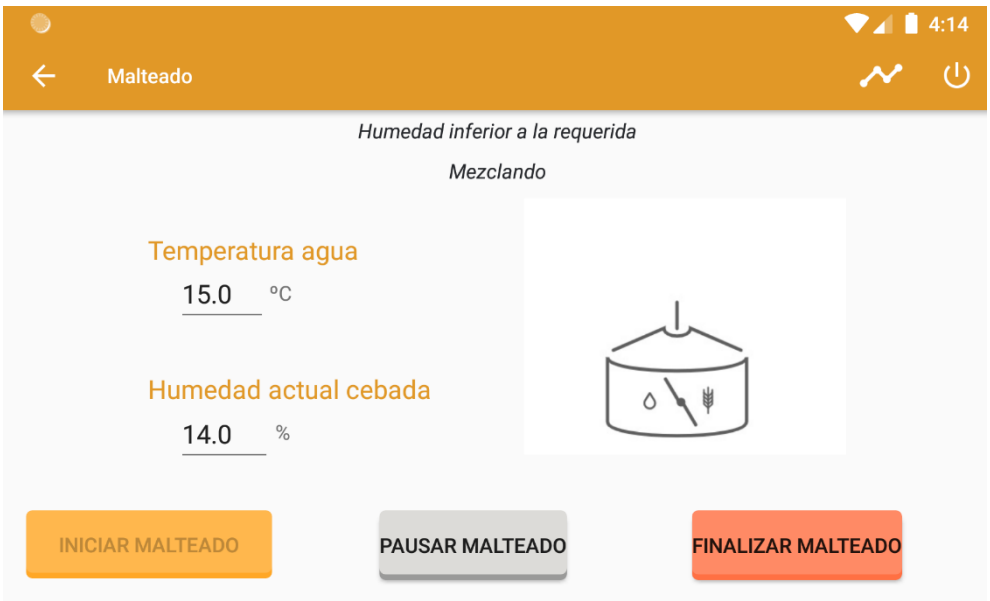

Figura 36: *Pantalla Malteado "Mezclando"*

Cuando la humedad del grano alcanza la humedad objetivo (45%) el proceso ha finalizado y se pasa a la fase de germinación. La aplicación nos lo indica mediante los *TextView*.
El botón de inicar malteado se vuelve a habilitar para poder comenzar de nuevo otro lote de producción. Aunque en nuestro caso la fabricación de varios lotes será una futura ampliación de la aplicación.

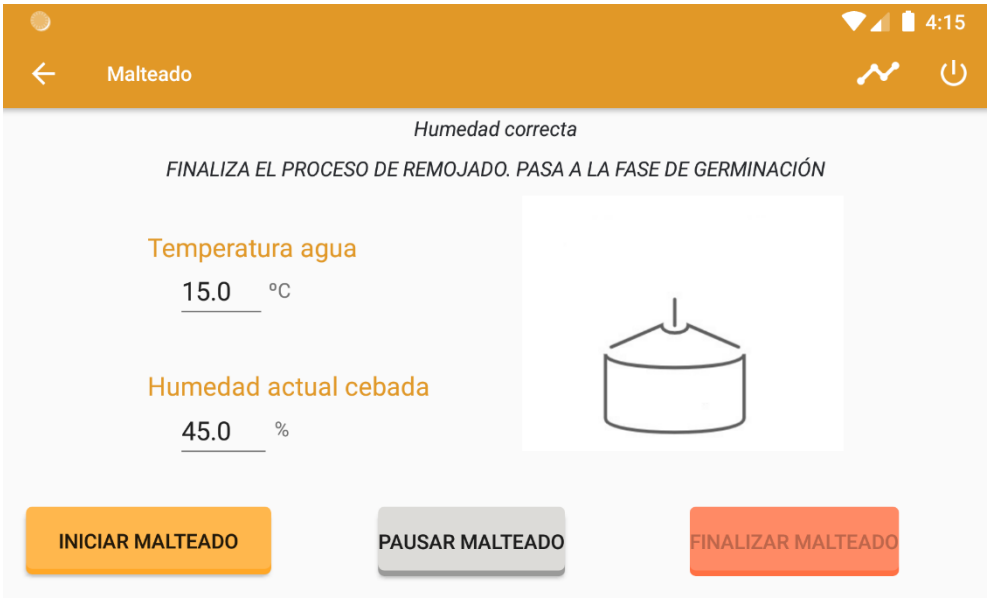

Figura 37: *Pantalla Malteado "Malteado finalizado"*

Por último, queda explicar el botón de las gráficas que es el siguiente:

Figura 38: *Botón gráficas*

Cuando pulsamos este botón se ejecuta la *activity MalteadoGraficas*, en la cual se muestra de forma gráfica el valor de la temperatura del agua y la humedad del grano en tiempo real.

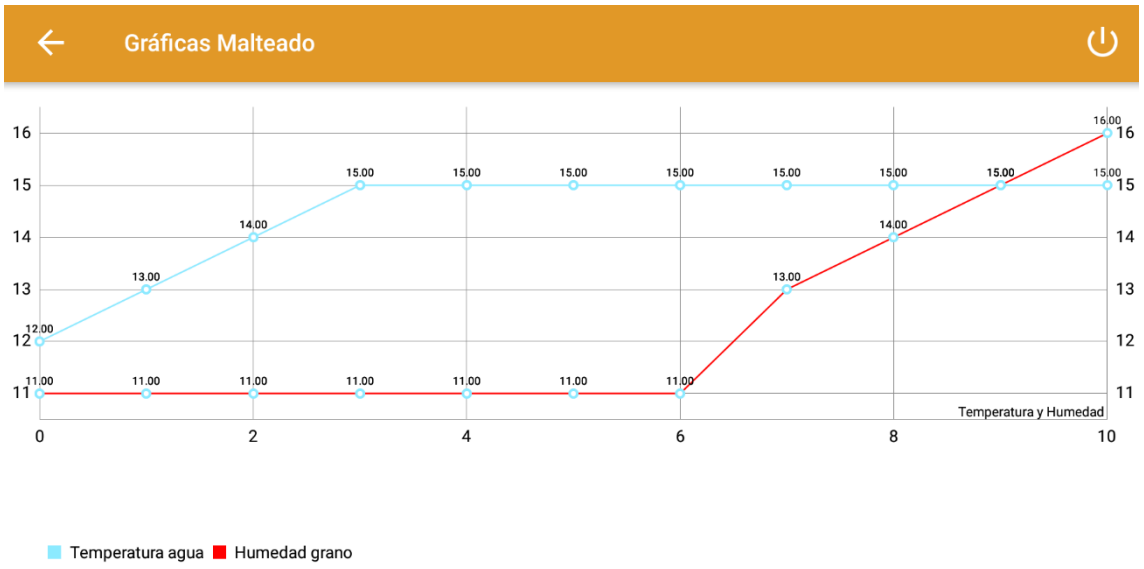

Figura 39: *Gráfico Malteado 1*

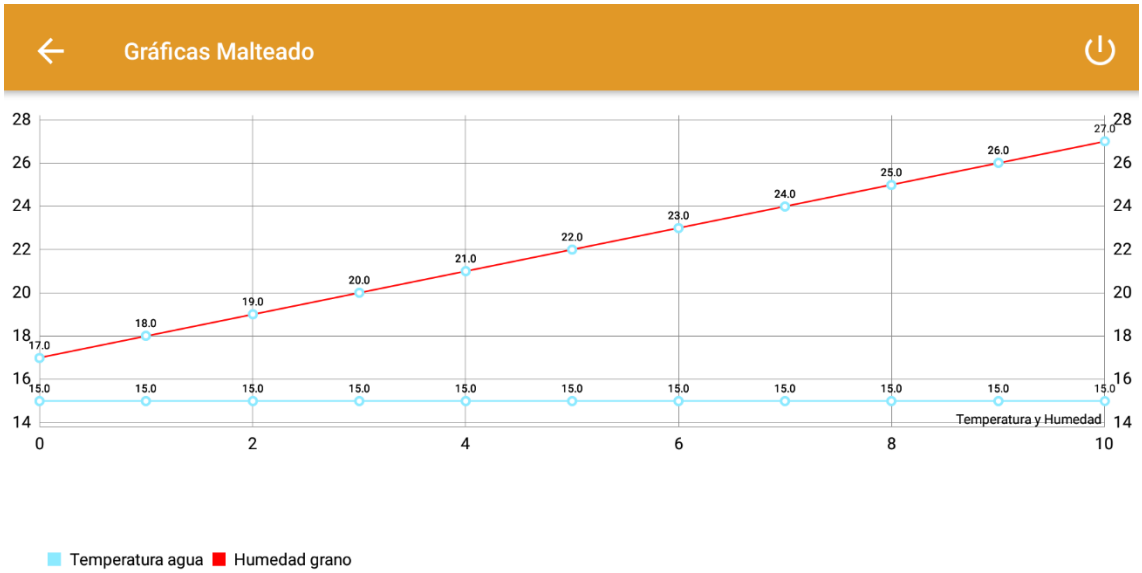

Figura 40: *Gráfico Malteado 2*

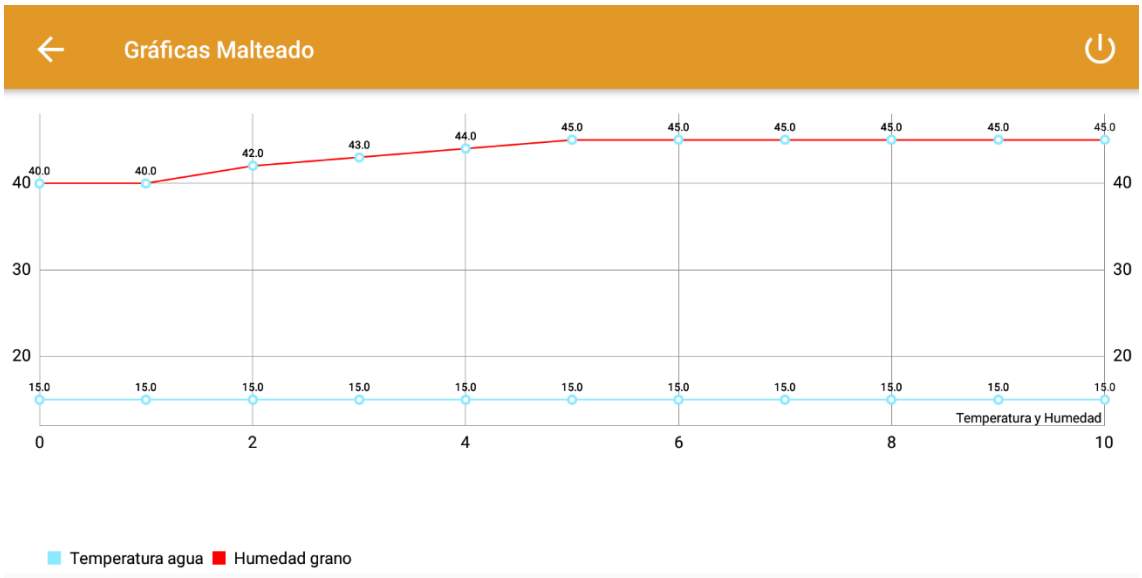

Figura 41: *Gráfico Malteado 3*

Cada 10 valores graficados el gráfico se limpia y comienza de nuevo para que la gráfica se pueda observar de forma clara.

Como se observa, la línea azul corresponde con la temperatura del agua, que comienza en 10°C y va aumentando hasta llegar a los 15°C, momento a partir del cual se estabiliza. En cuanto a la humedad del grano, se corresponde con la línea roja. Esta comienza en 11% y va aumentando de forma lineal con el tiempo al igual que la temperatura del agua. Cuando la humedad del grano alcanza el 45%, es decir, la humedad que deseamos conseguir con el proceso de malteado, esta se estabiliza y el proceso de malteado ha finalizado.

Además, tanto en la pantalla de Malteado como en la de las gráficas se incluye el botón de *Cerrar Sesión* y el de *Volver atrás*.

Germinación

Como se puede observar en la siguiente imagen el proceso de germinación ya ha empezado ya que el proceso de malteado ha finalizado y por lo tanto el cronómetro de la germinación ha comenzado. Nos muestra en un *TextView* el tiempo que ha pasado y además el círculo azul se trata de un *ProgressBar* que a medida que va pasando el tiempo, el círculo se va volviendo naranja.

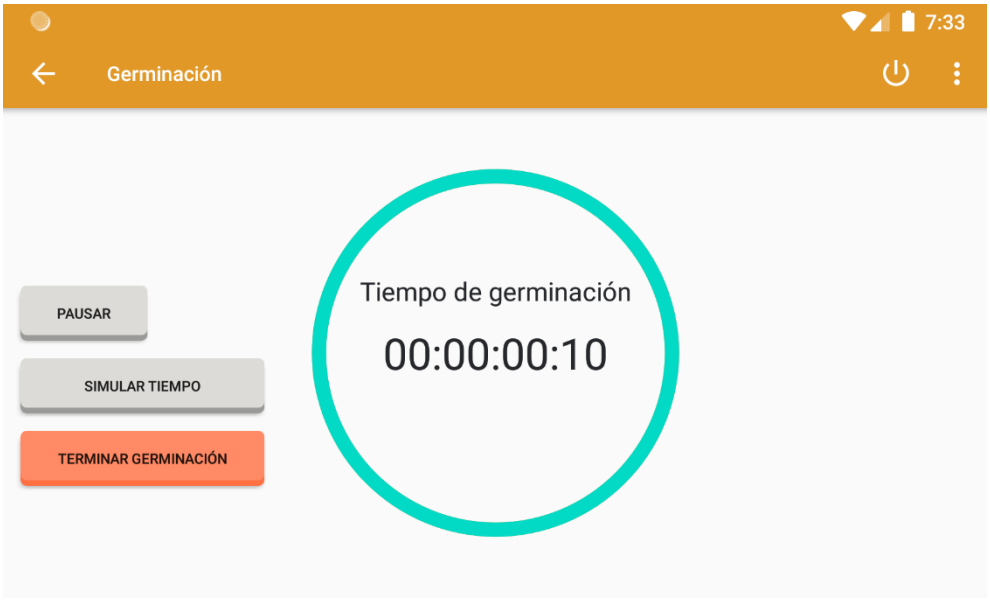

Figura 42: *Pantalla Germinación "Iniciar"*

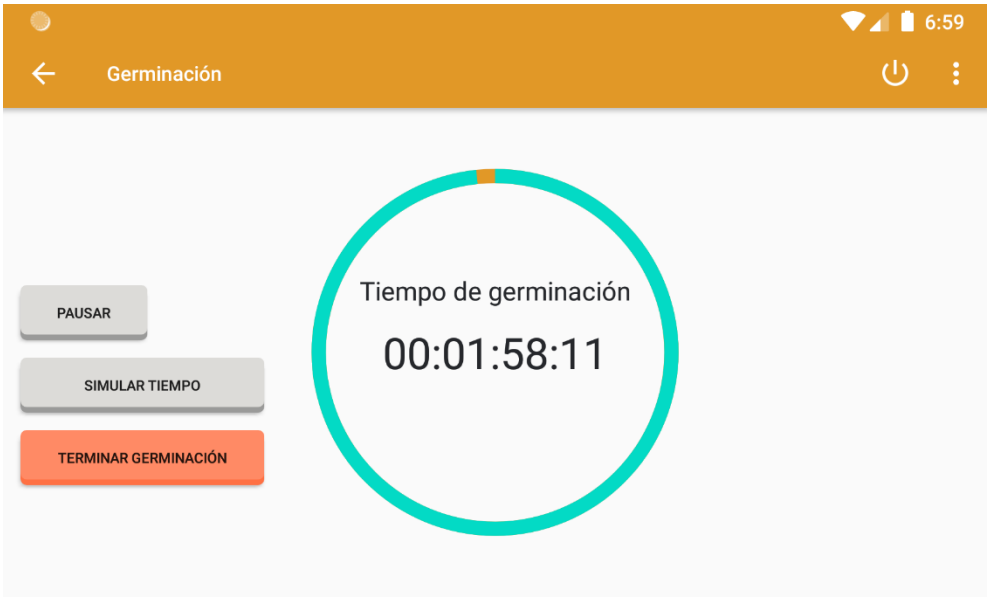

Figura 43: *ProgressBar Germinación*

En esta pantalla se observan 3 botones, aunque en realidad existen 4. El botón de Iniciar el cronómetro se ha ocultado ya que el proceso ya se ha iniciado.

Cuando pulsamos el botón de pausar, el cronómetro de la germinación se pausa, aunque en realidad el proceso no se pausaría ya que los granos de cebada se encuentran en la caja de germinación. Se trata de un botó para casos en los que se requiera reparar algo.

Cuando pulsamos el botón de pausar, este se oculta y se habilita el de Iniciar.

Figura 44: *Pantalla Germinación "Pausar"*

El botón de simular tiempo es para simular que el proceso esta apunto de acabar y que se pueda observar que pasaría en la realidad.

Y por último el botón de finalizar en caso de que ocurra alguna emergencia parar la germinación por completo. Si pulsamos este botón simulamos que el proceso de germinación ha finalizado y nos lo muestra por pantalla de las siguientes dos formas: con un *TextView*, y con un *AlertDialog.* Estas dos formas de indicarnos la información también se dan cuando el reloj llega a 5 días.

Figura 45: *Pantalla Germinación "Proceso Finalizado"*

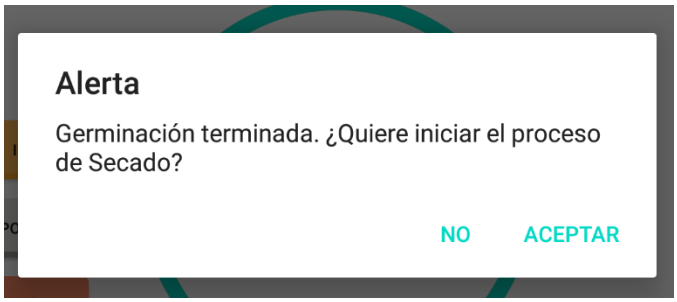

Figura 46: *Alerta "Proceso Germinación Finalizado"*

Si pulsamos el botón de *Aceptar* el proceso de Secado comenzará automáticamente, si pulsamos *No*, no comenzará el secado y se deberá iniciar de forma manual.

Secado o tostado

Cuando en el menú de los procesos elegimos el proceso de secado o tostado se ejecuta la *activity_secado_tostado*.

En primer lugar debemos introducir los datos de la temperatura del aire, la temperatura de la malta y la humedad de la malta de forma manual. De esta forma simulamos la lectura de los sensores correspondientes.

Además la temperatura maxima y mínima para el aire en la fase 2 se cargará en función de la receta que hayamos elegido, en este caso hemos elegido la receta Lager.

En caso de que la germinación hubiera acabado el proceso de secado y tostado se iniciaría automáticamente si en la Alerta de la figura 46 se hubiera pulsado aceptar, en este caso se pulsó que no, por lo tanto, lo haremos de forma manual con el botón iniciar.

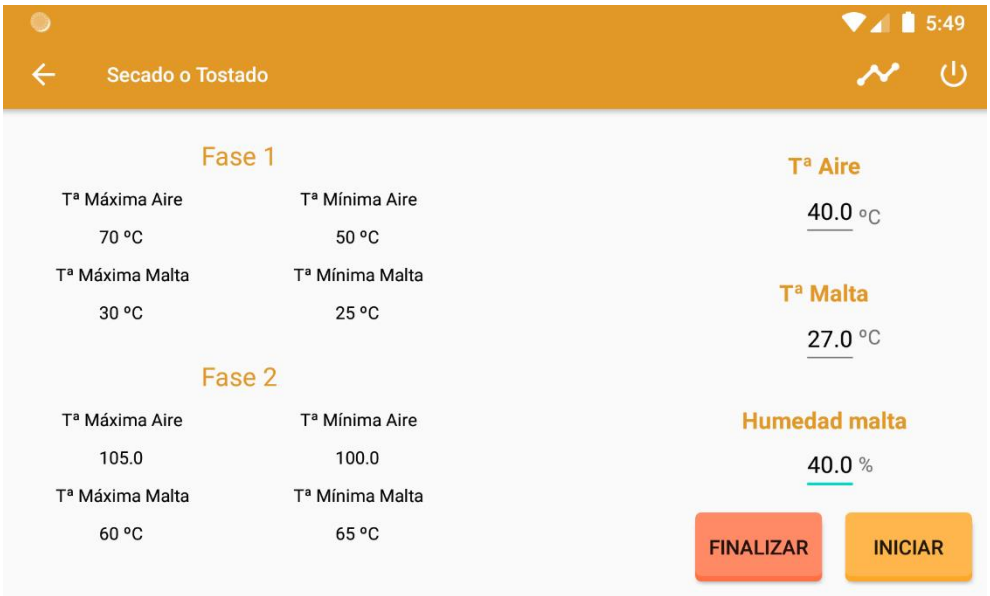

Figura 47: *Pantalla Proceso Secado y Tostado*

Cuando pulsamos el botón de iniciar, nos aparece una flecha que nos indica en que fase nos contramos. La temperatura del aire aumenta o disminuye para estar en el rango óptimo que se indica en los datos de la fase 1 que aparecen a la izquierda de la pantalla. Una vez la temperatura esté entre ese rango, la humedad empieza a dismiuir.

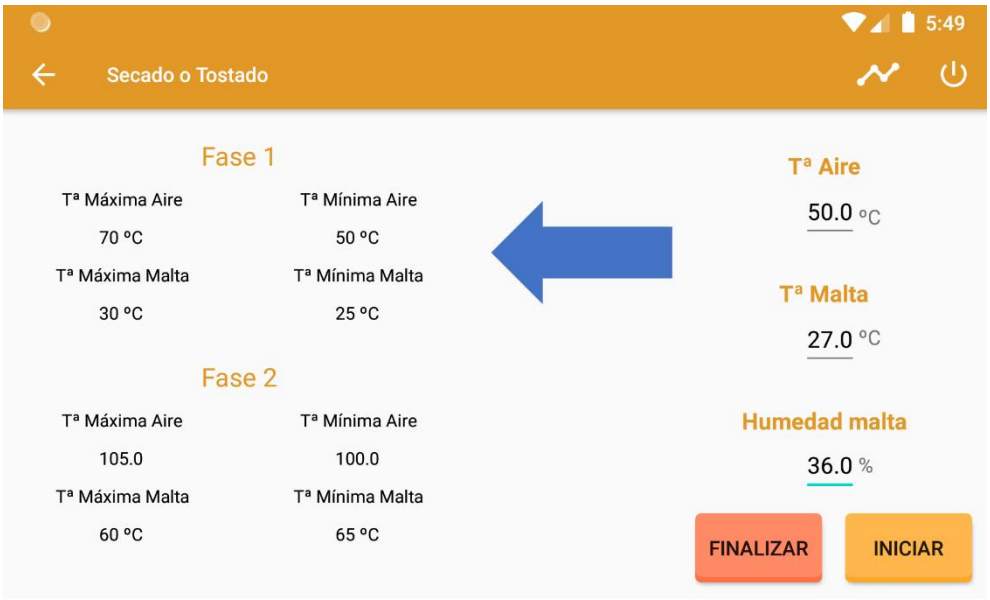

Figura 48: *Pantalla Proceso Secado y Tostado Fase 1*

A continuación se observa como cuando la humedad de la malta es del 10% o menor, la flecha de la fase 1 desaparece y la temperatura del aire y de la malta empiezan a aumentar o disminuir para estar en los rangos óptimos que se indican a la derecha en la fase 2.

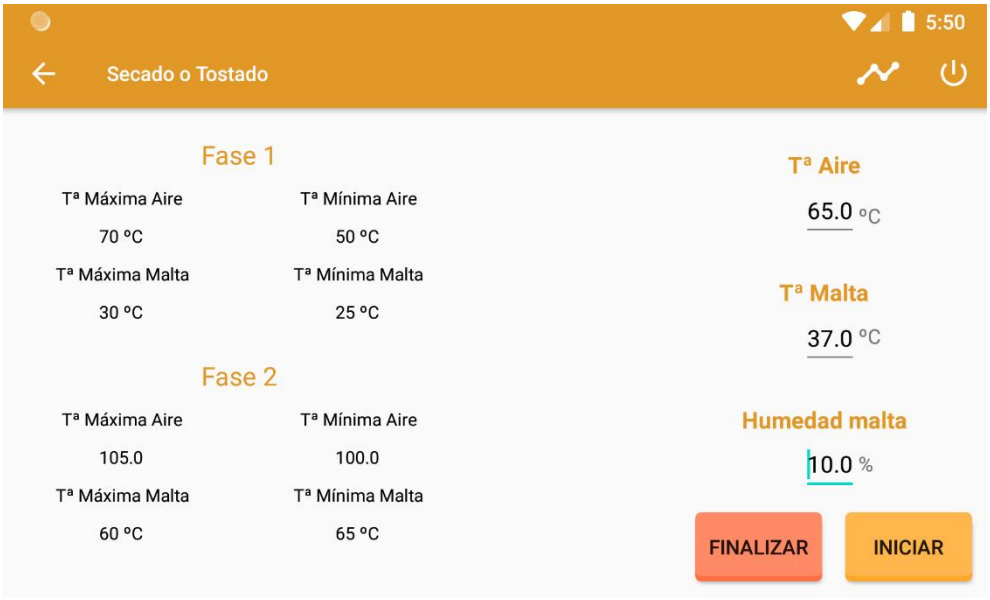

Figura 49: *Pantalla Proceso Secado y Tostado Fase 1 finalizada*

Cuando se alcanzan los valores de temperatura que se desean aparece de nuevo una flecha que nos indica que se esta ejecutando la fase 2. Simulamos que la humedad de la malta comienza a disminuir.

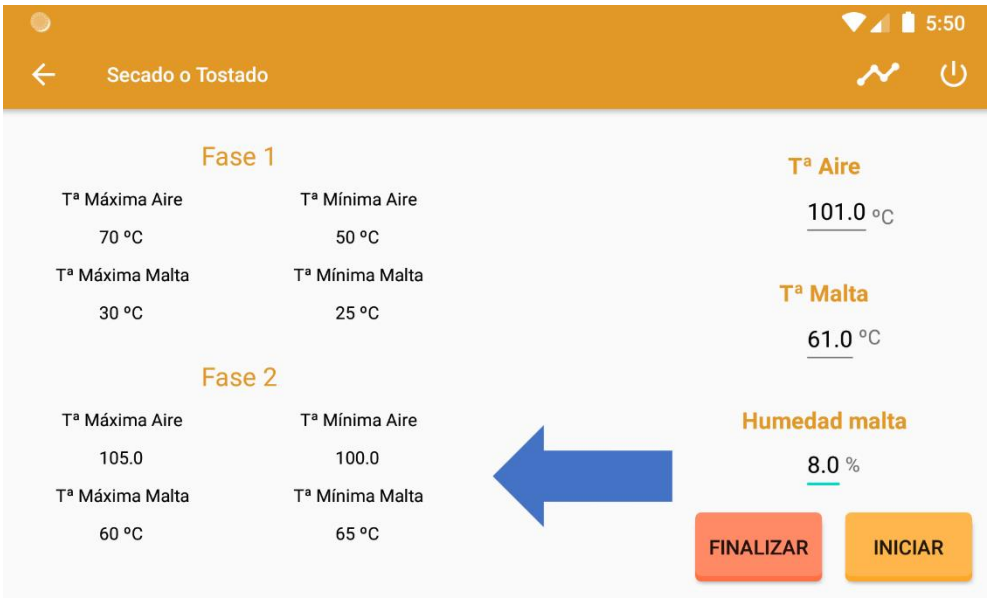

Figura 50: *Pantalla Proceso Secado y Tostado Fase 2*

Cuando la humedad de la malta es menor o igual al 5%, el proceso finaliza y nos muestra una ventana emergente indicándolo.

Figura 51: *Alerta "Proceso Secado Finalizado"*

En caso de que se pulsara el botón finalizar nos aparecería esta misma alerta y simularíamos que el proceso ha finalizado.

Por último, como se observa en la barra horizontal superior de la pantalla, tenemos la opción de ver los datos de forma gráfica. En este caso hemos graficado las distintas variables de forma separada.
Cuando pulsamos el botón de las gráficas se abre la *activity_secado_tostado_graficas* en la cual nos aparece la variable humedad de la malta. Esta variable se grafica en tiempo real y a continuación se muestra cómo.

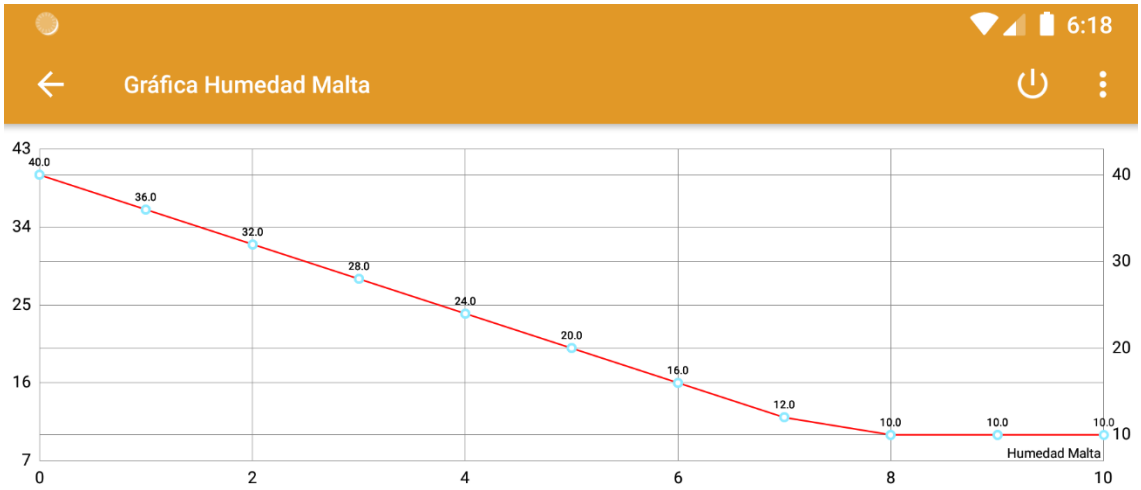

Figura 52: *Gráfica Humedad Malta Fase 1*

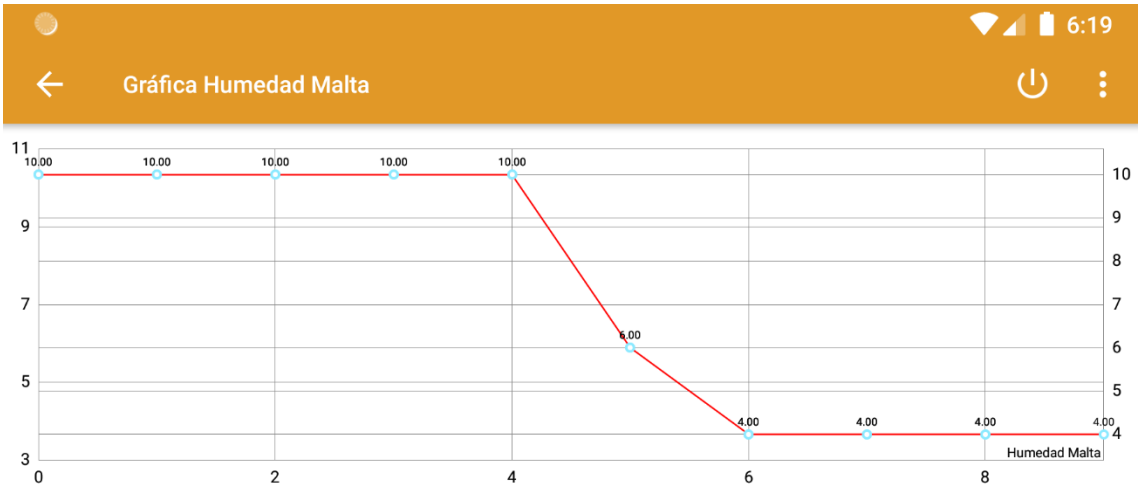

Figura 53: *Gráfica Humedad Malta Fase 2*

Como se observa las curvas son las esperadas, la humedad comienza en 40% para la fase 1 y va disminuyendo hasta alcanzar el 10%, una vez alcanzado se estabiliza durante un tiempo, tiempo en el cual simulamos que las temperaturas del aire y la malta estén dentro del rango óptimo y cuando esto ocurre, la humedad vuelve a disminuir hasta que esta es igual o menos al 5%, momento en el cual se vuelve a estabilizar ya que el proceso finaliza.

Para observar las distintas variables debemos pulsar sobre el siguiente botón:

Figura 54: *Botón opciones gráficas Secado*

Al pulsarlo, nos aparece lo siguiente:

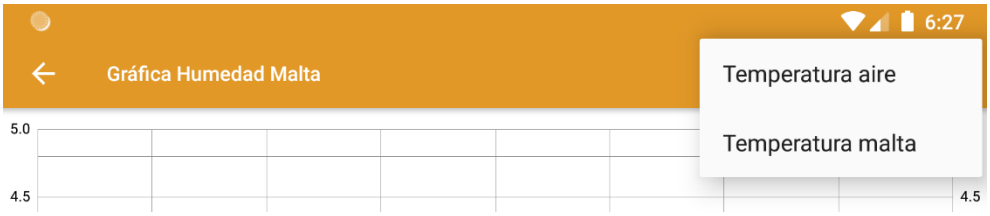

Figura 55: *Botón opciones gráficas Secado 2*

Si elegimos la opción de la variable temperatura aire se ejecutará y abrirá la *activity_temp_aire_secado* en la cual se grafica en tiempo real esta variable.

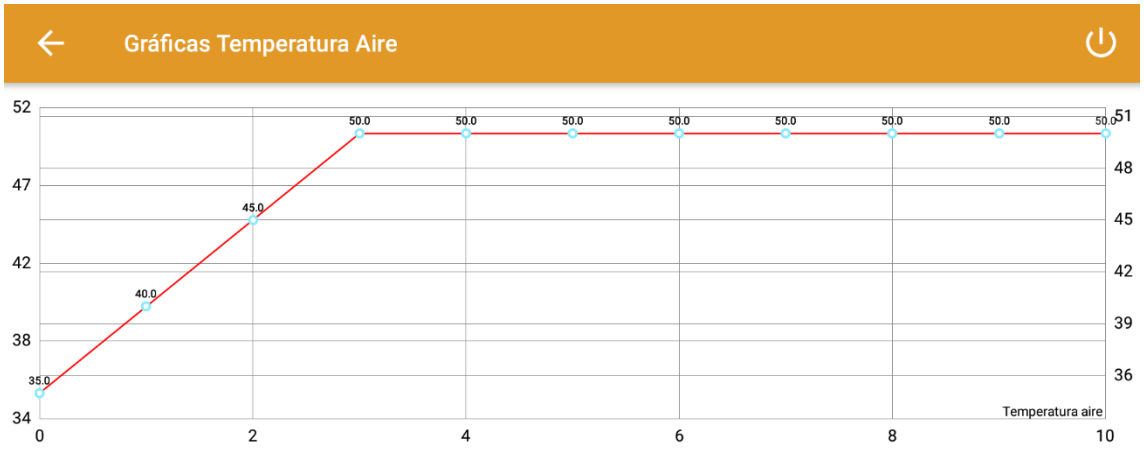

Figura 56: *Gráfica Temperatura Aire Fase 1*

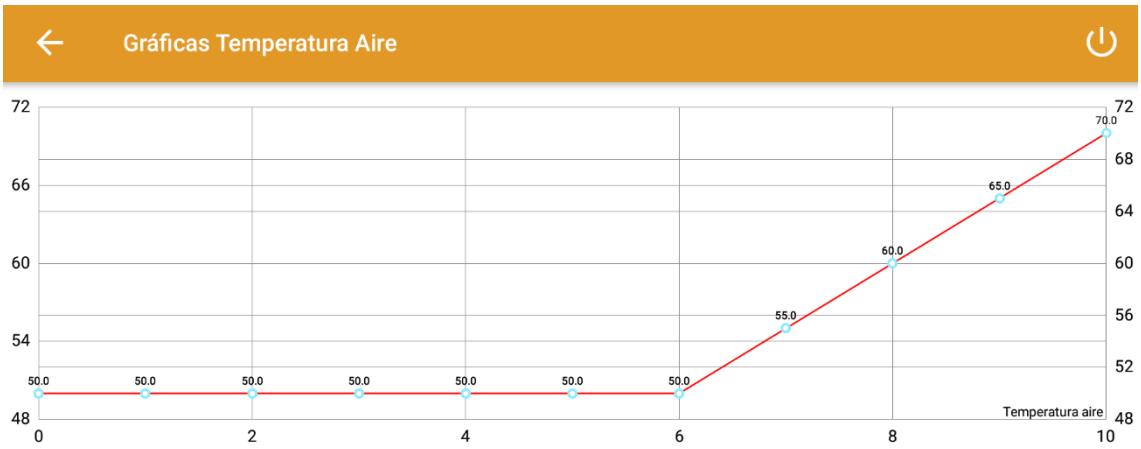

Figura 57: *Gráfica Temperatura Aire Fase 1-2*

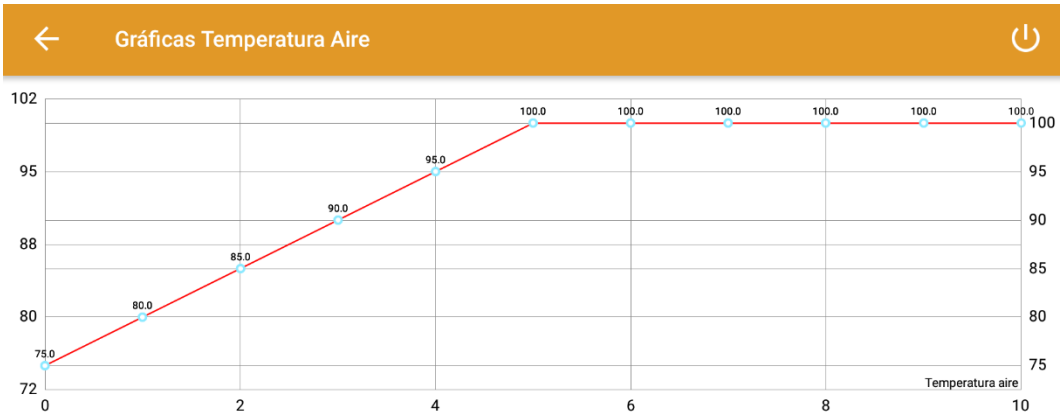

Figura 58: *Gráfica Temperatura Aire Fase 2*

Como se observa en las gráficas también son las esperadas, la temperatura inicial simulada era de 25°C, esta va aumentando hasta 50°C que es la temperatura objetivo-mínima para la fase 1, por lo tanto, en este valor se estabiliza durante un tiempo, tiempo en el cual la humedad de la malta está disminuyendo hasta llegar al 10%. Cuando la humedad es del 10%, la temperatura vuelve a comenzar a aumentar para alcanzar los 100°C que es la temperatura objetivo-mínima para la fase 2, momento en cual también se estabiliza y la humedad de la malta disminuye hasta alcanzar el 5% o menos y finaliza el proceso.

Si en la figura 55 elegimos la opción de la variable temperatura malta se ejecutará y abrirá la *activity_temp_malta_secado* en la cual se grafica en tiempo real esta variable.

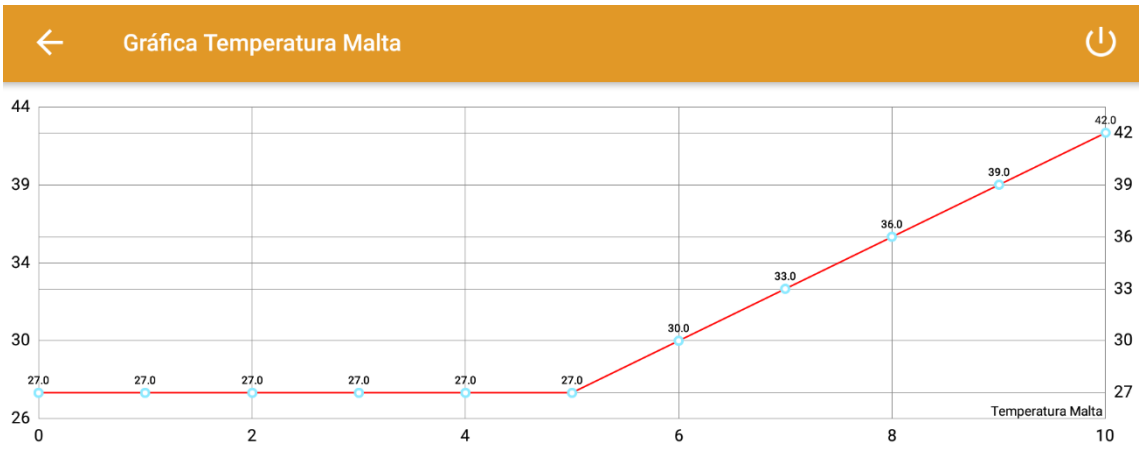

Figura 59: *Gráfica Temperatura Malta Fase 1*

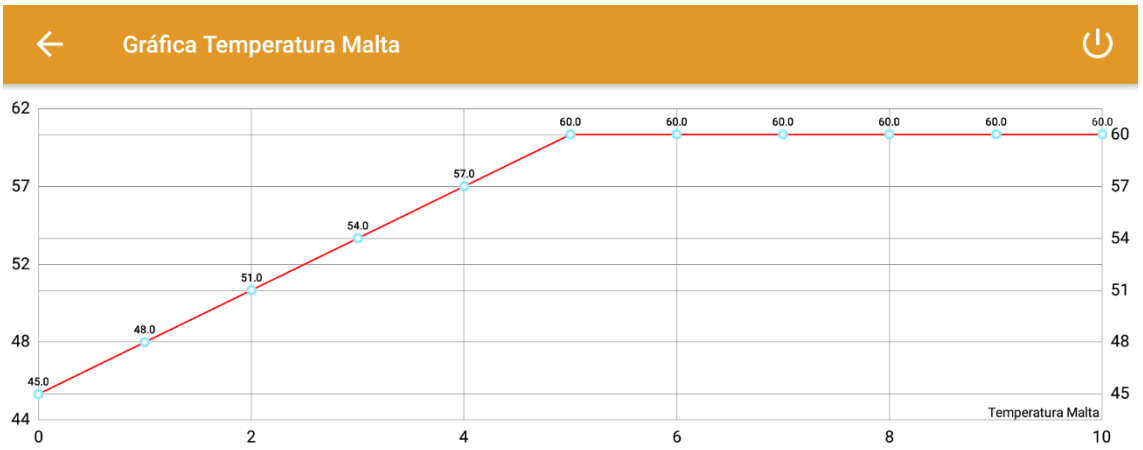

Figura 60: *Gráfica Temperatura Malta Fase 2*

Como se observa en las gráficas también son las esperadas, la temperatura inicial simulada era de 27°C, esta se mantiene constante que este valor está dentro del rango óptimo para la fase 1. Se mantiene constante hasta que la temperatura del aire es igual a 50°C y la humedad de la manta es del 10%, en este momento la temperatura de la malta comienza a aumentar hasta los 60°C que es la temperatura objetivo-mínima para la fase 2, momento en cual también se estabiliza y la humedad de la malta disminuye hasta alcanzar el 5% o menos y finaliza el proceso.

Por último, hay que destacar que en todas las pantallas se incluye el botón de cerrar sesión.

Molturación

Cuando pulsamos sobre el botón de Molturación en el menú de procesos se ejecuta y abre la *activity_molturación*. En primer lugar el sistema calcula el tiempo necesario de molturación necesario dependiendo de los litros que estemos produciendo.

Al igual que en el proceso de germinación este proceso se trata de un cronómetro que se inicia automaticamente o manualmente con el botón de start si el proceso de secado a acabado. En este caso no se ha iniciado automaticamente.

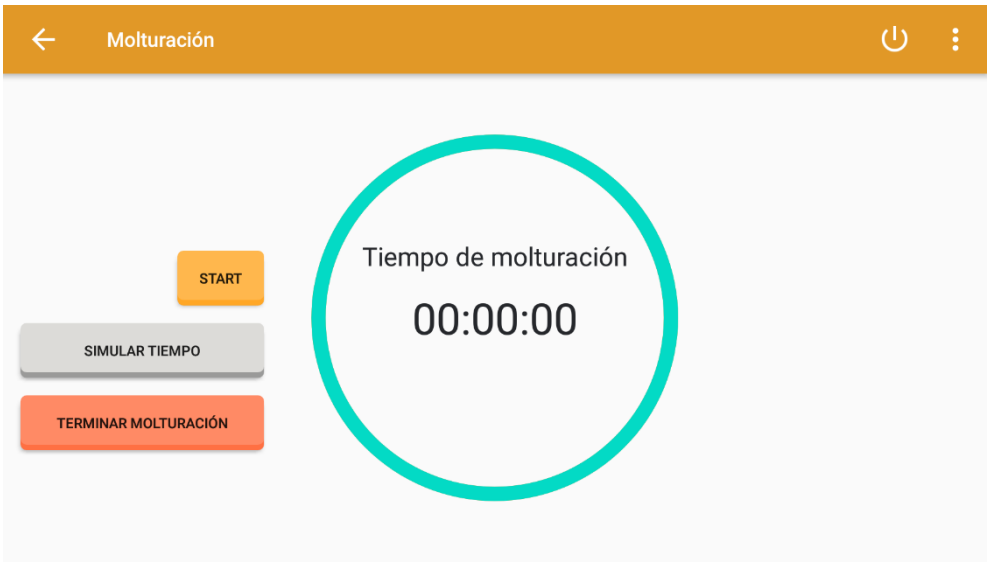

Figura 61: *Pantalla Molturación*

Cuando pulsamos el botón de start, el cronómetro se pone en marcha y este botón se oculta. Se habilita el botón de pausar por si se quiere pausar el proceso por algún motivo.

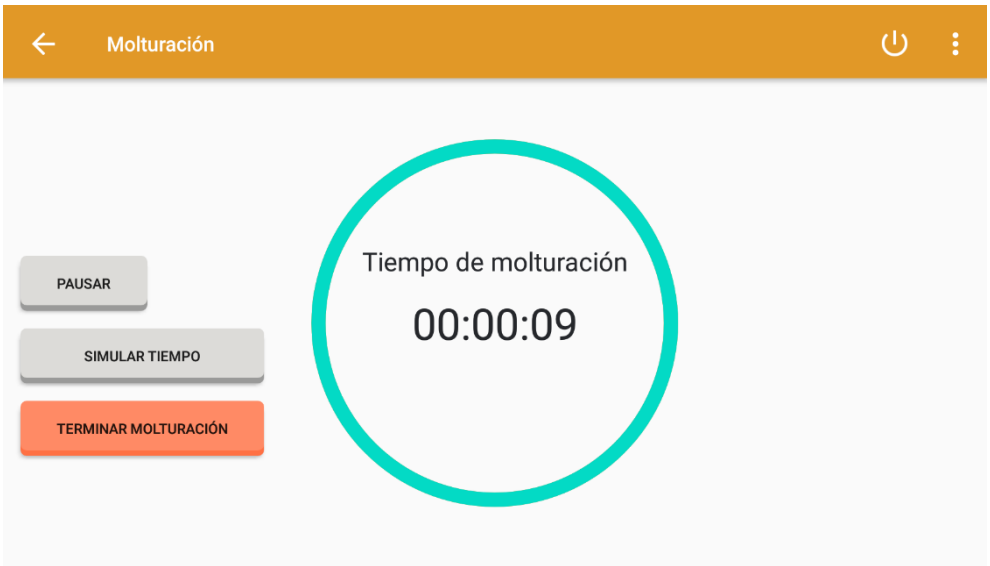

Figura 62: *Molturación iniciada*

El botón de simular tiempo nos sirve para simular que el proceso está a punto de acabar y observar que ocurre cuando este acaba.

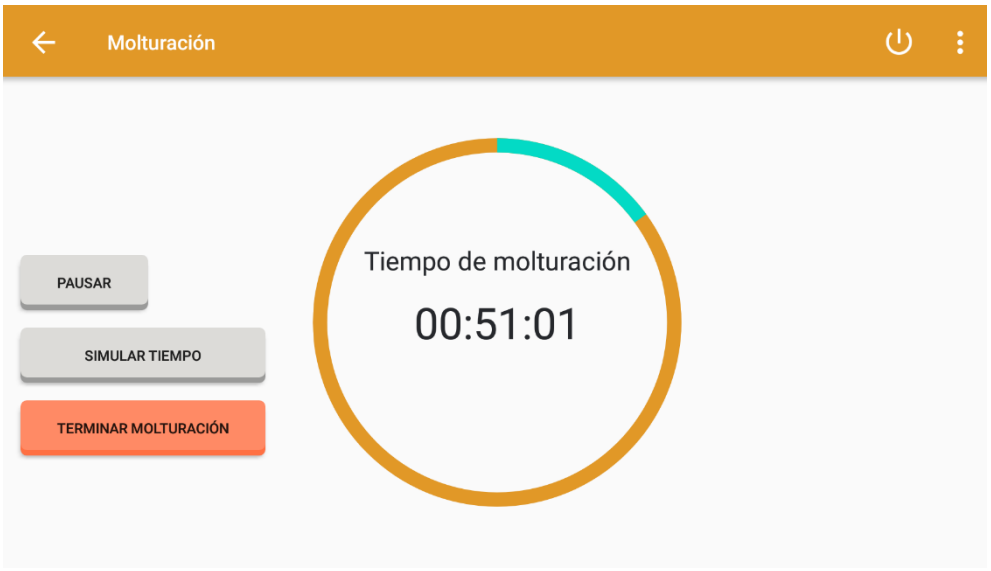

Figura 63: *Molturación iniciada*

Cuando el proceso alcanza el tiempo estimado la pantalla muestra que el proceso ha finalizado mediante un TextView y ademas nos muestra una ventana emergente indicandonos que ha finalizado y preguntando si queremos iniciar el siguiente proceso. Esto también ocurre con el botón de finalizar, con el cual simulamos que el proceso ha acabado y nos serviría para parar el proceso en caso de emergencia.

Figura 64: *Molturación finalizada*

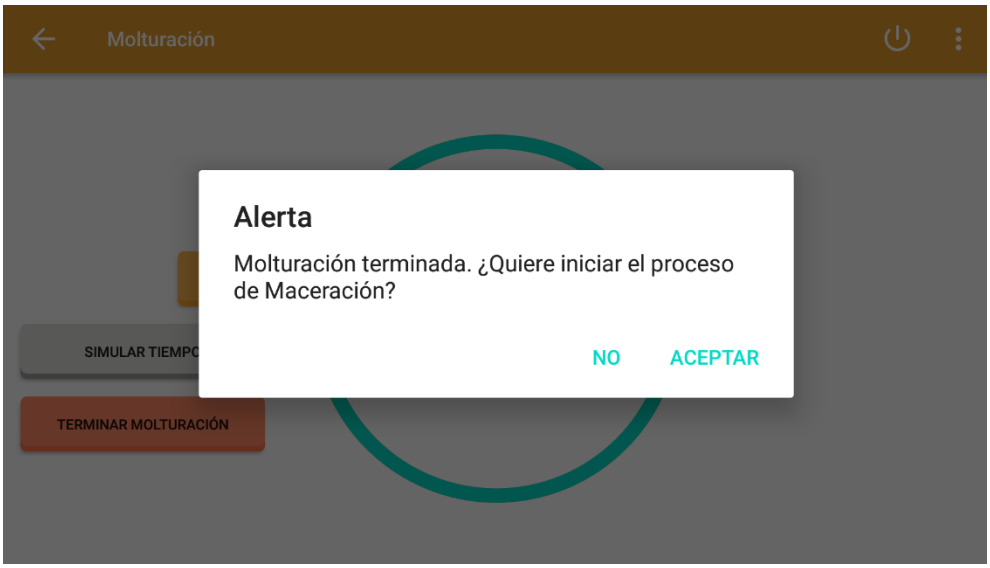

Figura 65: *Alerta Molturación finalizada*

Si en la alerta pulsamos sobre el botón de aceptar el proceso de maceración comenzará automáticamente. Si pulsamos sobre el botón no, deberemos iniciar la maceración de forma manual.

Maceración

Cuando pulsamos sobre el botón de Maceración en el menú de procesos se ejecuta y abre la *activity_maceracion*. Si en la alerta de molturación finalizada hemos cogido la opción aceptar este proceso se iniciará automáticamente, si no, se debe iniciar de forma manual con el boton iniciar como es este caso.

Figura 66: *Pantalla Proceso Maceración*

Cuando pulsamos el botón de iniciar el proceso comienza a simular el aumento de la temperatura del agua y nos lo muestra mediante un *TextView* que nos indica “Calentando agua”. Además, el botón de iniciar se oculta. Si pulsamos pausar el botón pausar se oculta al igual que antes de iniciar el proceso como vemos en la figura anterior:

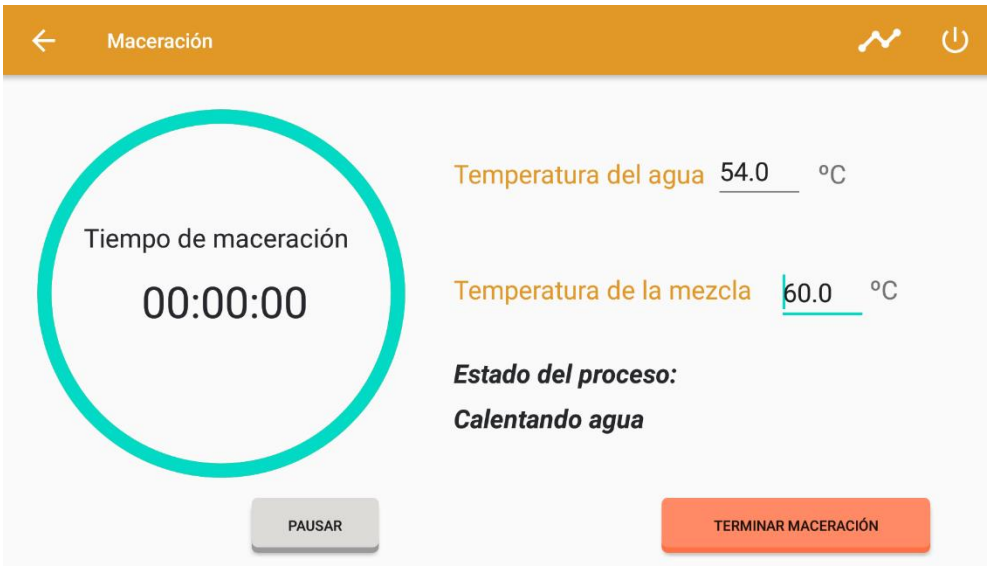

Figura 67: *Pantalla Maceración “Calentado agua”*

Cuando la temperatura del agua alcanza la temperatura de maceración objetivo dependiendo de la receta, simulamos simlamos que se introduce el agua en el depósito de maceración donde se encuentra la malta durante unos segundos.

Figura 68: *Pantalla Maceración “Temperatura correcta”*

Tras estos segundos el proceso de maceración comienza y cronómetro se inicia. En ese momento simulamos que se esta mezclando la mezcla a una emperatura de unos 9 grados inferior a la temperatura a la que se ha introducido el agua.

Figura 69: *Pantalla Maceración "La maceración ha empezado"*

Cuando el cronónometro indica que han pasado 2 horas, momento en el cual el proceso finalizar y nos lo muestra en una ventana emergente. Esta ventana emergente también la muestra cuando pulsamos el botón de terminar maceración. Simulamos que el proceso ha finalizado y nos sirve para casos de emergencia en los cuales queramos parar el proceso. Si pulsamos el botón de pausar, se pausa el cronómetro.

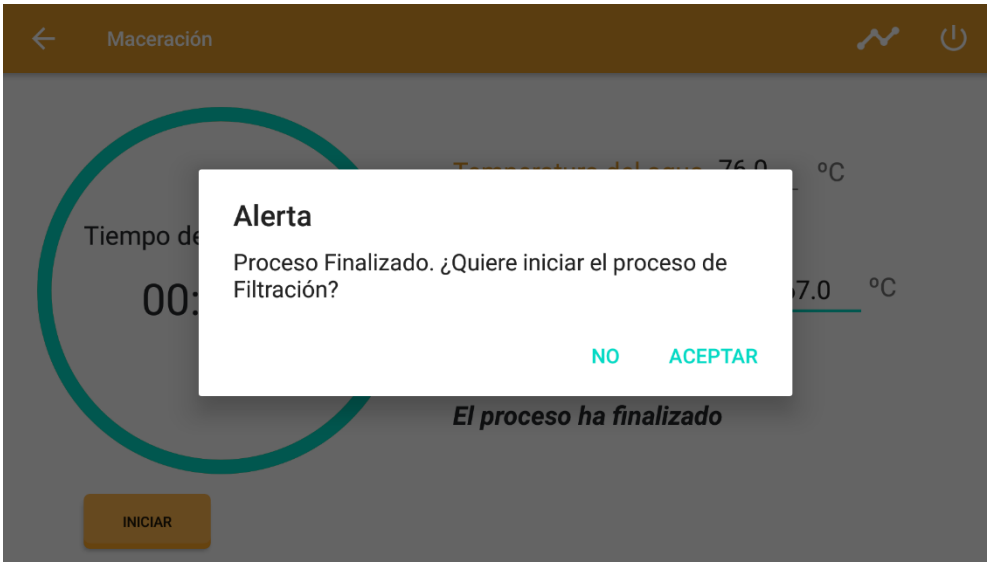

Figura 70: *Alerta Maceración "El proceso ha finalizado"*

Como se observa, los botones de pausar y terminar maceración se ocultan y vuelve a aparecer el de iniciar.

Si pulsamos sobre el boton aceptar, el proceso de filtración comenzará automáticamente, si pulsamos no, se deberá iniciar manualmente desde la pantalla de Filtración.

Por último, observamos que en este proceso tambien tenemos la opción de ver las variables de forma gráfica en tiempo real. Cuando pulsamos el botón de gráficas se ejecuta la *activity_maceracion_graficas*.

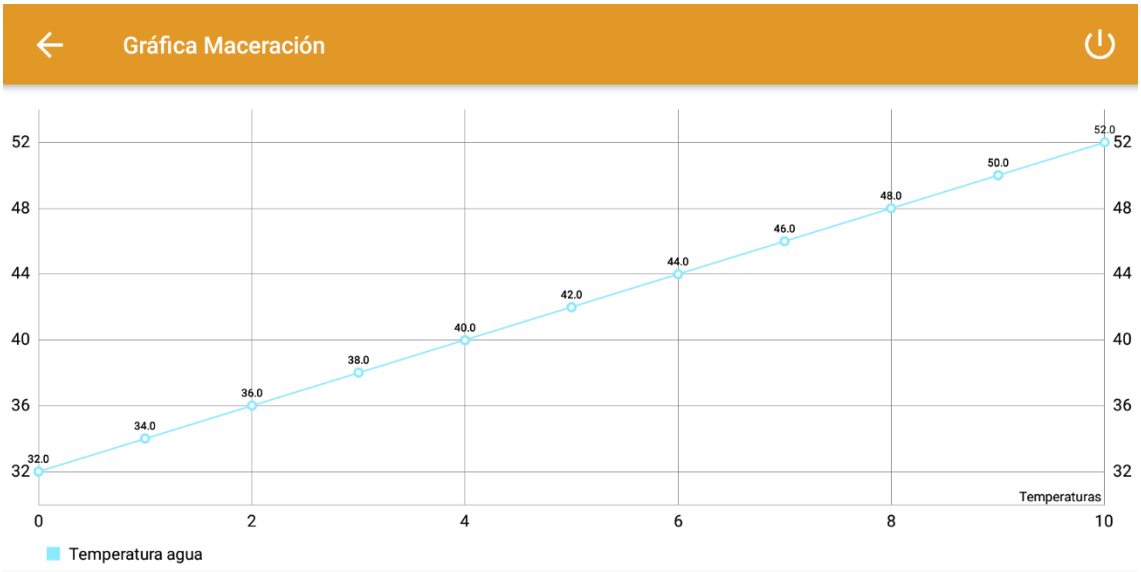

Figura 71: *Gráfico Temperatura agua 1*

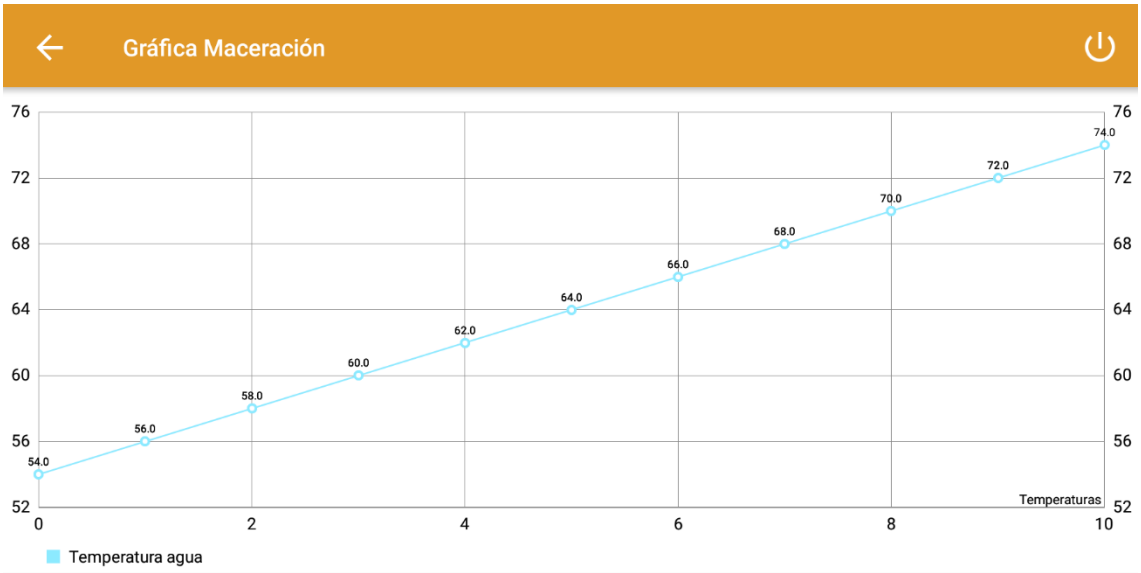

Figura 72: *Gráfico Temperatura agua 2*

Como se observa la temperatura del agua aumenta desde el valor inicial que hemos introducido que son 30°C hasta alcanzar la temperatura de maceración objetivo que este caso son 76°C. Cuando este valor se alcanza la gráfica cambia y empieza a graficarse la temperatura de la mezcla. Como vemos al principio tiene un valor inicial de 60°C y cuando el agua se introduce la temperatura pasa a ser 67°C.

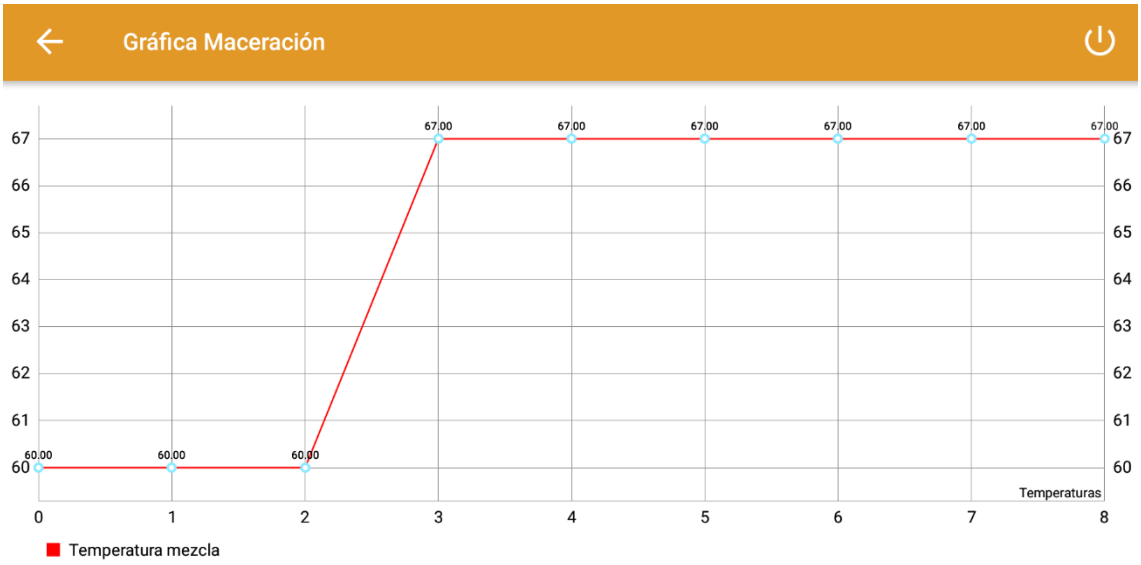

Figura 73: *Alerta Maceración "El proceso ha finalizado"*

Filtrado

Cuando pulsamos sobre el botón de Filtración en el menú de procesos se ejecuta y abre la *activity_filtrado*. Si en la alerta de maceración finalizada hemos cogido la opción aceptar este proceso se iniciará automáticamente, si no, se debe iciar de forma manual con el boton iniciar como es este caso.

En primer lugar se intruduce la temperatura inicial del agua manualmente simulando que estamos leyendo cual es su temperatura a través de un sensor.

Figura 74: *Pantalla Filtrado*

Una vez pulsamos el botón iniciar este se oculta y el proceso se pone en marcha. Iniciando el cronómetro que controla nuestro proceso. Y empieza la primera etapa, la primera filtracion del mosto que dura 45 minutos.

Figura 75: *Pantalla Filtrado "Iniciar"*

Cuando el cronómetro alcanza los 45 minutos comienza la segunda etapa del proceso, el lavado del bagazo y el agua necesaria para ello se calienta o se enfría antes de ser introducida. En este caso como la hemos iniciado en 30°C se calienta hasta alcanzar un valor entre 75 y 82°C. Y nos indica que la temperatura es correcta.

Figura 76: *Pantalla Filtrado "Calentando Agua"*

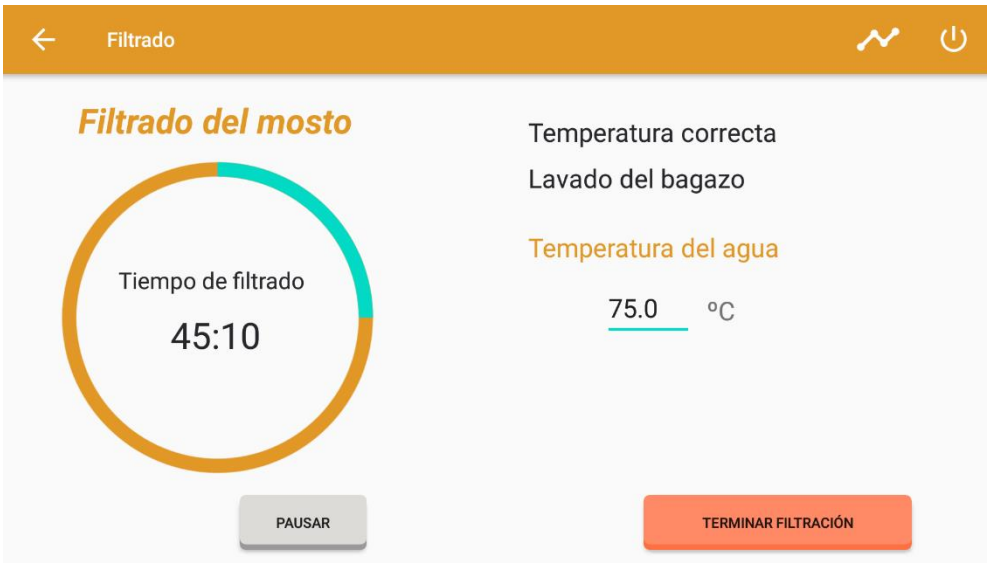

Figura 77: *Pantalla Filtrado "Temperatura correcta"*

Si pulsamos el botón de pausar el boton inicar vuelve a aparecer, el de pausar se oculta y el cronómetro se detiene.

Figura 78: *Pantalla Filtrado "Pausar"*

Una vez el proceso alcanza los 60 minutos o pulsamos el botón terminar filtración, el proceso se finaliza y nos muestra una alerta para indicarnos si queremos iniciar el proceso de cocción o enfriado.

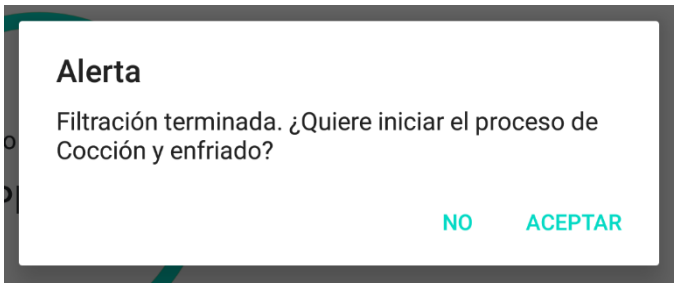

Figura 79: *Alerta Filtración terminada*

Por último, en la pantalla de gráficos observamos como la temperatura del agua ha ido aumentando una vez iniciado el lavado del bagazo hasta alcanzar la temperatura correcta para esta etapa.

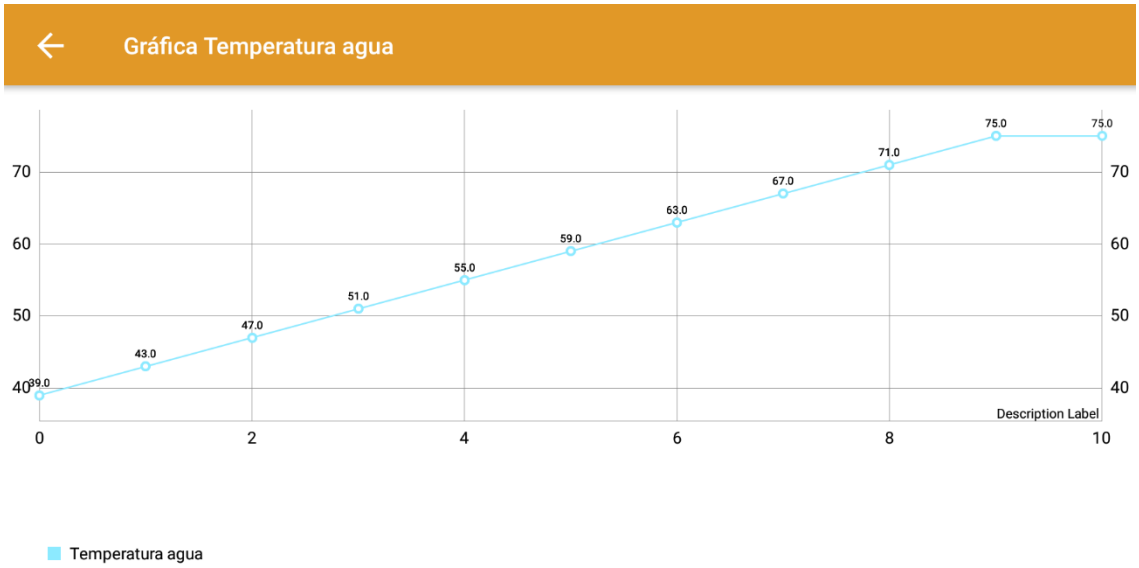

Figura 80: *Gráfica temperatura agua filtrado*

Cocción y enfriado

Cuando pulsamos sobre el botón de Cocción en el menú de procesos se ejecuta y abre la *activity_coccion*. Si en la alerta de filtrado finalizada hemos cogido la opción aceptar este proceso se iniciará automáticamente, si no, se debe iciar de forma manual con el boton iniciar como es este caso.

En primer lugar se intruduce la temperatura inicial del mosto manualmente simulando que estamos leyendo cual es su temperatura a través de un sensor.

Figura 81: *Pantalla Cocción y enfriado*

Cuando pulsamos el botón de iniciar la temperatura del mosto aumenta hasta 100°C para llevarlo a ebullición, momento en el cual comienza la cocción.

Figura 82: *Pantalla Cocción iniciada*

Si pulsamos el botón pausar, el cronómetro se detiene, se habilita el botón iniciar y se deshabilita el botón pausar.

Figura 83: *Pantalla Cocción y enfriado pausado*

Cuando la cocción finaliza, es decir, el cronómetro alcanza los 90 minutos, comienza el enfriado y la temperatura del mosto comienza a disminuir hasta alcanzar aproximadamente los 20°C. En este caso habíamos elegido la receta Ale, por lo que la temperatura a alcanzar se encuentra entre 16 y 20°C.

Figura 84: *Pantalla Enfriado iniciado*

El enfriado dura aproximadamente 75 minutos como observamos en la imagen anterior. Cuando el enfriado alcance este tiempo o pulsemos el botón de finalizar, el proceso se dará por finalizado. Y el sistema nos mostrará la siguiente alarma.

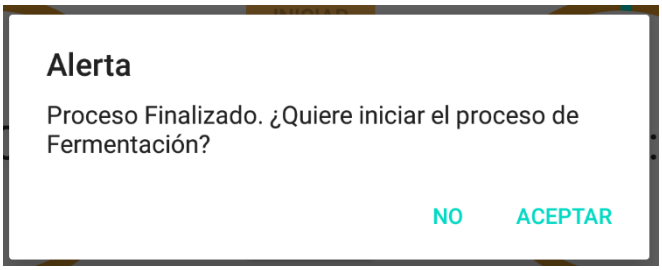

Figura 85: *Alerta Cocción y enfriado finalizado*

Por último, se pueden observar las gráficas cuando se lleva al mosto a ebullición para la etapa de cocción y cuando disminuimos la temperatura del mosto en el enfriado.

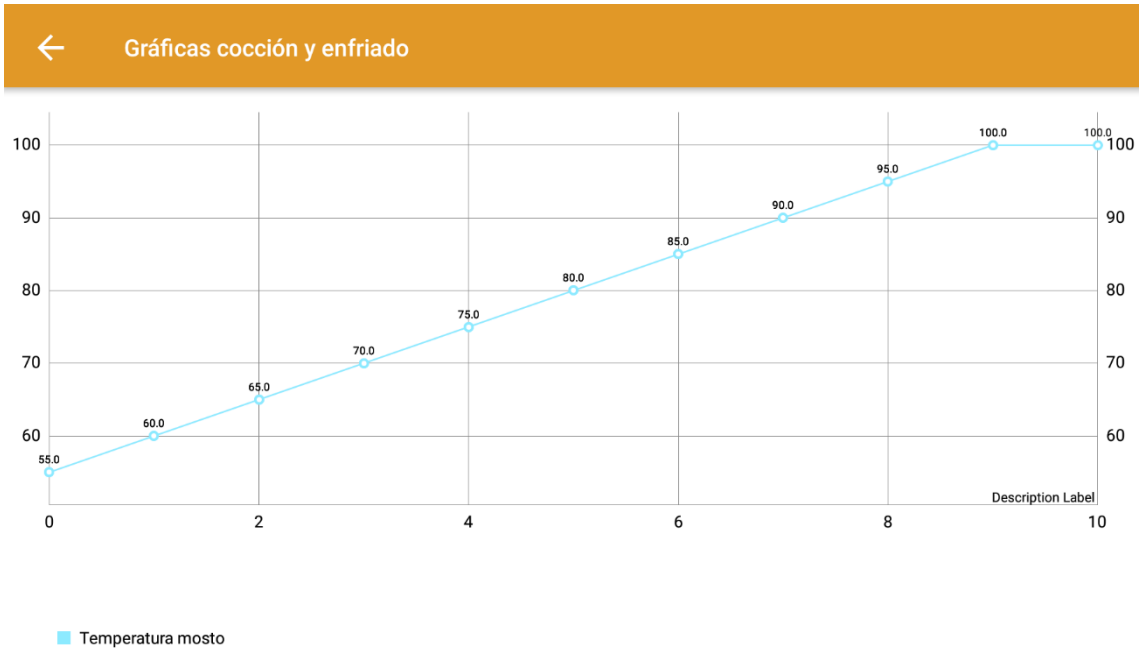

Figura 86: *Gráfica cocción*

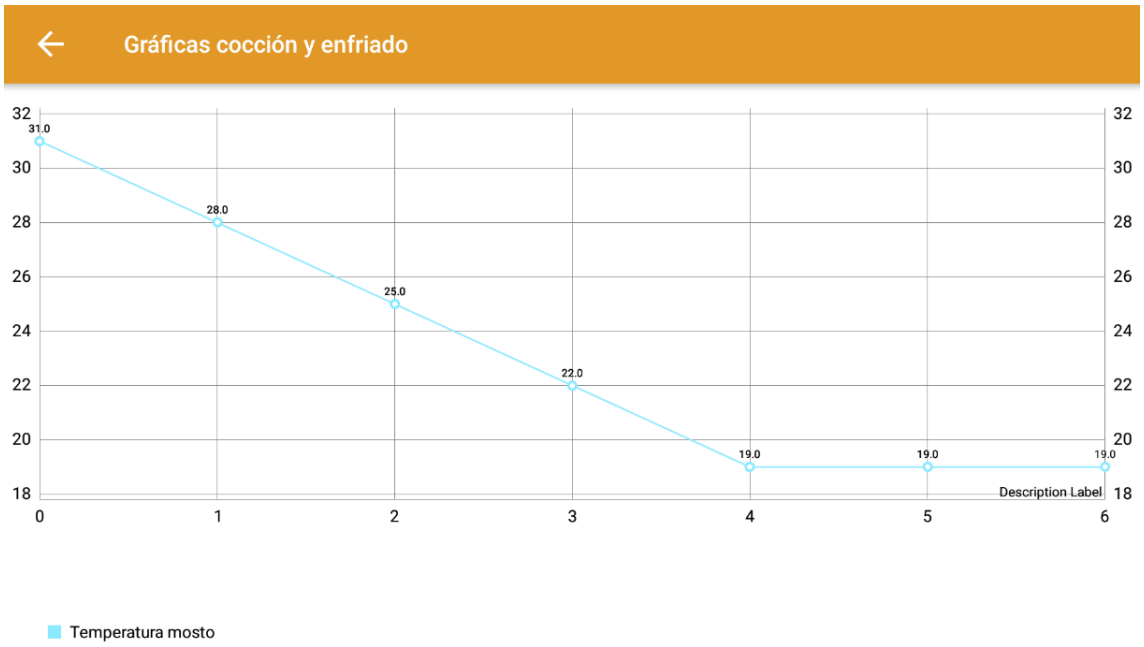

Figura 87: *Gráfica enfriado*

Fermentación

Cuando pulsamos sobre el botón de Fermentacion en el menú de procesos se ejecuta y abre la *activity_fermentacion*. Si en la alerta de cocción y enfriado finalizada hemos cogido la opción aceptar este proceso se iniciará automáticamente, si no, se debe iciar de forma manual con el boton iniciar como es este caso.

En primer lugar se intruduce la temperatura inicial del fermentador manualmente simulando que estamos leyendo cual es su temperatura a través de un sensor. Como se observa se introduce la levadura.

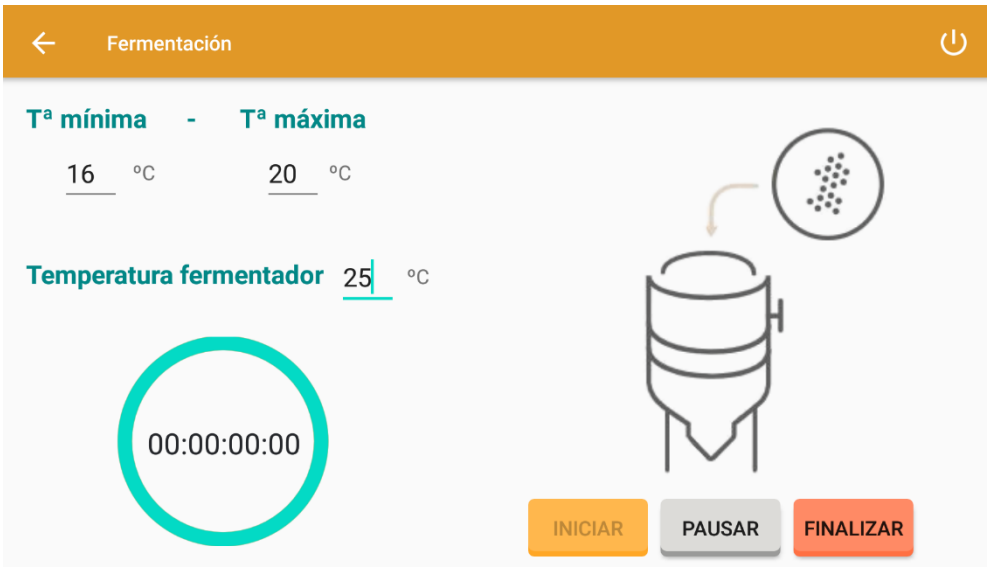

Figura 88: *Pantalla Fermentación "Introducir levadura"*

Pasados unos segundos de la introducción de la levadura, el fermentador aumenta o disminuye su temperatura para alcanzar la temperatura correcta que tiene que estar entre la máxima y la mínima que aparecen en la pantalla. Estas temperaturas se cargan de las variables globales ya que dependen de la receta elegida.

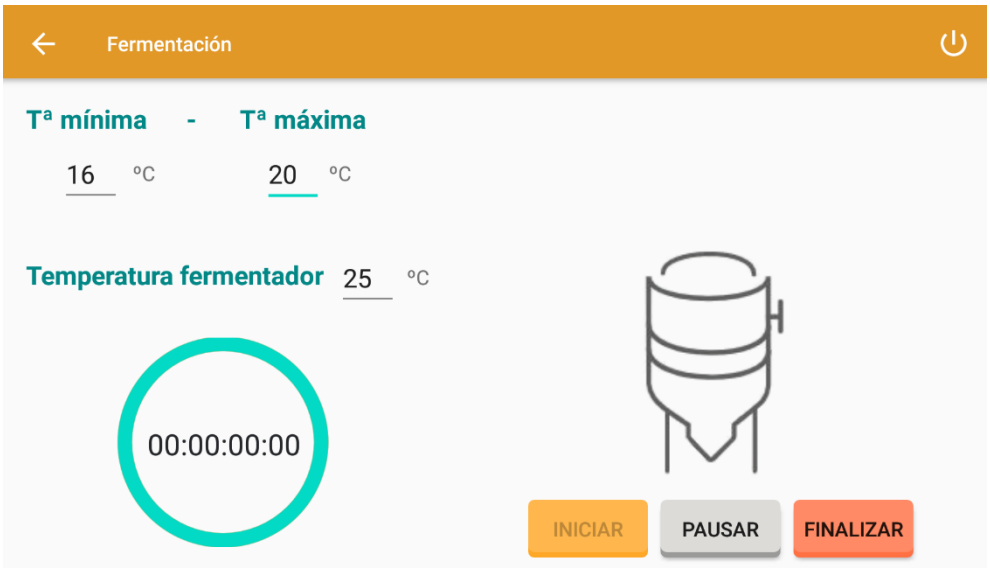

Figura 89: *Pantalla Fermentación*

Una vez el fermentador alcanza la temperatura correcta el proceso se inicia y se pone en marca el cronómetro. La temperatura del proceso también depende de la receta elegida.

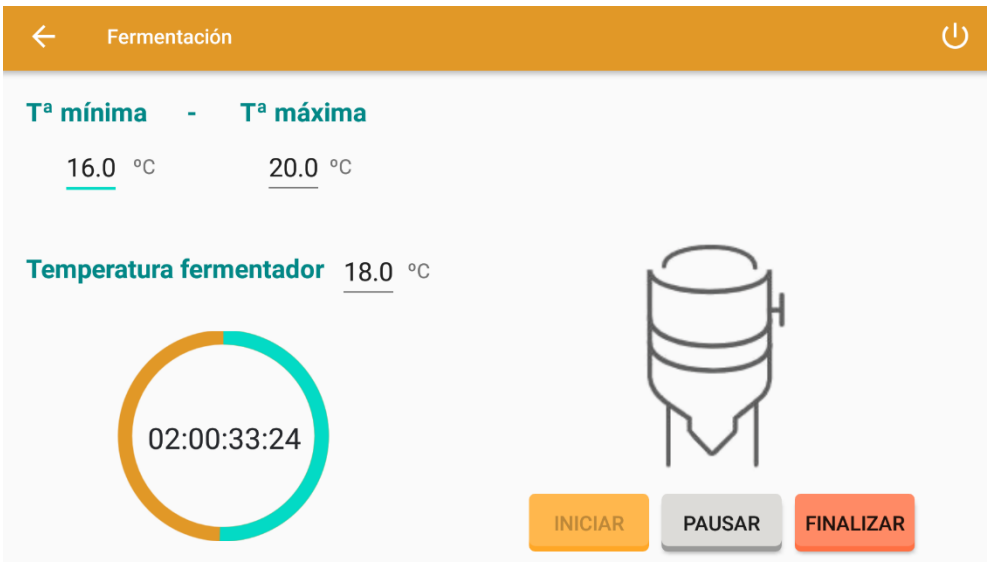

Figura 90: *Pantalla Fermentación "Enfriando fermentador"*

Como observamos la aplicación nos muestra el tiempo de fermentación en cada momento. Además, si pulsamos el botón de pausar el cronómetro se para, el botón de iniciar se habilita y el de pausar se habilita.

Figura 91: *Proceso Fermentación "Pausado"*

Una vez el cronómetro alcanza el tiempo objetivo o se pulsa el botón finalizar el proceso se da por finalizado y nos muestra la siguiente alerta. A continuación, podremos volver a empezar a fabricar otro lote de producción.

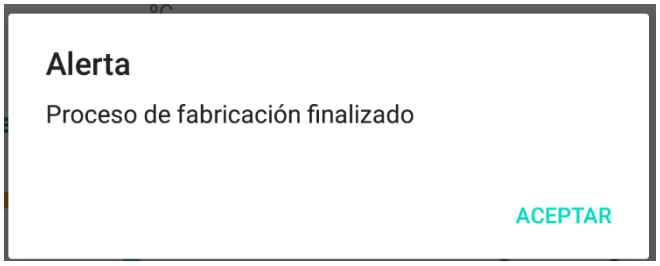

Figura 92: *Alerta finalización proceso*

5.6. Recetas

Cuando pulsamos sobre el botón de Recetas en el menú principal se ejecuta y abre la *activity_recetas*. Se trata de una ventana infomativa que nos muestra las recetas y sus datos correspondientes.

Figura 93: *Pantalla Recetas*

5.7. Historial

Cuando pulsamos sobre el botón de Historial en el menú principal se ejecuta y abre la *activity_list_lotes*. Se trata de una ventana infomativa que nos muestra los lotes que se producen, nos indica la hora y la fecha, la receta y los litros porducidos. Esta ventana carga la base de datos de lotes y los muestra en un *ListView*.

Figura 94: *Pantalla Historial*

6. Descripción detallada de la implementación

6.1. Base de datos SQLite

La forma más simplificada y típica para crear, actualizar y conectar una base de datos SQLlite es a través de una clase auxiliar *SQLiteOpenHelper* o una clase que derive de ella como es nuestro caso *ConexionSQLiteHelper.*

Esta clase contiene un constructor el cual no se debe modificar y dos métodos, onCreate() y onUpgrade(). Estos dos métodos sí que hay que personalizarlos con el código necesario para crear y actualizar la estructura de la base de datos respectivamente.

```
public class ConexionSQLiteHelper extends SQLiteOpenHelper {

    public ConexionSQLiteHelper(@Nullable Context context, @Nullable String name,
@Nullable SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Se ejecuta la sentencia SQL de creación de la tabla
        db.execSQL(Utilidades.CREAR_TABLA_USUARIO);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //Se elimina la versión anterior de la tabla
        db.execSQL("DROP TABLE IF EXISTS usuarios");
        //Se crea la nueva versión de la tabla
        onCreate(db);

    }
}
```

El método onCreate() será ejecutado cuando sea necesaria la creación de la base de datos, es decir, cuando aún no exista. En este método se realizan las tareas típicas de la creación de todas las tablas necesarias y la inserción de datos iniciales si fuera necesario. Para la creación de la tabla se utiliza el método execSQL() que se limita a ejecutar el código SQL que le pasemos como parámetro.
Por otra parte, el método onUpgrade() se ejecutará automáticamente cuando sea necesario actualizar la estructura de la base de datos o convertir los datos. Por ejemplo, si necesitamos añadir un campo a nuestra tabla, como podría ser el campo edad.

Una vez tenemos una referencia al objeto *ConexionSQLiteHelper,* llamaremos a los métodos getReadableDatabase() o getWritableDatabase(), dependiendo si necesitamos consultar los datos o también necesitamos modificarlos respectivamente.

Cuando llamamos a *Utilidades.CREAR_TABLA_USUARIO* estamos llamando a la siguiente clase, la que se encarga de crear la tabla con los campos que le indiquemos.

```
public class Utilidades {

    //Constantes campos tabla usuario
```

```
    public static final String TABLA_USUARIO ="usuario";
    public static final String CAMPO_NOMBRE ="nombre";
    public static final String CAMPO_DNI ="dni";
    public static final String CAMPO_CONTRA ="contra";
    public static final String CAMPO_TELEFONO ="telefono";
    public static final String CAMPO_TIPO ="tipo";

   public static final String CREAR_TABLA_USUARIO ="CREATE TABLE "+TABLA_USUARIO+"
("+CAMPO_NOMBRE+" TEXT, "+CAMPO_DNI+" TEXT, "+CAMPO_CONTRA+" TEXT, "+CAMPO_TELEFONO+"
TEXT, "+CAMPO_TIPO+" TEXT)";
}
```

Se ha utilizado la base de datos SQLite que sirve para almacenar datos estructurados en una base de datos privada. Android proporciona compatibilidad completa con bases de datos SQLite.

La mayoría de las aplicaciones móviles incluyen bases de datos SQLite, bien sea para la gestión total de los datos, o bien para al menos gestionar los datos almacenados localmente cuando las aplicaciones son parte de infraestructuras mayores que incluyen bases de datos centralizadas online o servicios Web para obtención de los datos.

Por este motivo en este proyecto utilizamos esta base de datos ya que se trata de un prototipo y en un futuro estos datos de usuarios y contraseñas se obtendrían de una base de datos online. Por lo tanto, las bases de datos creadas en este proyecto solo son accesibles mediante esta aplicación y es una base de datos local al teléfono.

Como se ha explicado antes en la pantalla de inicio de sesión se introduce el usuario y la contraseña. Cuando se pulsa el botón ingresar se comprueba si el usuario se encuentra en la base de datos y si la contraseña asociada es la correcta. El código implementado para ello es el siguiente:

```
//Creación de la conexión con la base de datos
conn = new ConexionSQLiteHelper(getApplicationContext(),"bd_usuarios",null,1);
//Evento al hacer click en el botón
BTN_ING.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        //Se recoge el usuario y contraseña introducidos
        String V_USR = TXT_USR.getText().toString();
        String V_PASS = TXT_PASS.getText().toString();
        //Modo lectura de la base de datos
        SQLiteDatabase db = conn.getReadableDatabase();

        String[] parametros = {TXT_USR.getText().toString()};
        //Campos que queremos que devuelva la consulta a la base de datos
        String[] campos={Utilidades.CAMPO_NOMBRE,Utilidades.CAMPO_CONTRA};

        try{
            //Se crea un cursor que se mueve hasta consultar el nombre del usuario y
la contraseña
            Cursor cursor =
db.query(Utilidades.TABLA_USUARIO,campos,Utilidades.CAMPO_NOMBRE+"=?",parametros,null
,null,null);
            cursor.moveToFirst();
            //Si el usuario y la contraseña coinciden con los datos de la base de
datos
            if (V_USR.equals(cursor.getString(0)) &&
V_PASS.equals(cursor.getString(1)))
            {   //Se inicia sesión
                Intent intent=new
```

```
Intent(getApplicationContext(),MenuPrincipal.class);
                startActivity(intent);

            }
            //Si no coinciden
            else
            {
                Toast.makeText(getApplicationContext(),"Usuario o Contraseña
Errónea", Toast.LENGTH_SHORT).show();
            }
            //Se cierra el cursor
            cursor.close();

        }catch (Exception e){
            Toast.makeText(getApplicationContext(),"El Usuario no existe" ,
Toast.LENGTH_SHORT).show();}

    }
});
```

El cursor que creamos tiene todos los datos que son los que se encuentran en la *TABLA_USUARIOS* devolviendo el nombre y la contraseña y buscando a través del nombre.

Se utiliza moveToFirst() para posicionarnos al principio del cursor y para obtener los valores usaremos el método getString(), recibiendo como único parámetro un entero que identifica la posición de la columna que queremos obtener. En nuestro caso la columna 0 correspondería con el nombre y la columna 1 con la contraseña ya que son los parámetros que se devuelven en ese orden.

Una vez finalicemos de manejar el cursor, es conveniente llamar al método close() para cerrarlo y así liberar todos sus recursos para no poder hacer uso de él más.

A continuación, se mostrará el código implementado más importante para registrar, eliminar, listar y consultar un usuario.

```
public void registrarUsuario (){
    //Conexión con la base de datos
    ConexionSQLiteHelper conn = new ConexionSQLiteHelper(this, "bd_usuarios", null,
1);
    //Modo escritura en la base de datos
    SQLiteDatabase db = conn.getWritableDatabase();
    //Datos que se desean insertar
    ContentValues values = new ContentValues();
    values.put(Utilidades.CAMPO_NOMBRE,edtnombre.getText().toString());
    values.put(Utilidades.CAMPO_DNI,edtdni.getText().toString());
    values.put(Utilidades.CAMPO_CONTRA,edtcontrasena.getText().toString());
    values.put(Utilidades.CAMPO_TELEFONO,edttelefono.getText().toString());
    values.put(Utilidades.CAMPO_TIPO,edttipo.getText().toString());
    //Registro del usuario
    Long idResultante =
db.insert(Utilidades.TABLA_USUARIO,Utilidades.CAMPO_NOMBRE,values);
    Toast.makeText(getApplicationContext(),"Nombre Registro" +idResultante,
Toast.LENGTH_SHORT).show();
    db.close();
}
```

Para el registro de usuarios hemos utilizado el método insert() que recibe tres parámetros. El primer argumento es el nombre de la tabla, el segundo indica que hacer en caso de que ContentValues esté vacío. Como se especifica una columna, el framework inserta una fila y establece el valor de esa columna como nulo. Y el tercer argumento son los valores que

queremos insertar, es decir, la cláusula WHERE, asociando a cada columna el valor del *EditText* correspondiente.

```
public void eliminarUsuario(){
    //Modo escritura en la base de datos
    SQLiteDatabase db = conn.getWritableDatabase();
    String[] parametros={edtnombrecons.getText().toString()};

    db.delete(Utilidades.TABLA_USUARIO,Utilidades.CAMPO_NOMBRE+"=?",parametros);
    Toast.makeText(getApplicationContext(),"Usuario eliminado",
Toast.LENGTH_SHORT).show();
    edtnombrecons.setText("");

    limpiar();
    db.close();
}
```

Para la eliminación de un usuario hemos utilizado el método delete() que recibe tres parámetros, el nombre de la tabla, la cláusula WHERE, es decir, que columna buscar en la base de datos que en nuestro caso es por el nombre, y el tercero los argumentos de la cláusula WHERE.

Con esta función eliminamos todos los datos correspondientes al usuario que insertemos en el *EditText.*

```
public void actualizarUsuario(){
    SQLiteDatabase db = conn.getWritableDatabase();
    String[] parametros={edtnombrecons.getText().toString()};

    ContentValues values = new ContentValues();
    values.put(Utilidades.CAMPO_NOMBRE,edtnombrecons.getText().toString());
    values.put(Utilidades.CAMPO_DNI,edtdnicons.getText().toString());
    values.put(Utilidades.CAMPO_CONTRA,edtcontrasenacons.getText().toString());
    values.put(Utilidades.CAMPO_TELEFONO,edttelefoncons.getText().toString());
    values.put(Utilidades.CAMPO_TIPO,edttipocons.getText().toString());

db.update(Utilidades.TABLA_USUARIO,values,Utilidades.CAMPO_NOMBRE+"=?",parametros);
    Toast.makeText(getApplicationContext(),"Usuario actualizado",
Toast.LENGTH_SHORT).show();
    db.close();
}
```

Para la actualización de datos en la base de dato hemos utilizado el método update(), funciona de la misma forma que el método insert() explicado con anterioridad, solo que este método recibe cuatro parámetros. En lugar de pasar directamente el argumento de la cláusula WHERE, le indicamos el valor '?' y usamos el cuarto parámetro para indicarle estos argumentos.

```
public void buscarUsuario(){
    SQLiteDatabase db = conn.getReadableDatabase();
    String[] parametros={edtnombrecons.getText().toString()};

    String[] campos={Utilidades.CAMPO_DNI, Utilidades.CAMPO_CONTRA,
Utilidades.CAMPO_TELEFONO, Utilidades.CAMPO_TIPO};

    try{
        Cursor cursor =
db.query(Utilidades.TABLA_USUARIO,campos,Utilidades.CAMPO_NOMBRE+"=?",parametros,null
,null,null);
        cursor.moveToFirst();
        edtdnicons.setText(cursor.getString(0));
        edtcontrasenacons.setText(cursor.getString(1));
```

```
        edttelefoncons.setText(cursor.getString(2));
        edttipocons.setText(cursor.getString(3));

        cursor.close();

    }catch (Exception e){
        Toast.makeText(getApplicationContext(),"El Usuario no existe" ,
Toast.LENGTH_SHORT).show();
        limpiar();
    }

}
```

El funcionamiento es simular que cuando se inicia sesión ya que en ambos casos debemos consultar la base de datos pero en este caso lo que hacemos es crear un cursor que nos devuelva todos los datos correspondientes al usuario consultado y nos los muestre en los *EditTexts* correspondientes mediante el método setText().

A continuación, veremos el código necesario para listar los usuarios:

```
private void consultarListaUsuarios(){
    SQLiteDatabase db=conn.getReadableDatabase();
    Usuario usuario=null;
    listaUsuarios=new ArrayList<Usuario>();
    Cursor cursor=db.rawQuery("SELECT * FROM "+ Utilidades.TABLA_USUARIO,null);
    while(cursor.moveToNext()){
        usuario = new Usuario();
        usuario.setNombre(cursor.getString(0));
        usuario.setDNI(cursor.getString(1));
        usuario.setContra(cursor.getString(2));
        usuario.setTelefono(cursor.getString(3));
        usuario.setTipo(cursor.getString(4));

        listaUsuarios.add(usuario);
    }
    obtenerLista();
}

private void obtenerLista() {
    listaInformación = new ArrayList<String>();
    for(int i=0; i<listaUsuarios.size();i++){
        listaInformación.add(listaUsuarios.get(i).getNombre()+" -
"+listaUsuarios.get(i).getDNI());
    }
}
```

Se trata de una consulta similar a la que realizamos cuando consultamos los datos de un usuario con la diferencia de que en este caso los datos que nos devuelve los guardamos en una variable llamada *Usuario*. Esta variable tiene varios parámetros y se crea en una clase independiente que mostraremos a continuación.

```
public Usuario(String nombre, String DNI, String contra, String telefono, String
tipo) {
    this.nombre = nombre;
    this.DNI = DNI;
    this.contra = contra;
    this.telefono = telefono;
    this.tipo = tipo;
}
```

Además también se crean todas las funciones get() y set() para cada parámetro. Con ellas podremos leer y guardar los valores que deseemos desde cualquier otra clase.

6.2. Card View

Para el menú principal se han utilizado *CardView* para acceder a cada una de las funciones de la aplicación mediante el siguiente código:

```
cardIngredientes=findViewById(R.id.cardview_ingredientes);
cardIngredientes.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intentingredientes=new
Intent(getApplicationContext(),MateriasPrimas.class);
        startActivity(intentingredientes);
    }
});
```

Con el evento *intent* ejecutamos la *activity* que corresponde.

6.3. Botón flecha hacia atrás

En cuanto al botón de volver atrás se habilita de la siguiente forma, ejecutando la función onSupportNavigateUp():

```
//Botón volver atrás
getSupportActionBar().setDisplayHomeAsUpEnabled(true);


public boolean onSupportNavigateUp() {
    onBackPressed();
    return false;
}
```

6.4. Función onResume. Ciclo de vida de las actividades

Cuando una actividad entra en estado Resumed, esta actividad pasa a primer plano, y invoca a la función llamada onResume(). La app permanece en este estado hasta que ocurre algún evento, como podría ser recibir una llamada telefónica, que se apague la pantalla o que el usuario navegue por otra actividad.

Cuando se vuelve a la *activity* se ejecuta este método que en nuestro caso llama a la función cargardatos().

```
//Función que se ejecuta nada más abrir la activity
protected void onResume() {

    super.onResume();
    cargardatos();
}

public void cargardatos(){
    //Cargamos los datos de las variables globales
    cantidadagua.setText(String.valueOf(global.getCantidadagua()));
    cantidadcebada.setText(String.valueOf(global.getCantidadcebada()));
    cantidadlupulo.setText(String.valueOf(global.getCantidadlupulo()));
    cantidadlevaduraale.setText(String.valueOf(global.getCantidadlevaduraale()));
    cantidadlevaduralager.setText(String.valueOf(global.getCantidadlevaduralager()));
```

```
}
```

La función cargardatos() lo que hace es actualizar los *EditText* con los datos que se han guardado en las variables globales.

Ahora bien, ¿qué son las variables globales?, ¿cómo las almacenamos?, ¿cómo accedemos a ellas?

6.5. Preferencias compartidas SharedPreferences

Para explicar el apartado anterior empezaremos por explicar la clase llamada *GlobalInfo* en la cual tenemos todas las variables y métodos necesarios para acceder a ellas y modificarlas.
Las variables globales son variables accesibles en todos los ámbitos de un programa informático.

Para explicarlo correctamente se va a explicar un ejemplo detalladamente. El código implementado para cada variable es el siguiente:

```
private double cantidadagua;

public double getCantidadagua() {
    SharedPreferences preferences = getSharedPreferences("credenciales",
Context.MODE_PRIVATE);
    String scantidadagua=preferences.getString("cantidadagua",
String.valueOf(1000.0));
    cantidadagua=Double.parseDouble(scantidadagua);
    return cantidadagua;
}

public void setCantidadagua(double cantidadagua) {
    SharedPreferences preferences = getSharedPreferences("credenciales",
Context.MODE_PRIVATE);
    SharedPreferences.Editor editor=preferences.edit();
    editor.putString("cantidadagua",String.valueOf(cantidadagua));
    editor.apply();
    this.cantidadagua = cantidadagua;
}
```

Para el almacenamiento de las variables globales utilizamos las preferencias compartidas. *SharedPreferences* se utiliza para almacenar datos cuando no se necesita una estructura o no se necesita almacenar muchos datos. Las preferencias compartidas se almacenan en ficheros XML.

SharedPreferences te permite leer y escribir pares persistentes de tipos de datos de primitivas: *booleanos, floats, ints, longs y strings*.

Para obtener acceso a una colección utilizaremos el método getSharedPreferences() al que pasaremos el identificador de la colección y un modo de acceso. En nuestro caso la colección es *credenciales* y el modo de acceso *MODE_PRIVATE,* mediante el cual solo nuestra aplicación tiene acceso a estas preferencias.

Una vez se ha obtenido una referencia a nuestra colección ya se podrá obtener, insertar o modificar las preferencias utilizando los métodos *get* y *put* correspondientes al tipo de dato. En nuestro caso se trata de un *string*.

Como se observa, al método getString() le pasamos el nombre de la preferencia que queremos recuperar y un segundo parámetro con un valor por defecto para cuando la preferencia solicitada no existe. A continuación, este *string* lo pasamos a tipo *double* mediante el método

Double.parseDouble(), el cual recibe un *string*. Este valor *double* será el que nos devuelva la función getCantidadagua().

Para actualizar o insertar nuevas preferencias no lo haremos directamente sobre el objeto *SharedPreferences*, si no sobre un objeto de edición *SharedPreferences.Editor.* A este método se accede mediante el método edit(). Una vez obtenida la referencia, se utilizan los métodos *put* correspondientes al tipo de datos para actualizar o insertar el valor de la preferencia.

Finalmente, se llamará al método apply() para confirmar los cambios.

Por último, ¿Cómo accedemos a ellas desde cualquier ámbito? Para ello se debe implementar el siguiente código:

```
GlobalInfo global = (GlobalInfo) getApplicationContext();
```

Colocando esta línea de código en cualquier clase de la aplicación tendremos acceso a todas las funciones *get* y *set* que hemos implementado en la clase *GlobalInfo*.

Para llamarlas deberemos escribir lo siguiente:

```
global.setCantidadagua(); //Modificar el dato
global.getCantidadagua(); //Leer el dato
```

6.6. Funciones addTextChangedListener() y isValidDouble()

Para que los litros de producción se guarden en la variable global correspondiente hacemos lo siguiente:

```
litros.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }
    @Override
    public void afterTextChanged(Editable s) {
        if (litros.getText().toString().isEmpty() == true) {
            handler.post(new Runnable() {
                public void run() {
                    litros.setError("Inserta una cantidad");
                }
            });
        }
        if (litros.getText().toString().isEmpty() == false) {

            if (isValidDouble(litros.getText().toString()) == false) {
                litros.setError("Tipo de valor erróneo");
            }
            if (isValidDouble(litros.getText().toString()) == true) {
                //Guardamos los litros de producción en la variable global
correspondiente
global.setProduccion(Double.parseDouble(litros.getText().toString()));
                handler.post(new Runnable() {
                    public void run() {
                       agua=Double.parseDouble(litros.getText().toString())*4.0;
                        String sagua = String.valueOf(agua);
                        txtagua.setText(sagua);
```

```
cebada=Double.parseDouble(litros.getText().toString())*175.0/1000.0;
                        String scebada = String.valueOf(cebada);
                        txtcebada.setText(scebada);


lupulo=Double.parseDouble(litros.getText().toString())*2.0/1000.0;
                        String slupulo = String.valueOf(lupulo);
                        txtlupulo.setText(slupulo);


levadura=Double.parseDouble(litros.getText().toString())*1.0/1000.0;
                        String slevadura = String.valueOf(levadura);
                        txtlevadura.setText(slevadura);
            }
        }
    });
        }
    }
});
```

Utilizamos el método *addTextChangedListener(**new** TextWatcher()),* este método incluye tres funciones, lo que quieres que ocurra antes de cambiar el texto del *EditText*, lo que ocurre mientras lo cambias y lo que ocurre después de cambiarlo. En nuestro caso vamos a utilizar este último, lo que queremos es que despues de cambiarlos guarde los litros de producción en la variable global correspondiente mediante la función setProduccion(). Pero antes de que esto ocurra comprobamos que el *EditText* no este vacío mediante la función isEmpty(). Si es *true*, es decir, está vacío, nos devuelve un error que nos indica que debemos introducir un valor. Si no está vacío, comprobamos que el valor introducido sea un *Double* con la función isValidDouble(). De la misma forma si es correcto nos guardara el valor correspondiente y si no es correcto nos mostrará un error indicándonos que el tipo de valor es erróneo.

```
private static boolean isValidDouble(String s) {
    boolean isValid = true;
    try {
        Double.parseDouble(s);
    } catch (NumberFormatException nfe) {
        isValid = false;
    }

    return isValid;
}
```

Esta función lo que hace es pasar el *string* que introducimos en el *EditText* a *double*. Si al convertirlo salta una excepción de formato de número, devuelve *false*, si no salta una excepción el tipo de dato es un *double* y por lo tanto devuelve *true*.

Además de guardar el valor, se calculan las cantidades de materias primas necesarias para la elaboración y las mostraremos por pantalla.

Por último, cuando pulsamos el botón comenzar producción se comprueba que la producción no sea menor a la mínima, es decir 200 litros y que las cantidades necesarias para la producción estén disponibles, si no hay suficientes materias primas, nos muestra un aviso a través de *AlertDialog* indicándote si quieres disminuir la producción.

6.7. AlertDialog

Un *AlertDialog* es una pequeña ventana que indica al usuario una información o indica que tiene que tomar una decisión. Generalmente, se usa para eventos que requieren que el usuario realice una acción para poder continuar.

```
//Comprobamos que la producción sea mayor o igual a 200 litros, si es menor salta una
alerta

if(global.getProduccion()<200.0){
    AlertDialog.Builder myBuild = new AlertDialog.Builder(Produccion.this);
    myBuild.setMessage("Cantidad de producción inferior a la mínima. ¿Desea producir
200 litros?");
    myBuild.setTitle("Alerta");
    myBuild.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            global.setProduccion(200.0);
            litros.setText(String.valueOf(global.getProduccion()));
        }
    });

    myBuild.setNegativeButton("No", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });

    AlertDialog dialog = myBuild.create();
    dialog.show();
}
//Si la producción no es menor de 200 litros
else {
    //Comprobamos que el agua disponible sea suficiente
    if (agua > global.getCantidadagua()) {
        AlertDialog.Builder myBuild = new AlertDialog.Builder(Produccion.this);
        myBuild.setMessage("No hay agua suficiente. Disminuya la producción");
        myBuild.setTitle("Alerta");
        myBuild.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                global.setProduccion(200.0);
                litros.setText(String.valueOf(global.getProduccion()));
            }
        });

        AlertDialog dialog = myBuild.create();
        dialog.show();
    }
```

El dialogo se crea mediante el método *AlertDialog.Builder*, se le asigna un título con el método setTitle() y un mensaje con el método setMessage(). También se crean los botones setPositiveButton() y setNegativeButton() indicándoles que tiene que ocurrir en cada caso.

Con el método show() el dialogo aparece en la pantalla.

En nuestro caso si aceptamos lo que ocurre es que la producción se disminuye a 200 litros que es la producción mínima y en caso de que no aceptemos el dialogo se cierra y la producción por lo tanto no puede comenzar.

Esta comprobación se hace con todas las variables.

6.8. Graficar Datos (LineChart)

A continuación, se va a explicar con detalles el código desarrollado para graficar la temperatura del agua y la humedad del grano en el proceso de malteado.

En primer lugar, se crean las variables necesarias:

```
private LineChart lineChart;
ArrayList<ILineDataSet> dataSets;

int graf=0;

Timer timergraf;
TimerTask timerTaskgraf;
GlobalInfo global;
```

A continuación, en la función onCreate de la activity Malteado Gráficas se escribe lo siguiente:

```
global = (GlobalInfo) getApplicationContext();

// Enlazamos al XML
lineChart=(LineChart) findViewById(R.id.lineChart);
//Habilitamos la descripción del gráfico
lineChart.getDescription().setEnabled(true);
lineChart.getDescription().setText("Temperatura y Humedad");

//Background del gráfico blanco y posicionamos el eje x en la parte inferior

lineChart.setBackgroundColor(Color.WHITE);
XAxis xAxis = lineChart.getXAxis();
xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);

lineChart.setExtraBottomOffset(50);

//Creamos un timer para que nos grafique las variables cada cierto tiempo (2
segundos)
timergraf = new Timer();
timerTaskgraf =new TimerTask() {
    public void run() {

        if(graf==11){

            graf=0;
            lineChart.clear();
            dataVals.clear();
            dataVals2.clear();
            dataSets.clear();
            lineChart.invalidate();
            lineChart.clear();
        }


        if (graf!=11) {
            // Creamos un set de datos
            dataValores1();
            dataValores2();
            // Unimos los datos al data set
            LineDataSet lineDataSet1 = new LineDataSet(dataValores1(), "Temperatura
agua");
            LineDataSet lineDataSet2 = new LineDataSet(dataValores2(), "Humedad
grano");
            lineDataSet2.setColor(Color.RED);
            dataSets = new ArrayList<>();
            dataSets.add(lineDataSet1);
```

```
            dataSets.add(lineDataSet2);
            // Asociamos al gráfico
            LineData data = new LineData(dataSets);
            lineChart.setData(data);
            lineChart.invalidate();
            graf = graf + 1;
        }
    }

};
//Ponemos en marcha el timer, cada 2 segundos
timergraf.schedule(timerTaskgraf,0, 2000);

//Habilitamos el botón de volver atrás
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Con el timer lo que se consigue es graficar en tiempo real los valores de las variables que nos interesan.

La comprobación del valor de la variable graf tiene como objetivo que el gráfico se pueda visualizar bien, es por ello por lo que cuando ya hay 10 valores graficados, limpiamos el gráfico y los datos, para que comience de nuevo un gráfico en blanco.

Las funciones dataValores1() y dataValores2() lo que hacen es añadir un valor nuevo a un ArrayList, como se muestra a continuación:

```
// Creamos un set de datos
ArrayList<Entry> dataVals=new ArrayList<>();
ArrayList<Entry> dataVals2=new ArrayList<>();

private ArrayList<Entry> dataValores1(){
    dataVals.add(new Entry(graf,(float) global.getTempagua_malteado()));
    return dataVals;
}
private ArrayList<Entry> dataValores2(){

    dataVals2.add(new Entry(graf,(float) global.getHumedadgrano_maltedado()));
    return dataVals2;
}
```

6.9. Timer

Un Timer es una herramienta útil para ejecutar una tarea cada cierto tiempo. En nuestro proyecto principalmente nos servirá para simular de manera periódica las lecturas de los sensores de nuestras variables más importantes. Así como para crear animaciones como ocurre en el caso que Graficamos datos, o que mostramos imágenes que nos indican los estados de los procesos.

Para explicar cómo funciona vamos a explicarlo con el código implementado para simular la fase 2 del proceso de Secado y Tostado.

```
timer1 = new Timer();
timerTask1 = new TimerTask() {
    @Override
    public void run() {

    // Tarea que queremos ejecutar
    }
```

```
};
timer1.schedule(timerTask1,0, 2000);
```

Como se observa en el código en primer lugar cargamos un *Timer* con la línea: `timer1 = new Timer()`, después se crear un objeto *TimerTask* y dentro de su método *run*() se debe implementar el código de la tarea que vamos a ejecutar.

Y, por último, con el método Schedule del *Timer* llamaremos a la tarea que queremos ejecutar, indicándole el tiempo que debe esperar para empezar que en nuestro caso es 0 y el tiempo tras el cual repetirá la tarea de forma indefinida.

6.10. Thread y Runnable

Thread (hilo, tarea) es la clase base de Java para definir hilos de ejecución concurrentes dentro de un mismo programa. La concurrencia está asociada a los objetos, son los objetos que actúan concurrentemente con otros.

Para poder usar hijos que pueda actuar concurrentemente la clase de un objeto debe extender de la clase Theard. En nuestro caso esto no es posible ya que nuestra clase hereda de AppCompactActivity y java solo admite heredar de una clase.

```
public class Germinacion extends AppCompatActivity{}
```

Pero java permite implementar múltiples interfaces. Proporciona un interfaz para clases concurrentes.
Para que una clase se haga concurrente se debe de implementar la interfaz Runable. Esta interfaz debe implementar el método run() al igual que con Theard.

```
cronos = new Thread(new Runnable() {
    @Override
    public void run() {
    // Proceso que se desea ejecutar
    }
});
cronos.start();
```

El inicio de la ejecución de una tarea se realiza mediante el método start() que hereda de Thread. Este método invoca a run() y devuelve inmediatamente el control a la tarea que lo ha llamado.

6.11. Calendario

El método getInstance () en la clase Calendar se usa para obtener un calendario usando la zona horaria actual y la configuración regional del sistema.

Este método lo utilizamos cuando creamos un nuevo lote para poder saber cuándo hemos creado el lote.

```
Date date = Calendar.getInstance().getTime();
DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss");
String strDate = dateFormat.format(date);
```

El método format () de la clase *DateFormat* se utiliza para convertir Date en *String*. *DateFormat* es una clase abstracta. La clase secundaria de *DateFormat* es *SimpleDateFormat*. Es la implementación de la clase *DateFormat*.

6.12. Proceso de fabricación

Por último, con respecto a código vamos a explicar las partes de código más importantes de cada uno de los procesos de fabricación.

Producción

El código más importante de este apartado es el siguiente:

```
if(radioButtonale.isChecked()==true){

    global.setTempaireminobjetivo(78.0);
    global.setTempairemaxobjetivo(82.0);
    global.setMaceracionmin(60.0);
    global.setMaceracionmax(65.0);
    global.setEnfriadomin(16.0);
    global.setEnfriadomax(20.0);
    global.setFermentacionmin(15.0);
    global.setFermentacionmax(26.0);
    global.setTiempofermentacion(4.0);
}

if(radioButtonlager.isChecked()==true){

     global.setTempaireminobjetivo(100.0);
    global.setTempairemaxobjetivo(105.0);
    global.setMaceracionmin(70.0);
    global.setMaceracionmax(75.0);
    global.setEnfriadomin(10.0);
    global.setEnfriadomax(15.0);
    global.setFermentacionmin(7.0);
    global.setFermentacionmax(14.0);
    global.setTiempofermentacion(7.0);
}
```

Con este código establecemos las variables objetivo del proceso dependiendo de la receta que se ha elegido.

Malteado

Para este proceso también hacemos uso de un *Timer*. A continuación, mostramos el código implementado con comentarios.

```
timer = new Timer();
timerTask = new TimerTask() {
    @Override
    public void run() {
        // Si la temperatura es menor que la mínima objetivo
        if (global.getTempagua_malteado() <
global.getTempagua_malteado_objetivo_min()) {
            handler.post(new Runnable() {
```

```
                public void run() {
                    labeltemp.setText("INICIADO EL PROCESO DE MALTEADO");
                    labelhumedad.setText("Calentando agua");
                     // Simulamos el aumento de la temperatura
                    global.setTempagua_malteado(global.getTempagua_malteado() + 1.0);
                }
            });
        }
         // Si la temperatura es mayor que la máxima objetivo
        if (global.getTempagua_malteado() >
global.getTempagua_malteado_objetivo_max()) {
            handler.post(new Runnable() {
                public void run() {
                    labeltemp.setText("INICIADO EL PROCESO DE MALTEADO");
                    labelhumedad.setText("Temperatura del agua demasiado alta");
                     // Simulamos la disminución de la temperatura
                    global.setTempagua_malteado(global.getTempagua_malteado() - 1.0);
                }
            });
        }
        // Si la temperatura es la correcta
        if (global.getTempagua_malteado() >=
global.getTempagua_malteado_objetivo_min() && global.getTempagua_malteado() <=
global.getTempagua_malteado_objetivo_max()) {
            // Si el proceso no había empezado
            if(global.getx()==0) {
                handler.post(new Runnable() {
                public void run() {
                     // Comienza el proceso de remojado
                    labeltemp.setText("INICIADO EL PROCESO DE MALTEADO");
                    labelhumedad.setText("Inicio del remojado");
                    noinit.setVisibility(View.INVISIBLE);
                    introcebada.setVisibility(View.INVISIBLE);
                    introagua.setVisibility(View.VISIBLE);
                    mezclando.setVisibility(View.INVISIBLE);
                }
            });
            // Dormimos la tarea durante 5 segundos
            try {
                //Ponemos a "Dormir" el programa durante los ms que queremos
                Thread.sleep(5*1000);
                //labeltemp.setText("Programa durmiendo");
            } catch (Exception e) {
                System.out.println(e);
            }

            handler.post(new Runnable() {
                public void run() {
                    labeltemp.setText("INICIADO EL PROCESO DE MALTEADO");
                    labelhumedad.setText("Mezclando");
                    noinit.setVisibility(View.INVISIBLE);
                    introcebada.setVisibility(View.INVISIBLE);
                    introagua.setVisibility(View.INVISIBLE);
                    mezclando.setVisibility(View.VISIBLE);
                }
            });
            // Guardamos 1 en la variable x para saber que el proceso ha comenzado ya
            global.setx(1);
            }
            // Si la humedad del grano es menor que la objetivo
            if (global.getHumedadgrano_maltedado() <
global.getHumedadgrano_malteado_objetivo()) {
                handler.post(new Runnable() {
                    public void run() {
                        labeltemp.setText("Humedad inferior a la requerida");
                        labelhumedad.setText("Mezclando");
```

```
                        introcebada.setVisibility(View.INVISIBLE);
                        noinit.setVisibility(View.INVISIBLE);
                        introagua.setVisibility(View.INVISIBLE);
                        mezclando.setVisibility(View.VISIBLE);

// Simulamos el aumento de la humedad
global.setHumedadgrano_maltedado(global.getHumedadgrano_maltedado() + 1.0);
                    }
                });
            }
            // Si la humedad del grano es igual o mayor que la objetivo
            if(global.getHumedadgrano_maltedado() >=
global.getHumedadgrano_malteado_objetivo()){
                handler.post(new Runnable() {
                    public void run() {
                        labeltemp.setText("Humedad correcta");
                        labelhumedad.setText("FINALIZA EL PROCESO DE REMOJADO. PASA A
LA FASE DE GERMINACIÓN");

                        introcebada.setVisibility(View.INVISIBLE);
                        noinit.setVisibility(View.VISIBLE);
                        introagua.setVisibility(View.INVISIBLE);
                        mezclando.setVisibility(View.INVISIBLE);

                        btniniciarmalteado.setEnabled(true);
                        btnfinalizarmalteado.setEnabled(false);
                    }
                });
                // El proceso acaba, iniciamos la molturación y cancelamos la tarea
                       global.setx(0);
                       global.setMalteado(1);
                       global.setOn(true);
                       timerTask.cancel();

                   }

               }

               //ACTUALIZAR VALORES
               handler.post(new Runnable() {
                   public void run() {

       tempagua.setText(String.valueOf(global.getTempagua_malteado()));
       humedadgrano.setText(String.valueOf(global.getHumedadgrano_maltedado()));
                   }
               });

           }
       };
       timer.schedule(timerTask,0, 2000);
```

Germinación

En cuanto a la germinación se trata de un cronómetro como hemos explicado en el apartado de 8.10.

```
cronos = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true){
            if (global.isOn()){
```

```
btn_start.setVisibility(View.INVISIBLE);
try {
    Thread.sleep(1000);
}catch (InterruptedException e){
    e.printStackTrace();
}
global.setSeg(global.getSeg()+1);
if(global.getSeg()==59){
    global.setMin(global.getMin()+1);
    global.setSeg(0);
}
if(global.getMin()==59){
    global.setHoras(global.getHoras()+1);
    global.setMin(0);
}
if(global.getHoras()==24){
    global.setDias(global.getDias()+1);
    global.setHoras(0);
}
if(global.getDias()==5){
    global.setSeg(0);
    global.setMin(0);
    global.setHoras(0);
    global.setDias(0);
    progressbar.setProgress(7200);

    h.post(new Runnable() {
        public void run() {
            labelclock.setText("PROCESO FINALIZADO");
            btn_start.setVisibility(View.VISIBLE);
            btn_pausar.setVisibility(View.INVISIBLE);
        }
    });

    global.setOn(false);
}
h.post(new Runnable() {
    @Override
    public void run() {
        String s="",mi="",h="",d="";
        if(global.getSeg()<10){
            s="0"+global.getSeg();
        }else{
            s=""+global.getSeg();
        }
        if(global.getMin()<10){
            mi="0"+global.getMin();
        }else{
            mi=""+global.getMin();
        }
        if(global.getHoras()<10){
            h="0"+global.getHoras();
        }else{
            h=""+global.getHoras();
        }
        if(global.getDias()<10){
            d="0"+global.getDias();
        }else{
            d=""+global.getDias();
        }
        labelclock.setText(d+":"+h+":"+mi+":"+s);
        if(global.getDias()==0){
           p=7200-((global.getHoras()*60)+global.getMin());
        }
        if(global.getHoras()==0){
```

```
                            p=7200-((global.getDias()*24*60)+global.getMin());
                        }
                        if(global.getDias()!=0 || global.getHoras()!=0){
                            p=7200-
(((global.getDias()*24*60)+(global.getHoras()*60))+global.getMin());

                        }
                        progressbar.setProgress(p);
                    }

                });
            }}

    }
});
cronos.start();
```

La tarea que ejecutamos se trata de un cronómetro que se ha creado para que vayan aumentando los segundos, los minutos y los días.

Lo primero se comprueba que la variable *isOn* es igual a true, a continuación se usa el método sleep() para dormir el hilo durante un segundo, despues incrementamos el valor de los segundos en una unidad. Si los segundos son igual a 59, incrementamos los minutos en una unidad, si los minutos son 59 incrementamos las horas en una unidad y por último si las horas son 24 incrementamos los días en una unidad.

A continuación, comprobamos si los días son iguales a 5, en este caso damos por finalizado el proceso y cambiamos la variable *isOn* a false para que una vez finalice la tarea no se vuelva a ejecutar.

En caso de que los días no sean 5 continuamos hasta finalizar la tarea, de forma que creamos otra tarea concurrente en la cual comprobamos el valor de los segundos horas, minutos y días, si el valor es menos a 10 cuando queramos mostrarlo por pantalla deberemos poner un “0” delante. Por último, mediante setText() mostramos el tiempo que lleva ejecutándose y actualizamos el progreso en la *progressbar*.

Aquí finaliza la tarea, si los días son igual a 5 no volverá a ejecutarse ya que isOn será false y si no es igual a 5, la tarea se ejecutará de nuevo.

Secado o Tostado

Como ya sabemos este proceso consta de dos fases que se llevan a cabo con dos timer. En primer lugar, comprobamos el valor de la humedad de la malta y dependiendo de su valor, iniciamos una fase u otra.

```
if(global.getHumedad_secado()<=5.0){
    AlertDialog.Builder myBuild = new AlertDialog.Builder(SecadoTostado.this);
    myBuild.setMessage("El proceso ha finalizado");
    myBuild.setTitle("Información");
    myBuild.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });
```

```
    AlertDialog dialog = myBuild.create();
    dialog.show();
}
if(global.getHumedad_secado()<10.0 && global.getHumedad_secado()>5.0){
    iniciarFase2();
}

if(global.getHumedad_secado()>=10.0){
    iniciarFase1();
}
```

El código de la primera fase es el siguiente:

```
    public void iniciarFase1(){

        tempmalta.setFocusableInTouchMode(false);
        tempaire.setFocusableInTouchMode(false);
        humedadmalta.setFocusableInTouchMode(false);

        timer1 = new Timer();
        timerTask1 = new TimerTask() {
            @Override
            public void run() {

                if(global.getTempaire()<50.0){
                    fase1.setVisibility(View.INVISIBLE);
                    fase2.setVisibility(View.INVISIBLE);
                    global.setTempaire(global.getTempaire()+5.0);
                }
                if(global.getTempaire()>70.0){
                    fase1.setVisibility(View.INVISIBLE);
                    fase2.setVisibility(View.INVISIBLE);
                    global.setTempaire(global.getTempaire()-5.0);
                }

                if(global.getTempaire()>=50.0 && global.getTempaire()<=70.0){
                    handler.post(new Runnable() {
                        public void run() {
                            fase1.setVisibility(View.VISIBLE);
                            fase2.setVisibility(View.INVISIBLE);
                        }
                    });

                    global.setHumedad_secado(global.getHumedad_secado()-2.0);
                    //Actualizar valores
                    handler.post(new Runnable() {
                        public void run() {

    humedadmalta.setText((String.valueOf(global.getHumedad_secado())));
                        }
                    });

                    if(global.getHumedad_secado()<=10.0){

                        fase1.setVisibility(View.INVISIBLE);
                        fase2.setVisibility(View.INVISIBLE);
                        iniciarFase2();
                        timerTask1.cancel();

                    }
                }

                //Actualizar valores
                handler.post(new Runnable() {
                    public void run() {
                        tempaire.setText(String.valueOf(global.getTempaire()));
```

```
    humedadmalta.setText((String.valueOf(global.getHumedad_secado())));
                    }
                });
            }
        };
        timer1.schedule(timerTask1,0, 2000);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    }
```

El objetivo de este código es que cada 2 segundos nos compruebe en primer lugar la temperatura del aire del proceso, si esta es menor que 50, suba 5 grados su temperatura, si es mayor que 70 se disminuyan 5 grados.

Cuando la temperatura del aire este entre 50 y 70 grados, lo que hacemos es disminuir la humedad de la malta, ya que el objetivo de este proceso es que la humedad este por debajo de 5% al final del proceso.

Una vez disminuida la guardamos en la variable global correspondiente, la leemos de nuevo esta variable y si está por debajo de 10% iniciamos la fase 2 y dejamos de ejecutar esta tarea.

Si esta no está por debajo del 10%, actualizamos los datos de los *EditText* y la tarea no finaliza, por lo que dentro de 2 segundos se repetirá este proceso.

Para la fase 2 tenemos el siguiente código:

```
public void iniciarFase2(){

     timer2 = new Timer();
    timerTask2 = new TimerTask() {
        @Override
        public void run() {

            if(global.getTempaire()<global.getTempaireminobjetivo()){


                fase1.setVisibility(View.INVISIBLE);
                fase2.setVisibility(View.INVISIBLE);

                global.setTempaire(global.getTempaire()+3.0);


            }
            if (global.getTempmalta()<60.0){
                fase1.setVisibility(View.INVISIBLE);
                fase2.setVisibility(View.INVISIBLE);

                global.setTempmalta(global.getTempmalta()+2.0);
            }

            if(global.getTempaire()>global.getTempairemaxobjetivo()){

                fase1.setVisibility(View.INVISIBLE);
                fase2.setVisibility(View.INVISIBLE);

                global.setTempaire(global.getTempaire()-3.0);

            }

            if (global.getTempmalta()>65.0){
                fase1.setVisibility(View.INVISIBLE);
```

```
                fase2.setVisibility(View.INVISIBLE);

                global.setTempmalta(global.getTempmalta()-2.0);
            }

            if(global.getTempaire()>=global.getTempaireminobjetivo() &&
global.getTempaire()<=global.getTempairemaxobjetivo() && global.getTempmalta()>=60.0
&& global.getTempmalta()<=65.0){

                handler.post(new Runnable() {
                    public void run() {
                        fase1.setVisibility(View.INVISIBLE);
                        fase2.setVisibility(View.VISIBLE);
                    }
                });

                global.setHumedad_secado(global.getHumedad_secado()-2.0);
                //Actualizar el valor
                handler.post(new Runnable() {
                    public void run() {

humedadmalta.setText((String.valueOf(global.getHumedad_secado())));
                    }
                });
                if(global.getHumedad_secado()<=5){

                    handler.post(new Runnable() {
                        public void run() {
                            fase1.setVisibility(View.INVISIBLE);
                            fase2.setVisibility(View.INVISIBLE);
                            AlertDialog.Builder myBuild = new
AlertDialog.Builder(SecadoTostado.this);
                            myBuild.setMessage("El proceso ha finalizado");
                            myBuild.setTitle("Información");
                            myBuild.setPositiveButton("Aceptar", new
DialogInterface.OnClickListener() {
                                @Override
                       public void onClick(DialogInterface dialog, int which) {
                                    dialog.cancel();
                                }
                            });
                            AlertDialog dialog = myBuild.create();
                            dialog.show();
                        }
                    });
                    timerTask2.cancel();
                }
            }
            //Actualizar valores
            handler.post(new Runnable() {
                public void run() {
                    tempaire.setText(String.valueOf(global.getTempaire()));
                    tempmalta.setText(String.valueOf(global.getTempmalta()));

humedadmalta.setText((String.valueOf(global.getHumedad_secado())));
                }
            });
        }
    };
    timer2.schedule(timerTask2,0, 2000);
}
```

En esta fase se realiza lo mismo que en la anterior solo que las Temperaturas del aire ahora se tratan de variables globales, ya que dependiendo de la receta que elijamos, ese valor es uno u otro. En la fase 1 eran temperaturas constantes elijamos la receta que elijamos.

En primer lugar, comprobamos si la temperatura del aire está por debajo de la temperatura mínima objetivo, si es así, aumentamos su valor 3°C, si está por encima de la temperatura máxima, disminuimos su valor 3°C. A continuación, se comprueba si la temperatura de la malta es menor que 60°C, si esto ocurre aumentamos la temperatura de la malta 2°C, si esta está por encima de 65°C disminuimos su valor 2°C. Cuando la temperatura del aire se encuentra entre la máxima y la mínima objetivo y la temperatura de la malta se encuentra entre 60 y 65°C, disminuimos el valore de la humedad un 2% guardándolo en la variable global correspondiente.

Leemos esta variable, si su valor es igual o menos al 5% el proceso ha finalizado, la aplicación nos mostrará un *AlertDialog* y se cancelará la tarea. Si no, actualizamos los valores de los *EditText* y dentro de 2 segundos se repetirá la tarea.

Molturación

Al igual que en la germinación este proceso se trata de un cronómetro, pero el tiempo del proceso varía dependiendo de los litros que se produzcan. A continuación, se muestra el código implementado para el cálculo del tiempo.

```
//Calculo tiempo necesario
tiemponecesario=global.getProduccion()*30/200;
//Hacemos que el progressbar tenga tantas divisiones como minutos del tiempo
necesario
progressbarmoltulacion.setMax((int)tiemponecesario);
progressbarmoltulacion.setProgress((int)tiemponecesario);
```

Se crea una variable de tipo *int* llamada *tiemponecesario.* Sabiendo que para 200 litros son 30 minutos hacemos la regla de 3 para calcular el tiempo necesario. Seguidamente hacemos que la barra de progreso tenga tantas divisiones como minutos va a durar el proceso para que nos muestre el progreso.

```
cronos2 = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true){
            if (global.isOn2()){
                try {
                    Thread.sleep(1000);
                }catch (InterruptedException e){
                    e.printStackTrace();
                }
                global.setSeg2(global.getSeg2()+1);

                if(global.getSeg2()==59){
                    global.setMin2(global.getMin2()+1);
                    global.setSeg2(0);
                }

                if(tiemponecesario>=30 || tiemponecesario<=60){

                    if(global.getMin2()==(int)tiemponecesario){
                        global.setSeg2(0);
                        global.setMin2(0);
                        global.setHoras2(0);
```

```
                        global.setOn2(false);
                    }
                }

                if(tiemponecesario>60){
                    horas=(int)tiemponecesario/60;
                    minutos=(int)tiemponecesario-(horas*60);

                    if(global.getMin2()==59){
                        global.setHoras2(global.getHoras2()+1);
                        global.setMin2(0);
                    }
                    if(global.getHoras2()==horas && global.getMin2() == minutos){
                        global.setSeg2(0);
                        global.setMin2(0);
                        global.setHoras2(0);

                        global.setOn2(false);
                    }
                }

                h.post(new Runnable() {
                    @Override
                    public void run() {
                        String s2="",mi2="",h2="";
                        if(global.getSeg2()<10){
                            s2="0"+global.getSeg2();
                        }else{
                            s2=""+global.getSeg2();
                        }
                        if(global.getMin2()<10){
                            mi2="0"+global.getMin2();
                        }else{
                            mi2=""+global.getMin2();
                        }
                        if(global.getHoras2()<10){
                            h2="0"+global.getHoras2();
                        }else{
                            h2=""+global.getHoras2();
                        }
                        labeltiempomolienda.setText(h2+":"+mi2+":"+s2);
                        if(global.getHoras2()==0){

                            p2=(int)tiemponecesario-(global.getMin2());
                        }
                        if(global.getHoras2()!=0){
                            p2=(int)tiemponecesario-
(((global.getHoras2()*60))+global.getMin2());
                        }
                        progressbarmoltulacion.setProgress(p2);
                    }
                });
            }}
    }
});
cronos2.start();
```

Como observamos el cronometro se trata de un hilo Theard y un Runnable. La diferencia con el que se ha explicado en apartados anteriores es que existe una condición más en el hilo. Si el tiempo está en 30 y 60 minutos, las horas que se deben alcanzar serán cero y los minutos serán igual al tiempo necesario calculado. Cuando los minutos del cronómetro sean igual a los minutos necesarios el proceso finalizará.

Se ha puesto la condición de mayor o igual que 30 minutos ya que la producción mínima son 200 litros y por lo tanto el tiempo mínimo siempre será 30 minutos.

En caso de que el tiempo sea superior a una hora, se calculan las horas necesarias dividendo entre 60, cada vez que los minutos alcancen 59 sumamos 1 a las horas y cuando las horas sean igual a las horas necesarias y los minutos a los minutos necesarios, el proceso finalizará.

Maceración

La tarea que realizamos de forma periódica es la siguiente:

```
timermaceracion = new Timer();
timerTaskmaceracion = new TimerTask() {
    @Override
    public void run() {
        if(global.getAguamaceracion()<global.getminmaceracion()){
            global.setAguamaceracion(global.getAguamaceracion()+2.0);
            handler.post(new Runnable() {
                public void run() {
                    estado.setText("Calentando agua");
                }
            });
        }
        if(global.getAguamaceracion()>global.getmaxmaceracion()){
            global.setAguamaceracion(global.getAguamaceracion()-2.0);
            handler.post(new Runnable() {
                public void run() {
                    estado.setText("Enfriando agua");
                }
            });
        }
        if(global.getAguamaceracion()>=global.getminmaceracion()&&
global.getAguamaceracion()<=global.getmaxmaceracion()){
            handler.post(new Runnable() {
                public void run() {
                    estado.setText("Se introduce el agua en el depósito");
                    estado2.setText("Estado del proceso: Temperatura correcta");
                }
            });
            try {
                //Ponemos a "Dormir" el programa durante los ms que queremos
                Thread.sleep(5*1000);
                //labeltemp.setText("Programa durmiendo");
            } catch (Exception e) {
                System.out.println(e);
            }

            global.setMezclamaceracion(global.getAguamaceracion()-9.0);
            handler.post(new Runnable() {
                public void run() {
                    estado.setText("La maceración ha empezado");
                }
            });
            global.setOn3(true);
        }
        //ACTUALIZAR VARIABLES
        handler.post(new Runnable() {
            public void run() {
                aguamaceracion.setText(String.valueOf(global.getAguamaceracion()));
                mezcla.setText(String.valueOf(global.getMezclamaceracion()));
            }
        });
```

```
    }
};
timermaceracion.schedule(timerTaskmaceracion,0, 2000);
```

En primer lugar, se comprueba si la temperatura del agua está dentro de la máxima y la mínima temperatura objetivo de maceración. Si está por debajo de la mínima se aumenta la temperatura del agua, si está por encima de la máxima se disminuye la temperatura.

En caso de que la temperatura se encuentre en el rango óptimo, se introduce el agua en el depósito de maceración y tras 5 segundos, comienza la maceración. Se pone en marcha el cronómetro y se actualizan los datos.

Filtrado

```
cronosfiltrado = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true){
            if (global.isOn4()){
                btn_startfiltrado.setVisibility(View.INVISIBLE);
                try {
                    Thread.sleep(1000);
                }catch (InterruptedException e){
                    e.printStackTrace();
                }
                global.setSeg4(global.getSeg4()+1);
                if(global.getSeg4()==59){
                    global.setMin4(global.getMin4()+1);
                    global.setSeg4(0);
                }

                if(global.getMin4()<45){
                    h.post(new Runnable() {
                        public void run() {
                            label2.setText("Primera Filtración del mosto");
                            label3.setText("PROCESO ACTIVO");
                        }
                    });
                }
                if(global.getMin4()>=45){
                    h.post(new Runnable() {
                        public void run() {
                            label2.setText("Lavado del bagazo");
                            label3.setText("Calentando agua");
                        }
                    });
                    if(global.getTempaguafiltrado()<75.0){
                        global.setTempaguafiltrado(global.getTempaguafiltrado()+5.0);
                        h.post(new Runnable() {
                            public void run() {

tempaguafiltrado.setText(String.valueOf(global.getTempaguafiltrado()));
                                label3.setText("Calentando agua");
                            }
                        });
                    }
                    if(global.getTempaguafiltrado()>82.0){
                        global.setTempaguafiltrado(global.getTempaguafiltrado()-5.0);
                        h.post(new Runnable() {
                            public void run() {
```

```
tempaguafiltrado.setText(String.valueOf(global.getTempaguafiltrado()));
                                label3.setText("Enfriando agua");
                            }
                        });
                    }
                    if(global.getTempaguafiltrado()>=75.0 &&
global.getTempaguafiltrado()<=82.0){
                        h.post(new Runnable() {
                            public void run() {
                                label3.setText("Temperatura correcta");
                            }
                        });
                    }

                }

                if(global.getMin4()==tiempo){
                    global.setSeg4(0);
                    global.setMin4(0);
                    progressbarfiltrado.setProgress(60);

                    h.post(new Runnable() {
                        public void run() {
                            label3.setText("PROCESO FINALIZADO");
                            btn_startfiltrado.setVisibility(View.VISIBLE);
                            btn_pausarfiltrado.setVisibility(View.INVISIBLE);
                        }
                    });


                    AlertDialog.Builder myBuild = new
AlertDialog.Builder(Filtrado.this);
                    myBuild.setMessage("Filtración terminada. ¿Quiere iniciar el
proceso de Coccio y enfriado?");
                    myBuild.setTitle("Alerta");
                    myBuild.setPositiveButton("Aceptar", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {

                        }
                    });

                    myBuild.setNegativeButton("No", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            dialog.cancel();
                        }
                    });



                    global.setOn4(false);


                }
                h.post(new Runnable() {
                    @Override
                    public void run() {
                        String s4="",mi4="";
                        if(global.getSeg4()<10){
                            s4="0"+global.getSeg4();
                        }else{
                            s4=""+global.getSeg4();
                        }
```

```
                        if(global.getMin4()<10){
                            mi4="0"+global.getMin4();
                        }else{
                            mi4=""+global.getMin4();
                        }
                        labelclockfiltrado.setText(mi4+":"+s4);
                        pfiltrado= tiempo -(global.getMin4());
                        progressbarfiltrado.setProgress(pfiltrado);
                    }

                });
            }}

    }
});
cronosfiltrado.start();
```

Como se observa se trata de un cronómetro en el cual cada segundo verificamos el tiempo del proceso que lleva funcionando, si es menor de 45 minutos indicamos que el proceso que está activo es la primera filtración del mosto, si es mayor de 45 minutos, comprobamos la temperatura del agua, si es menor de 75°C aumentamos la temperatura en 5°C, si la temperatura es mayor de 82°C, la disminuimos. Una vez la temperatura del agua se encuentre entre 75 y 82 °C, se introduce el agua en el depósito de filtrado y comienza el lavado del bagazo. Cuando se alcanzan los 60 minutos del cronómetro, el proceso finaliza y nos muestra una alerta.

Cocción y enfriado

Este proceso consta de dos cronómetros similares a los explicados anteriormente, uno de 90 minutos para la cocción y otro de 75 minutos para el enfriado.

También haremos uso de dos timer uno para llevar al mosto a ebullición nada más empezar el proceso y otro para enfriar el mosto que comenzará cuando la cocción haya terminado.

A continuación, se muestra el código implementado:

```
public void iniciarTimercoccion(){
    timer = new Timer();
    timerTask = new TimerTask() {
        @Override
        public void run() {

            if(global.getTemperaturamosto()<100.0){
                global.setTemperaturamosto(global.getTemperaturamosto()+5.0);
            }
            if(global.getTemperaturamosto()>=100.0){
                global.setOn(true);
                timerTask.cancel();
            }
            //ACTUALIZAR VALORES
            h.post(new Runnable() {
                public void run() {
tempmosto.setText(String.valueOf(global.getTemperaturamosto()));
                }
            });
        }
    };
    timer.schedule(timerTask,0, 2000);
}

public void iniciarTimerenfriado(){
```

```
    timer = new Timer();
    timerTask = new TimerTask() {
        @Override
        public void run() {

            if(global.getTemperaturamosto()<global.getTemperaturaminmosto()){
                global.setTemperaturamosto(global.getTemperaturamosto()+3.0);
            }
            if(global.getTemperaturamosto()>global.getTemperaturamaxmosto){
                global.setTemperaturamosto(global.getTemperaturamosto()-3.0);
            }
            if(global.getTemperaturamosto()<=global.getTemperaturamaxmosto() &&
global.getTemperaturamosto()>=global.getTemperaturaminmosto()){
                global.setOn2(true);
                timerTask.cancel();
            }
            //ACTUALIZAR VALORES
            h.post(new Runnable() {
                public void run() {
                    tempmosto.setText(String.valueOf(global.getTemperaturamosto()));
                }
            });

        }
    };
    timer.schedule(timerTask,0, 120000);
}
```

En el timer de la cocción comprobamos si la temperatura es menor 100°C y aumentamos su valor. Si es igual o mayor que 100°C la tarea finaliza.

En el segundo timer comprobamos si la temperatura es menor que la temperatura mínima objetivo y si es así aumentamos su valor, si es mayor que la temperatura máxima objetivo, disminuimos su valor y si está en el rango deseando, la tarea se da por finalizada. Este proceso tardará aproximadamente 75 minutos, es por ello por lo que aumentamos o disminuimos cada 120000 milisegundos que son aproximadamente 2 minutos.

La temperatura mínima y máxima objetivo dependerán de la receta que hayamos elegido.

Fermentación

En primer lugar, se cargan las temperaturas objetivo guardadas anteriormente al elegir la receta mediante el siguiente código implementado.

```
public void cargardatos(){
    tminfermentacion.setText(String.valueOf(global.getFermentacionmin()));
    tmaxfermentacion.setText(String.valueOf(global.getFermentacionmax()));
}
```

A continuación, llevamos al fermentador a la temperatura deseada de la siguiente forma:

```
if(global.getTemperaturafermentador()<global.getFermentacionmin()){
    global.setTemperaturafermentador(global.getTemperaturafermentador()-1.0);
}
if(global.getTemperaturafermentador()>global.getFermentacionmax()){
    global.setTemperaturafermentador(global.getTemperaturafermentador()-1.0);
}
if(global.getTemperaturafermentador()<global.getFermentacionmax() &&
global.getTemperaturafermentador()>global.getFermentacionmin()){
```

```
    global.isOnfermentacion(true);
}
```

Por último, activamos el cronómetro con la función isOnfermentacion(). Este cronómetro es similar a los anteriores, la única diferencia es que el tiempo que deseamos que alcance se encuentra en la variable global getTiempofermentacion() que se ha guardado a la hora de elegir la receta.

6.13. Historial

En cuanto al historial se trata de una tabla en la base de datos que almacena los datos de cada lote.

En primer lugar, creamos la clase Lote, para crear los lotes que se producen, cada lote contendrá una serie de parámetros: la fecha en la que se produjo, la cantidad de cerveza y la receta. También implementaremos las funciones de leer y escribir para todos los parámetros de cada lote.

```
public Lote(String receta, String fecha, double litros) {
        this.receta = receta;
        this.fecha = fecha;
        this.litros = litros;
    }

    public Lote() {
    }
    public String getReceta() { return receta; }

    public void setReceta(String receta) { this.receta = receta; }

    public String getFecha() { return fecha; }

    public void setFecha(String fecha) { this.fecha = fecha; }

    public double getLitros() { return litros; }

    public void setLitros(double litros) { this.litros = litros; }
```

A continuación, tenemos la conexión a la base de datos.

```
public class ConexionSQLiteHelperLotes extends SQLiteOpenHelper {

    public ConexionSQLiteHelperLotes(@Nullable Context context, @Nullable String
name, @Nullable SQLiteDatabase.CursorFactory factory, int  ersión) {
        super(context, name, factory,  ersión);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(UtilidadesLote.CREAR_TABLA_LOTES);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)  {
        db.execSQL("DROP TABLE IF EXISTS lotes");
        onCreate(db);
    }
}
```

Cuando llamamos a *UtilidadesLote.CREAR_TABLA_LOTES* estamos llamando a la siguiente clase, la que se encarga de crear la tabla con los campos que le indiquemos.

```
public class UtilidadesLote {
    //Constantes campos tabla lotes
    public static final String TABLA_LOTES ="lote";
    public static final String CAMPO_RECETA ="receta";
    public static final String CAMPO_FECHA ="fecha";
    public static final String CAMPO_LITROS ="litros";
    public static final String CREAR_TABLA_LOTES ="CREATE TABLE "+TABLA_LOTES+"
("+CAMPO_RECETA+" TEXT, "+CAMPO_FECHA+" TEXT, "+CAMPO_LITROS+" TEXT)";
}
```

Cuando se pulsa el botón de la ventana de producción se crea un nuevo lote y se almacena en la base de datos llamando a la función registrarlote().

```
public void registrarlote (){

    date = Calendar.getInstance().getTime();
    dateFormat = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss");
    strDate = dateFormat.format(date);

    ConexionSQLiteHelperLotes conn = new ConexionSQLiteHelperLotes(this, "bd_lotes",
null, 1);

    SQLiteDatabase db = conn.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(UtilidadesLote.CAMPO_RECETA,receta);
    values.put(UtilidadesLote.CAMPO_LITROS,litros.getText().toString());
    values.put(UtilidadesLote.CAMPO_FECHA,strDate);

    Long idResultante =
db.insert(UtilidadesLote.TABLA_LOTES,UtilidadesLote.CAMPO_FECHA,values);
    Toast.makeText(getApplicationContext(),"Lote registrado" +idResultante,
Toast.LENGTH_SHORT).show();
    db.close();

}
```

A continuación, veremos el código necesario para listar los usuarios:

```
private void consultarListaLotes(){
    SQLiteDatabase db=conn.getReadableDatabase();
    Lote lote=null;
    listalotes=new ArrayList<Lote>();
    Cursor cursor=db.rawQuery("SELECT * FROM "+ UtilidadesLote.TABLA_LOTES,null);
    while(cursor.moveToNext()){
        lote = new Lote();
        lote.setReceta(cursor.getString(0));
        lote.setFecha(cursor.getString(1));
        lote.setLitros(cursor.getDouble(2));

        listalotes.add(lote);
    }
    obtenerLista();
}

private void obtenerLista() {
    listaInformacionlotes = new ArrayList<String>();
    for(int i=0; i<listalotes.size();i++){
        listaInformacionlotes.add(listalotes.get(i).getReceta()+" -
"+listalotes.get(i).getFecha()+"-"+listalotes.get(i).getLitros());
    }
}
```

7. Pruebas y ajustes finales o de servicio

Entrada	Salida esperada	Resultado
Registrarse	Registro del usuario en la base de datos.	Funcionamiento correcto
Registrarse con algún dato sin completar	No permitir el registro	Funcionamiento correcto
Dar de baja al usuario	Se elimina el usuario de la base de datos	Funcionamiento correcto
Inicio de sesión	Permite el inicio de sesión en la aplicación	Funcionamiento correcto
Inicio de sesión sin indicar algún campo	No permite iniciar sesión	Funcionamiento correcto
Inicio de sesión con algún campo erróneo	No permite iniciar sesión	Funcionamiento correcto
Cerrar sesión	Se permite el cierre de sesión y se vuelve a la pantalla de identificación	Funcionamiento correcto
Se edita algún usuario	Edita los datos del usuario en la base de datos	Funcionamiento correcto
Se edita algún usuario con un campo vacío	No se editan los datos del usuario	Funcionamiento correcto
Se consultan los datos de un usuario	Muestra la consulta	Funcionamiento correcto
Se accede mediante los botones a las distintas funciones de la aplicación	Se accede correctamente a las ventanas correspondientes	Funcionamiento correcto
Se introducen las variables del proceso de malteado	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de malteado	El proceso de malteado se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de malteado	El proceso de malteado se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de malteado	El proceso de malteado finaliza	Funcionamiento correcto
Se introducen las variables del proceso de germinación	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de germinación	El proceso de germinación se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de germinación	El proceso de germinación se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de germinación	El proceso de germinación finaliza	Funcionamiento correcto
Se pulsa el botón simular tiempo del proceso de germinación	El proceso de germinación avanza hasta quedar 5 minutos para finalizar	Funcionamiento correcto
Se introducen las variables del proceso de secado	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de secado	El proceso de secado se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de secado	El proceso de secado se pausa	Funcionamiento correcto

Se pulsa el botón finalizar del proceso de secado	El proceso de secado finaliza	Funcionamiento correcto
Se introducen las variables del proceso de molturación	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de molturación	El proceso de molturación se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de molturación	El proceso de molturación se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de molturación	El proceso de molturación finaliza	Funcionamiento correcto
Se pulsa el botón simular tiempo del proceso de molturación	El proceso de molturación avanza hasta quedar 5 minutos para finalizar	Funcionamiento correcto
Se introducen las variables del proceso de maceración	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de maceración	El proceso de maceración se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de maceración	El proceso de maceración se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de maceración	El proceso de maceración finaliza	Funcionamiento correcto
Se introducen las variables del proceso de filtración	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de filtración	El proceso de filtración se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de filtración	El proceso de filtración se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de filtración	El proceso de filtración finaliza	Funcionamiento correcto
Se introducen las variables del proceso de cocción y enfriado	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de cocción y enfriado	El proceso de cocción y enfriado se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de cocción y enfriado	El proceso de cocción y enfriado se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de cocción y enfriado	El proceso de cocción y enfriado finaliza	Funcionamiento correcto
Se introducen las variables del proceso de fermentación	Se guardan estas variables en las variables globales sin saltar ninguna excepción	Funcionamiento correcto
Se pulsa el botón iniciar del proceso de fermentación	El proceso de fermentación se inicia	Funcionamiento correcto
Se pulsa el botón pausar del proceso de fermentación	El proceso de fermentación se pausa	Funcionamiento correcto
Se pulsa el botón finalizar del proceso de fermentación	El proceso de fermentación finaliza	Funcionamiento correcto
Se introduce la cantidad disponible de los ingredientes en la ventana de ingredientes	Se guardan las cantidades en las variables globales	Funcionamiento correcto
Se introducen menos de 200 litros para producir en la ventana de producción	Se muestra un error y se pide que se aumente la producción. No permite iniciar la producción	Funcionamiento correcto
Se introducen una cantidad de litros a producir superior a los ingredientes disponibles	Se muestra un error y se pide que se disminuya la producción. No permite iniciar la producción	Funcionamiento correcto
Se introducen más de 200 litros en la ventana de producción	Se comprueba si los ingredientes disponibles son suficientes y comienza la producción	Funcionamiento correcto

Se pulsa el botón iniciar producción y los ingredientes están disponibles. La cantidad es superior a 200 litros	Se crea un nuevo lote en la base de datos.	Funcionamiento correcto
Se pulsa sobre el CardView de Historial en la ventana de Menú Principal	Se muestran los lotes producidos	Funcionamiento correcto
Se pulsa sobre el botón de volver atrás	Se vuelve a la ventana anterior	Funcionamiento correcto
Se pulsa el botón de gráficas en los procesos que lo incluyan.	Se abre la ventana de las gráficas correspondiente y se observan los datos en tiempo real.	Funcionamiento correcto
Se lee una variable global	No salta ningún error y se guarda	Funcionamiento correcto
Se cambia el valor de una variable global	No se produce ningún error y se guarda	Funcionamiento correcto
Se introduce un tipo de valor erróneo en las variables a controlar	Se muestra un error y el proceso no puede comenzar	Funcionamiento correcto
Se deja un EditText vacío	Se muestra un error y el proceso no puede comenzar	Funcionamiento correcto

Tabla 3: Pruebas finales

8. Presupuesto

En este apartado del proyecto se muestran los costes de los materiales, mano de obra y herramientas empleadas para llevar a cabo el proyecto.

APLICACIÓN MÓVIL				
MATERIALES				
Uds.	Denominación	Cantidades	Precio	Total
u	Ordenador Lenovo ideapad 530S-15IKB	1	905,75 €	905,75 €
u	Google Pixel 2	1	339,00 €	339,00 €
meses	Conexión a internet y telefonía fija Movistar	12	38,00 €	456,00 €
			SUBTOTAL MATERIALES	**1.700,75 €**
MANO DE OBRA				
Uds.	Profesional	Tiempo	Precio	Total
h	Programador	400	10,00 €	4.000,00 €
h	Analista	100	9,00 €	900,00 €
			SUBTOTAL MANO DE OBRA	**4.900,00 €**
LICENCIAS				
Uds.	Denominación	Tiempo	Precio	Total
meses	Office 365 Empresa (subscripción mínima: anual)	12	8,80 €	105,60 €
			SUBTOTAL MANO DE OBRA	**105,60 €**

Tabla 4: Presupuesto aplicación móvil desglosado

A continuación, se muestra la tabla resumen del presupuesto en la cual añadimos el coste del IVA y un beneficio del 12%.

PEM					
	Materiales	Mano de obra	Licencias	Costes Directos Complementarios (%)	Total
Aplicación	1.700,75 €	4.900,00 €	105,60 €		7.376,99 €
Beneficio				12%	885,24 €
IVA				21%	1.549,17 €
				SUBTOTAL	**9.811,39 €**

Tabla 5: Presupuesto total

9. Conclusiones

9.1. Valoración Personal

Gracias a este TFG He aprendido sobre la importancia de una buena planificación, diseño e identificación de los requisitos necesarios para la realización de un proyecto. He podido aprender muchas cosas sobre programación en Android y he ampliado mis conocimientos sobre el proceso de fabricación de la cerveza además de que he podido observar y visitar una de las fábricas de Valencia.

En cuanto al desarrollo de la aplicación es cierto que he tenido bastantes problemas a la hora de comenzar con ella ya que surgían problemas de actualizaciones y no sabía muy bien cómo encontrar esos errores y resolverlos. Además, no disponía de experiencia en el desarrollo de aplicaciones Android ni del lenguaje de programación java, por lo que me ha llevado bastante tiempo buscar información y material necesario para saber desarrollas las funcionalidades de la aplicación. Gracias a esto he llevado a cabo una formación autodidacta sobre el entorno y el lenguaje de programación y he mejorado en la capacidad de investigación y resolución de problemas.

Por otro lado, en cuanto al proceso de fabricación y el proyecto he tenido varias dudas sobre el dimensionamiento del proyecto ya que se trata de un proceso complejo del cual no me han podido dar mucha información detallada debido a la privacidad de datos de la empresa.

También cabe destacar que el proyecto realizado responde a las expectativas y los requerimientos que se tenían como punto de partida. Desarrollar un primer prototipo de aplicación para poder controlar y observar el proceso de fabricación de cerveza. Aunque es cierto que me hubiera gustado implementar más funcionalidades.

En conclusión, creo que mi involucración con este proyecto ha sido completa y el tiempo invertido y dedicación ha sido mucho con el fin de poder diseñar un primer prototipo de una aplicación para una fábrica de cerveza.

9.2. Futuras ampliaciones

En cuanto a posibles ampliaciones del prototipo que se ha desarrollado se podrían destacar dos principalmente.

La primera de ellas sería la obtención de datos a través de servidores SQL, para la gestión de usuarios, recetas y la obtención de historiales útiles para el rendimiento de la elaboración y no bases de datos locales ya que estas no se encontrarían en todos los dispositivos en los cuales instalemos la aplicación.

Por otro lado, otra posible ampliación seria la producción en cadena en el prototipo, ya que es lo que realmente ocurre en una fábrica de elaboración de cerveza, es decir la posibilidad de producción de varios lotes de cerveza al mismo tiempo. Poder visualizar el estado de cada uno de ellos y sus variables importantes, teniendo en cuenta los tiempos de cada etapa y los depósitos disponibles en la fábrica para evitar que el proceso se pare debido al mal cálculo de los tiempos.

Por otro lado, un futuro trabajo sería la implementación de la aplicación como tal y no solo un prototipo lo que conllevaría la conexión de la aplicación a los distintos servicios de la fábrica como sería la lectura de sensores, bases de datos, etc.

Referencias bibliográficas

[1] "Breve historia del origen de la cerveza." https://grannaria.com/breve-historia-del-origen-de-la-cerveza/ (accessed Sep. 01, 2020).

[2] "Ingredientes de la cerveza: El Agua | MEGA | Diccionario cervecero." https://mundoestrellagalicia.es/agua-ingrediente-cerveza/ (accessed Sep. 09, 2020).

[3] "La cebada, insumo esencial en la fabricación de cerveza | Cervecería Källa." http://www.cerveceriakalla.com/blog/la-cebada-insumo-esencial-en-la-fabricacion-de-la-cerveza/ (accessed Sep. 09, 2020).

[4] "The 'Food' for the Brew | SpringerLink." https://link.springer.com/chapter/10.1007/978-3-319-46394-0_5 (accessed Jun. 15, 2020).

[5] "El lúpulo, componente esencial para la elaboración de cerveza." https://www.labarracadelaspapas.com/lupulo-la-esencia-aromatica-la-cerveza/ (accessed Sep. 09, 2020).

[6] "Cerveza Artesana | La guía definitiva de la levadura." https://www.cervezartesana.es/blog/post/la-guia-definitiva-de-la-levadura.html (accessed Sep. 09, 2020).

[7] "Las Temperaturas de Maceración, Ciencia y Arte. – BrewMasters. Insumos e Ingredientes para Elaborar Cerveza." https://brewmasters.com.mx/las-temperaturas-de-maceracion-ciencia-y-arte/ (accessed Aug. 29, 2020).

[8] "¿Cuáles son los sistemas operativos más usados o utilizados en 2019? - ITSoftware." https://itsoftware.com.co/content/sistemas-operativos-mas-usados/ (accessed Sep. 09, 2020).

[9] "Arquitectura de la plataforma | Desarrolladores de Android." https://developer.android.com/guide/platform?hl=es-419 (accessed Sep. 09, 2020).

[10] "Distribution data for Android." https://androiddistribution.io/#/ (accessed Sep. 09, 2020).

[11] "Sistema operativo de código abierto Vs. código cerrado." https://techlandia.com/sistema-operativo-codigo-abierto-vs-codigo-cerrado-info_477610/ (accessed Sep. 09, 2020).

[12] "Cómo interpretar el ciclo de vida de una actividad." https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es (accessed Sep. 09, 2020).

[13] "Panel de distribución | Desarrolladores de Android." https://developer.android.com/about/dashboards (accessed Sep. 09, 2020).

[14] "Qué es un APK de Android, cómo se instala y diferencias con las apps normales." https://www.xatakandroid.com/aplicaciones-android/que-apk-android-como-se-instala-diferencias-apps-normales (accessed Sep. 09, 2020).

[15] "Calendar (Java Platform SE 7)." https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html#getInstance() (accessed Sep. 09, 2020).

[16] "Runnable | Desarrolladores de Android | Android Developers." https://developer.android.com/reference/java/lang/Runnable (accessed Aug. 31, 2020).

[17] "Thread | Desarrolladores de Android | Android Developers." https://developer.android.com/reference/java/lang/Thread#public-methods (accessed Aug. 31, 2020).

[18] "Diálogos | Desarrolladores de Android | Android Developers." https://developer.android.com/guide/topics/ui/dialogs?hl=es-419 (accessed Aug. 30, 2020).

[19] "Descripción general del almacenamiento de archivos y datos." https://developer.android.com/guide/topics/data/data-storage?hl=es-419#pref (accessed Aug. 29, 2020).

[20] "Cómo interpretar el ciclo de vida de una actividad." https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es#alc (accessed Aug. 28, 2020).

[21] "Bases de Datos en Android (I): Primeros pasos | sgoliver.net." https://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/ (accessed Aug. 28, 2020).

[22] "SQLite en App Android: actualizar, eliminar y consultar datos – Academia Android." https://academiaandroid.com/sqlite-en-app-android-actualizar-eliminar-y-consultar-datos/ (accessed Aug. 28, 2020).

[23] "SQLite en Android: creación y acceso base de datos e inserción de registros – Academia Android." https://academiaandroid.com/sqlite-android-creacion-acceso-base-datos-insercion/ (accessed Aug. 28, 2020).

[24] "Descripción general del almacenamiento de archivos y datos." https://developer.android.com/guide/topics/data/data-storage?hl=es-419 (accessed Aug. 28, 2020).

[25] "Download Android Studio and SDK tools | Android Studio." https://developer.android.com/studio (accessed Jul. 18, 2020).

[26] F. Alfonso, "El proceso de," *Neurología*, vol. 25, no. 9, pp. 521–529, 2010, Accessed: May 21, 2020. [Online]. Available: https://www.sabeer.es/cultura-cervecera/el-proceso-de-malteado-en-3-pasos/.

[27] "Malteado | Procesos de la elaboración de la cerveza - 3BTours." https://3btourspraga.com/tour-de-la-cerveza/elaboracion-de-la-cerveza/malteado/ (accessed May 21, 2020).

[28] R. Sancho Saurina, “Diseño de una micro-planta de fabricación de cerveza y estudio de técnicas y procesos de producción,” *Tesis doctoral. Universitat Politècnica de Catalunya (UPC)*, p. 121, 2015, Accessed: May 21, 2020. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/76575/02_Memoria.pdf?sequence=5&isAllowed=y%0Ahttps://upcommons.upc.edu/bitstream/handle/2117/76575/02_Memoria.pdf.

[29] John Palmer and Colin Kaminski. Water a comprehensive guide for brewers. Colorado:Brewers Publications, 2013. ISBN: 978-0-937381-99-1