TECNOLÓGICO NACIONAL DE MÉXICO

CAMPUS OAXACA

INSTITUTO TECNOLÓGICO DE OAXACA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

TOPICOS AVANZADOS DE PROGRAMACIÓN

"DOCUMENTACIÓN
PROYECTO: 'MEZCALERA' "

INTEGRANTES:

CHÁVEZ SÁNCHEZ KEVIN EDILBERTO
GARCÍA GARCÍA JOSÉ ÁNGEL

FECHA: 16/06/2020

Oaxaca de Juárez, Oax.

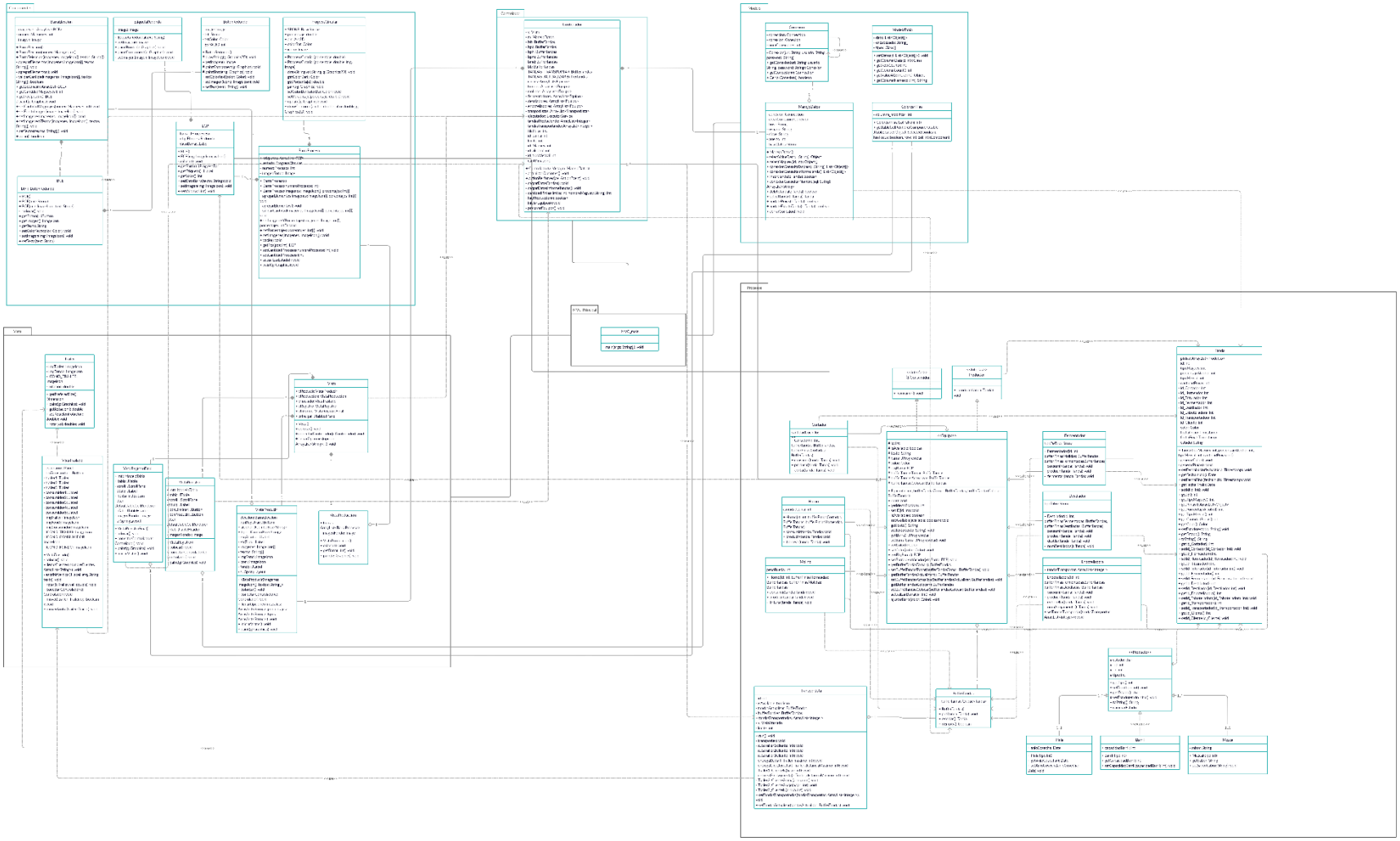
ÍNDICE

PLANTEAMIENTO DEL PROBLEMA

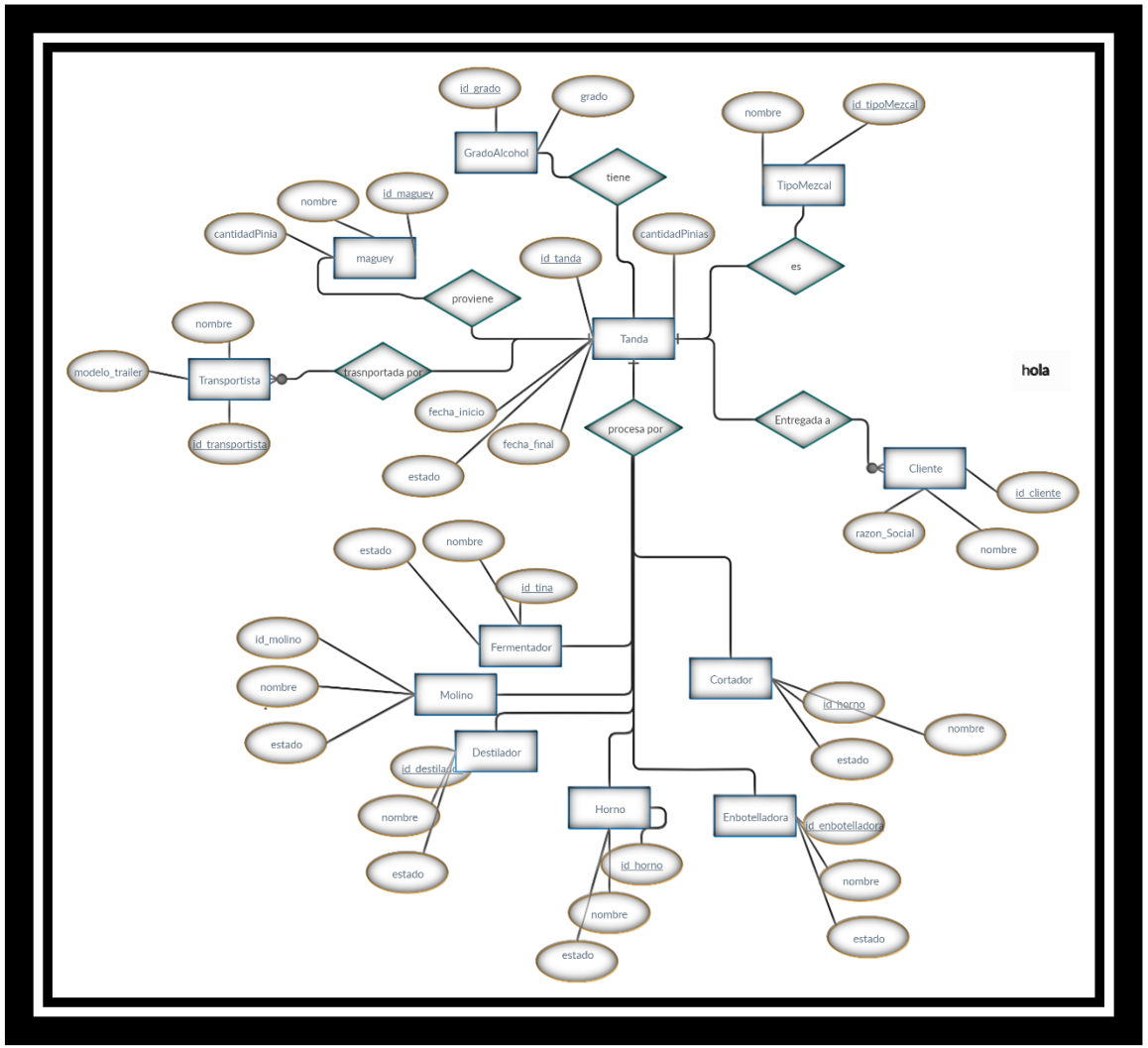
El ahorrar tiempo, además de manejar grandes cantidades de datos a través de una aplicación, que es necesario tener control sobre su funcionamiento y el proceso para la elaboración del mezcal. Tras adquirir el conocimiento esencial en el curso actual de la materia “Tópicos Avanzados de Programación”, se desarrolló e implementó una aplicación mostrando los procesos que pasa el maguey para convertirse en mezcal y así poder ser comercializado.

Ocupando así interfaces amigables, realizar una conexión a base de datos y el uso y manejo de hilos.

DÍAGRAMA DE CLASES UML



DÍAGRAMA ENTIDAD – RELACIÓN (BASE DE DATOS)



CÓDIGO FUENTE (ESENCIAL)

BCE.java

```
package Componentes;

import java.awt.Color;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.SpringLayout;

/**
 * Clase de un componente que tiene un bóton redondo y una etiqueta abajo.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class BCE extends JComponent{

    // Variable de instacia - Bóton redondo con imágen.
    private BotonRedondo btn;

    /**
     * Constructor para objetos de BCE.
     */
    public BCE(){
        btn = new BotonRedondo();
        colocar();
```

```
}

/**
 * Constructor para objetos de BCE con imagen y texto.
 *
 * @param img Imagen que tendrá.
 * @param text Texto de la etiqueta.
 */
public BCE(ImageIcon img, String text){
    btn = new BotonRedondo();
    setImagen(img);
    setTexto(text);
    colocar();
}

/**
 * Constructor para obetos BCE con texto y fondo circular.
 *
 * @param text Texto de la etiqueta
 */
public BCE(String text){
    btn = new BotonRedondo();
    setTexto(text);
    colocar();
}

/**
 * Coloca el bóton redondo en el componente.
```

```
 */
private void colocar(){
    SpringLayout s = new SpringLayout();
    setLayout(s);
    add(btn);
    s.putConstraint(SpringLayout.NORTH, btn, 0, SpringLayout.NORTH, this);
    s.putConstraint(SpringLayout.WEST, btn, 0, SpringLayout.WEST, this);
    s.putConstraint(SpringLayout.EAST, btn, -0, SpringLayout.EAST, this);
    s.putConstraint(SpringLayout.SOUTH, btn, -0, SpringLayout.SOUTH, this);
}

/**
 * Coloca el texto dado a la etiqueta del componente.
 *
 * @param text Texto que tendrá la etiqueta.
 */
public void setTexto(String text){
    btn.setText(text);
    repaint();
}

/**
 * @return texto que tiene la etiqueta.
 */
public String getText(){
    return btn.getText();
}
```

```
/**
 * Establece la imagen que contendrá el bóton.
 *
 * @param img Imagen a colocar.
 */
public void setImagen(ImageIcon img){
    btn.setImagen(img);
}

/**
 * Establece el color del texto de la etiqueta.
 *
 * @param color Color a usar.
 */
public void setColorText(Color color){
    btn.setColorText(color);
}

/**
 * @return imágen contenida en el bóton redondo.
 */
public ImageIcon getImagen(){
    return new ImageIcon(btn.getImagen());
}

/**
 * Retorna el bóton redondo.
 *
```

```
     * @return el boton que contiene.
     */
    public JButton getBoton(){
       return btn;
    }
}
```

BarraProceso.java

```
package Componentes;

import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.util.ArrayList;
import javax.swing.ImageIcon;
import javax.swing.JPanel;

/**
 * Clase de un componente tipo JPanel que representa una Barra de procesos (Etiquetas).
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class BarraProceso extends JPanel {

    // Variable de instancia - La cantidad de etiquetas que tendrá.
    private ArrayList<ECP> etiquetas;
```

```
    // Variable de instancia - Progress circular que indica el proceso de cada barra.
    private ProgressCircular estado;

    // Variable de instancia - La cantidad de procesos que tiene.
    private int numeroProcesos;

    // Variable de intancia - La imagende fondo que tiene la barra.
    private Image imagenFondo = new ImageIcon(getClass().getResource("/Imagenes/fondo_barra.jpg")).getImage();

    /**
     * Constructor para objetos de BarraProceso.
     */
    public BarraProceso() {
        etiquetas = new ArrayList<>();
        estado = new ProgressCircular(0);
    }

    /**
     * Constructor para objetos de BarraProceso con un
     * numero de etiquetas dado.
     *
     * @param numeroProcesos Cantidad de ECP.
     */
    public BarraProceso(int numeroProcesos) {
        estado = new ProgressCircular(0);
        setCantidadProcesos(numeroProcesos);
        agregarElemetos();
        actualizarEstado();
```

```
}

/**
 * Constructor para botones con imagenes y porcentajes dados.
 * Si son de diferente tamaño, se colocan unicamente
 * las imagenes.
 *
 * @param imagenes Arreglo de imagenes que tendrá las etiquetas.
 */
public BarraProceso(ImageIcon[] imagenes, int[] porcentajes) {
    estado = new ProgressCircular(0);
    if(validarCantidad(imagenes, porcentajes)){
        setCantidadProcesos(imagenes.length);
        agregarElementos(imagenes, porcentajes);
    }else{
        setCantidadProcesos(imagenes.length);
        agregarElemetos();
        setImagenes(imagenes);
    }
    actualizarEstado();
}

/**
 * Agrega los elementos con su imagen y porcentaje correspondiete.
 *
 * @param imagenes Arrelgo de imagenes a establecer.
 * @param porcentajes Arreglo de porcentajes a establecer.
 */
```

```
private void agregarElementos(ImageIcon[] imagenes, int[] porcentajes){
    setLayout(new GridLayout(1, 0));
    setSize(700, 220);
    for (int i = 0; i < numeroProcesos; i++){
        ECP e = new ECP(imagenes[i], porcentajes[i]);
        etiquetas.add(e);
        add(e);
    }
    /** Para nuestro proyecto no lo usaremos */
    //add(estado);
}

/**
 * Agrega los elementos al array y al panel.
 */
private void agregarElemetos(){
    setLayout(new GridLayout(1, 0));
    for (int i = 0; i < numeroProcesos; i++) {
        ECP e = new ECP();
        etiquetas.add(e);
        add(e);
    }
    add(estado);
}

/**
 * Valida que el numero de imagenes y porcentajes sea el correcto.
 *
```

```
     * @return true si son iguales y false de lo contrario
     */
    private boolean validarCantidad(ImageIcon[] imagenes, int[] porcentajes) {
        return  (imagenes == null || porcentajes == null) ? false :(imagenes.length == porcentajes.length);
    }

    /**
     * Coloca a las etiquetas las imagenes y porcentajes dados.
     * Si son de diferentes cantidades no las coloca a las etiquetas.
     *
     * @param imagenes Arreglo de imagenes a colocar.
     * @param porcentajes Arreglo de porcentajes a colocar.
     */
    public void setImagenesYPorcentajes(ImageIcon[] imagenes, int[] porcentajes) {
        if (validarCantidad(imagenes, porcentajes) && !vacio()) {
            setPorcentajes(porcentajes);
            setImagenes(imagenes);
        }
    }

    /**
     * Coloca los porcentajes dados a las etiquetas.
     *
     * @param porcentajes Arreglo de porcentajes a colocar.
     */
    public void setPorcentajes(int[] porcentajes){
        if (!vacio() && porcentajes.length == numeroProcesos)
            for (int i = 0; i < numeroProcesos; i++)
```

```
            etiquetas.get(i).setValor(porcentajes[i]);
    }

    /**
     * Coloca las imagenes a las etiquetas.
     *
     * @param imagenes Arreglo de imagenes a colocar.
     */
    public void setImagenes(ImageIcon[] imagenes) {
        if (!vacio() && imagenes.length == numeroProcesos)
            for (int i = 0; i < numeroProcesos; i++)
                etiquetas.get(i).setImagen(imagenes[i]);
    }

    /**
     * @return true si no tiene etiquetas y false de lo contrario.
     */
    private boolean vacio() {
        return numeroProcesos == 0;
    }

    /**
     * Retorna la etiqueta en una posición dada, en un rango de 0 a n.
     *
     * @param pos Posición dada.
     * @return ECP componente de la posición dada, si está vacía retorna un null.
     */
    public ECP getPos(int pos) {
```

```
        return (vacio()) ? null : etiquetas.get(pos);
    }

    /**
     * Establece la cantidad de procesos.
     *
     * @param numeroProcesos Cantidad de componentes ECP a tener.
     */
    public void setCantidadProcesos(int numeroProcesos) {
        this.numeroProcesos = (numeroProcesos < 0) ? 0 : numeroProcesos;
        this.etiquetas = (etiquetas == null) ? new ArrayList<>(numeroProcesos) : new
ArrayList<>();
    }

    /**
     * Devuelve la cantidad de etiquetas que tiene la barra.
     *
     * @return Cantidad de etiquetas que tiene.
     */
    public int getCantidadProcesos() {
        return numeroProcesos;
    }

    /**
     * Actualiza el estado de la barra de procesos.
     */
    public void actualizarEstado(){
        double promedio = 0;
        for (ECP e : etiquetas)
```

```
            promedio += e.getValor();
        estado.setPorcentaje(promedio / numeroProcesos);
        estado.repaint();
    }

    /**
     * Método para pintar la imágen de fondo.
     * @param g Graphics para pintar.
     */
     @Override
    public void paint(Graphics g) {
        g.drawImage(imagenFondo, 0, 0, getWidth(), getHeight(),this);
        setOpaque(false);
        super.paint(g);
    }
}
```

BarraEleccion.java

```
package Componentes;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.util.ArrayList;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
```

```
/**
 * Clase de un componente tipo JPanel que representa una Barra de elección (Botones).
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class BarraEleccion extends JPanel{

    // Variable de instancia - Array de Botones para los tipos de magueyes.
    private ArrayList<BCE> magueyes;

    // Variable de instancia - Cantidad de magueyes.
    private int numeroMagueyes;

    // Variable de instancia - Imagen de fondo.
    private Image imagen;

    /**
     * Constructor para objetos de BarraEleccion.
     */
    public BarraEleccion(){
        magueyes = new ArrayList<>();
    }

    /**
     * Constructor para objetos de BarraEleccion con una cantidad
     * de botones dada.
```

```
 *
 * @param numeroMagueyes Cantidad de botones.
 */
public BarraEleccion(int numeroMagueyes) {
    setCantidadMagueyes(numeroMagueyes);
    agregarElemetos();
}

/**
 * Constructor para objetos de BarraEleccion con los elementos
 * de imagenes y textos dados.
 * Si son de diferente tamaño, se colocan unicamente las imagenes.
 *
 * @param imagenes Arreglo de imagenes que tendrán los botones.
 * @param textos Arreglo de textos que tendrán los botones.
 */
public BarraEleccion(ImageIcon[] imagenes, String[] textos) {
    if(validarCantidad(imagenes, textos)){
        setCantidadMagueyes(imagenes.length);
        agregarElementos(imagenes, textos);
    }else{
        setCantidadMagueyes(imagenes.length);
        agregarElemetos();
        setImagenes(imagenes);
    }
}

/**
```

```
 * Agrega los elementos con su imagen y texto correspondiente.
 *
 * @param imagenes Arrelgo de imagenes a establecer.
 * @param textos Arreglo de textos a establecer.
 */
private void agregarElementos(ImageIcon[] imagenes, String[] textos){
    setLayout(new GridLayout(1, 0));
    for (int i = 0; i < numeroMagueyes; i++){
        BCE e = new BCE(imagenes[i], textos[i]);
        magueyes.add(e);
        add(e);
    }
}

/**
 * Agrega los elementos al array de botones y al panel.
 */
private void agregarElemetos(){
    setLayout(new GridLayout(1, 0));
    for (int i = 0; i < numeroMagueyes; i++) {
        BCE e = new BCE();
        magueyes.add(e);
        add(e);
    }
}


/**
```

```
     * Valida que el numero de imagenes y textos sea el correcto.
     *
     * @return true si son iguales y false de lo contrario.
     */
    private boolean validarCantidad(ImageIcon[] imagenes, String[] textos) {
        return (imagenes == null || textos == null) ? false : imagenes.length == textos.length;
    }

    /**
     * Método que indica si hay o no botones.
     *
     * @return true si no tiene botones y false de lo contrario.
     */
    public boolean vacio(){
        return numeroMagueyes == 0;
    }

    /**
     * Retorna el botón en una posición dada, en un rango de 0 a n.
     *
     * @param pos Posición dada.
     * @return Botón de la posición dada, si está vacía retorna un null.
     */
    public BCE getPos(int pos) {
        return (vacio()) ? null : magueyes.get(pos);
    }

    /**
```

```
 * Coloca las imagenes y textos dados a los botones.
 * Si son de tamaños diferentes no se colocan.
 * Si no hay botones, tampoco se colocan los elementos dados.
 *
 * @param imagenes Arreglo de imagenes a colocar.
 * @param textos Arreglo de textos a colocar.
 */
public void setImagenesYTextos(ImageIcon[] imagenes, String[] textos) {
    if (validarCantidad(imagenes,textos) && !vacio()) {
        setTextos(textos);
        setImagenes(imagenes);
    }
}

/**
 * Coloca las imagenes a los botones.
 * Si está vacia o la cantidad de imagenes suepera la de los botones,
 * no se establecen las imagenes dadas.
 *
 * @param imagenes Arreglo de imagenes a colorcar.
 */
public void setImagenes(ImageIcon[] imagenes) {
    if (!vacio() && imagenes.length == numeroMagueyes)
        for (int i = 0; i < numeroMagueyes; i++)
            magueyes.get(i).setImagen(imagenes[i]);
}

/**
```

```
     * Coloca los texos a los botones.
     * Si está vacia o la cantidad de textos suepera la de los botones,
     * no se establecen los textos dados.
     *
     * @param textos Arreglo de textos a colorcar.
     */
    public void setTextos(String[] textos){
        if (!vacio() && textos.length == numeroMagueyes)
            for (int i = 0; i < numeroMagueyes; i++) {
                magueyes.get(i).setTexto(textos[i].toUpperCase());
                magueyes.get(i).setFont(new Font("Arial", Font.BOLD, 14));
                magueyes.get(i).setColorText(Color.WHITE);
            }
    }

    /**
     * Establece la cantidad de magueyes que tendrá la barra.
     *
     * @param numeroMagueyes Cantidad de magueyes.
     */
    public void setCantidadMagueyes(int numeroMagueyes){
        this.numeroMagueyes = (numeroMagueyes > 0 ) ? numeroMagueyes : 0;
        this.magueyes = (magueyes == null) ? new ArrayList<>(numeroMagueyes) :
new ArrayList<>();
    }

    /**
     * Retorna la cantidad de botones que tiene la barra.
     *
```

```
 * @return La cantidad de magueyes que hay.
 */
public int getCantidadMagueyes(){
    return numeroMagueyes;
}

/**
 * Retorna el array que contiene a los botones.
 *
 * @return magueyes ArrayList de botones.
 */
public ArrayList<BCE> getBotones(){
    return magueyes;
}

/**
 * Método para colocar una imágen de fondo.
 *
 * @param imagen Imagen a colocar de fondo.
 */
public void setFondoImagen(ImageIcon imagen){
    this.imagen = imagen.getImage();
}

/**
 * Método para pintar la imágen de fondo.
 * @param g Graphics para pintar.
 */
```

```
    @Override
    public void paint(Graphics g) {
        g.drawImage(imagen, 0, 0, getWidth(), getHeight(), this);
        setOpaque(false);
        super.paint(g);
    }
}
```

Controlador.java

```
package Controlador;

import Modelo.ColorearFilas;
import Modelo.ManejoDatos;
import Procesos.*;
import Vista.Vista;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * Clase controladora, aquí se encuentra la parte lógica del proyecto.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
```

```
 * @version 1.0 14/06/2020
 */
public class Controlador implements ActionListener{

    // Variable de instancia - Vista del proyecto.
    private Vista v;

    // Variable de instancia - Modelo del proyecto.
    private ManejoDatos m;

    // Variables de instancia - Buffers de Tandas
    private BufferTandas bft = new BufferTandas(),
                         bpc = new BufferTandas(),
                         bph = new BufferTandas(),
                         bpm = new BufferTandas(),
                         bpf = new BufferTandas(),
                         bmd = new BufferTandas(),
                         bb  = new BufferTandas(),
            TANDAS_TRANSPORTAR = new BufferTandas(),
            TANDAS_ACTUALIZAR  = new BufferTandas();

    // Variables de instancia - Equipos
    private ArrayList<Equipo> cortes              = new ArrayList<>(),
                              hornos              = new ArrayList<>(),
                              molinos             = new ArrayList<>(),
                              fermentadores       = new ArrayList<>(),
                              destiladores        = new ArrayList<>(),
                              enbotelladores      = new  ArrayList<>();
```

```
    private ArrayList<Transportista> transportistas = new ArrayList<>();

    // Variable de instancia - Ejecutador de los Hilos.
    private ExecutorService ejecutador      = Executors.newCachedThreadPool();

    // Variable de instancia - Array de los ID de Tandas en proceso.
    private ArrayList<Integer>   tandasProduciendo  = new ArrayList<Integer>(),
                    tandasTransportando  = new ArrayList<Integer>();

    // Variables para las acciones de los botones.
    private int filaPulsada = 0, id_tanda = 0, limite = 0,id_Maguey = 0, id_alcohol = 0,
id_tipoMezcal = 0, cantPinias = 0;

    /**
     * Constructor para objetos de Controlador
     *
     * @param m Modelo para realizar consultas.
     * @param v Vista a mostrar.
     */
    public Controlador(Vista v, ManejoDatos m){
        this.v = v;
        this.m = m;
        prepararEquipos();
        IniciarEquipos();
        actualizarOpciones();
        cargarDatosTandas();
        cargarInformeTandas();
        v.addWindowListener(new WindowAdapter() {
            @Override
```

```
        public void windowClosing(WindowEvent e) {
            if(hayProduccion()){
                JOptionPane.showMessageDialog(v,"NO PUEDE ABANDONAR, HAY
MEZCALES PRODUCIENDOSE");

v.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
                return;
            }else{
                m.cerrarConexion();
                System.exit(0);
            }
        }});
    }


    /**
     * Consulta para rellenar las opciones de las diferentes vistas.
     */
    private void actualizarOpciones(){
        ArrayList<String> porcentajes = m.conexionConsultarNombre("select * from
mezcal.gradoalcohol"),
                          tipos       = m.conexionConsultarNombre("select * from
mezcal.tipomezcal"),
                          mezcales    = m.conexionConsultarNombre("select * from
mezcal.maguey order by id_maguey"),
                          clientes    = m.conexionConsultarNombre("select * from
mezcal.cliente");
        v.llenarOpciones(mezcales, porcentajes, tipos, clientes);
    }


    /**
```

```
 * Método que controla las acciones de los botones de cada vista.
 */
@Override
public void actionPerformed(ActionEvent ae) {
    Tanda t;
    Object datos[] = null;
    String o = ae.getActionCommand();
    if(o.matches("[0-9]*")){
        datos = m.selectMaguey(Integer.parseInt(o));
        System.out.println("Seleccionó el maguey " + (String) datos[1]);
        id_Maguey = (int) datos[0];
        limite = (int) datos[2];
        System.out.println(limite);
        cantPinias = cantidadPinas(limite, (String) datos[1]);
    }
    switch(o){
        /** Registra la tanda de acuerdo a los campos rellenados */
        case "registrar":
            System.out.println(id_Maguey + " <<<<->>> " + cantPinias);
            if(id_Maguey > 0 && cantPinias > 0){
                v.principal.setSelectedIndex(1);
                id_tipoMezcal = v.vProducir.tipo.getSelectedIndex() + 1;
                id_alcohol = v.vProducir.alcohol.getSelectedIndex() + 1;
                t = new Tanda(id_Maguey,id_alcohol,id_tipoMezcal, cantPinias);
                System.out.println(t);
                m.insertTanda(t);
                v.principal.setSelectedIndex(1);
                cargarDatosTandas();
```

```
                v.vRegistro.tabla.updateUI();
                v.vRegistro.tabla.revalidate();
            }else
                JOptionPane.showMessageDialog(v,"No rellenó correctamente");
            break;
        /** Elimina el registro de la tabla de tandas disponibles para producir */
        case "eliminar":
            filaPulsada = v.vRegistro.tabla.getSelectedRow();
            if (filaPulsada >= 0) {
                t = new Tanda();
                id_tanda = Integer.parseInt((String) v.vRegistro.mtt.getValueAt(filaPulsada, 0));
                t.setId(id_tanda);
                int respuesta = JOptionPane.showConfirmDialog(v, "¿Está seguro de eliminar la tanda con id: " + t.getId());
                if(!tandasProduciendo.contains(id_tanda) && respuesta == JOptionPane.YES_OPTION){
                    m.deleteTanda(t);
                    cargarDatosTandas();
                    v.vRegistro.tabla.updateUI();
                    JOptionPane.showMessageDialog(v,"Eliminó a tanda con id " + t.getId() );
                }else if(respuesta == JOptionPane.YES_OPTION)
                    JOptionPane.showMessageDialog(v, "No lo puede eliminar, está en produccion");
                else
                    JOptionPane.showMessageDialog(v,"Tanda no eliminada...");
            }
            break;
        case "producir":
```

```
            filaPulsada = v.vRegistro.tabla.getSelectedRow();
            if (filaPulsada >= 0) {
                id_tanda = Integer.parseInt((String) v.vRegistro.mtt.getValueAt(filaPulsada, 0));
                t = new Tanda();
                t.setId(id_tanda);
                Tanda tandaProducir = m.selectTanda(t);
                if(!tandasProduciendo.contains(t.getId())){
                    tandasProduciendo.add(t.getId());
                    m.updatePinias(tandaProducir);
                    bft.put(tandaProducir);
                    tandaProducir.setEstado("Produciendo");
                    m.updateEstadoTanda(tandaProducir);
                    cargarDatosTandas();
                }else
                    JOptionPane.showMessageDialog(v, "Ya está en proceso esa tanda");
            }
            break;
        case "transportar":
            if(bb.isEmpty()){
                JOptionPane.showMessageDialog(v,"No hay barriles para transportar");
                return;
            }
            transportistas.stream()
                    .filter(transportista -> !transportista.isAlive())
                    .forEach(Thread::start);
            break;
```

```
            case "salir":
                if(hayProduccion()){
                    JOptionPane.showMessageDialog(v,"NO PUEDE ABANDONAR, HAY MEZCALES PRODUCIENDOSE");
                    return;
                }
                m.cerrarConexion();
                ejecutador.shutdown();
                System.exit(0);
                break;
        }
    }


    /**
     * Carga información a la tabla de TANDAS DISPONIBLES.
     */
    public void cargarDatosTandas(){
        ColorearFilas c = new ColorearFilas(5);
        String consultaTandas = "select * from mezcal.tanda where status != 'Entregada'";
        /**
         *  Se lo dejamos al Thread del despachador de eventos
         *  para que la actualización de la tabla sea instantanea.
         */
        SwingUtilities.invokeLater(() ->{
            v.vRegistro.mtt.setDatos(m.conexionConsultaTanda(consultaTandas));
            v.vRegistro.tabla.getColumnModel().getColumn(5).setCellRenderer(c);
            v.vRegistro.tabla.updateUI();
            v.vRegistro.updateUI();
```

```
        });
    }

    /**
     * Carga la informacion de las tandas a la ultima tabla.
     *
     */
    private void cargarInformeTandas() {
        v.vInforme.mti.setDatos(m.conexionConsultaInformeTanda());
        v.vInforme.tabla.updateUI();
    }

    /**
     * Solicita la cantidad de piñas y las valida para el maguey dado.
     *
     * @param limite Cantidad maxima de piñas a usar.
     * @param nombreMaguey Nombre del maguey.
     */
    private int cantidadPinas(int limite, String nombreMaguey){
        int cantidad = 0;
        boolean op = false;
        while (!op) {
            try{
                String msj = JOptionPane.showInputDialog(v, "Introduce la cantidad de piñas a usar del maguey " +nombreMaguey);
                if(msj != null){
                    if (msj.isEmpty())
                        msj = "0";
                        cantidad = Integer.parseInt(msj);
```

```
                if(cantidad > limite)
                    new Exception("Limite pasado");
                else
                    op = true;
            }
            op = true;
        }catch(NumberFormatException e){
            System.out.println(e.toString());
        }
    }
    return cantidad;
}

/**
 * Valida si hay tandas pendientes en producción o traslado.
 *
 */
private boolean hayProduccion(){
    return !tandasProduciendo.isEmpty() || !tandasTransportando.isEmpty();
}

/**
 * Inicia todos los equipos (Hilos)
 */
public void IniciarEquipos(){
    for (int i = 0; i < 3; i++) {
        ejecutador.submit(cortes.get(i));
        ejecutador.submit(hornos.get(i));
```

```
            ejecutador.submit(molinos.get(i));
            ejecutador.submit(fermentadores.get(i));
            ejecutador.submit(destiladores.get(i));
            ejecutador.submit(enbotelladores.get(i));
        }
        ejecutador.submit(new MiHilo(TANDAS_ACTUALIZAR));
        ejecutador.submit(new HiloVista());
    }

    /**
     * Método que permite preparar los equipos
     */
    private void prepararEquipos(){
        int id_Equipo = 0;
        for (int i = 0; i < 3; i++) {
            id_Equipo = i + 1;
            cortes.add(new Cortador(id_Equipo, bft, bpc));
            hornos.add(new Horno(id_Equipo, bpc, bph));
            molinos.add(new Molino(id_Equipo , bph, bpm));
            fermentadores.add(new Fermentador(id_Equipo, bpm, bpf));
            destiladores.add(new Destilador(id_Equipo, bpf, bmd));
            enbotelladores.add(new Enbotelladora(id_Equipo, bmd, bb));
            transportistas.add(new Transportista(id_Equipo, bb, v.vTraslado));
            cortes.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(0));
            hornos.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(1));
            molinos.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(2));

fermentadores.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(3));
```

```
destiladores.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(4));

enbotelladores.get(i).setBarraIdentificador(v.vProduccion.getBarra(i).getPos(5));
            cortes.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            hornos.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            molinos.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            fermentadores.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            destiladores.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            enbotelladores.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);

((Enbotelladora)enbotelladores.get(i)).setTandasTransportar(tandasTransportando
);
            transportistas.get(i).setTandasActualizar(TANDAS_ACTUALIZAR);
            transportistas.get(i).setTandasTransportadas(tandasTransportando);
        }
    }


    /**
     * Clase que te permite hacer la actualización de las tablas cada que cambien de
estado la tanda dada.
     */
    class MiHilo extends Thread{


        // Variable de instancia - Tandas a actualizar.
        private BufferTandas tandas_actualizar;


        /**
         * Constructor para objetos de MiHilo.
         * @param tandas_actualizar
```

```
     */
    public MiHilo(BufferTandas tandas_actualizar){
        this.tandas_actualizar = tandas_actualizar;
    }

    public void run(){
        for(;;) {
            Tanda t = this.tandas_actualizar.remove();
            if(t != null){
                m.updateEstadoTanda(t);
                cargarDatosTandas();
                if(t.getEstado().equals("Entregada")) {
                    tandasProduciendo.remove(Integer.valueOf(t.getId()));
                    tandasTransportando.remove(Integer.valueOf(t.getId()));
                    System.out.println(tandasProduciendo.toString());
                    cargarInformeTandas();
                    cargarDatosTandas();
                    }
                }
         }
    }
}

/**
 * Clase que nos permitirá ajustar el tamaño de las ventanas al instante.
 */
class HiloVista extends  Thread{
```

```
        // Variable de instancia - Determina si estás en la vista o no.
        private boolean aquiYa = false;

        public void run(){
            int op = 0, aux = 0;
            while (true) {
                op = v.principal.getSelectedIndex();
                if ((op == 1 || op == 2 || op == 3 || op == 4) && !aquiYa) {
                    v.setSize(1020, 725);
                    v.setLocationRelativeTo(null);
                }else if (op == 0 && !aquiYa) {
                    v.setSize(770, 700);
                    v.setLocationRelativeTo(null);
                }while(aux == op){
                    aquiYa = true;
                    aux = v.principal.getSelectedIndex();
                    System.out.print("");
                }
                aquiYa = false;
                aux = op;
            }
        }
    }
}
```

Tanda.java

```
package Procesos;

import java.awt.Color;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;

/**
 * Clase para representar una Tanda de producción.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class Tanda {

    // Variable de instancia - Array de piñas.
    private ArrayList<Producto> pinias;

    // Variable de instancia - Identificadores de cada equipo.
    private int  id, tipoMaguey, porcentajeAlcohol, tipoMezcal, cantidadPinias,
                 id_Cortador, id_Horneador, id_Triturador, id_Fermentador,
                 id_Destilador, id_Enbotelladora, id_Transportador, id_Cliente;

    // Variable de instancia - Color que la identifica.
    private Color color;
```

```
    // Variable de clase - Identificador.
    private static int id_Contador = 1;

    // Variable de instancia - Fechas de producción de la tanda.
    private Timestamp fechaInicio, fechaFinal;

    // Variable de instancia - Estado de la tanda.
    private String estado;

    /**
     * Constructor para objetos de Tanda.
     */
    public Tanda(){

    }

    /**
     * Constructor para objetos de Tanda.
     *
     * @param tipoMaguey Valor de la FK para el tipo de maguey.
     * @param porcentajeAlcohol Valor de la FK para el porcentaje de alcohol.
     * @param tipoMezcal Valor de la FK para el tipo de mezcal.
     * @param cantidadPinias Cantidad de piñas que tendrá la tanda.
     */
    public Tanda(int tipoMaguey, int porcentajeAlcohol, int tipoMezcal, int
cantidadPinias) {
        this.tipoMaguey = tipoMaguey;
        this.porcentajeAlcohol = porcentajeAlcohol;
        this.tipoMezcal = tipoMezcal;
```

```
    this.cantidadPinias = cantidadPinias;
    //id = id_Contador++;
    generarColor();
    generarPinias();
    estado = "Registrada";
}

/**
 * Genera un color de tono claro para identificar la tanda.
 */
private void generarColor(){
    /** Generamos un color, jamás será el negro*/
    Random ran = new Random();
    float r = (float) (ran.nextFloat() / 2f + 0.5);
    float g = (float) (ran.nextFloat() / 2f + 0.5);
    float b = (float) (ran.nextFloat() / 2f + 0.5);
    color = new Color(r, g, b);
}

/**
 * Genera objeto de las piñas que se utilizarán.
 */
private void generarPinias(){
    pinias = new ArrayList();
    for (int i = 0; i < cantidadPinias ; i++)
        pinias.add(new Pinia(tipoMaguey));
}
```

```
/**
 * Establece la fecha de inicio de producción.
 *
 * @param fechaInicio Fecha a establecer.
 */
public void setFechaInicio(Timestamp fechaInicio){
    this.fechaInicio = fechaInicio;
}

/**
 * Retorna la fecha en que se inición la producción.
 *
 * @return Fecha de inicio.
 */
public Date getFechaInicio(){
    return fechaInicio;
}

/**
 * Establce la fecha final de la producción.
 *
 * @param fechaFinal Fecha a establecer.
 */
public void setFechaFinal(Timestamp fechaFinal){
    this.fechaFinal = fechaFinal;
}

/**
```

```
 * Retorna la fecha final de la producción.
 *
 * @return Fecha final.
 */
public Date getFechaFinal(){
    return fechaFinal;
}

/**
 * Establece el id a la tanda.
 *
 * @param id Valor a establecer.
 */
public void setId(int id){
    this.id = id;
}

/**
 * Retorna el id de la tanda.
 *
 * @return Identificador único de la tanda.
 */
public int getId(){
    return id;
}

/**
 * Retorna el identificador del tipo de maguey.
```

```
     *
     * @return Tipo de maguey.
     */
    public int getTipoMaguey() {
        return tipoMaguey;
    }

    /**
     * Retorna el array de las piñas.
     *
     * @return Array de pinias.
     */
    public ArrayList<Producto> getPinias() {
        return pinias;
    }

    /**
     * Retorna el valor del porcentaje de alcohol.
     *
     * @return Porcentaje de alcohol.
     */
    public int getPorcentajeAlcohol() {
        return porcentajeAlcohol;
    }

    /**
     * Retorna el identificador del tipo de mezcal.
     *
```

```
 * @return Tipo de mezcal.
 */
public int getTipoMezcal() {
    return tipoMezcal;
}

/**
 * Retorna la cantidad de piñas que posee la tanda.
 *
 * @return Cantidad de piñas.
 */
public int getCantidadPinias() {
    return cantidadPinias;
}

/**
 * Retorna el color que identifica a la tanda.
 *
 * @return Color de la tanda.
 */
public Color getColor(){
    return color;
}

/**
 * Establece el estado a la tanda.
 *
 * @param estado Estado a establecer.
```

```
 */
public void setEstado(String estado){
    this.estado = estado;
}

/**
 * Retorna el estado en el que se encuentre la tanda.
 *
 * @return Estado actual de la tanda.
 */
public String getEstado(){
    return estado;
}

/**
 * Representación de la tanda.
 *
 * @return Datos basicos de la tanda.
 */
@Override
public String toString() {
    return "Tanda{" +
            "pinias=" + pinias +
            ", id=" + id +
            ", tipoMaguey=" + tipoMaguey +
            ", porcentajeAlcohol=" + porcentajeAlcohol +
            ", tipoMezcal=" + tipoMezcal +
            ", cantidadPinias=" + cantidadPinias +
```

```
            ", id_Cortador=" + id_Cortador +
            ", id_Horneador=" + id_Horneador +
            ", id_Triturador=" + id_Triturador +
            ", id_Fermentador=" + id_Fermentador +
            ", id_Destilador=" + id_Destilador +
            ", id_Enbotelladora=" + id_Enbotelladora +
            ", id_Transportador=" + id_Transportador +
            ", id_Cliente=" + id_Cliente +
            ", color=" + color +
            ", fechaInicio=" + fechaInicio +
            ", fechaFinal=" + fechaFinal +
            ", estado='" + estado + '\'' +
            '}';
    }

    //@Override
    //public String toString() {
      //  return "Tanda{" + "tipoMaguey=" + tipoMaguey + ", porcentajeAlcohol=" + porcentajeAlcohol + ", tipoMezcal=" + tipoMezcal + ", cantidadPinias=" + cantidadPinias + '}';
    //}

    /**
     * Retorna el identificador del cortador que trabajó la tanda.
     *
     * @return Valor del identificador.
     */
    public int getId_Cortador() {
        return id_Cortador;
```

```
}

/**
 * Establece el identificador del cortador que trabajó la tanda.
 *
 * @param id_Cortador Valor del identificador.
 */
public void setId_Cortador(int id_Cortador) {
    this.id_Cortador = id_Cortador;
}

/**
 * Retorna el identificador del horneador que trabajó la tanda.
 *
 * @return Valor del identificador.
 */
public int getId_Horneador() {
    return id_Horneador;
}

/**
 * Establece el identificador del horneador que trabajó la tanda.
 *
 * @param id_Horneador Valor del identificador.
 */
public void setId_Horneador(int id_Horneador) {
    this.id_Horneador = id_Horneador;
}
```

```
/**
 * Retorna el identificador del triturador que trabajó la tanda.
 *
 * @return Valor del identificador.
 */
public int getId_Triturador() {
    return id_Triturador;
}

/**
 * Establece el identificador del triturador que trabajó la tanda.
 *
 * @param id_Triturador Valor del identificador.
 */
public void setId_Triturador(int id_Triturador) {
    this.id_Triturador = id_Triturador;
}

/**
 * Retorna el identificador del fermentador que trabajó la tanda.
 *
 * @return Valor del identificador.
 */
public int getId_Fermentador() {
    return id_Fermentador;
}
```

```
/**
 * Establece el identificador del fermentador que trabajó la tanda.
 *
 * @param id_Fermentador Valor del identificador.
 */
public void setId_Fermentador(int id_Fermentador) {
    this.id_Fermentador = id_Fermentador;
}

/**
 * Retorna el identificador del destilador que trabajó la tanda.
 *
 * @return Valor del identificador.
 */
public int getId_Destilador() {
    return id_Destilador;
}

/**
 * Establece el identificador del destilador que trabajó la tanda.
 *
 * @param id_Destilador Valor del identificador.
 */
public void setId_Destilador(int id_Destilador) {
    this.id_Destilador = id_Destilador;
}

/**
```

```
     * Retorna el identificador de la enbotelladora que trabajó la tanda.
     *
     * @return Valor del identificador.
     */
    public int getId_Enbotelladora() {
        return id_Enbotelladora;
    }

    /**
     * Establece el identificador de la enbotelladora que trabajó la tanda.
     *
     * @param id_Enbotelladora Valor del identificador.
     */
    public void setId_Enbotelladora(int id_Enbotelladora) {
        this.id_Enbotelladora = id_Enbotelladora;
    }

    /**
     * Establece el identificador del que transportó la tanda.
     *
     * @param id_Transportador Valor del identificador.
     */
    public void setId_Transportador(int id_Transportador){this.id_Transportador =
id_Transportador;}

    /**
     * Retorna el identificador del transportador de la tanda.
     *
     * @return Valor del identificador.
```

```
     */
    public int getId_Transportador(){return id_Transportador;}

    /**
     * Retorna el identificador del cliente que compró la tanda.
     *
     * @return Valor del identificador.
     */
    public int getId_Cliente() {
        return id_Cliente;
    }

    /**
     * Establece el identificador del cliente que compró la tanda.
     *
     * @param id_Cliente Valor del identificador.
     */
    public void setId_Cliente(int id_Cliente) {
        this.id_Cliente = id_Cliente;
    }
}
```

BufferTandas.java

```
package Procesos;

import java.util.LinkedList;
import java.util.Queue;
```

```
/**
 * Clase para mantener las tandas antes de pasarla a otro proceso.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class BufferTandas {

    // Variable de instancia - Almacenamiento de tandas.
    private Queue<Tanda> bufferTandas;

    /**
     * Constructor para objetos de BufferTandas.
     */
    public BufferTandas(){
        bufferTandas = new LinkedList();
    }

    /**
     * Almacena una tanda a la cola.
     * @param tanda Tanda a almacenar.
     */
    public synchronized void put(Tanda tanda){
        bufferTandas.add(tanda);
        notifyAll();
    }
```

```
    /**
     * Remueve la primera tanda de la cola.
     * @return Tanda de la cola.
     */
    public synchronized Tanda remove(){
        while (isEmpty()) {
            System.out.println("Espera tanda inicial ::: " +
Thread.currentThread().getName());
            try {
                wait();
            } catch (InterruptedException ex) {
                System.out.println("Error esperando tanda inicial ::: " + ex.getCause());
            }
        }
        notifyAll();
       return bufferTandas.remove();
    }

    /**
     * Verifica si el buffer está vacio.
     *
     * @return tre si está vacio y false de lo contario.
     */
    public boolean isEmpty(){
        return bufferTandas.isEmpty();
    }
}
```

Conexion.java

```
package Modelo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;

/**
 * Clase Conexion para conectar con la BD utilizando el patrón de diseño Singleton.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
 * @version 1.0 14/06/2020
 */
public class Conexion {

    // Variable de clase - Contenida en el pquete SQL.
    private static Connection coneccion = null; //

    // Variable de calse - Instancia a utilizar.
    private static Conexion conexion = null;

    // Variable de clase - Número de conexiones.
    private static  int numConexiones = 0;

    /**
     * Constructor para un único objeto de la clase
     * @param password Contraseña de la BD
```

```
     * @param url Dirreción de la BD
     * @param usuario Usuario de la BD
     */
    private Conexion(String url, String usuario, String password){
        try {
            Class.forName("org.postgresql.Driver");
            coneccion = DriverManager.getConnection(url,usuario, password);
        } catch (SQLException e) {

java.util.logging.Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,
null,e);
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }


    /**
     * Método de la clase para retornar una instancia de la misma.
     * @param password Contraseña de la BD
     * @param url Dirreción de la BD
     * @param usuario Usuario de la BD
     *
     * @return instancia de la clase.
     */
    public static Conexion getConexion(String url, String usuario, String password){
        numConexiones++;
        if(conexion == null)
            return conexion = new Conexion(url, usuario, password);
```

```
   return conexion;
}

/**
 * Retorna un objeto de Conecccction.
 *
 * @return instancia de Conecction.
 */
public static Connection getConeccion(){
    return coneccion;
}

/**
 * Método para cerrar la conexión con la BD
 *
 * return true si fue cerrada correctamente y
 *        false de lo contrario.
*/
public boolean CerrarConexion(){
   try {
       if (coneccion != null)
         if(numConexiones == 1){
            coneccion.close();
             System.out.println("Conexion cerrada.");
            return true;
       }else
           numConexiones--;
       return false;
```

```
        } catch (SQLException e) {
            System.err.println(" Error al tratar de cerrar la conexion " + e);
        }
        return false;
    }
}
```

ManejoDatos.java

```
package Modelo;

import Procesos.Tanda;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase que controla las consultas requeridas para el proyecto.
 * @author García García José Ángel
 * @author Sánchez Chávez Kevin Edilberto
```

```
 * @version 1.0 14/06/2020
 */
public class ManejoDatos {

    // Variable de instancia - Acceso a conexion.
    private Connection conexion;

    // Variable de instancia - Crear conexion a la BD.
    private Conexion crearConexion;

    // Variables de instancia - Atributos de la BD
    private String host = "localhost";
    private String usuario = "postgres";
    private final String clave = "Dexter1998";
    private int puerto = 5432;
    private String baseDatos = "Mezcalera";

    /**
     * Constructor para objetos de ManejoDatos
     */
    public ManejoDatos() {
        try {
            crearConexion = crearConexion.getConexion("jdbc:postgresql://" + host + ":"
+ puerto
                    + "/" + baseDatos, usuario, clave);
            conexion = crearConexion.getConeccion();
        } catch (Exception e) {
            System.out.println(e.getCause());
        }
```

```
    System.out.println("Conectado a " + baseDatos);
}

/**
 * Consulta un campo especifico de una tabla.
 *
 * @param sql Instrucción sql a ejecutar.
 */
private Object selectValueDe(String sql){
    PreparedStatement ps;
    Object dato = null;
    try {
        Statement st = conexion.createStatement();
        ResultSet rs = st.executeQuery(sql);
        dato =  (rs.next()) ? rs.getObject(1) : null;
    } catch (SQLException e) {
        System.out.println("Error al obtener dato \n " + e);
    }
    return dato;
}

/**
 * Retorna los datos de un maguey.
 *
 * @param id Id del maguey.
 */
public Object[] selectMaguey(int id) {
    PreparedStatement ps;
```

```
        Object datos[] = new Object[3];
        try {
            Statement st = conexion.createStatement();
            ResultSet rs = st.executeQuery("select * from mezcal.maguey where id_maguey=" + id);
            if(rs.next()){
                datos[0] = rs.getInt(1);
                datos[1] = rs.getString(2);
                datos[2] = rs.getInt(3);
            }
        } catch (SQLException e) {
            System.out.println("Error al consultar las piñas del maguey " + id + " \n " + e);
        }
        return datos;
    }

    /**
     * Retorna campos especificos de la tabla tanda.
     *
     * @param sql Instrucción sql a ejecutar.
     * @return Datos especificos de la tanda.
     */
    public List<Object[]> conexionConsultaTanda(String sql) {
        PreparedStatement ps;
        ResultSet rs;
        List<Object[]> datos = new ArrayList<Object[]>();
        try {
            ps = conexion.prepareStatement(sql);
```

```
            rs = ps.executeQuery();
            while (rs.next()) {
                String dat[] = new String[6];
                dat[0] = String.valueOf((Integer) rs.getInt(1));
                dat[1] = (String) selectValueDe("select nombre from mezcal.maguey where id_maguey = " + rs.getString(2));
                dat[2] = String.valueOf((Double)selectValueDe("select valor from mezcal.gradoalcohol where id_grado = " + rs.getInt(3)));
                dat[3] = (String) selectValueDe("select nombre from mezcal.tipomezcal where id_tipo = " + rs.getInt(4));
                dat[4] = String.valueOf((Integer) rs.getInt(5));
                dat[5] = rs.getString(6);
                datos.add(dat);
            }
        } catch (SQLException e) {
            // System.err.println("Error al CARGAR DATOS " + e);
        }
        return datos;
    }

    /**
     * Método par aobtener todos los campos de la tanda.
     *
     * return Lista con los datos de las tandas.
     */
    public List<Object[]> conexionConsultaInformeTanda() {
        PreparedStatement ps;
        ResultSet rs;
        List<Object[]> datos = new ArrayList<Object[]>();
```

```
try {
    ps = conexion.prepareStatement("select * from mezcal.tanda where status = 'Entregada'");
    rs = ps.executeQuery();
    while (rs.next()) {
        String dat[] = new String[15];
        dat[0] = String.valueOf((Integer) rs.getInt(1));
        dat[1] = (String) selectValueDe("select nombre from mezcal.maguey where id_maguey = " + rs.getString(2));
        dat[2] = String.valueOf((Double)selectValueDe("select valor from mezcal.gradoalcohol where id_grado = " + rs.getInt(3)));
        dat[3] = (String) selectValueDe("select nombre from mezcal.tipomezcal where id_tipo = " + rs.getInt(4));
        dat[4] = String.valueOf((Integer) rs.getInt(5));
        dat[5] = (String) selectValueDe("select nombre from mezcal.cortador where id_cortador = " + rs.getInt(7));
        dat[6] = (String) selectValueDe("select nombre from mezcal.horno where id_horno = " + rs.getInt(8));
        dat[7] = (String) selectValueDe("select nombre from mezcal.molino where id_molino = " + rs.getInt(9));
        dat[8] = (String) selectValueDe("select nombre from mezcal.fermentador where id_fermentador = " + rs.getInt(10));
        dat[9] = (String) selectValueDe("select nombre from mezcal.destilador where id_destilador = " + rs.getInt(11));
        dat[10] = (String) selectValueDe("select nombre from mezcal.enbotelladora where id_enbotelladora = " + rs.getInt(12));
        dat[12] = (String) selectValueDe("select nombre from mezcal.cliente where id_cliente = " + rs.getInt(13));
        SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
        dat[13] = (rs.getTimestamp(14) == null) ? "Null" : sdf.format(rs.getTimestamp(14));
```

```
            dat[14] = (rs.getTimestamp(15) == null) ? "Null" :
sdf.format(rs.getTimestamp(15));
            dat[11] = (String) selectValueDe("select nombre from mezcal.transportista
where id_transportista = " + rs.getInt(16));
            datos.add(dat);
          }
       } catch (SQLException e) {
         System.err.println("ERROR AL CARGAR DATOS DE INFORME " +
e.getCause());
       }
       return datos;
   }

   /**
    * Insertar un registro de tanda.
    *
    * @param t Tnada a insertar
    * @return true si se insertó correctamente y
    *         false de lo contrario.
    */
   public boolean insertTanda(Tanda t) {
      PreparedStatement ps;
      String sqlInsertTanda = "insert into mezcal.tanda
(tipomaguey,gradoAlcohol,tipoMezcal,cantidadPinias"
            + ",status,fecha_inicio) values (?,?,?,?,?,?);";
      try {
         ps = conexion.prepareStatement(sqlInsertTanda);
         ps.setInt(1, t.getTipoMaguey());
         ps.setInt(2, t.getPorcentajeAlcohol());
         ps.setInt(3, t.getTipoMezcal());
```

```
            ps.setInt(4, t.getCantidadPinias());
            ps.setString(5, t.getEstado());
            ps.setDate(6, (Date) t.getFechaInicio());
            ps.executeUpdate();
            return true;
        } catch (SQLException e) {
            System.err.println("Error en la INSERCIÓN " + e.getMessage());
            return false;
        }
    }

    /**
     * Consulta el nombre de una de las tablas maguey, cliente.
     *
     * @param sql Instrucción a ejecutar.
     * @return Array con los datos los registros.
     */
    public ArrayList<String> conexionConsultarNombre(String sql) {
        ArrayList<String> datos = new ArrayList<>();
        try {
            Statement ps = conexion.createStatement();
            ResultSet rs = ps.executeQuery(sql);
            while (rs.next())
                datos.add(rs.getString(2));
        } catch (SQLException e) {
            System.out.println("ERROR AL OBTENER NOMBRE: " + e.getCause());
        }
        return datos;
```

```
}

/**
 *  Eliminar una tanda con id dado.
 *
 * @param t Tanda con ID dado
 * @return true si se eliminó correctamente y
 *         false de lo contrario.
 */
public boolean deleteTanda(Tanda t) {
    PreparedStatement ps;
    String sqlDeleteCliente = "delete from mezcal.tanda where id_tanda = ?;";
    try {
        ps = conexion.prepareStatement(sqlDeleteCliente);
        ps.setInt(1, t.getId());
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        System.err.println("Error en el BORRADO " + e);
        return false;
    }
}

/**
 * Retorna la tanda seleccionada.
 *
 * @param t Tanda seleccionada.
 * @return Tanda con los datos de la seleccionada.
```

```
 */
public Tanda selectTanda(Tanda t){
    PreparedStatement ps;
    ResultSet rs;
    Tanda tandaEncontrada = null;
    String sqlConsulta = "select * from mezcal.tanda where id_tanda = ?;";
    try{
        ps  = conexion.prepareStatement(sqlConsulta);
        ps.setInt(1,t.getId());
        rs  = ps.executeQuery();
        if(rs.next()){
            tandaEncontrada = new Tanda(rs.getInt(2),rs.getInt(3),rs.getInt(4),rs.getInt(5));
            tandaEncontrada.setId(rs.getInt(1));
            tandaEncontrada.setEstado(rs.getString(6));
            tandaEncontrada.setId_Cortador(rs.getInt(7));
            tandaEncontrada.setId_Horneador(rs.getInt(8));
            tandaEncontrada.setId_Triturador(rs.getInt(9));
            tandaEncontrada.setId_Fermentador(rs.getInt(10));
            tandaEncontrada.setId_Destilador(rs.getInt(11));
            tandaEncontrada.setId_Enbotelladora(rs.getInt(12));
            tandaEncontrada.setId_Cliente(rs.getInt(13));
            tandaEncontrada.setFechaInicio(rs.getTimestamp(14));
            tandaEncontrada.setFechaFinal(rs.getTimestamp(15));
        }
    }catch (SQLException e) {
        System.err.println("Error al CARGAR TANDA " + e);
    }
    return tandaEncontrada;
```

```
    }

    /**
     * Método para la modifiación de la cantidad de piñas.
     *
     * @param t Tanda que modifica la piñas.
     * @return true si se modificó correctemente y
     *         false de lo contrario.
     */
    public boolean updatePinias(Tanda t){
        PreparedStatement ps;
        String sqlUpdateMaguey = "update mezcal.maguey set cantidadPinia = ? where id_maguey = " + t.getTipoMaguey() + ";";
        try{
            ps  = conexion.prepareStatement(sqlUpdateMaguey);
            int valor = (int) selectValueDe("select cantidadPinia from mezcal.maguey where id_maguey = " + t.getTipoMaguey())
                    - t.getCantidadPinias();
            ps.setInt(1,valor);
            ps.executeUpdate();
            return true;
        }catch (SQLException e) {
            System.err.println("Error en la MODIFICACION de piñas \n" + e);
            return false;
        }
    }

    /**
     * Método para actualizar la BD
```

```
    *
    * @param t Tanda a actualizar.
    * @return true si se modificó correctamente
    *         false de lo contrario.
    */
   public boolean updateEstadoTanda(Tanda t){
      PreparedStatement ps;
      String sqlUpdateTanda = "";
      boolean completa = false;
      if (t.getEstado().equals("Entregada")) {
         sqlUpdateTanda = "update mezcal.tanda set "
            + "status = ?, id_Cortador = ?, id_Horno = ?, id_Molino = ?,"
            + "id_Fermentador = ?, id_Destilador = ?, id_Enbotelladora = ?"
            + ",id_Cliente = ?, id_Transportista = ?, fecha_inicio = ?, fecha_final = ?
where id_tanda = ?;";
         completa = true;
      }
      else
         sqlUpdateTanda = "update mezcal.tanda set status = ? where id_tanda = ?;";
      try{
         ps  = conexion.prepareStatement(sqlUpdateTanda);
         System.out.println(t);
         ps.setString(1,t.getEstado());
         if(completa){
            ps.setObject(2, t.getId_Cortador());
            ps.setObject(3, t.getId_Horneador());
            ps.setInt(4, t.getId_Triturador());
            ps.setInt(5, t.getId_Fermentador());
            ps.setInt(6, t.getId_Destilador());
```

```
            ps.setInt(7, t.getId_Enbotelladora());
            Timestamp d1 = new Timestamp(t.getFechaInicio().getTime());
            Timestamp d2 = new Timestamp(t.getFechaFinal().getTime());
            ps.setInt(8,t.getId_Cliente());
            ps.setInt(9,t.getId_Transportador());
            ps.setTimestamp(10, d1);
            System.out.println("La fecha de inicio es : " + d1 + "\nLA final es :"+ d2);
            ps.setTimestamp(11, d2);
            ps.setInt(12,t.getId());
        }else
            ps.setInt(2, t.getId());
        ps.executeUpdate();
        return true;
    }catch (SQLException e) {
        System.err.println("Error en la MODIFICACION de Estado de tanda \n" + e);
        return false;
    }
}

/**
 * Método para cerrar la conexión con la BD
 */
public void cerrarConexion(){
    crearConexion.CerrarConexion();
}
}
```

EVIDENCIA (IMÁGENES DE EJECUCIÓN)

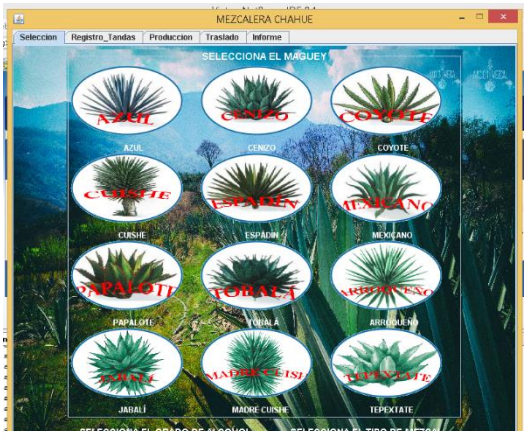


Ilustración 1 Podemos observar el inicio de la aplicación, además de los diferentes tipos de magueyes que manejamos. Así poder elegir los que queramos

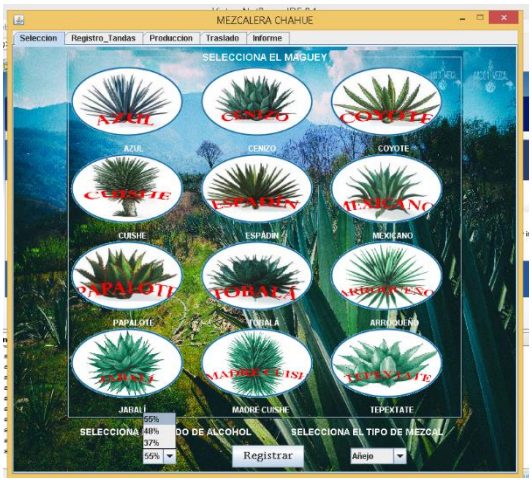


Ilustración 2 Vemos la opción de elegir el grado de alcohol que queramos, cabe mencionar que dependiendo el alcohol se demorara más en el proceso de la destilación.

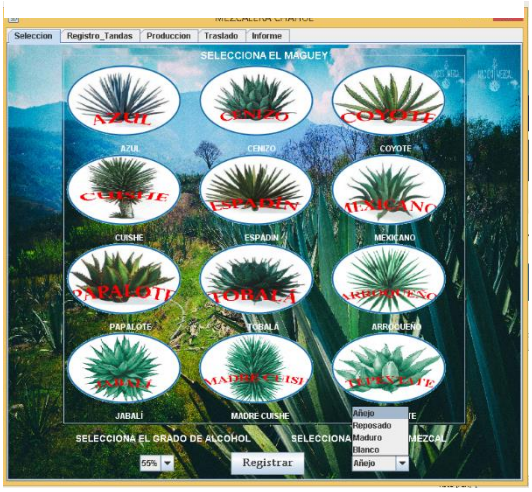


Ilustración 4 Ahora nos muestra el tipo de mezcal que queramos, teniendo en cuenta que dependiendo la elección en el embotellamiento tendrá que ejecutarse más veces para conseguir el tipo de mezcal deseado. No olvides registrar tu pedido

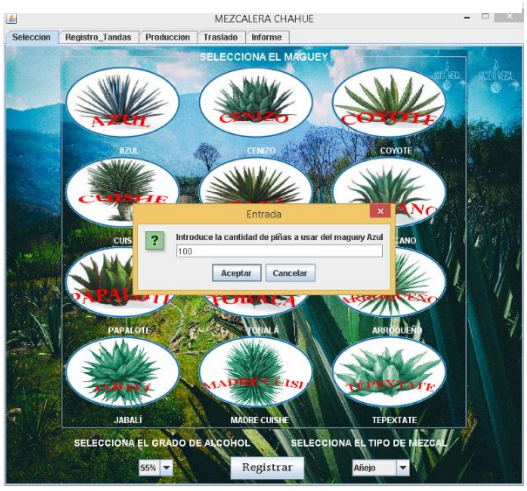


Ilustración 3 Como podemos observar, al momento de elegir algún maguey, podemos asignar las piñas que serán procesadas, entre más piñas más será la espera para obtener nuestro mezcal

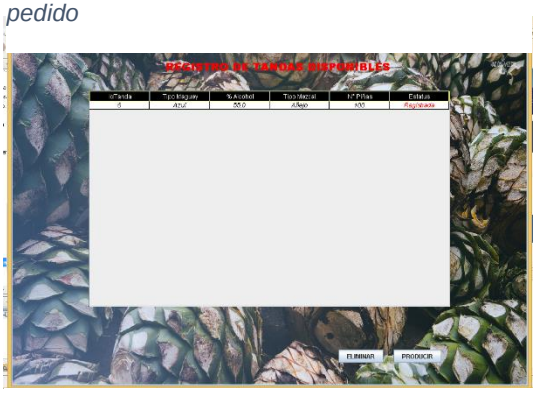


Ilustración 5 Cuando tengamos la cantidad de piñas a procesar, pasará a la siguiente pestaña para así ver en detalle nuestro pedido.

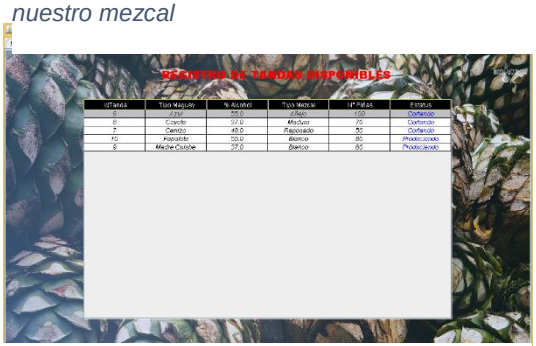


Ilustración 6 Cuando presionamos el botón "producir" empezará el proceso de nuestras piñas para obtener mezcal.

Ilustración 8 Aquí nos muestra los procesos que se llevan a cabo para obtener el mezcal, pasando por 6 procesos (el tiempo que tarda cada proceso, es logrado gracias a los hilos.) NOTA: cada proceso activa 3 hilos para que se pudiese observar con claridad. Además, de cada pedido contiene un color que lo diferencia de los demás.

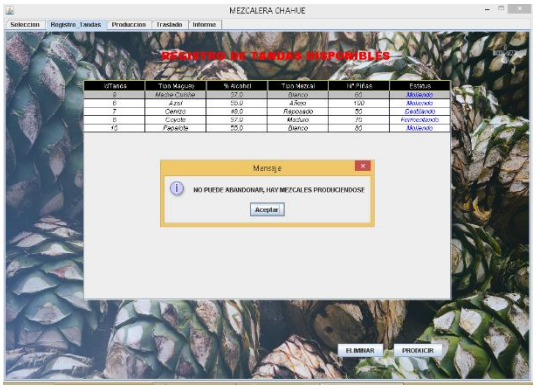


Ilustración 7 Para que el "cliente" no se salga de la aplicación mientras se esté llevando a cabo el proceso, decidimos poner validaciones, así evitamos que el cliente cierre el programa por error.

Ilustración 10 Con ayuda de los hilos, simulamos que el trailer realice una entrega especial a una de las sucursales.

Ilustración 9 Cada trailer se activa de manera aleatoria, como también el destino es una opción aleatoria.

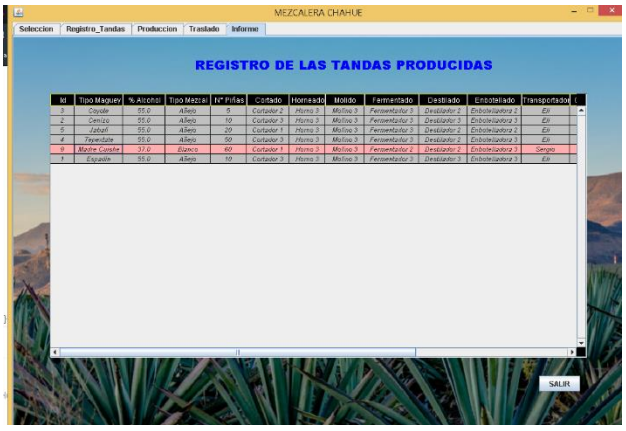


Ilustración 11 Tenemos un apartado especial para cuando el proceso termine, así tener un informe de lo que se está produciendo.