UNIVERSIDAD DE SONORA

División de Ingeniería

Departamento de Ingeniería Industrial

CAJERO AUTOMÁTICO

Memoria de Prácticas Profesionales

Que para obtener el título de

INGENIERO EN SISTEMAS DE INFORMACIÓN

Presenta

Alan Alberto Lerma Molina

Asesor

Dr. Gerardo Sánchez Schmitz

Hermosillo, Sonora. Diciembre 2019

Índice de contenido

1: Introducción **3**

1.1: Definición del proyecto 3

1.2: Objetivos del proyecto 3

1.3: Acerca de la empresa 3

1.3.1: Misión de LYF 4

1.3.2: Visión de LYF 5

1.3.3: Política de calidad de LYF 5

2: Estructura y desarrollo del proyecto. 5

2.1: Composición externa del cajero 5

2.2: Composición interna 5

2.2.1: Estructura para pago 6

2.2.2: Estructura Administrativa 9

2.3: Uso de pantallas 14

2.3.1: Pantalla Principal: MainWindow 14

2.3.2: User controls de apoyo 16

2.4: Clases de control 18

2.5: Clases de modelos 41

3: Análisis sobre el proyecto 42

3.1: Experiencia Adquirida 43

3.2: Análisis general del programa, su diseño, desarrollo y organización. 43

3.3: Conclusión final 43

4: Referencias **43**

Índice de Imágenes

Imagen 1.1: Logotipo de la empresa 3

Imagen 1.2: Vista de las oficinas LYF 3

Imagen 1.3: Vista de ubicación sacada de google maps 3

Imagen 2.1: User Control mostrado en Welcome View 5

Imagen 2.2: User Control mostrado al momento de ingresar póliza 5

Imagen 2.3: User control mostrado para revisar datos del usuario 6

Imagen 2.4: User control mostrado al elegir metodo de pago 6

Imagen 2.5: User control mostrado al pagar con efectivo 7

Imagen 2.6: User control mostrado al pagar con tarjeta 7

Imagen 2.7: User control mostrado al finalizar pago 8

Imagen 2.8: User control mostrado al momento de ingresar al panel administrativo 9

Imagen 2.9: User control mostrado al momento de hacer dotación. 9

Imagen 2.10: User control mostrado para agregar usuarios. 10

Imagen 2.11: User control mostrado para dar indicaciones de apertura permitida de puertas 11

Imagen 2.12; User control mostrado para dar los niveles de efectivo en el cajero 11

Imagen 2.13: User control mostrado para la reimpresión de tickets. 12

Imagen 2.14: Diagrama de Navegación de la aplicación 13

Imagen 2.14: Representación inicial de Welcome View. 15

Imagen 2.15: Representación de User control que ayuda a vaciar monedas 15

Imagen 2.16: Representación de User control que permite mandar mensajes en diferentes puntos. 16

Imagen 2.17: User control de pantalla de carga, el mensaje es dinámico, y puede tener mensajes secundarios con un tamaño de letra menor 16

Imagen 2.18: User control que muestra la cantidad de dinero y papel al cliente 17

Imagen 2.19: Representación visual que se usó para comprender los distintos casos 22

1.- Introducción.

Las prácticas profesionales son un requisito actual para poder obtener la titulación como alumno dentro de la institución (Universidad de Sonora), por lo cual, es muy importante completarlas en su totalidad. Estas tratan sobre aplicar conocimientos establecidos como alumno, de tal forma que se pueda constatar de forma fidedigna con un trato formal a una empresa que el susodicho tiene tales aptitudes.
Los requisitos que esta instancia profesional tiene que cumplir son el de al menos durar 340 horas, lo que se traducen a 20 créditos en la institución.

1.1.- Definición del proyecto

El proyecto tiene como objetivo el crear una aplicación con función de cajero automático, el cual tiene como función el poder hacer pagos o adeudos para una compañía de seguros en panamá.
El proyecto está siendo desarrollado en C# y Sqlite, haciendo usos de programas como Visual Studio, Visual Studio Code, Postman y herramientas de control de software como puede ser Gitlab, Jira y Trello.

1.2. - Objetivos del proyecto

El objetivo principal del proyecto es finalizar la aplicación cajero con todos los requerimientos establecidos por el cliente, teniendo como funcionalidades primordiales las transacciones monetarias y el actualizado de transacciones de una forma correcta.
Funcionalidades de uso Vital:

a) Feriado Correcto
b) Actualización de datos del cliente correcto en base de datos
c) Actualización de datos del cliente correcto en el servidor de servicio web
d) Lectura de usuarios tipo cliente
e) Funcionamiento estable a través de la pantalla
f) Cumplir con los estándares visuales del cliente

1.3.- Acerca de la empresa.

La empresa LYF es una empresa fundada en el año 2008 por Francisco Eduardo Lopez Lopez y Benny Fernandez Parada, los cuales son ingenieros en electrónica con especialidad en sistemas digitales, al recibir una petición de proyecto de CFE el cual se realizó de forma satisfactoria por lo cual se decidió a generar una empresa.

Imagen 1.1: Logotipo de la empresa

La empresa está ubicada en la calle calle congreso 384 en la colonia ley 57 con número telefónico de 6622603825

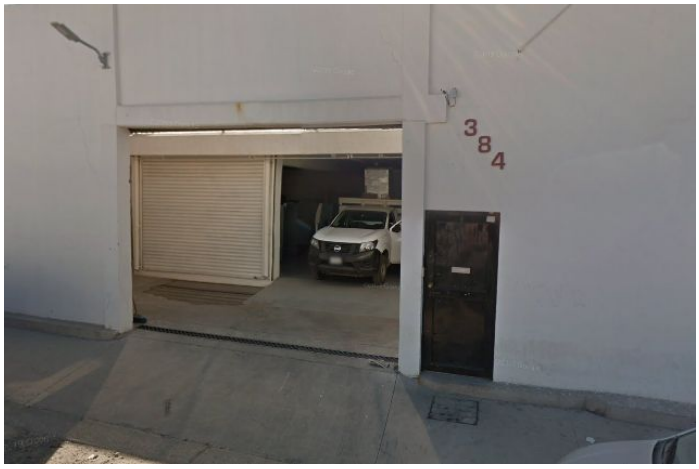

Imagen 1.2: Vista de las oficinas LYF

Imagen 1.3: Vista de ubicación sacada de google maps

1.3.1 Misión de LYF

"Crear soluciones enfocadas a la seguridad y eficiencia en procesos de manejo de dinero de nuestros clientes, implementando sistemas de gestión de calidad y mejora continua, mediante el desarrollo integral de nuestros colaboradores."

1.3.2 Visión de LYF

"Ser en el 2022 la empresa líder en automatización de procesos de manejo de dinero en México y América Latina."

1.3.3 Política de calidad de LYF

"En LYF INGENIERÍA nuestro compromiso es ofrecer las mejores soluciones del mercado para el manejo automatizado de valores, a través de la implantación de sistemas de Gestión de Calidad y Mejora Continua, cumpliendo estándares internacionales, requisitos aplicables y reglamentarios, con personal altamente competente."

2 Estructura y desarrollo del proyecto.

El desarrollo de la aplicación empezó a inicios de septiembre del 2019 y sigue en curso actualmente (mes de diciembre del año 2019), han existido ciertos factores que han hecho un atrasos directos al dia de entrega como la falta de accesorios físicos enviados por el cliente como es el uso del pin pad, que es totalmente ajeno a los dispositivos que otorga la empresa LYF, y la apertura de los servicios web, por lo que en cuanto a función y diseño, el tiempo muerto se ha utilizado para pulir y mejorar las funciones, actualmente el proyecto se encuentra un noventa por ciento terminado en base al número de “issues” generados en gitlab.

2.1 Composición externa del cajero

Composición externa es todo aquello que va ligado al hardware del cajero, es decir todos los dispositivos con los que se comunicara el cajero:

a) Contenedores de monedas para dispensación: llamados Hoppers, son especificados por el cliente, en este caso se pidieron tres contenedores, uno para monedas de 25 centavos americanos, 5 centavos americanos y 1 penny.
b) Contenedores de billetes para dispensación: llamados cassette, igualmente especificados por el cliente, en este caso dos de un dólar y uno de 10 dólares americanos.
c) Contenedores de retención: Son llamados cashbox y es donde son almacenadas las cantidades de dinero ingresadas por el cliente, existen de dos tipos, de moneda y de billetes.
d) Dispensador de monedas: Dispositivo el cual tiene como función principal el poder otorgar cambio al cliente, tomando en cuenta solamente monedas.
e) Dispensador de billetes: cumple con la función del dispositivo anterior, tomando en cuenta que solo dispensa billetes
f) Controlador principal: Cumple con la función de controlar diferentes características del cajero, como los sensores de las puertas de seguridad, las luces led, o la impresora.
g) Pantalla touch: Es la forma en que el cliente se comunicará con el servicio.

2.2. Composición interna:

Cuando se habla de la composición interna nos referimos a todo aquello que la aplicación deberá ser acreedora, es decir tanto pantallas visuales como lógica.

El proyecto tiene una estructura visual que fue dada por el cliente externo, la cual consta de las siguientes vistas:

2.2.1 Estructura para Pago

Estas pantallas son las que el cliente directo de la empresa contratadora usará, es decir, son aquellas por las cuales podrá generar transacciones, y son las pantallas de uso e interés más grande.

a) Pantalla de bienvenida: Es la pantalla default que se mostrará, la cual consta de una imagen de bienvenida, con la única instrucción de pulsar la pantalla para continuar, muestra fondo de pantalla proporcionado por el cliente:

Imagen 2.1: User Control mostrado en Welcome View

b) Pantalla de ingreso de usuario: aquí es donde el cliente ingresa su información, este puede ser mediante un lector de QR o manualmente, esta pantalla es mostrada después de la pantalla de bienvenida.

Imagen 2.2: User Control mostrado al momento de ingresar póliza

c) Pantalla de chequeo de datos: Esta pantalla es mostrada después de que se ingresa el usuario, muestra la deuda actual que tiene el cliente, y si desea continuar o salir, además, en esta misma pantalla se muestra la posibilidad de poder pagar otra cantidad.

Imagen 2.3: User control mostrado para revisar datos del usuario

d) Pantalla de forma de pago: Esta pantalla es mostrada después de la pantalla de chequeo de datos, y muestra dos posibilidades para pago a elegir por el cliente:

- pago en efectivo
- pago en Tarjeta

Imagen 2.4: User control mostrado al elegir metodo de pago

e) Pantalla de pago en efectivo: Pantalla mostrada en caso de elegir la opción a del punto anterior, en esta pantalla se mostrarán las diferentes denominaciones y las cantidades de efectivo que son ingresadas en tiempo real, además, al ser pagada la deuda, o al finalizar de forma manual, nos lleva a la siguiente pantalla.

Imagen 2.5: User control mostrado al pagar con efectivo

f) Pantalla de pago de tarjeta: Pantalla mostrada en caso de elegir la opción b, la cual debido a problemas con el cliente no está implementada, solo simulada.

Imagen 2.6: User control mostrado al pagar con tarjeta

g) Pantalla de pago terminado, en esta pantalla se muestra el ticket virtual, y el estatus del pago en base a los servicios web o en base la situación del pago, es decir, pago completo, pago incompleto, pago interrumpido, pago incompleto interrumpido, pago parcial, pago parcial interrumpido etc.
En esta pantalla se mostrará la opción de hacer otra transacción o terminar los servicios, redireccionando al usuario a diferentes pantallas dependiendo el caso, pudiendo ser la pantalla principal o la pantalla de ingreso de usuario.

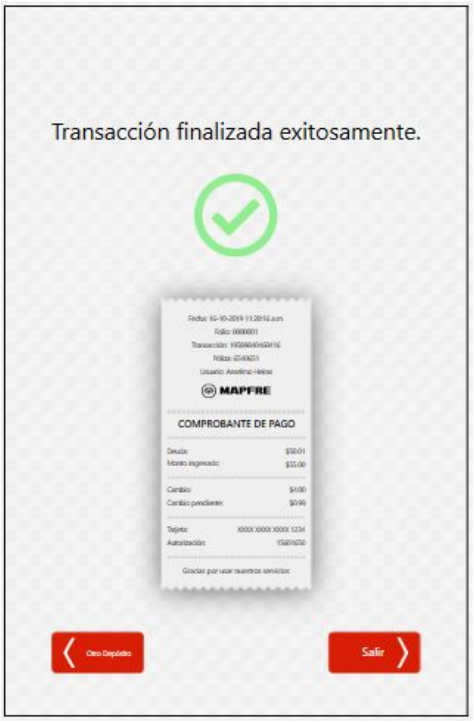

Imagen 2.7: User control mostrado al finalizar pago

2.2.2 Estructura Administrativa

Esta estructura incluye una libertad creativa mayor por los implicados en el proyecto más que por los clientes, y está dada en base a proyectos pasados, la cual fue creada en función de las necesidades que pueda tener el cliente.

a) Menú administrativo: es un menú el cual cuenta con ocho botones, siete de estos siendo opciones para el usuario administrador para dar servicio al cajero, los cuales son los siguientes:

Imagen 2.8: User control mostrado al momento de ingresar al panel administrativo

b) Dotación de efectivo: Conjunto de pantallas que ayudan a dotar dinero, agregando cantidades de efectivo en la base de datos para mantener controlado ingresado.

Imagen 2.9: User control mostrado al momento de hacer dotación.

c) Agregar usuarios Administrativos: Conjunto de operaciones que ayudan a agregar diferentes tipos de usuarios para el manejo de las pantallas administrativas, ya sea para dar servicio, agregar dinero, o inclusive agregar más usuarios, dando diferentes rangos de movilidad por el panel administrativo.

Imagen 2.10: User control mostrado para agregar usuarios.

d) Retiro de efectivo: Conjunto de pantallas que ayudan a retirar dinero y actualizarlo en la base de datos, tomando en cuenta solo el dinero ingresado por el cliente (contenedores de retención o cashbox)
e) Retiro total de efectivo: Cumple con una función similar al anterior punto, con la diferencia que este toma todo el efectivo actual en el cajero.
f) Servicio de cajero: pantalla que activa la posibilidad de manejo total físicamente del cajero, evitando que este se bloquee en caso de una violación, para poder dar mantenimiento o chequeo.

Imagen 2.11: User control mostrado para dar indicaciones de apertura permitida de puertas

g) Niveles de efectivo en dispensadores: pantalla que muestra niveles de dispensación, dando valores recomendados sobre qué cantidad de billetes se deben de tener dentro del cajero para estar en un funcionamiento óptimo (Seguro para operar).

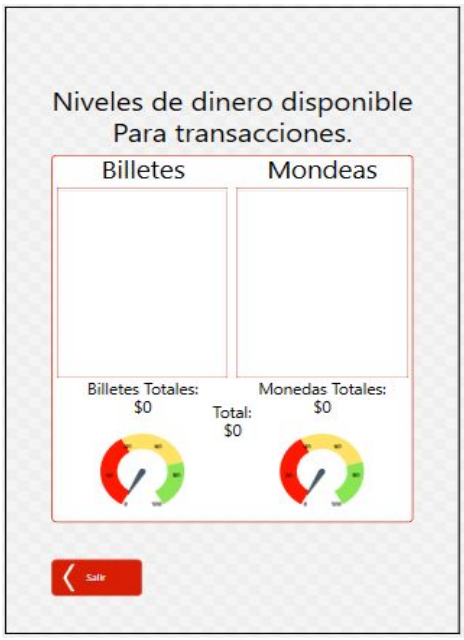

Imagen 2.12; User control mostrado para dar los niveles de efectivo en el cajero

h) Reimpresión de ticket: Pantalla que sirve para poder ver las transacciones pasadas y obtener una copia exacta de ticket ya impreso con anterioridad, con una marca de copia para evitar conflictos.

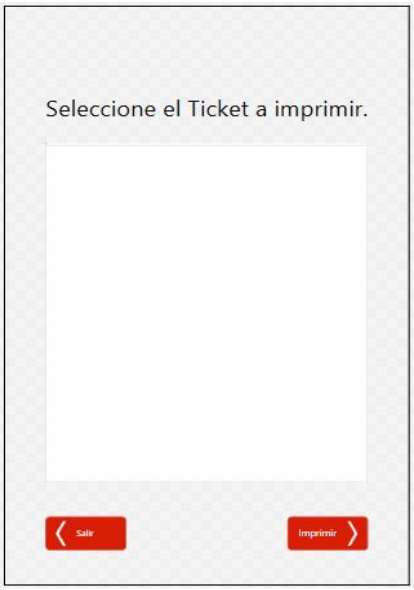

Imagen 2.13: User control mostrado para la reimpresión de tickets.

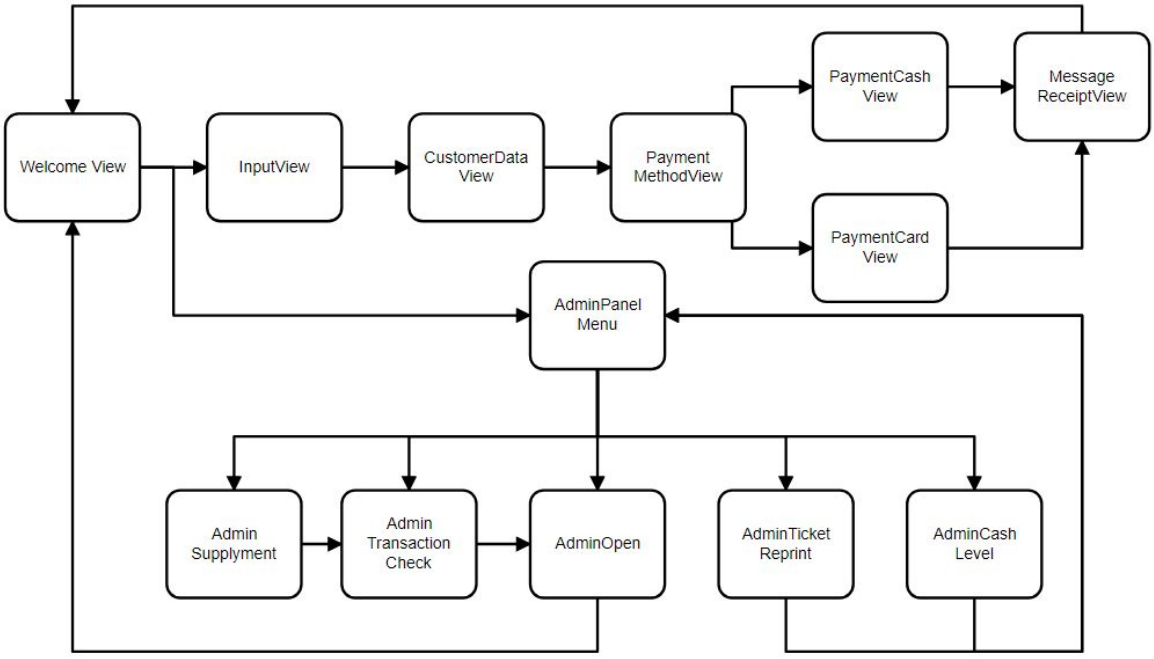

Imagen 2.14: Diagrama de Navegación de la aplicación

En la imagen 2.14 podemos ver el flujo de las pantallas de una forma gráfica, donde todo inicia en el Welcome View, y donde de forma opcional desde esa misma pantalla podemos entrar al Admin Panel Menú

2.3 Uso de pantallas

Las pantallas son una parte muy importante dentro de la aplicación, ya que además de llevar lógica en ellas, son la principal interacción con el usuario directo, es decir, si estas pantallas fallan la aplicación será inusable por lo tanto ya teniendo el diagrama de manejo de las pantallas, solamente fue cuestión de seguir la forma ya establecida y buscar ramificaciones de otros posibles casos que generen problemas.

2.3.1 Pantalla Principal: MainWindow

la forma en la que el proyecto funciona es con una pantalla principal, que es la base de todo lo mostrado en el cajero, la cual va alternando con diferentes user controls, de esta forma el proyecto logra ser más ligero y mejor optimizado, esta misma es llamada MainWindow tan solo incluye el fondo de pantalla otorgado por el cliente con sus respectivos marcos, es la única pantalla que tiene el tamaño correspondiente a la pantalla a usarse en el cajero, es decir una resolución 1920x1080, aquí se almacenan diferentes métodos los cuales pueden ser invocados en los demas user controls, a continuación se muestran los respectivos métodos:

a) GetContent: método utilizado para obtener de respuesta el user control que está almacenando en ese momento la pantalla MainWindow, útil para saber en qué pantalla está en momentos específicos

```
public UserControl GetContent()
        {
        if (ContentGrd.Children.Count <= 0) return null;
        return (UserControl)ContentGrd.Children[0];
        }
```

b) SetContent: método cuya principal función es cambiar el contenido de la vista base, es decir, modificar el user control activo, limpiando antes este para evitar problemas de solapamiento.

```
public void SetContent(UserControl control)
    {
        ContentGrd.Children.Clear();
        ContentGrd.Children.Add(control);
        CurrentControl = control.GetType();
```

```
}
```

c) UserControlSetter: se encarga de guardar la pantalla base en un objeto de tipo user Control con el fin de tener de respaldo la pantalla con toda la información guardada para interrupciones indebidas del cajero.

```
public void UserControlSetter()
    {
        var waiter = new ManualResetEvent(false);
        App.Window.Dispatcher.Invoke(() =>
        {
            ControlChildren = App.Window.GetContent();
            waiter.Set();
        });

        waiter.WaitOne();
        waiter.Close();
    }
```

d) CompareCurrentView: Se encarga de hacer una comparación de user controls, tomando en cuenta el user control actual con otro especificado, útil para ver de forma dinamica contenido dentro de la pantalla principal.

```
public bool CompareCurrentView<T>() {
        return CurrentControl == typeof(T);
    }
```

Imagen 2.14: Representación inicial de Welcome View.

2.3.2 User controls de apoyo

A lo largo del proyecto se usaron diferentes vistas que no están dentro del flujo principal pero que sirven de apoyo, es decir son un paso intermedio o son pantallas que pueden ser llamadas un sin fin de veces pero que no dependen de un flujo específico.

AdminDrain: Esta pantalla es de apoyo para el retiro total ya que para vaciar las monedas totalmente se necesita de ayuda de software, por lo que de esta forma se aseguran de poder dejar todas las cantidades de cero.

Imagen 2.15: Representación de User control que ayuda a vaciar monedas

MessageView: Pantalla que fue creada para mostrar mensajes específicos bajo ciertas condiciones, como errores en el sistema o indicaciones, la cual simplemente al ser llamada se debe indicar a qué pantalla regresará una vez se presione un botón y el respectivo mensaje a mostrar.

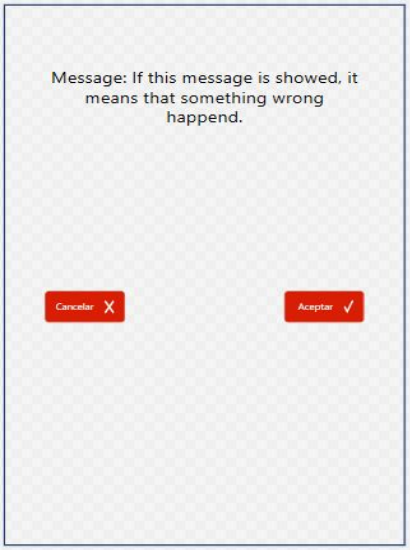

Imagen 2.16: Representación de User control que permite mandar mensajes en diferentes puntos.

LoadingScreen: Pantalla de carga la cual sirve para denotar que algo se está trabajando de fondo, es llamada en los procesos más longevos de la aplicación como el feriado o las consulta a la base de datos.
Tiene incorporado un timer configurable el cual denota un error de los procesos si la pantalla de carga está activa más de un tiempo estimado.

Imagen 2.17: User control de pantalla de carga, el mensaje es dinámico, y puede tener mensajes secundarios con un tamaño de letra menor

MessageLevelsView: Pantalla que es mostrada en caso de que el nivel de billetes para dispensar, monedas a dispensar o papel de impresora son bajos, de esta forma el cliente sabrá que está haciendo una transacción riesgosa, (Lo óptimo aquí es que nunca se muestre esta pantalla)

Imagen 2.18: User control que muestra la cantidad de dinero y papel al cliente

2.4. Clases de control

Las clases de control son todas aquellas que nos ayudan a tener métodos de una forma contextualizada, de esta forma se tiene una buena estructura dentro del proyecto, a continuación se mencionan las creadas.

Devices:
Aquí se incluyen un sin fin de métodos que son necesarios para controlar los dispositivos, a continuación se mostrarán los métodos que incluye esta clase.
Cabe destacar que todos los metodos estan usando librerías internas de la empresa LYF, que a su vez están usando controladores de los respectivas empresas que proporcionan los dispositivos, por lo cual, muchas veces se estan llamando métodos internos dentro de los mismos de esta clase.

a) Open: Método encargado de todas las órdenes necesarias para poder encender todos los dispositivos al iniciar el cajero, esto solo se hace una vez, ya que si se tuviesen que prender los dispositivos para cada transacción, habría una espera muy grande para las transacciones.

Lo que hace especial este método es que simplemente se encarga de agrupar la inicialización de los diferentes dispositivos, así como asignar valores de inicialización de los sensores para poder comprobar el estado de las puertas.
En este metodo tambien se abren los respectivos eventos los cuales van a ser ejecutados en todo momento en el inicializador de la aplicacion (app.cs), de tal forma que estos eventos puedan ser ejecutados siempre.

```
public static void Open()
{
        _masterBoard.Open(Config.Current.MasterBoardConfig.Port);
        _masterBoard.Inputs(Config.Current.MasterBoardConfig.Address, true);
        var inputs =_masterBoard.Input(Config.Current.MasterBoardConfig.Address);
        IsKnobClosed = inputs.SensorStatus.Ch01;
        IsUpperDoorClosed = inputs.SensorStatus.InPut03;
        IsInternalDoorClosed = inputs.SensorStatus.InPut02;
        IsExternalDoorClosed = inputs.SensorStatus.InPut01;
        _masterBoard.EMessageReceive += OnMessageMasterBoard;
        _billAcceptor.MessageReceived += OnMessageBillAcceptor;
        _billAcceptor.OpenSync();
        _billDispenser.e_MessageReceive += BillDispenser_MessageReceived;
}
```

b) AfterOpeningResetter: Al momento de usar una opción de administrador, muchas de las restricciones para poder abrir el cajero son desactivadas, de tal forma que el cajero no pueda ser abierto de nuevo:

```
public static void AfterOpeningResetter()
    {
        //Revisa si alguna compuerta está abierta
        var problemWithSensors = !IsUpperDoorClosed ||
            !IsKnobClosed ||
            !IsExternalDoorClosed ||
            !IsInternalDoorClosed;

        AvailableToOpen = false;
        MainWindow.ControlChildren = null;
        KioskNotOpened = true;
        IsCurrentTransactionAService = false;

        if (problemWithSensors)
        {
            SensorChecker();
        }
    }
```

c) OpenDoorsSensorChecker: Indica el comportamiento de bloqueos con sensores cuando éstos estén permitidos, es decir, nos indica como los sensores estan comportandose cuando las puertas están permitidas para abrirse, de esta forma el cajero sabe cuando se cierran todas las puertas para continuar.

```
public static void OpenDoorsSensorChecker(User user = null, List<Document> dataDocuments =
null, MoneyTransaction moneyTransaction = null, TransactionType ticketType =
TransactionType.Undefined)
      {
         if (IsUpperDoorClosed || (!IsExternalDoorClosed && !IsInternalDoorClosed &&
!IsKnobClosed))
         {
            App.Window.Dispatcher.Invoke(() =>
            {
               App.Window.SetContent(new Views.Admin.AdminOpen(ticketType, user,
dataDocuments, moneyTransaction));
            });
         }
      }
```

d) DoorsAvailable: Regresa un booleano que indica si alguna puerta está abierta dependiendo si está permitido o no abrir la puerta externa

```
private static bool DoorsAvailable()
      {
         return (MainWindow.ControlChildren.GetType() == typeof(Views.Admin.AdminDrain) ||
          App.Window.CompareCurrentView<Views.Admin.AdminDrain>()) ?
          !IsInternalDoorClosed || !IsKnobClosed || !IsUpperDoorClosed :
         !IsExternalDoorClosed || !IsInternalDoorClosed || !IsKnobClosed || !
IsUpperDoorClosed;
 }
```

e) DoorClosed: Regresa un booleano indicando si las puertas están cerradas dependiendo si está permitido o no abrir la puerta externa, este método fue creado ya que el cliente en ciertas pantallas quería que fuese posible abrir la puerta externa del cajero.

```
private static bool DoorClosed() {
return (MainWindow.ControlChildren.GetType() == typeof(Views.Admin.AdminDrain) ||
App.Window.CompareCurrentView<Views.Admin.AdminDrain>()) ?
  IsInternalDoorClosed && IsKnobClosed && IsUpperDoorClosed :
  IsExternalDoorClosed && IsInternalDoorClosed && IsKnobClosed
  }
```

f) SensorChecker: Probablemente el método más complejo de la clase devices, ya que tiene que tomar en cuenta todas las posibles pantallas para saber cuando bloquearse o

no la aplicación, en resumen lo que hace es ver si al momento de abrir una compuerta tiene que bloquearse la aplicación, y a su vez devolverla al punto donde se encontraba después de que todo esté en orden.

```
public static void SensorChecker()
      {
          //revisar si el cajero tiene permitido abrirse
         if (AvailableToOpen || IsCurrentTransactionAService)
         {
             //Si la apertura está disponible, loggear el proceso
            App.Log.Message(string.Empty,$"OPENING-AVAILABLE");
         }
         else
         {
             //revisar si tiene guardado un user control de repuesto
            if (MainWindow.ControlChildren == null)
            {
                //De no ser el caso, vamos a guardar pantalla de respaldo
               App.Window.UserControlSetter();
            }
             // ver que puertas estan abiertas
            if (DoorsAvailable())
            {
               App.Window.Dispatcher.Invoke(() =>
               {
                   //Se mostrará una pantalla de cargado con un mensaje que indique
                   //las puertas que estan abiertas
                  string externalSensor = IsExternalDoorClosed ||
                     App.Window.CompareCurrentView<Views.Admin.AdminDrain>() ?
                     string.Empty :
                     Config.Current.Gui.LoadingScreen.ExternalSensorOpen;

                  string upperSensor = IsUpperDoorClosed ?
                     string.Empty :
                     Config.Current.Gui.LoadingScreen.SensorUpperOpen;

                  string internalSensor = IsInternalDoorClosed ?
                     string.Empty :
                     Config.Current.Gui.LoadingScreen.InternalSensorOpen;

                  string knobSensor = IsKnobClosed ?
                     string.Empty :
                     Config.Current.Gui.LoadingScreen.SensorKnobOpen;

                  string secondaryMessage =
$"{externalSensor}{internalSensor}{knobSensor}{upperSensor}";
```

```
                    App.Window.SetContent(
                        new LoadingScreenView(
                            Config.Current.Gui.LoadingScreen.SensorErrorMessage,
                            secondaryMessage,
                            TextAlignment.Left
                        )
                    );

                    App.Log.Message($"Upper-sensor-" +
                        $"{(IsUpperDoorClosed ? "CLOSED" : "OPENED")}" +
                        $", External-sensor-" +
                        $"{(IsExternalDoorClosed ? "CLOSED" : "OPENED")}" +
                        $", Internal-sensor-" +
                        $"{(IsInternalDoorClosed ? "CLOSED" : "OPENED")}" +
                        $", Knob-sensor-" +
                        $"{(IsKnobClosed ? "CLOSED" : "OPENED")}",
                        $"SENSOR-STATUS-OPENING-ALLOWED"
                    );
                });
            }
        //si las puertas se cierran tendremos diferentes tipos de opciones
            else if (MainWindow.ControlChildren != null && DoorClosed())
            {
                App.Window.Dispatcher.Invoke(() =>
                {
                    // Revisaremos a donde te llevara el metodo dependiendo la pantalla donde se
violaron las puertas
                    // las siguientes pantallas no irán a ningún lado
                    var screenNotChanging =
                      MainWindow.ControlChildren.GetType() == typeof(Views.PaymentCashView);

                    // Pantallas que iran a WelcomeView
                    var screenGoingToWelcomeView =
                        MainWindow.ControlChildren.GetType() != typeof(LoadingScreenView)
                        && MainWindow.ControlChildren.GetType() != typeof(WelcomeView)
                        && MainWindow.ControlChildren.GetType() !=
typeof(Views.PaymentCashView);

                    if (screenNotChanging)
                    {
                        App.Log.Message(
                            $"Ingoring origin view: {MainWindow.ControlChildren.ToString()}",
                            "GUI-VIEW");
                    }
                    else if (screenGoingToWelcomeView)
                    {
                        App.Window.SetContent(MainWindow.ControlChildren);
                        App.Log.Message($"Returning to origin view: " +
```

```
                    $"{MainWindow.ControlChildren.ToString()}", "GUI-VIEW");
                }
                else
                {
                    //Pantallas que volveran a su origen
                    App.Window.SetContent(new WelcomeView());
                    App.Log.Message($"Returning to origin view: " +
                        $"Mapfre.Views.WelcomeView", "GUI-VIEW");
                }
                MainWindow.ControlChildren = null;
            });

        }
    }
}
```

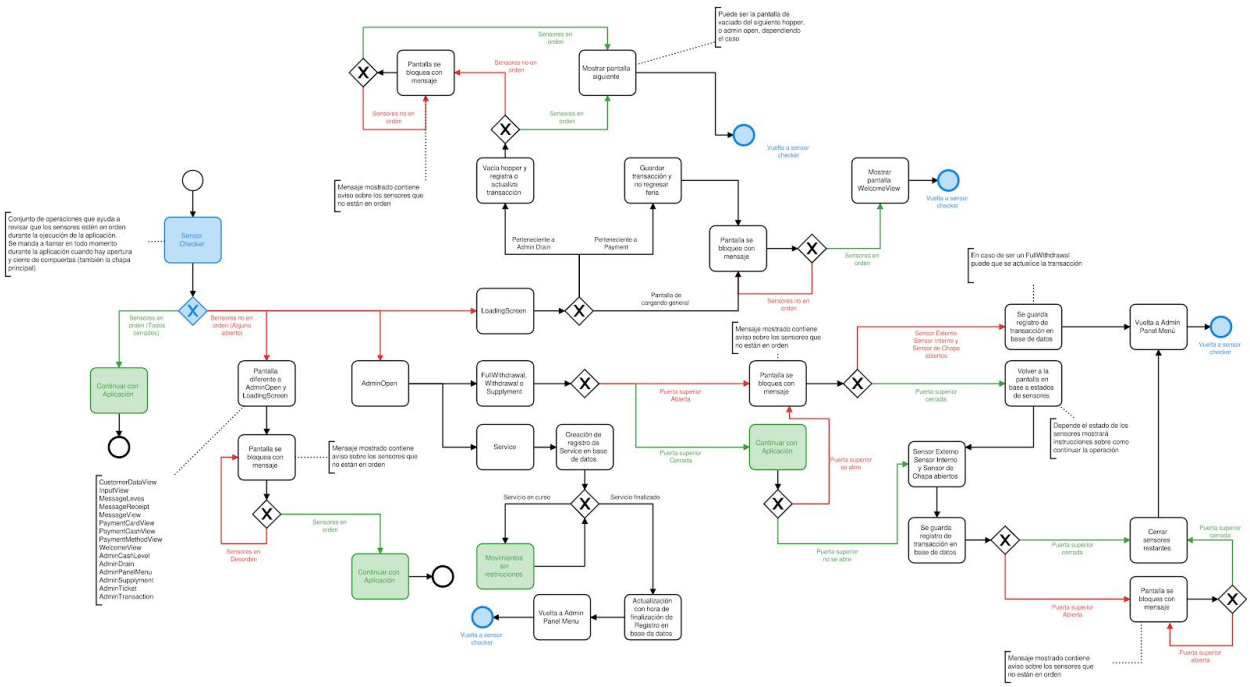

Imagen 2.19: Representación visual que se usó para comprender los distintos casos pósibles para los sensores, imagen en tamaño original [1]

g) Enable: Método que se encarga de activar los dispositivos, de esta forma se ahorra tiempo, es decir, desactiva el bloqueo de uso

```
public static void Enable()
    {
        _billAcceptor.EnableSync();
        UploadSiteIfIfNeeded(
```

```
            _coinInterface.EnableDisable_Sync(
                ActiveDeactive.ENABLE_COINS,
                (StatusNodes.NODES)Config.Current.CcTalkConfig.AcceptorIndex
            )
        );
        App.Log.Message(string.Empty, "DEVICES-ENABLED");
    }
```

h) Disable: Método que hace lo contrario al anterior.

```
        _billAcceptor.DisableSync();
        UploadSiteIfNeeded(
            _coinInterface.EnableDisable_Sync(
                ActiveDeactive.DISABLE_COINS,
                (StatusNodes.NODES)Config.Current.CcTalkConfig.AcceptorIndex
            )
        );
        App.Log.Message(string.Empty, "DEVICES-DISABLED");
```

i) IsPrinterPaperLevelLow: nos indica si el nivel de papel es óptimo con un booleano.

```
 public static bool IsPrinterPaperLevelLow()
  {
 #if DEV_Sensors || DEV_Doors
 try
 {
    var isPrinterPaperLevelLow =
        _masterBoard == null ||
        _masterBoard?.SensorStatus.Printer.Key == SensorPrinter.UpdStatusPaperEnd
        || _masterBoard?.SensorStatus.Printer.Key == SensorPrinter.UpdStatusNearEnd
        || _masterBoard?.SensorStatus.Printer.Key == SensorPrinter.UpdStatusOtherErr;

    App.Log.Message(
        isPrinterPaperLevelLow ? "Paper Level Low" :
        "Paper Level OK", "PRINTER-PAPER-LEVEL"
    );

    return isPrinterPaperLevelLow;
 }
 catch (Exception ex)
 {
    App.HandleException(ex);
    return true;
 }
}
```

j) IsCoinsLevelLow: nos indica si el nivel de monedas para dispensación es optimo.

```
public static bool IsCoinsLevelLow()
    {
        var Coindocuments = App.DataBase
            .GetCurrentValues()
            .Where(x => x.Value > 0 &&
            x.Count > 0 &&
            x.ContainerId != App.BillCashboxContainerId &&
            x.Type == DocumentType.Coin)
            .ToList();

        decimal CoinTotalDoc = 0;

        for (int i = 0; i < Coindocuments.Count; i++)
        {
            CoinTotalDoc = Coindocuments.ElementAt(i).Count + CoinTotalDoc;
        }

        decimal CoinCapacity = (Config.Current.GeneralData.HopperCapacity) *
            (Config.Current.CcTalkConfig.Hoppers.Length);

        decimal CoinCapacityV = 0;

        if ((CoinTotalDoc / CoinCapacity) > 1)
        {
            CoinCapacityV = 100;
        }
        else
        {
            CoinCapacityV = (CoinTotalDoc / CoinCapacity) * 100;
        }
        return CoinCapacityV < 15;
    }
```

k) IsBillsLevelLow: nos indica si el nivel de billetes para dispensacion es optimo.

```
public static bool IsBillsLevelLow()
    {
        var Billdocuments = App.DataBase
            .GetCurrentValues()
            .Where(x => x.Value > 0 &&
            x.Count > 0 &&
            x.ContainerId != App.BillCashboxContainerId &&
            x.Type == DocumentType.Bill)
            .ToList();

        decimal BillTotalDoc = 0;
```

```
            for (int i = 0; i < Billdocuments.Count; i++)
            {
                BillTotalDoc = Billdocuments.ElementAt(i).Count + BillTotalDoc;
            }
            decimal BillCapacity = (Config.Current.GeneralData.CassetteCapacity) *
                    (Config.Current.F53Config.Cassettes.Length);

            decimal BillCapacityV = 0;

            if ((BillTotalDoc / BillCapacity) > 1)
            {
                BillCapacityV = 100;
            }
            else
            {
                BillCapacityV = (BillTotalDoc / BillCapacity) * 100;
            }

            return BillCapacityV < 15;
        }
```

l) OpenKnob: Método que se encarga de abrir la cerradura de seguridad la cual solo puede ser abierta con software:

```
public static void OpenKnob()
{
   var knob = Config.Current.MasterBoardConfig.Knob;
   _masterBoard.Out(Config.Current.MasterBoardConfig.Address, knob, false);
   _masterBoard.Out(Config.Current.MasterBoardConfig.Address, knob, true);
}
```

m) EmtpyHopper: método que vacía contenedor de monedas

```
public static void EmtpyHopper(CcTalkNode hopper)
      {
         var response = _coinInterface.Empty_Sync((StatusNodes.NODES)hopper.Index);
         UploadSiteIfNeeded(response);
}
```

Además de los métodos anteriormente mencionados se implementan eventos, los cuales son disparados en situaciones específicas:

a) Evento que es disparado al momento de movimiento con sensores, se encarga de actualizar los valores de los sensores y ejecuta el metodo que revisa si todas las

puertas están en orden, de tal forma que al momento de recibir que un sensor es modificado, se bloqueara si es el caso:

```
private static void OnMessageMasterBoard(MasterBoard.Response e)
    {
        if (e.SensorChange.Key == Config.Current.MasterBoardConfig.ExternalSensor)
        {
            IsExternalDoorClosed = e.SensorChange.Value;
            App.Log.Message($"Upper-sensor-" +
                $"{(IsUpperDoorClosed ? "CLOSED" : "OPENED")}" +
                $", External-sensor-" +
                $"{(IsExternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Internal-sensor-" +
                $"{(IsInternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Knob-sensor-" +
                $"{(IsKnobClosed ? "CLOSED" : "OPENED")}",
                $"SENSOR-STATUS"
            );
        }
        if (e.SensorChange.Key == Config.Current.MasterBoardConfig.InternalSensor)
        {
            IsInternalDoorClosed = e.SensorChange.Value;
            App.Log.Message($"Upper-sensor-" +
                $"{(IsUpperDoorClosed ? "CLOSED" : "OPENED")}" +
                $", External-sensor-" +
                $"{(IsExternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Internal-sensor-" +
                $"{(IsInternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Knob-sensor-" +
                $"{(IsKnobClosed ? "CLOSED" : "OPENED")}",
                $"SENSOR-STATUS"
            );
        }
        if (e.SensorChange.Key == Config.Current.MasterBoardConfig.UpperSensor)
        {
            IsUpperDoorClosed = e.SensorChange.Value;
            App.Log.Message($"Upper-sensor-" +
                $"{(IsUpperDoorClosed ? "CLOSED" : "OPENED")}" +
                $", External-sensor-" +
                $"{(IsExternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Internal-sensor-" +
                $"{(IsInternalDoorClosed ? "CLOSED" : "OPENED")}" +
                $", Knob-sensor-" +
                $"{(IsKnobClosed ? "CLOSED" : "OPENED")}",
                $"SENSOR-STATUS"
            );
        }
```

```
            if (e.SensorChange.Key == Config.Current.MasterBoardConfig.KnobSensor)
            {
                IsKnobClosed = e.SensorChange.Value;
                App.Log.Message($"Upper-sensor-" +
                    $"{(IsUpperDoorClosed ? "CLOSED" : "OPENED")}" +
                    $", External-sensor-" +
                    $"{(IsExternalDoorClosed ? "CLOSED" : "OPENED")}" +
                    $", Internal-sensor-" +
                    $"{(IsInternalDoorClosed ? "CLOSED" : "OPENED")}" +
                    $", Knob-sensor-" +
                    $"{(IsKnobClosed ? "CLOSED" : "OPENED")}",
                    $"SENSOR-STATUS"
                );
            }

            if (e.SensorChange.Key != Sensor.Printer)
            {
                SensorChecker();
            }

            DoorStatusChanged?.Invoke();

            // send error to SITEF if any door was opened unexpectedly
            if (App.Window.GetContent().GetType() != typeof(Views.Admin.AdminOpen))
            {
                var opened = "Abierto";
                var closed = "Cerrado";
                var code = "Door_UnauthorizedOpening";
                var date = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

                var description = "Se abrieron una o mas puertas del quiosco en un momento
 inesperado:" +
                        $"\n  • Puerta externa: {(IsExternalDoorClosed ? closed : opened)}" +
                        $"\n  • Puerta interna: {(IsInternalDoorClosed ? closed : opened)}" +
                        $"\n  • Puerta superior: {(IsUpperDoorClosed ? closed : opened)}" +
                        $"\n  • Cerradura: {(IsKnobClosed ? closed : opened)}";

                UploadSitefAlert(code, description, date);
            }
        }
```

b) Evento que es disparado cuando se introduce una moneda: se encarga de tomar los valores de la moneda que es introducida si es que no hay error alguno con este.

```
 private static void OnMessageCoinAcceptor(CCTALKV2.Response response)
{
decimal coin = (decimal)(response.AceptorCoin.Value);
```

```
if (coin != 0)
{
   var doc = new Document {
     Type = DocumentType.Coin,
      Value = coin,
      Count = 1,
      Currency = Config.Current.CcTalkConfig.Currency,
      ContainerId = App.CoinCashboxContainerId,
      Inserted = true
   };
   DocumentAccepted?.Invoke(doc);
   }

   // notify errors to SITEF ws
   UploadSiteflfNeeded(response);
}
```

c) Evento que es disparado cuando el aceptador de billetes tiene movimientos: se encarga de revisar que tipo de movimiento es el que tiene el aceptador de billetes, de tal forma que dependiendo el caso tomará el valor de los billetes o ejecutara diferente tipos de acciones.

```
private static void OnMessageBillAcceptor(Scx sender, Message message)
      {
         // manage autostack
         if (message.Type == MessageType.Escrow)
         {
            _billAcceptor.StackSync();
         }
         // manage stacks
         else if (message.Type == MessageType.Stacked)
         {
            if (!(message is DocumentMessage docMessage))
            {
               return;
            }

            DocumentAccepted?.Invoke(new Document
            {
               Value = (decimal)docMessage.Document.Value / 100,
               Type = DocumentType.Bill,
               Count = 1,
               ContainerId = App.BillCashboxContainerId,
               Currency = Utils.GetCurrency(docMessage.Document.Nation),
               Inserted = true
            });
```

```
            }
            // notify errors to SITEF ws
            else
            {
                var description = GetBillAcceptorErrorDescription(message);

                if (description != null)
                {
                    var code = $"BillAcceptor_{message.Type}";
                    var date = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                    UploadSitefAlert(code, description, date);
                }
            }
        }
```

Payment: Esta clase es la que se encarga de subir el pago a un webservice y ejecutar todos los comandos correspondientes tras de este, como el feriado, esta clase no se va a profundizar ya que contiene información del cliente y algoritmos delicados de la empresa.
Hay que destacar los servicios que son llamados de forma Sitef para contextualizar, que son todos esos procesos que son procesados por un web service.

Printer: Aquí van todos los métodos que se encargan de imprimir los tickets, es usado tanto al final de una transacción como en la reimpresión de tickets, lo único que pide cada método es un conjunto de variables que van a ser ingresados a un formato ya preestablecido en una librería interna de LYF.
los métodos que contiene son los siguientes:

a) PrintPaymentReceipt: Este es el recibo que es impreso al momento de generar un pago, lo que pide es el valor de la deuda, el usuario, lo introducido por el cliente, la fecha, el folio de la transacción, que tipo de transacción es, el cambio recibido, si faltó cambio, si es copia y si la transacción falló, a continuación el código que da formato:

```
    public static void PrintPaymentReceipt(decimal debt, string user, decimal introduced, DateTime
date, string folio, string transaction, decimal change, decimal remaining, bool isCopy, bool failed, string
policy)
    {
        // dont print if payment failed and remaining is 0
        if (failed && remaining <= 0) return;

        // continue
        var doc = new LibreR.SystemPrinting.PrintDocument();
        doc.Document.PrintController = new StandardPrintController();
        if (isCopy)
        {
            doc.AddLine("--- Copia ---", alignment: System.Windows.TextAlignment.Center);
            DateTime fecha_actual = DateTime.Now;
```

```
            doc.AddLine($"Fecha de Reimpresión: {fecha_actual}", alignment:
System.Windows.TextAlignment.Center);
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 218 });
        }
        else
        {
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 178 });
        }
        doc.NewLine();

        doc.AddLine(
            $"Quiosco: {Models.Config.Current.GeneralData.Kiosk}",
            alignment: System.Windows.TextAlignment.Center
        );

        doc.AddLine($"Fecha: {date:dd-MM-yyyy hh:mm:ss tt}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Folio: {folio.PadLeft(6, '0')}", alignment: System.Windows.TextAlignment.Center);

        doc.AddLine(
            $"Transacción: {transaction.ToString().PadLeft(16, '0')}",
            alignment: System.Windows.TextAlignment.Center
        );

        doc.AddLine($"Póliza: {policy}", alignment: System.Windows.TextAlignment.Center);
        doc.AddLine($"Usuario: {user}", alignment: System.Windows.TextAlignment.Center);
        doc.NewLine();
        doc.NewLine();
        doc.NewLine();
        doc.Separator('=');

        var header = "COMPROBANTE DE PAGO";

        if (introduced < debt)
        {
            header += " PARCIAL";
        }

        if (failed)
        {
            header += $" FALLIDO";
        }

        if (remaining > 0)
        {
            header += " CON CAMBIO PENDIENTE";
        }
```

```
            doc.AddLine(
                header,
                12,
                FontStyle.Bold,
                null,
                alignment: System.Windows.TextAlignment.Center
            );

            if (remaining > 0)
            {
                doc.Separator('-');

                doc.AddLine(
                    "Por favor presente en sucursal este comprobante para reclamar su cambio pendiente",
                    alignment: System.Windows.TextAlignment.Center
                );
            }

            doc.Separator('-');
            doc.AddLine(
                new Column(new Line("Deuda:"), 0, 149),
                new Column(new Line($"{debt:C}") { Alignment = System.Windows.TextAlignment.Right }, 150,
250));
            doc.AddLine(
              new Column(new Line("Monto ingresado:"), 0, 149),
              new Column(new Line($"{introduced:C}") { Alignment = System.Windows.TextAlignment.Right },
150, 250));
            doc.Separator('-');
            doc.AddLine(
                new Column(new Line(failed ? "Devolución:" : "Cambio:"), 0, 149),
                new Column(new Line($"{change:C}") { Alignment = System.Windows.TextAlignment.Right },
150, 250));

            if (remaining > 0)
            {
                doc.AddLine(
                    new Column(new Line("Pendiente:"), 0, 149),
                    new Column(new Line($"{remaining:C}") { Alignment =
System.Windows.TextAlignment.Right }, 150, 250)
                );
            }

            doc.Separator('-');
            doc.AddLine("Gracias por usar nuestros servicios", alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine($"Contacto: {Models.Config.Current.GeneralData.ContactInfo}", alignment:
System.Windows.TextAlignment.Center);
            doc.Print();
```

```
            App.Log.Message(string.Empty, "TICKET-PRINTED");
        }
```

b) PrintServiceReceipt: Es el método que se encarga de imprimir un ticket después de un servicio al cajero, los datos que pide son el usuario, la fecha de inicio del servicio, la fecha de finalización de servicio, el folio y si es un ticket de copia:

```
        public static void PrintServiceReceipt(string user, DateTime beginning, DateTime finish, string
folio, bool copia)
        {
            var doc = new LibreR.SystemPrinting.PrintDocument();
            doc.Document.PrintController = new StandardPrintController();
            if (copia)
            {
                doc.AddLine("-Copia-", alignment: System.Windows.TextAlignment.Center);
                DateTime fecha_actual = DateTime.Now;
                doc.AddLine($"Fecha de Reimpresión: {fecha_actual}", alignment:
System.Windows.TextAlignment.Center);
                doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 175 });
            }
            else
            {
                doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 135 });
            }
            doc.NewLine();
            doc.AddLine($"Quiosco: {Models.Config.Current.GeneralData.Kiosk}", alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine($"Fecha: {beginning}", alignment: System.Windows.TextAlignment.Center);
            doc.AddLine($"Folio: {folio.ToString().PadLeft(6, '0')}", alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine($"Usuario: {user}", alignment: System.Windows.TextAlignment.Center);
            doc.NewLine();
            doc.NewLine();
            doc.NewLine();
            doc.Separator('=');
            doc.AddLine("APERTURA DE SERVICIO", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.Separator('-');
            doc.AddLine($"Inicio {beginning}");
            doc.AddLine($"Fin {finish}");
            doc.Separator('=');
            doc.AddLine($"{Models.Config.Current.GeneralData.GeneralFooter}", alignment:
System.Windows.TextAlignment.Center);
            doc.Print();
        }
```

c) PrintSupplymentReceipt: método que imprime un ticket para una transacción de dotación de efectivo, lo único que pide es un objeto de tipo MoneyTransaction que contiene toda la información a ingresar en el ticket y si el ticket es una copia

```
public static void PrintSupplymentReceipt(MoneyTransaction transaction, bool copia)
{
    var doc = new LibreR.SystemPrinting.PrintDocument();
    doc.Document.PrintController = new StandardPrintController();
    if (copia)
    {
        doc.AddLine("-Copia-", alignment: System.Windows.TextAlignment.Center);
        DateTime fecha_actual = DateTime.Now;
        doc.AddLine($"Fecha de Reimpresión: {fecha_actual}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 175 });
    }
    else
    {
        doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 135 });
    }
    doc.NewLine();
    doc.AddLine($"Quiosco: {Models.Config.Current.GeneralData.Kiosk}", alignment:
System.Windows.TextAlignment.Center);
    doc.AddLine($"Fecha: {transaction.CreatedAt}", alignment:
System.Windows.TextAlignment.Center);
    doc.AddLine($"Folio: {transaction.Id.ToString().PadLeft(6, '0')}", alignment:
System.Windows.TextAlignment.Center);

    doc.AddLine(
        $"Usuario: {transaction.Users.FirstOrDefault()?.ToString() ?? "---"}",
        alignment: System.Windows.TextAlignment.Center
    );

    doc.NewLine();
    doc.NewLine();
    doc.NewLine();
    doc.Separator('=');
    doc.AddLine("DOTACIÓN DE EFECTIVO", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);

    Document[] documents = transaction.Documents.ToArray();
    int totalBill = documents.Count(x => x.Type == DocumentType.Bill);
    int totalCoin = documents.Count(x => x.Type == DocumentType.Coin);

    if (totalCoin > 0)
    {
        doc.Separator('-');
```

```
            doc.AddLine("Monedas", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine(
              new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
              new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 84, 167),
              new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 168, 250));

            foreach (var document in documents)
            {
               if (document.Type == DocumentType.Coin)
               {
                  doc.AddLine(
                     new Column(
                        new Line($"{document.Value.ToString("C")}")
                        {
                           Alignment = System.Windows.TextAlignment.Right,
                           Style = FontStyle.Bold
                        },
                        0,
                        83
                     ),
                     new Column(
                        new Line($"{document.Count}")
                        {
                           Alignment = System.Windows.TextAlignment.Right,
                           Style = FontStyle.Bold
                        },
                        84,
                        167
                     ),
                     new Column(
                        new Line($"{(document.Value * document.Count).ToString("C")}")
                        {
                           Alignment = System.Windows.TextAlignment.Right,
                           Style = FontStyle.Bold
                        },
                        168,
                        250
                     )
                  );
               }
            }
         }

         if (totalBill > 0)
```

```
        {
            doc.Separator('-');
            doc.AddLine("Billetes", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine(
            new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
            new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 84, 167),
            new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 168, 250));

            foreach (var document in documents)
            {
                if (document.Type == DocumentType.Bill)
                {
                    doc.AddLine(
                      new Column(new Line($"{document.Value.ToString("C")}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 0, 83),
                      new Column(new Line($"{document.Count}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 84, 167),
                      new Column(new Line($"{(document.Value * document.Count).ToString("C")}") {
Alignment = System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 168, 250)
                    );
                }
            }
        }

        doc.Separator('-');
        doc.AddLine($"Total {documents.Sum().ToString("C")}", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
        doc.Separator('=');
        doc.AddLine($"{Models.Config.Current.GeneralData.GeneralFooter}", alignment:
System.Windows.TextAlignment.Center);
        doc.Print();
    }
```

d) PrintPartialWithdrawalReceipt: Imprime ticket de un retiro, y solamente pide un objeto de Money transaction y booleano que indica si es copia.

```
     public static void PrintPartialWithdrawalReceipt(MoneyTransaction transaction, bool copia)
    {
        var doc = new LibreR.SystemPrinting.PrintDocument();
        doc.Document.PrintController = new StandardPrintController();
        if (copia)
        {
            doc.AddLine("-Copia-", alignment: System.Windows.TextAlignment.Center);
```

```
            DateTime fecha_actual = DateTime.Now;
            doc.AddLine($"Fecha de Reimpresión: {fecha_actual}", alignment:
System.Windows.TextAlignment.Center);
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 175 });
        }
        else
        {
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 135 });
        }
        doc.NewLine();
        doc.AddLine($"Quiosco: {Models.Config.Current.GeneralData.Kiosk}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Fecha: {transaction.CreatedAt}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Folio: {transaction.Id.ToString().PadLeft(6, '0')}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Usuario: {transaction.Users[0].ToString()}", alignment:
System.Windows.TextAlignment.Center);
        doc.NewLine();
        doc.NewLine();
        doc.NewLine();
        doc.Separator('=');
        doc.AddLine("RETIRO DE EFECTIVO", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);

        Document[] documents = transaction.Documents.ToArray();
        int totalBill = documents.Count(x => x.Type == DocumentType.Bill);
        int totalCoin = documents.Count(x => x.Type == DocumentType.Coin);

        if (totalCoin > 0)
        {
            doc.Separator('-');
            doc.AddLine("Monedas", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);

            doc.AddLine(
              new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
              new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 84, 167),
              new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 168, 250)
            );

            foreach (var document in documents)
            {
              if (document.Type == DocumentType.Coin)
              {
```

```
                    doc.AddLine(
                        new Column(new Line($"{document.Value.ToString("C")}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 0, 83),
                        new Column(new Line($"{document.Count}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 84, 167),
                        new Column(new Line($"{(document.Value * document.Count).ToString("C")}") {
Alignment = System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 168, 250)
                    );
                }
            }
        }

        if (totalBill > 0)
        {
            doc.Separator('-');
            doc.AddLine("Billetes", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine(
            new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
            new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 84, 167),
            new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 168, 250));

            foreach (var document in documents)
            {
                if (document.Type == DocumentType.Bill)
                {
                    doc.AddLine(
                        new Column(new Line($"{document.Value.ToString("C")}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 0, 83),
                        new Column(new Line($"{document.Count}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 84, 167),
                        new Column(new Line($"{(document.Value * document.Count).ToString("C")}") {
Alignment = System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 168, 250)
                    );
                }
            }
        }

        doc.Separator('-');

        doc.AddLine(
            $"Total {(documents.Sum() * -1).ToString("C")}",
            12,
            FontStyle.Bold,
            null,
```

```
            alignment: System.Windows.TextAlignment.Center
        );

        doc.Separator('=');
        doc.AddLine($"{Models.Config.Current.GeneralData.GeneralFooter}", alignment:
System.Windows.TextAlignment.Center);
        doc.Print();
    }
```

e) PrintFullWithdrawalReceipt: Imprime ticket de un retiro total, y solamente pide un objeto de Money transaction y booleano que indica si es copia.

```
    public static void PrintFullWithdrawalReceipt(MoneyTransaction transaction, bool copia)
    {
        var doc = new LibreR.SystemPrinting.PrintDocument();
        doc.Document.PrintController = new StandardPrintController();
        if (copia)
        {
            doc.AddLine("-Copia-", alignment: System.Windows.TextAlignment.Center);
            DateTime fecha_actual = DateTime.Now;
            doc.AddLine($"Fecha de Reimpresión: {fecha_actual}", alignment:
System.Windows.TextAlignment.Center);
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 175 });
        }
        else
        {
            doc.AddImage(@"Sources\receipt.png", new Point { X = 90, Y = 135 });
        }
        doc.NewLine();
        doc.AddLine($"Quiosco: {Models.Config.Current.GeneralData.Kiosk}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Fecha: {transaction.CreatedAt}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Folio: {transaction.Id.ToString().PadLeft(6, '0')}", alignment:
System.Windows.TextAlignment.Center);
        doc.AddLine($"Usuario: {transaction.Users[0].ToString()}", alignment:
System.Windows.TextAlignment.Center);
        doc.NewLine();
        doc.NewLine();
        doc.NewLine();
        doc.Separator('=');
        doc.AddLine("RETIRO TOTAL DE EFECTIVO", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);

        Document[] documents = transaction.Documents.ToArray();
        int totalBill = documents.Count(x => x.Type == DocumentType.Bill);
        int totalCoin = documents.Count(x => x.Type == DocumentType.Coin);
```

```
        if (totalCoin > 0)
        {
            doc.Separator('-');
            doc.AddLine("Monedas", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine(
              new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
              new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 84, 167),
              new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style
= FontStyle.Bold }, 168, 250));

            foreach (var document in documents)
            {
                if (document.Type == DocumentType.Coin)
                {
                    doc.AddLine(
                      new Column(new Line($"{document.Value.ToString("C")}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 0, 83),
                      new Column(new Line($"{document.Count}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 84, 167),
                      new Column(new Line($"{(document.Value * document.Count).ToString("C")}") {
Alignment = System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 168, 250)
                    );
                }
            }
        }

        if (totalBill > 0)
        {
            doc.Separator('-');
            doc.AddLine("Billetes", 12, FontStyle.Bold, null, alignment:
System.Windows.TextAlignment.Center);
            doc.AddLine(
            new Column(new Line("Denominación") { Alignment = System.Windows.TextAlignment.Right,
Style = FontStyle.Bold }, 0, 83),
            new Column(new Line("Cantidad") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 84, 167),
            new Column(new Line("Subtotal") { Alignment = System.Windows.TextAlignment.Right, Style =
FontStyle.Bold }, 168, 250));

            foreach (var document in documents)
            {
                if (document.Type == DocumentType.Bill)
                {
                    doc.AddLine(
```

```
                    new Column(new Line($"{document.Value.ToString("C")}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 0, 83),
                    new Column(new Line($"{document.Count}") { Alignment =
System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 84, 167),
                    new Column(new Line($"{(document.Value * document.Count).ToString("C")}") {
Alignment = System.Windows.TextAlignment.Right, Style = FontStyle.Bold }, 168, 250)
                );
            }
        }
    }

    doc.Separator('-');

    doc.AddLine(
        $"Total {(documents.Sum() * -1).ToString("C")}",
        12,
        FontStyle.Bold,
        null,
        alignment: System.Windows.TextAlignment.Center
    );

    doc.Separator('=');
    doc.AddLine($"{Models.Config.Current.GeneralData.GeneralFooter}", alignment:
System.Windows.TextAlignment.Center);
    doc.Print();
  }
 }
}
```

Clase Utils: En esta clase hay un conjunto de métodos que nos ayudarán a diferentes tipos de transacciones.

Clase webservice, aquí existe el conjunto de métodos de webservices que ayudan a subir las transacciones al respectivo servidor.

2.5 Clases de modelos

Conjunto de clases que se encarga de la creación de objetos para la utilización de diferentes transacciones, la cual debido a su extensión demasiado grande solo se explicaran los objetos y la clase de origen:

Clase config: La función de esta clase es la de generar objetos con valores asignados los cuales serán ingresados en un archivo de configuración, de esta forma, el cajero leerá de un archivo de tipo texto diferentes opciones de configuración, las siguientes son las principales clases que almacenan, a continuación se muestra la composición usada para el archivo de configuración:

a) AdminPanel: Contiene la configuración configurable de la pantalla AdminPanelView, aquí solamente se destaca un booleano cuya función es indicar si la pantalla a regresar después de una dotación o retiro es el mismo menú o la pantalla welcome view.
b) MasterBoardConfig: Contiene toda la información necesaria para poder controlar el dispositivo con nombre de master board (encargado de las luces led, impresora y sensores de puertas)
c) CcTalkConfig: Contiene toda la información de configuración para el aceptador y de monedas.
d) F53Config: Contiene los archivos de configuración para el aceptador de billetes.
e) Scx: Contiene los archivos de configuración para el dispositivo que se encarga del feriado de billetes (al contrario del dispositivo que acepta monedas que hace ambas funciones, el dispensador y aceptador de billetes son dos aparatos distintos).
f) LoadingData: Datos configurables de la pantalla de cargado, como el tiempo de espera.
g) GeneralData: Datos generales que la aplicación va a tomar, como la capacidad de los contenedores, la versión del cajero, la información de contacto etc.
h) PaymentConfig: Configuración que es necesaria para el pago, actualmente solamente tiene la opción de poder pagar una cantidad mayor a la deuda.
i) Gui: Aquí va toda la configuración que va en cuanto lo visual, por ejemplo el contenido de los botones o los strings que se leerán en los labels o textbox, de esta forma no es necesario compilar todo de nuevo.

InsertionData: Aqui esta toda esa información que involucra el ingreso de billetes y monedas al cajero, se divide en lo siguiente:

a) InsertionData: contiene los valores de interés del documento introducido, como el valor, la cantidad, la nacionalidad y el tipo que es (moneda o billete)
b) ReceiptData: información que es utilizada para la impresión del ticket, como el folio de la transacción, el tipo de transacción que fue, cantidad ingresada al cajero y retirada, lo regresado, la fecha, los detalles, un arreglo de documentos y el usuario.
c) PaymentDetails: detalles generales del pago, como la deuda, el cambio, el restante, si fue exitosa, los mensajes que envía etc.

User: Información de los usuarios que podrán ingresar al panel administrativo, contiene Id, usuario, contraseña, si está activo y que tipo de usuario es.

3. Análisis sobre el proyecto

En la siguiente sección se mostrará las conclusiones y análisis finales del proyecto, de tal forma de tratar de expresar todas las experiencias y habilidades adquiridas.

3.1 Experiencia Adquirida:

En cuanto experiencia adquirida creo que me sirvió demasiado, el simple hecho de haber trabajado con un modelo de metodología ágil como lo fue en este caso con scrum, me hizo ver todas las virtudes, a su vez, la forma en que el proyecto fue avanzando y tomando forma, tanto por uso de gitlab que con anterioridad lo usaba de una forma muy básica.

Creo que si bien el trabajar con C# con anterioridad lo sentía como mi fortaleza mas grande al momento de programar, el trabajar directamente con un equipo experto en este lenguaje hizo ver que realmente mi campo de conocimientos era mínimo, y que podría mejorar más, lo cual se agradece de una forma muy grande, sobre todo en la estructura en la que las clases y los métodos, donde ahora son de una forma ramificada para evitar dependencias, por lo que cada clase

3.2 Análisis general del programa, su diseño, desarrollo y organización.

La administración del proyecto final, si bien no es la mejor ya que estuve aprendiendo mediante trabajaba, creo que para ser mi primer trabajo en ámbito laboral es de una forma correcta y bien estructurada, sobre todo por que el proyecto estuvo siendo elaborado a la par con compañeros, los cuales no dejaban que se uniesen nuevos fragmentos de código si este no cumplia con los estándares de calidad.

La retroalimentación diaria hizo que el proyecto tuviese una mejor organización, que si bien en momentos yo era el único dedicado al proyecto totalmente, el hecho de que el equipo aportase sus ideas ayudó mucho

3. 3 Conclusión final

Creo que como estudiante el tener una experiencia laboral antes de poder concluir tus estudios es un agregado muy importante para la realización de una formación, ya que la seguridad que da el poder expresar las habilidades adquiridas ayuda demasiado a que se sienta más confianza para poder salir de la institución.

El proyecto final fue un trabajo que en sí dará prestigio y clientes a LYF por lo cual fue de suma ayuda para ellos y creo que el darme la confianza de hacer este proyecto de talla internacional, si bien en un principio fue un poco intimidante con el tiempo fui agarrando confianza, sin miedo a preguntar las dudas y la oportunidad de absorber información de otros programadores.

4 Referencias

GmbH, C. (2019). Cawemo. Retrieved 4 December 2019, from https://cawemo.com

[1] https://cawemo.com/share/7438d1b4-002a-43c3-aae5-e148e265895a

The first single application for the entire DevOps lifecycle - GitLab. (2019). Retrieved 4 December 2019, from https://about.gitlab.com/

UNIVERSIDAD DE SONORA

COORDINACIÓN DIVISIONAL DE INGENIERIA

PRÁCTICAS PROFESIONALES

FPP-4

DEPARTAMENTO: INGENIERÍA INDUSTRIAL

UNIDAD REGIONAL CENTRO CAMPUS HERMOSILLO

REPORTE FINAL DE ACTIVIDADES

Periodo: Del ____ / _____ / 201___ al ____ / _____ / 201___

Cantidad de 340 Horas de un total de 340 Avance: 100 %

Nombre del practicante: Alan Alberto Lerma Molina

Expediente: 214214225 **Programa Educativo (Licenciatura):** ISI

Nombre del Programa/Proyecto: Creación de Software para Cajeros Automáticos

Datos de la Unidad Receptora (Razón Social): LYF Ingeniería S.A. de C.V.

Responsable de la Unidad Receptora (Nombre/Puesto): José Rafael Olivas Izaguirre

Contacto: Teléfono/UR: _____ Ext. _____ Celular: 6623007272

DESCRIPCIÓN GENERAL DE ACTIVIDADES
Se desarrolló el software para un Kiosco de automatización de pagos el cual se integró con distintos dispositivos, los cuales se encargan de la aceptación de monedas, la devolución de estas, pagos por tarjeta de credito, debito, seguridad del Kiosco y otras funciones.
Tambien se desarrolló el funcionamiento de pantallas pensando en la usabilidad.

RETROALIMENTACIÓN (Comentarios del tutor)

En caso de requerirse, anexar reportes, formatos, diagramas que apoyen las actividades realizadas.

Para las Ingenierías deberá anexar ***reporte técnico*** en archivo electrónico ≤ 2 MB y carta de terminación de prácticas firmada por el responsable de la empresa.

Observaciones Generales:

Alan Alberto Lerma Molina		L Y F INGENIERIA S.A. DE C.V. DESARROLLO INTEGRACION Y VENTA DE EQUIPO ELECTRONICO Y DE CÓMPUTO. RFC LIF1080423JF1
Nombre y firma del alumno	**Nombre y firma del tutor de prácticas profesionales UniSon.** Osvaldo Sanchez Schmitz	**Nombre y firma del responsable de la unidad receptora Sello de la UR**

Original entregar en físico al *Coordinador o Responsable de Prácticas Profesionales de la carrera.*
Copia *para Tutor de Prácticas Profesionales y* ***Copia*** *alumno.*
Enviar en PDF los documentos *al coordinador/responsable de prácticas profesionales de la carrera.*

(25/04/2018)

LYF Ingenieria S.A. de C.V.

LYF INGENIERIA

Hermosillo, Sonora a 04 de Febrero de 2020

A QUIEN CORRESPONDA:

Por medio de la presente hago de su constar que **Alan Alberto Lerma Molina** con número de expediente **214214225** de la carrera de Ingeniería en sistemas de información en la Universidad de Sonora llevó a cabo sus prácticas profesionales en nuestra empresa **LYF Ingeniería S.A de C.V**, ubicada en Congreso 384 Col Ley 57 entre Arizona y 12 de Octubre en Hermosillo Sonora México con código postal 83100 en el periodo de Septiembre del año 2019 a Noviembre del año 2019 en el área de desarrollo, ayudando en la creación de software para Kiosco de pago de servicios, el cual constó de 340 horas.

Sin más por el momento quedo a sus ordenes

LYF INGENIERIA

LYF INGENIERIA S.A. DE C.V.
DESARROLLO INTEGRACION Y
VENTA DE EQUIPO
ELECTRONICO Y DE CÓMPUTO.
RFC:LFI080423JF1

Atentamente.

José Rafael Olivas Izaguirre
Gerente de Desarrollo
LYF Ingenieria S.A. de C.V.