Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
IE1119 — Sistemas Incrustados
I-2020
Profesor: Esteban Ortiz

# Tarea de Diseño

Dispensador de alimentos IoT

**Circuititos S.A.**
Leonardo Hernández B43262
Willy Villalobos B17170

## 1. Marco Teórico

### 1.1. Definiciones

1. IoT: Hace referencia a la tendencia de conectar toda clase de dispositivos físicos al Internet, en donde se emplean varios protocolos de comunicación y toda clase de funcionalidades que se pueden combinar entre si, sin necesidad de que exista interacción con las personas. De este término empieza a arraigarse más la idea de casas e incluso ciudades inteligentes donde existe todo un ecosistema de dispositivos comunicándose entre sí y recolectando información de los usuarios y su entorno [2].

2. MQTT: Es un protocolo para comunicaciones máquina-máquina especialmente centrado en la idea del IoT. La principal premisa detrás de este protocolo es el transporte de mensajes en un sistema basado en suscripciones y publicaciones, y el requerimiento de código y recursos computacionales es relativamente bajo [1].

3. QT: Es un framework multiplataforma ampliamente utilizado en el desarrollo de aplicaciones con una interfaz gráfica de usuario (GUI). Tiene la ventaja de ser software libre y abierto, con una amplia comunidad y empresas que soportan su desarrollo y mantenimiento. Utiliza C++ como lenguaje de programación de forma nativa, pero también existen bibliotecas para su uso en Python [4].

## 2. Requerimientos de software del sistema completo

La implementación del sistema de punto de venta y de sistema central de monitoreo tiene requerimientos divididos para cada sistema. En general se manejan las siguientes categorías:

- Interacción con el usuario.
- Procesamiento de datos.
- Transferencia de datos.
- Administración de servicios.

Estas categorías engloban las diferentes áreas que se considera de mayor importancia para poder especificar, diseñar e implementar el software del punto de venta en el dispensador de alimentos. El cuadro 1 muestra dichos requerimientos con su respectiva categoría y código.

Cuadro 1: Requerimientos de Software del punto de venta del dispensador de alimentos.

| Código | Categoría | Descripción |
|---|---|---|
| PVSW-001 | Interacción con el usuario | El sistema deberá mostrar en la interfaz gráfica una pantalla principal con botones para acceder a una ventana de cliente y una ventana de administrador. |
| PVSW-002 | Interacción con el usuario | El sistema deberá mostrar en la pantalla principal un indicador de la temperatura interna de la maquina en grados Celsius. |
| PVSW-003 | Interacción con el usuario | El sistema deberá mostrar el mensaje **Powered by Circuititos S.A.** en la parte inferior de la pantalla principal. |
| PVSW-004 | Interacción con el usuario | La ventana de cliente deberá desplegar una lista de los productos disponibles e información relevante como cantidad disponible y costo. En caso de que la maquina este vacía debe desplegar un mensaje indicándolo. |
| PVSW-005 | Interacción con el usuario | La ventana de cliente, en caso de haber productos disponibles, deberá incluir un conjunto de espacios al lado de cada producto para seleccionar la cantidad que se desea comprar. Estas cajas de espacios no deben aceptar valores inválidos tales como negativos, caracteres alfabéticos, fracciones, o enteros más grandes que la cantidad disponible. |
| PVSW-006 | Interacción con el usuario | La ventana de cliente, en caso de haber productos disponibles, deberá incluir un conjunto de botones que le permitan al usuario gestionar la selección y compra de un carrito (debe ser posible adquirir varios productos en una misma compra). Las opciones mínimas que debe soportar son:<br>▪ Actualizar carrito: Se incluye al carrito la selección de productos realizada a través de los espacios en PVSW-005.<br>▪ Limpiar carrito.<br>▪ Comprar carrito. |
| PVSW-007 | Interacción con el usuario | La ventana de cliente, en caso de haber productos disponibles, deberá incluir información al lado de cada espacio para la selección de producto, indicando el costo por los productos seleccionados y en la parte inferior el costo total. Esta información debe ser actualizada al actualizar el carrito. |
| PVSW-008 | Interacción con el usuario | Tanto la ventana de cliente como la ventana de administrador deben contar con un botón que le permita al usuario volver a la pantalla principal. |
| PVSW-009 | Interacción con el usuario | La ventana de administración deberá desplegar el estado real de la máquina en un arreglo rectangular de 5x5, donde cada casilla corresponde a una cola de productos. Cada casilla debe desplegar una foto del producto e información relevante como la cantidad disponible y precio. Además, si la casilla está vacía se debe incluir en cambio una imagen e información que así lo refleje. |

| Código | Categoría | Descripción |
|---|---|---|
| PVSW-010 | Interacción con el usuario | La ventana de administración deberá desplegar un conjunto de cajas de selección para editar el producto y la cantidad en una cola de la máquina. |
| PVSW-011 | Interacción con el usuario | La ventana de administración deberá incluir una serie de botones que le permitan al usuario hacer efectiva la actualización del estado de la máquina descrito en PVSW-010. Estos deben funcionar como se indica:<br>▪ Remover Todo: Este botón facilita vaciar todas las colas de la máquina.<br>▪ Actualizar: Este botón permite realizar las modificaciones a las colas indicadas mediante a las cajas de selección según PVSW-010.<br>▪ Cancelar Selección: Permite limpiar el estado actual de las cajas de selección. |
| PVSW-012 | Interacción con el usuario | La ventana de administración deberá contar con un botón que le permita al usuario cambiar el tipo de producto que se desea manejar. Además en consecuencia al cambio de tipo producto, automáticamente se debe vaciar la máquina y las cajas de selección deben reflejar el inventario correspondiente. |
| PVSW-013 | Administración de servicios. | Tras presionar el botón para cambiar el tipo de producto en la ventana de administración, la cámara debe cambiar a 27°C para temperatura ambiente y 4°C para frío. . |
| PVSW-014 | Administración de servicios. | Tras presionar el botón para comprar el carrito, la máquina debe dispensar el producto de la/las cola/colas con la menor cantidad disponible. |
| PVSW-015 | Transferencia de datos. | El punto de ventas debe generar un conjunto de alertas, tal como se describe en la sección 2.1 |
| PVSW-016 | Transferencia de datos | El punto de venta deberá comunicarse con un centro de control remoto que gestionará las alertas de compras e inventario. Generadas en PVSW-015 |
| PVSW-017 | Transferencia de datos | El protocolo de comunicación entre el punto de venta y el servidor central será MQTT cifrado. |

De forma similar, los requerimientos para el sistema central de monitoreo se muestran en el cuadro 2.

Cuadro 2: Requerimientos de Software del sistema central de monitoreo.

| Código | Categoría | Descripción |
|---|---|---|
| SMSW-001 | Procesamiento de datos | El sistema deberá gestionar de forma continua las compras y disponibilidad de los productos, a través de alertas provenientes de los puntos de venta. |
| SMSW-002 | Transferencia de datos | La comunicación con los puntos de venta se hará mediante protocolo MQTT. |

## 2.1. Alertas generadas.

En esta implementación el punto de venta enviará mensajes en los cuales se seguirá un formato estándar que facilite su manejo por una aplicación externa de monitoreo e interpretación de mensajes. Dicho formato será **[ID punto de**

**venta] mensaje :: Sat Jul 4 14:51:56 2020** y se dará para los siguientes eventos:

- Compras realizadas, donde se enviará la cantidad de unidades compradas por producto. Ej: **[vending_lw_001] sold 1 of Fanta Grape :: Sat Jul 4 14:50:46 2020**
- Alertas por cada producto que se agota en el punto de venta. Ej: **[vending_lw_001] Powerade out of stock. :: Sat Jul 4 14:51:56 2020**
- Alerta por punto de venta sin stock de productos. Ej: **[vending_lw_001] unit is empty. Please restock. :: Sat Jul 4 15:11:42 2020**

El timestamp se agregará en el momento que el servidor recibe el mensaje y lo escribe en un log file dedicado a este propósito. El log file en general tendría un contenido similar al mostrado:

```
[vending_lw_001] sold 24 of Powerade :: Sat Jul  4 17:38:28 2020
[vending_lw_001] Powerade out of stock. :: Sat Jul  4 17:38:28 2020
[vending_lw_001] sold 15 of Fanta Grape :: Sat Jul  4 17:38:28 2020
[vending_lw_001] Fanta Grape out of stock. :: Sat Jul  4 17:38:28 2020
[vending_lw_001] sold 22 of Fanta Red :: Sat Jul  4 17:38:28 2020
[vending_lw_001] Fanta Red out of stock. :: Sat Jul  4 17:38:28 2020
[vending_lw_001] sold 5 of Fanta Orange :: Sat Jul  4 17:38:28 2020
[vending_lw_001] Fanta Orange out of stock. :: Sat Jul  4 17:38:28 2020
[vending_lw_001] sold 11 of Coke :: Sat Jul  4 17:38:29 2020
[vending_lw_001] Coke out of stock. :: Sat Jul  4 17:38:29 2020
[vending_lw_001] unit is empty. Please restock. :: Sat Jul  4 17:38:32 2020
```

# 3. Arquitectura de la solución

## 3.1. Protocolo MQTT mediante el programa Mosquitto

Mosquitto es un servidor de mensajes el cual implementa el protocolo MQTT [3]. Es una aplicación liviana ideal para sistemas incrustados de pocos recursos que se desea emplear para aplicaciones IoT o simple paso de mensajes con otros dispositivos. Este emplea un sistema de subscripción y publicación de mensajes.

Para el caso del sistema de máquina dispensadora inteligente se emplea este programa para facilitar un canal de subscripción en el cual los distintos puntos de venta y el servidor central estarán inscritos. Los puntos de venta enviarán mensajes a este canal, pero solamente el servidor central tomará acciones según los mensajes recibidos, según se muestra en la figura 1. El canal de subscripción será provisto por Mosquitto, y en cada punto de venta habrá una clase en python llamada Broker con los métodos para conectarse, suscribirse, publicar mensajes y extraerlos del canal para su interpretación.

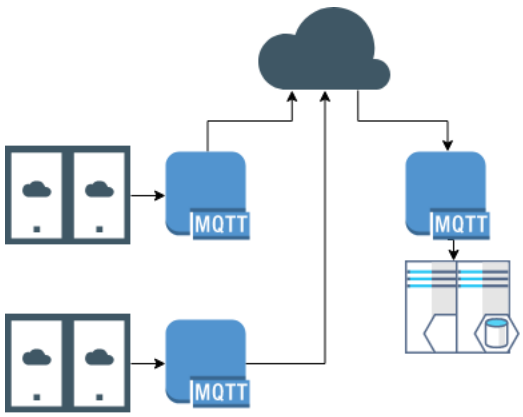


Figura 1: Diagrama de comunicación entre las máquinas expendedoras y el servidor central usando el protocolo MQTT.

# 4. Clases y métodos

## 4.1. Manejo de la interfaz gráfica

### 4.1.1. NewProduct

Esta clase básicamente permite crear objetos que representan un producto. Los atributos de la clase corresponden a las características que nos interesan del producto, por lo tanto se tienen:

- Name = Representa el nombre o marca que identifica al producto. Por ejemplo Çoke", "Fanta", "Pringles".
- Cost = Corresponde a un valor entero que representa el precio en colones del producto, por unidad.
- Temperature: Caracter que ubica el producto entre el tipo que debe almacenarse frío ('C') o a temperatura ambiente ('R').
- LogoPath: Corresponde al dirección donde se encuentra la imagen del logo que distingue al producto.

### 4.1.2. NewQueue

Esta clase permite crear objetos que representan a una cola de productos. Cada cola cuenta con 4 atributos, que son útiles para interactuar con ellas a lo largo del programa. Se detallan a continuación:

- Row = Representa la fila en la que se ubica la cola dentro de la máquina expendedora. De izquierda a derecha, las filas se denotan con A,B,C,D,E respectivamente.
- Column = Representa la columna en la que se ubica la cola dentro de la máquina expendedora. De arriba a abajo, las columnas se denotan con los números del 1 al 5, respectivamente.
- Product: Objeto tipo NewProduct, representando al producto disponible en la cola.
- Quantity: Cantidad disponible del producto.

  Además esta clase tiene un conjunto de métodos que le permiten al usuario modificar su estado, entre ellas:

Cuadro 3: Métodos de la clase NewQueue.

| Nombre | Uso |
|---|---|
| Add_Product(Product) | Permite agregar un objeto NewProduct a la cola. |
| Remove_Product() | Permite remover una unidad del producto presente en la cola. |
| Add_Products(Product,Quantity) | Llama a la función Add_Product, "Quantity"veces. |
| Remove_Product(Quantity) | Llama a la función Remove_Product, "Quantity"veces. |

Cabe destacar que estos métodos son lo suficientemente robustos para manejar las siguientes condiciones:

- Una cola no puede tener más de 5 productos o menos de cero.
- Una cola solamente puede tener un tipo de producto. Para agregar otro tipo es necesario primero vaciar la cola.
- Si la cola está vacía el atributo Product presente es un objeto NewProduct que denote vacío.

### 4.1.3. NewVendingMachine

Esta clase permite crear objetos que representan un máquina expendedora. Los atributos de esta clase son:

- list = Mediante una lista de 25 posiciones se almacena en cada una de estas un objeto tipo NewQueue que representa una cola.
- Temp = Mediante este atributo, se designa si el tipo de producto presente en la máquina expendedora corresponde a un producto que se debe almacenar frío o caliente.

Cuadro 4: Métodos de la clase NewQueue.

| Nombre | Uso |
|---|---|
| Add_Product_To_Queue(Product,Row,Col) | Permite agregar un producto a la cola en (Row,Col) de la máquina. |
| Remove_Product_From_Queue(Row, Col) | Permite remover una unidad del producto de la cola (Row,Col). |
| Check_Availability(Product) | Retorna la cantidad disponible del producto en la máquina. |
| UpdateQueue(self, Row, Col, Product, Quantity) | Cambia forzadamente el producto y cantidad de la cola en (Row,Col). |
| ClearVM() | Vacía la máquina expendedora. |
| Extract_Product( Product) | Función para comprar una unidad de Producto. |
| Extract_A_Quantity_Of_Products( Product, Quantity) | Llama a la función Extract_Product, "Quantity"veces. |

Algunos detalles que vale la pena destacar son:

- La realizar una compra de un producto, el producto se extrae de la cola con la menor cantidad disponible de este.
- Se tiene una serie de funciones de ayuda para pasar de (Row,Col) a una posición 0,24 de la lista que representa la máquina expendedora, y viceversa.
- Hay ciertos escenarios posibles a la hora de editar el producto de una cola, desde la clase NewVending Machine, algunos se detallan a continuación.

    1. Si el administrador selecciona Empty como el tipo de producto y presiona el botón para actualizar inventario, la selección de Quantity es irrelevante, pues la cola estará vacía.
    2. Si la cola está vacía, seleccionar un producto sin seleccionar la cantidad, o viceversa, no tiene efecto y la cola permanecerá vacía luego de presionar update.
    3. Si la cola no está vacía es posible cambiar el tipo de producto manteniendo la cantidad del producto anterior, y viceversa, es posible cambiar la cantidad de un producto sin cambiar este último por otro.

Para mejorar el orden del programa y tener mayor modularidad, se creó un archivo que almacena todos los productos, denominado Catalog.py. Este aparte de las instancias de los objetos tipo NewProduct, correspondientes a productos legítimos, también incluye una instancia denominada NoProduct, que representa en la clase NewQueue la ausencia de producto en una cola. Además se creó otro archivo denominado InitVendingMachine.py que permite crear una instancia de un objeto tipo NewVendingMachine, y además implementa un par de listas, que sirven como inventarios. Se tiene una lista de productos para almacenar en frío y otras para productos que deben mantenerse a temperatura ambiente; ambos inventarios implementados a través de listas de instancias de objetos tipo NewProduct.

## 4.2. Manejo de comunicaciones con MQTT

Para las comunicaciones con MQTT existe una biblioteca en Python llamada paho-mqtt la cual incluye toda la estructura y métodos que el protocolo utiliza para suscribirse a un tema, conectarse a un host remota, publicar y recibir mensajes. Dada la naturaleza de la aplicación, se decidió implementar una clase wrapper llamada broker, la cual facilita la inicialización y la comunicación de loss puntos de venta, dejando por defecto muchos de los parámetros que paho-mqtt pide por cuenta propia.

### 4.2.1. broker

Esta clase permite manejar de forma más transparente las llamadas a paho-mqtt para facilitar la comunicación entre el punto de venta y el servidor central que recibe también los mensajes al inscribirse al tópico de interés. Esta clase recibe los siguientes atributos:

- broker_adr = dirección IP del host del tema de MQTT al cual nos suscribiremos.
- topic = tema al cual el cliente, en este caso cada máquina dispensadora, se suscribirá para publicar el estado local.
- client_name = corresponde al nombre del cliente que se suscribirá al tema en el servidor de MQTT.
- port = puerto a través del cual nos conectaremos al servidor, dado que este puede ser cambiado en la configuración del tema.
- keepalive = número de segundos que se mantendrá viva la conexión en caso de inactividad.

El cuadro 5 muestra los métodos de la clase.

Cuadro 5: Métodos de la clase broker.

| Nombre | Uso |
|---|---|
| client_init() | Inicializa una instancia de paho-mqtt con los parámetros de broker. |
| client_connect() | Inicia la conexión del cliente con el servidor y tema de MQTT. |
| client_disconnect() | Finaliza la conexión con el servidor y tema de MQTT. |
| client_subscribe() | Suscribe el cliente al tema de MQTT en el servidor. |
| client_publish(payload) | Publica en el tema y servidor indicados el mensaje contenido en la variable payload. |
| start_payload_loop() | Inicializa un bucle para recibir y procesar los mensajes del tema en el servidor de MQTT. |
| stop_payload_loop() | Detiene el bucle de procesamiento de mensajes. |

Para la ejecución a nivel de cliente basta con solo inicializar la conexión con el servidor MQTT y llamar al método client_publish() cada vez que se desea publicar un mensaje, particularmente mensajes relacionados con actualización de inventario, compras o similares. A nivel de servidor, este debe también suscribirse al tema y agregar un método on_message(client, userdata, message) el cual tendrá toda la lógica que se desea realizar con cada mensaje recibido. Este método se agregara a la variable client de la clase broker en la instancia donde se desea realizar monitoreo y procesamiento de los mensajes. Como muestra se puede analizar el siguiente ejemplo de código:

```
def on_message(client, userdata, message):
    msg_received = str(message.payload.decode("utf-8"))
    msg_topic    = message.topic
    msg_qos      = message.qos
    msg_retain   = message.retain
    print("message received ", msg_received)
    print("message topic=", msg_topic)
    print("message qos=", msg_qos)
    print("message retain flag=", msg_retain)

    # Writes received messages to a log file with a timestamp.
    f = open("vending_machine.log", "a")
    f.write(f"{msg_received} :: {time.ctime(time.time())}\n")
    f.close()

lil_client = broker(broker_adr, topic, client_name, port)
lil_client.client_init()
lil_client.client.on_message = on_message # Acá se agrega el método a la funcionalidad
    del cliente.
```

## 4.3. Manejo de la interfaz gráfica

Para la implementación de la interfaz gráfica se decidió utilizar PyQt5. Se creó un programa denominado App.py que corresponde a la interfaz gráfica de la aplicación, esta cuenta con 1 ventana principal tipo QMainWindow, 3 ventanas secundatrias tipo QMainWindow y una ventana de confirmación más simple, tipo QMessageBox. Estas se describen brevemente a continuación:

### 4.3.1. Ventana Principal

Esta incluye un par de botones para navegar hacia las ventadas de compra en caso del cliente, o para administración en caso del personal de soporte y mantenimiento. En el futuro se debe agregar algún tipo de autenticación para ingresar al modo de administración. Además en la esquina superior derecha se despliega la temperatura de la cámara interna de la máquina.

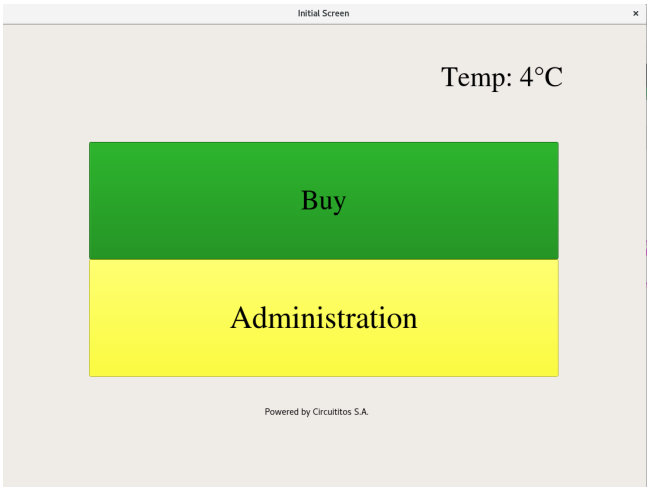


Figura 2: Ventana Principal de la interfaz gráfica.

### 4.3.2. Ventana de administración.

Esta ventana permite al administrador del equipo modificar los productos presentes en la máquina. Para hacer efectivo el cambio del producto actual debe presionar el botón 'update' tras la selección adecuada de las colas por modificar. Presenta una opción para representar una cola vacía, botones para ejecutar funciones útiles que facilitan la labor del personal administrativo como limpiar selección (Clear) y remover todos los elementos de la máquina. Finalmente el botón 'Change Product Type' cambia la temperatura de la cámara, alternando entre frío y temperatura ambiente, consecuentemente limpia la máquina y hace un cambio en el inventario que se puede agregar ahora a cada cola.

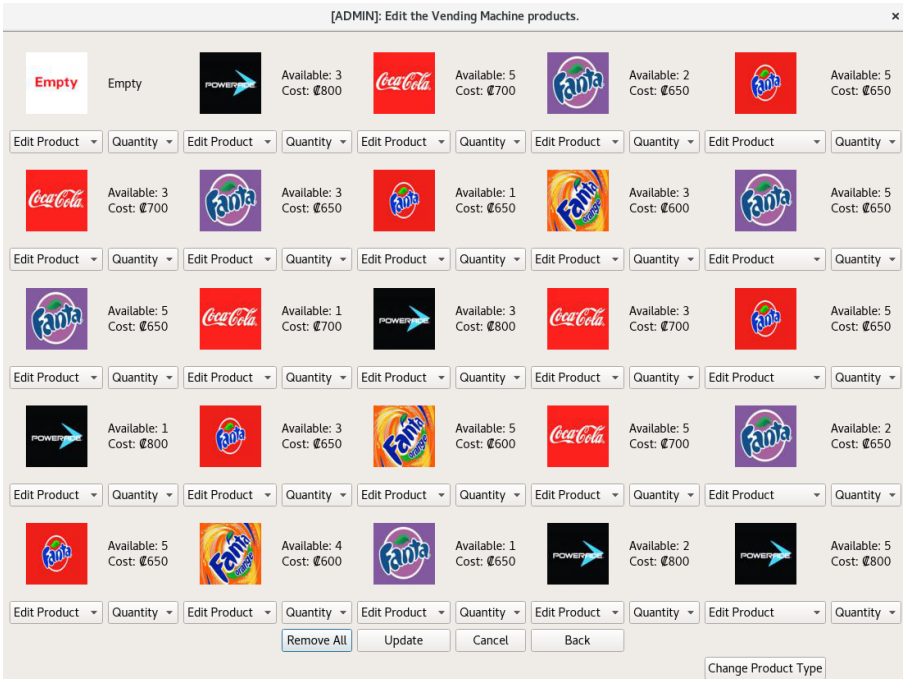


Figura 3: Ventana de adminstración.

### 4.3.3. Ventana de Inventario.

Esta es una ventana de despliegue de productos. Es muy similar a la ventana de administración sin embargo esta no incorpora las cajas de selección y solo incluye un botón para volver a la ventana antes mencionada, por lo tanto es visualmente más simple al usuario.

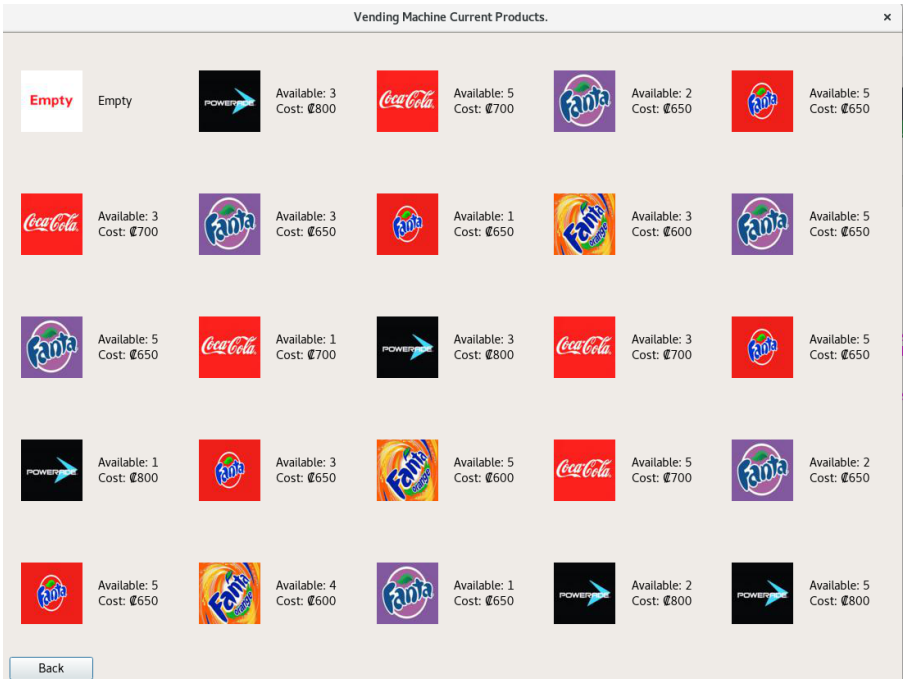


Figura 4: Ventana de despliegue de inventario.

### 4.3.4. Ventana de venta.

Esta es la ventana de compra dirigida al cliente. Básicamente se presentan los productos disponibles, e información relevante como cantidad y precio. El cliente debe introducir la cantidad al lado de los productos que desea comprar, es importante mencionar que los espacios están implementados con objetos tipo QSpinBox(), que nos permiten rechazar el ingreso de caracteres alfabéticos y se configuró los valores máximos y mínimos para no exceder la cantidad disponible de producto, ni aceptar valores negativos por obvia razón. Una vez seleccionado el producto que se desea comprar el usuario debe presionar el botón de 'Update Chart' para visualizar el precio que debe pagar por su compra, y luego de presionar el botón de comprar carrito 'Buy Chart' se despliega un mensaje de confirmación. En caso de recibir confirmación el sistema realiza la compra exitosamente, entrega el producto y en caso de aplicar, envía los mensajes de alerta que correspondan a la central de monitoreo.

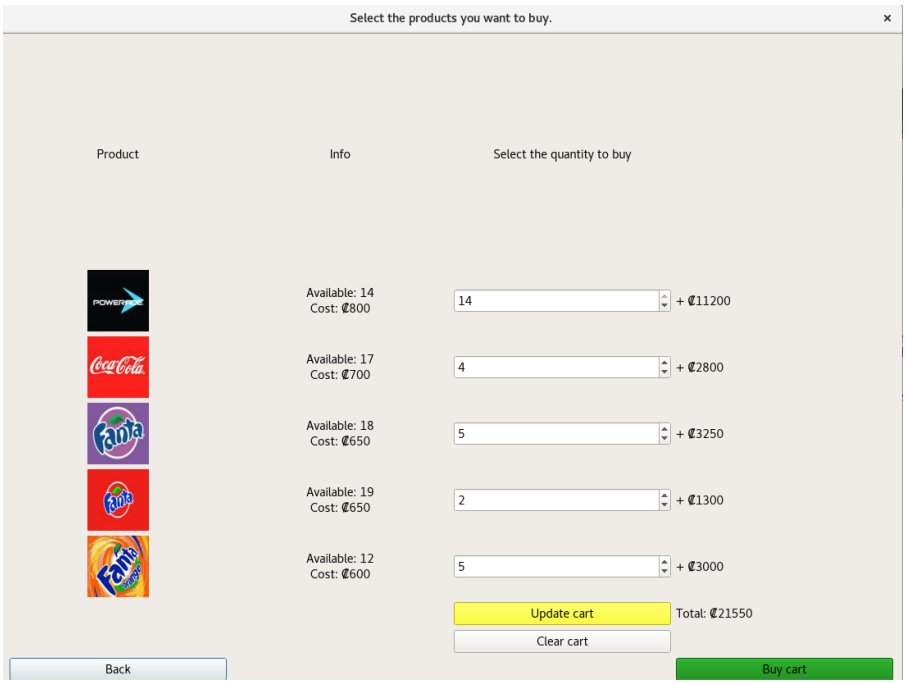


Figura 5: Ventana para la compra de productos.

# 5. Anexo.

- A continuación se presenta el link, a un video de youtube con un demo del funcionamiento del sistema: `https://youtu.be/yxcP3P6iJww`
- El código fuente del proyecto está presente en [5].

# Referencias


[1] MQTT. `http://mqtt.org/` Extraído el 23 de Junio del 2020.

[2] ¿Qué es el Internet de las cosas (IoT)?. `https://www.redhat.com/es/topics/internet-of-things/what-is-iot` Extraído el 23 de Junio del 2020.

[3] Eclipse Mosquitto: An open source MQTT broker. `https://mosquitto.org/` Extraído el 28 de Junio del 2020.

[4] QT: Cross-platform software development for embedded and desktop. `https://www.qt.io/` Extraído el 28 de Junio del 2020.

[5] Repositorio del proyecto. `https://github.com/leoher1996/VendingMachine`