AutoTicket: Sistema de cobro por QR con plataforma ESP32

Santiago Rotger, Lucas I. Calderon, Roberto M. Murdocca, Sergio F. Hernandez Velazquez
Departamento de Electrónica
Universidad Nacional de San Luis
San Luis, Argentina
e-mail: {santi.rotger, calderonlucasignacio, mmurdocc, sergio.sfhv}@gmail.com

Resumen - En el presente proyecto se desarrolla un sistema automático de cobro haciendo uso del microcontrolador ESP32. El sistema se aprovecha de las ventajas ofrecidas por las herramientas de pago digital de Mercado Pago: generando, cobrando y validando de forma automática y sin intermediarios un pago realizado a través de QR. Se implementa un servidor elaborado con JavaScript para realizar la autenticación del usuario de modo tal que pueda operar con el sistema. Mediante las capacidades de conectividad Wi-Fi de la ESP32, se implementaron solicitudes HTTP para interactuar con la base de datos y la API de Mercado Pago.

Palabras clave: ESP32, Mercado Pago, QR, Node.js, HTTP.

I. Introducción

La idea inicial que impulsó este proyecto surge de la observación directa de una problemática presente en el comedor universitario de la Universidad Nacional de San Luis (UNSL), en muchas ocasiones los alumnos no cuentan con dinero en efectivo pero sí en las distintas billeteras digitales que no son aceptadas como medio de pago actualmente. A dicha observación se suma los tiempos de espera para realizar la compra del ticket.

A fin de dar respuesta a esta problemática y aprovechando el fácil acceso a las billeteras virtuales, surge el desarrollo de AutoTicket, que permite automatizar y mejorar la experiencia de pago.

El sistema se implementó utilizando una ESP32, esto ya que cuenta con funcionalidades de conectividad Wi-Fi lo que le permite conectarse a un servidor para mantener un control de los alumnos y comunicarse con la API de Mercado Pago.

II. Descripción

El sistema integra un microcontrolador ESP32 con un servidor basado en Node.js que permiten la validación de los datos del alumno y del pago que se realice.

El usuario que realizará la compra deberá ingresar su DNI a través de un teclado matricial para así realizar la autenticación.

Tanto la ESP32 como el servidor interactúan con la API de Mercado Pago mediante peticiones HTTP para gestionar en tiempo real las transacciones, proporcionando un flujo continuo desde la generación del código QR hasta la confirmación del pago.

Si todas las comprobaciones son correctas se procede con la impresión del ticket. En la Fig.1 se muestra un diagrama en bloques completo del sistema Auto Ticket.

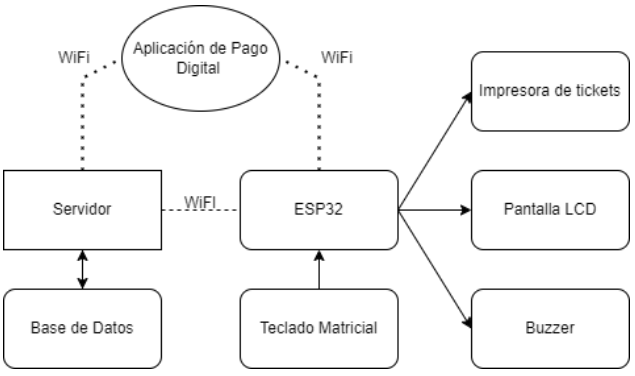

Fig. 1. Diagrama en bloques del sistema

Un aspecto a considerar en la implementación del sistema es la falta de una impresora térmica de tickets compatible con el microcontrolador, debido a su alto costo. Actualmente, la confirmación del pago se realiza visualmente a través del display y de forma auditiva con el buzzer. Aunque la integración de la impresora está pendiente, no afecta la validación del pago ni la operatividad

del sistema. La referencia a la impresión del ticket indica que el proceso de compra se ha completado con éxito.

Para contener el microcontrolador ESP32 junto con el teclado matricial, el buzzer, la pantalla LCD y la impresora de tickets con los que el usuario interactúa, se planeó el housing observable en la Fig.2, dicho gabinete se puede fijar a la pared a una altura apropiada o apoyarse sobre un mostrador.

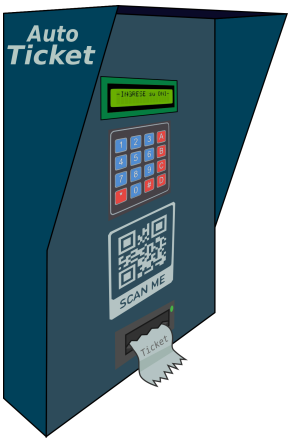

Fig. 2: Housing Propuesto

III. Arquitectura Física

La arquitectura está conformada por un microcontrolador ESP32 del fabricante Espressif (ESP32_DEVKIT_V1.1), un teclado matricial 4x4 (HC-543), un buzzer (TMB12A05), una pantalla LCD de dimensiones 16x2 (LCD-016N002B) y un Módulo que permite comunicación I2C (I2C-PCF8574).

La selección del microcontrolador ESP32 para este proyecto se debe a la capacidad de procesamiento que posee ya que tiene un procesador potente y eficiente. Esto es esencial para un proyecto que requiere la gestión de peticiones HTTP, procesamiento de datos, y el control de hardware [1]. Más específicamente las funcionalidades utilizadas del ESP32 en el sistema son: la conectividad WiFi ya que trabaja con el protocolo TCP-IP y además permite trabajar con el protocolo de comunicación I2C, usando únicamente el GPIO21 como línea serial de datos (SDA) y GPIO22 como línea de reloj serial (SCL) facilitando la comunicación con la pantalla LCD y permitiendo utilizar menos pines de salida. Además este microcontrolador cuenta con funcionalidades extras que aunque aquí no se utilicen pueden integrarse en futuras versiones de AutoTicket.

El modelo utilizado “ESP32_DEVKIT_V1.1”, permite utilizar una fuente de corriente continua que suministre 5 (V), ya que contiene un regulador de tensión que la reduce a 3,3 (V), y que entregue por lo menos 500 (mA) ya que los demás componentes se alimentan a partir del microcontrolador.

La pantalla LCD 16x2 se llama así porque tiene 16 columnas y 2 filas. Por lo tanto, tendrá (16×2=32) 32 caracteres en total y cada carácter está formado por 5x8 píxeles [7].

El teclado está formado por una matriz de pulsadores que se encuentran dispuestos en filas (L1,L2,L3,L4) y columnas (C1,C2,C3,C4), esta organización es de acuerdo a reducir el número de pines requeridos para su conexión y programación [5]. En la Fig.3 se muestra el diagrama de conexiones del hardware. En la Fig.4 se observa el prototipo del sistema.

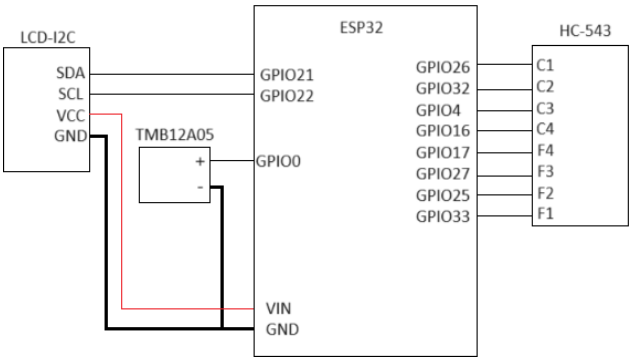

Fig. 3: Diagrama de conexiones

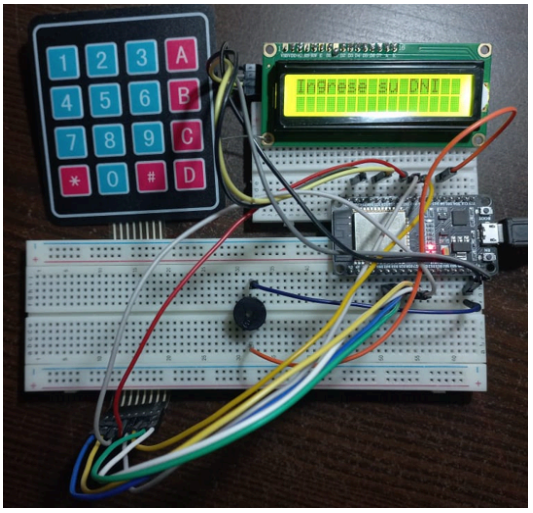

Fig. 4: Prototipo

IV. Arquitectura del Sistema

A. ESP32

Para el funcionamiento en conjunto de las distintas partes del sistema, se hace uso de las funciones de las siguientes librerías:

- Wifi.h: proporciona funciones para iniciar la conexión WiFi, verificar el estado de la conexión, la dirección IP, etc [4].
- HTTPClient.h: permite al dispositivo realizar solicitudes HTTP a servidores [4].
- Keypad.h: se encarga de gestionar la entrada de datos desde el teclado matricial [15].
- LiquidCrystal_I2C.h: permite controlar pantallas LCD a través de I2C [16].

B. Servidor

El servidor está basado en Node.JS que se utiliza para crear aplicaciones de y permite gestionar muchas conexiones de forma simultánea [2], en él se integra las siguientes herramientas para su funcionamiento:

- *Express:* es un framework, que brinda las herramientas necesarias para la construcción del servidor [12].
- *body-parse*r: es un middleware que analiza los cuerpos de las solicitudes HTTP y permite trabajar con datos en formato JSON [13].
- *Librería axios:* utilizada para interactuar con APIs externas permitiendo enviar y recibir datos de una forma sencilla [14].
- *Módulo sqlite3:* brinda las herramientas para la creación y trabajo con la base de datos SQL [8].

C. Base de Datos

La base de datos se construyó utilizando SQLite, se almacena en ella el DNI, el nombre y el apellido de los alumnos autorizados para comprar en el comedor universitario.

Se optó por SQLite ya que es fácil de integrar y brinda las herramientas suficientes para el número de datos que se están utilizando en este proyecto. También se tuvo en cuenta la gran cantidad de información disponible sobre esta herramienta debido a ser un proyecto de código abierto con una gran comunidad [8].

Aunque la base de datos almacena por el momento la información justa y necesaria para la compra, es una herramienta que da flexibilidad al sistema ya que permitiría llevar un control sobre distintas variables como por ejemplo si un alumno ya compró su comida, cuántas veces utilizó el servicio en la semana, implementar una chequera, etc.

V. Mercado Pago Developers

Una parte fundamental en el funcionamiento de AutoTicket es la conexión con la API de Mercado Pago Developers, esta brinda la oportunidad de trabajar en la creación de todas las etapas necesarias para una integración adecuada, segura y personalizable, a través de peticiones HTTP [3].

Mercado Pago no permite trabajar con su API si no se tienen los permisos necesarios en la cuenta, y estos son validados en cada petición, a través de lo que se denomina “Acces Token”[3].

La ESP32 está vinculada a una “Caja” que posee un QR fijo, dicha “Caja” pertenece a una Sucursal creada en Mercado Pago. Estas partes, como se mencionó anteriormente, fueron creadas a través de peticiones HTTP de URL’s específicas brindadas por la API ya que este método permite un control y seguimiento mayor sobre las características e identificadores específicos de la Caja y de la Sucursal, que son necesarios para llevar un control y manejar las solicitudes.

La creación de la orden de compra se hace a través de una solicitud HTTP tipo PUT que utilizando los respectivos identificadores, carga la información en el QR de la Caja. Las variables que definen la compra, como lo son el nombre, la descripción, el código interno, el precio, etc. se definen en el body de la petición en formato JSON como se puede ver en la Fig.5.

También es importante indicar que a través de este mecanismo la orden de pago que se genera tiene el monto definido, sin la posibilidad que el usuario elija el valor, evitando dificultades a la hora de que el usuario efectúe el pago[3].

```
String body = "{"
  "\"cash_out\":{"
  "\"amount\":0"
  "},"
  "\"description\":\"Menu del Dia\","
  "\"external_reference\":\"12345\","
  "\"items\":["
    "{"
      "\"sku_number\":\"[redacted]\","
      "\"category\":\"COMIDA\","
      "\"title\":\"AutoTicket\","
      "\"description\":\"Menu del Dia comprado a traves de AutoTicket\","
      "\"unit_price\":1,"
      "\"quantity\":1,"
      "\"unit_measure\":\"unit\","
      "\"total_amount\":1"
    "}"
  "],"
  "\"notification_url\":\"" + String(url) + "/webhook/mercadopago\","
  "\"title\":\"Almuerzo\","
  "\"total_amount\":1"
"}";
```

Fig. 5: body formato json

VI. Funcionalidad

Para explicar el funcionamiento se debe tener en cuenta que AutoTicket es un sistema compuesto por 3 partes, aquella contenida dentro de la Arquitectura Física, otra compuesta por el servidor y otra por la base de datos. Ambas se comunican con la API de Mercado Pago. A continuación se detalla la serie de tareas realizada por cada una de estas partes.

A. ESP32

Las acciones realizadas por el conjunto ESP32 e interfaz de usuario se pueden dividir en tres partes fundamentales:

1. *Ingreso y comprobación del DNI*
2. *Generación del QR de pago*
3. *Recepción de la validación del pago*

1. Ingreso y comprobación del DNI

Haciendo uso de la pantalla LCD se solicita que se ingrese el DNI del alumno que está por llevar a cabo la compra del ticket.

A través del teclado matricial, el alumno ingresa su DNI digito por digito y un zumbido generado por un buzzer indica la lectura de cada dígito ingresado. Por medio de la pantalla LCD se va mostrando el DNI a medida que se ingresa. En caso de que se ingrese un número incorrecto se puede borrar, mediante el botón “D” del teclado. El programa ignora el ingreso de los botones que no son necesarios como “A”, “B”, “C”, “#” y “*”.

Una vez ingresado completamente el DNI, se genera una petición HTTP POST, hacia el servidor en Node.js. El “body” de esta petición contiene únicamente el DNI ingresado, para corroborar que el alumno se encuentre en la base de datos del sistema y evitar que una persona no registrada, pueda comprar el ticket. Luego el sistema queda a la espera de la validez del DNI ingresado y una vez que se comprobó, la respuesta de la petición indica si es válido o no.

En caso de que sea incorrecto o no se encuentre en la base de datos, el buzzer emite un sonido dando a entender que lo ingresado es invalido y por la pantalla LCD se solicita nuevamente que se ingrese un DNI.

2. Generación del QR

En el momento en que se certifica que el DNI es válido, se genera una nueva petición HTTP, pero ahora de tipo PUT, a la API de Mercado Pago Developers, que genera una orden de pago y la carga en el QR.

Una vez que está cargado el QR, la ESP32 espera la confirmación del servidor de que el usuario ha escaneado y realizado el pago. En esta instancia se contemplan dos situaciones particulares que hay que tener en cuenta, por si el alumno se arrepiente de la compra o simplemente se va.

En caso de que el alumno se arrepienta, puede cancelar la compra presionando la tecla “D”, la ESP32 genera una petición HTTP DELETE y cancela el pago. En ausencia de la confirmación de pago del usuario, y transcurrido un intervalo de tiempo predefinido, el sistema procede a cancelar automáticamente la transacción, emitiendo una solicitud HTTP DELETE. En ambos casos se indica por pantalla que se canceló la compra y también se genera un zumbido a través del buzzer.

3. Recepción de la validación del pago

La ESP32 tiene en este caso funcionalidades propias de un servidor, esto primeramente podría parecer innecesario, pero su utilidad recae en la posibilidad de escuchar peticiones desde el servidor en Node.js, esto permite que la información de la confirmación del cobro sea mucho más fluida ya que no se requiere que la ESP32 espere un tiempo y pregunte al servidor si el pago se realizó, si no que el servidor le

avisará con una petición GET que el pago se completó.

La confirmación del pago genera que la ESP32, emita una melodía, encienda un led e imprima el ticket. El formato de impresión se puede observar en la Fig.6

```
------------------
      -UNSL-
   -Auto-Ticket-
------------------
    -dd/mm/aa-
   -Hora:minuto-
------------------
      Alumno:
  Nombre-Apellido
------------------
```

Fig. 6. Formato impresión de ticket

B. Servidor en Node.js

El servidor realiza las siguientes acciones:

1. *Validación del DNI*
2. *Recepción y envío de la confirmación de pago.*

1. *Validación del DNI.*

El servidor recibe a través de una petición HTTP POST la solicitud de la ESP32 para validar la existencia de un DNI que llega en el "body" de dicha petición en formato JSON.

El servidor revisará la existencia del DNI en la base de datos y a través de la respuesta de la petición enviará una confirmación o negación.

2. *Recepción y envío de confirmación de pago.*

Cuando la orden de pago es creada y cargada en el QR, el servidor empieza a recibir información del estado de dicha orden ya que se encuentra configurado como webhook.

Webhook es un mecanismo de comunicación que permite que una web envíe notificaciones o datos en tiempo real [11]. En lugar de tener que consultar constantemente para verificar si ha sucedido algún evento, este sistema permite que la API de Mercado Pago notifique automáticamente al servidor cuando sucede algo.

Mediante la verificación de la existencia del ID de pago se comprueba que la transacción de dinero se ha completado correctamente ya que dicho ID existe únicamente al concretarse exitosamente el cobro.

Cuando el pago es confirmado, el servidor avisará mediante una petición HTTP GET a la ESP32, que se encuentra esperando.

En la Fig.7 se puede observar el funcionamiento del sistema AutoTicket a través de una Máquina de Estados Finitos (MEF).

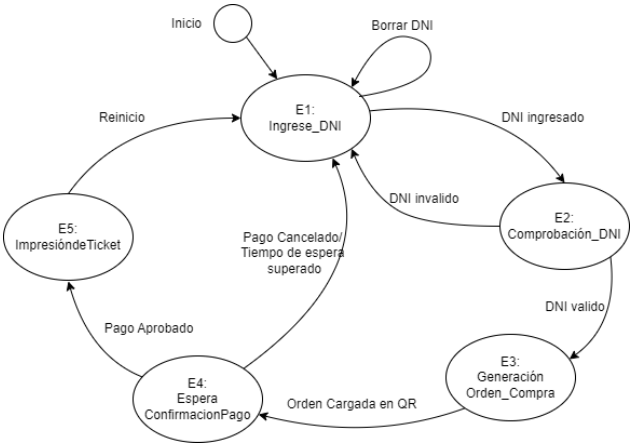

Fig. 7: Máquina de estados

VII. Resultados

Las pruebas del servidor se realizaron utilizando una herramienta llamada Ngrock.

Esta permite crear un túnel entre un puerto local en el que se encuentra el servidor y una URL accesible a través de internet ofrecida por Ngrok gratuitamente [9], de esta forma se realizaron pruebas de las distintas peticiones y también el funcionamiento como webhook del servidor. Además cuenta con una interfaz que permite observar las respuestas a cada petición, y así validar si la comunicación entre las distintas partes, funciona correctamente. En la Fig.8 se puede observar la interfaz de Ngrok con algunas peticiones realizadas.

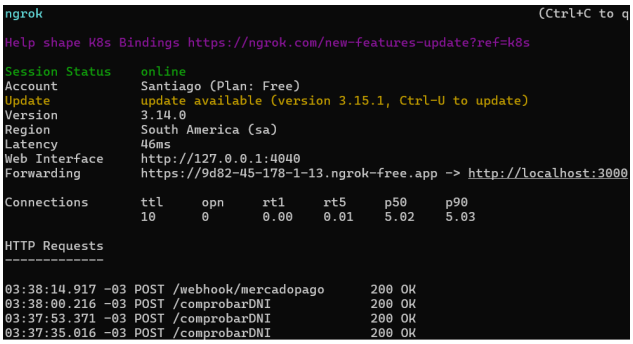

Fig. 8: Interfaz Ngrok

Inicialmente las pruebas de las peticiones a la API de Mercado Pago Developers para la creación de las sucursales, las cajas y las órdenes de pago se realizaron utilizando “Postman”, plataforma que permite realizar peticiones y ver su respuesta [10], dicha herramienta también se utilizó para probar las peticiones realizadas entre el servidor y el microcontrolador.

La integración de un buzzer probó tener gran utilidad para las pruebas ya que tener una señal sonora del correcto funcionamiento permitió mantener mayor fluidez en las pruebas, por lo que se optó por incluirlo definitivamente en el sistema [6].

Se ha elaborado una maqueta, Fig. 8 para representar y demostrar el funcionamiento del sistema, permitiendo visualizar de manera tangible cómo operan todos los componentes y cómo se integran en el sistema general. Se puede observar en la Fig.9.

Fig. 9. Maqueta Funcional AutoTicket

En el QR de la Fig.10, se puede visualizar algunas pruebas hechas y varias situaciones posibles que se pueden dar, las cuales fueron explicadas anteriormente.

Fig. 10. Pruebas de AutoTicket

VIII. ESCALABILIDAD

Inicialmente, el desarrollo de este proyecto se pensó con un único objetivo, tal como se menciona en la introducción. Sin embargo, a medida que el proyecto avanzaba, se hizo evidente su potencial para escalar, revelando su adaptabilidad a diversas aplicaciones relacionadas con la venta automatizada.

Esta adaptabilidad se manifiesta tanto en las implementaciones de hardware como de software. Por ejemplo, el sistema podría incorporar autenticación biométrica, como reconocimiento de huellas dactilares o tecnología RFID, como alternativas al teclado matricial. En cuanto al software, la flexibilidad del sistema permite su operación con cualquier tipo de servidor, facilitando la gestión de múltiples ofertas de venta y variaciones de precios.

La escalabilidad de este proyecto abre numerosas posibilidades. Podría ampliarse para coordinar múltiples sistemas de "AutoTicket", permitiendo que todas las transacciones se dirijan a la misma cuenta o a diferentes cuentas según sea necesario.

El proyecto ofrece la capacidad de satisfacer una amplia gama de necesidades, con el potencial de soportar aplicaciones mucho más amplias de lo inicialmente previsto.

IX. Conclusiones

En este proyecto se desarrolló un sistema con las capacidades necesarias para poder llevar a cabo ventas automatizadas.

Se pudo implementar el sistema completo dentro de una ESP32 y un servidor en JavaScript dando la posibilidad de probar el sistema con pocos recursos.

Se han cumplido los objetivos inicialmente propuestos, dando solución a la problemática. Además en el proceso se reveló el potencial que brindan la combinación de las tecnologías utilizadas, por lo que se puede asegurar que AutoTicket tiene mucho potencial por explotar.

Referencias

[1] Espressif Systems. (2024). ESP32 datasheet. Obtenido de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[2] OpenJS Foundation. (2024). Node.js. Obtenido de https://nodejs.org/en/about

[3] Mercado Libre. (2024, 8 de agosto). Mercado Pago Developers. Obtenido de https://www.mercadopago.com.ar/developers/es/reference

[4] Libreria WiFi.h. y HTTPclient.h Obtenido de https://github.com/espressif/arduino-esp32/blob/master/libraries/README.md#httpclient

[5] Parallax Inc. (2011, 16 de diciembre). Product documentation. Obtenido de https://cdn.sparkfun.com/assets/f/f/a/5/0/DS-16038.pdf

[6] (s.f.). Obtenido de https://www.quick-teck.co.uk/Management/EEUploadFile/1420788438.pdf

[7] VISHAY. (2013, 18 de marzo). LCD016N002BCFHE. Obtenido de https://www.vishay.com/docs/37484/lcd016n002bcfhet.pdf

[8] SQL. (2024, 10 de agosto). SQLite. Obtenido de https://www.sqlite.org/index.html

[9] Ngrok Docs. Obtenido de https://ngrok.com/docs/

[10] What is Postman?. Obtenido de https://www.postman.com/home

[11] Webhooks to revolutionize the web. Obtenido de https://web.archive.org/web/20150826052314/http://progrium.com/blog/2007/05/03/web-hooks-to-revolutionize-the-web/

[12] OpenJs Foundation. (s.f.). Express. Obtenido de https://expressjs.com/

[13] OpenJs Foundation. (s.f.). Express. Obtenido de https://github.com/expressjs/body-parser

[14] The Axios Project: Copyright © 2014-present Matt Zabriskie and contributors. (s.f.). Axios. Obtenido de https://axios-http.com/es/

[15] Libreria Keypad.h. Obtenido de https://www.arduino.cc/reference/en/libraries/keypad/

[16] Libreria LiquidCrystal_I2C.h. Obtenido de https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/