Taller de Proyecto II
Sistema RFID

Proyecto Nº 5

Basanta, Sofía - 524/1
Discoli, Tomas - 543/4
Minig Traverso, Marcelo - 489/7

UNLP
Facultad de Informática
Instituto de Investigación en Informática - LIDI

La Plata, 5 de Octubre de 2017

Profesor:
Tinetti, Fernando G.

Ayudantes:
Marón, Gastón Ariel
Meroni, Milton Alberto

Contenido

1. Propuesta Original del Proyecto

En la propuesta original se propuso implementar una API que permita potenciar el desarrollo de futuras aplicaciones que interactúen con RFID, proporcionando la estructura base para las mismas.

El objetivo es proveer a futuras camadas de alumnos con un sistema autocontenido que posea los endpoints más comunes a la hora de diseñar una plataforma y que utilice identificación por radiofrecuencia como entrada fundamental del sistema. Lateralmente, se desarrollará un sistema que haga uso de esta API para que pueda apreciarse su potencial. Para ello se propuso un sistema de *trazabilidad de ganado*. El mismo consiste en poner en las orejas de los animales tags RFID para que al pasar por una manga, un lector pueda identificar al animal y permitir ver a los trabajadores rurales la información concerniente al mismo.

Los dispositivos previstos para implementar el proyectos fueron los siguientes:

- Microcontrolador Arduino Uno R3
- Módulo ESP8266
- Módulo RFID RC522 1356 mhz
- Llaveros o tarjetas RFID

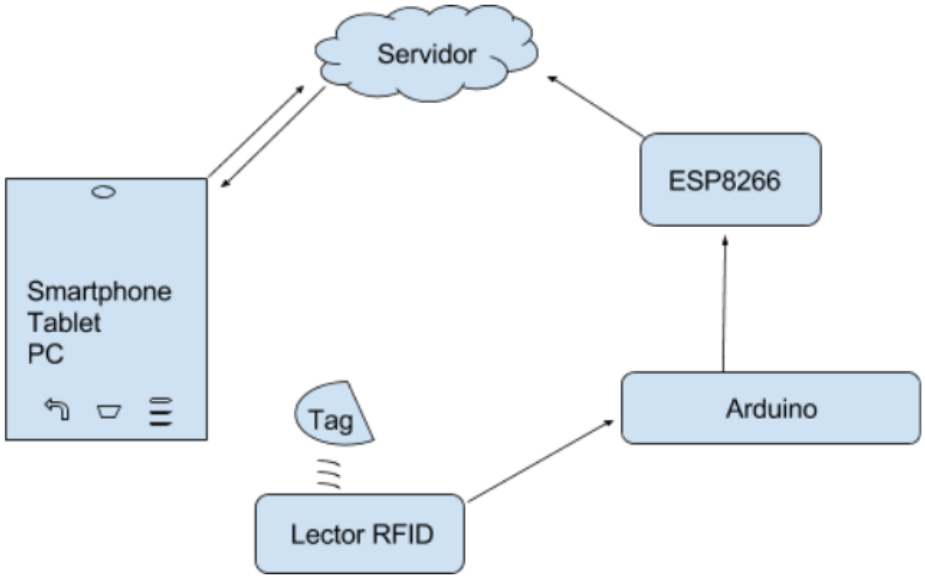


Figura 1: Diagrama de la interacción de los componentes de sistema.

Flujo de la información:

En el diagrama de la Figura 1 se muestra la interacción básica de los componentes del sistema y cómo interactúan entre ellos pasándose información.

En el Apéndice A se encuentra la propuesta original completa presentada a la cátedra.

2. Correcciones y Cambios de la Propuesta

2.1. Indicadas por la Cátedra

- Ante la propuesta opcional de implementar el sistema con una Raspberry Pi 3 Model B la cátedra nos indicó que la solución debía ser implementada con Arduino.
- Se nos indico que el módulo ESP8266 tenía que estar configurado como Access Point haciendo que el mismo genere un red WIFI propia y el sistema funcione dentro de ella. El motivo de esto es facilitar las pruebas, ya que el sistema no depende de una red WIFI ajena y al mismo tiempo el sistema está autocontenido.

2.2. Definidas por el Avance y la Disponibilidad

- Algunas de las pruebas por disponibilidad de materiales se realizan sobre un Arduino Mega en vez de hacerse todo sobre el Arduino UNO propuesto.
- Como sistema de ejemplo para el proyecto se eligió el de trazabilidad del ganado en el que los animales poseen un tag RFID que los identifica y facilita el acceso a la información del mismo. El esquema de la Figura 2 ilustra la idea del sistema planteado.

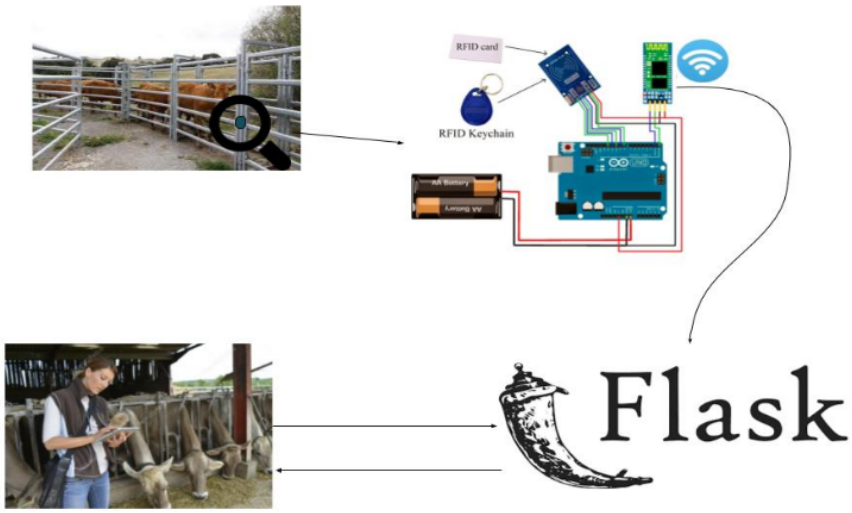


Figura 2: Esquema del sistema de trazabilidad de ganado.

3. Descripción de Hardware y Conexiones

El Arduino UNO se conecta con dos dispositivos. El primero es la placa lectora RFID que se conecta por SPI y funciona como entrada de información, capturando los IDs de los tags. El otro módulo es el ESP8266 que se conecta mediante el puerto serie (UART), se utiliza para crear la red WIFI del sistema al funcionar como AP y para generar peticiones HTTP que se envían al servidor (PC).

Para realizar el proyecto primeramente planteamos el esquema de conexiones que se ve en la Figura 3.

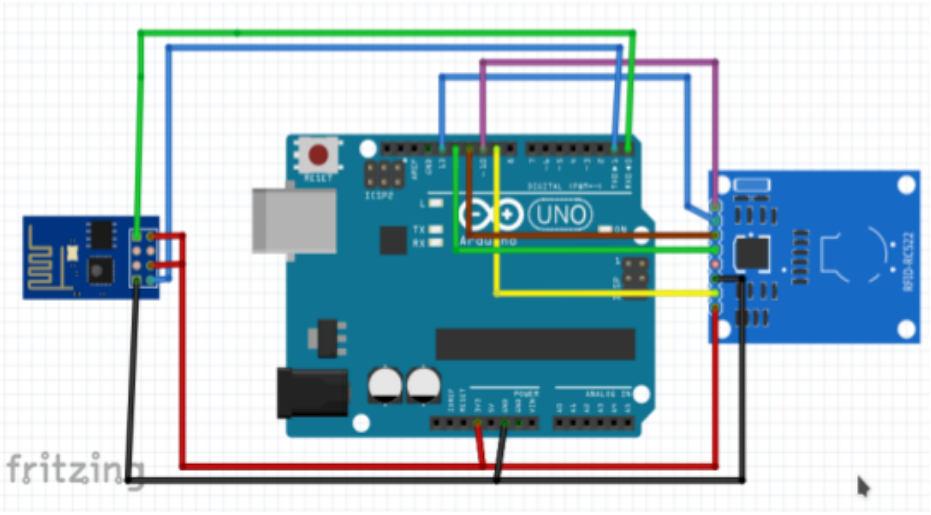


Figura 3: Conexiones del Arduino UNO con el ESP8266 y el RFID-RC522

En la Figura 4 se muestra el diseño que realizamos para desarrollar un prototipo de pruebas, la implementación del mismo se se puede observar en las Figuras 5 y 6.

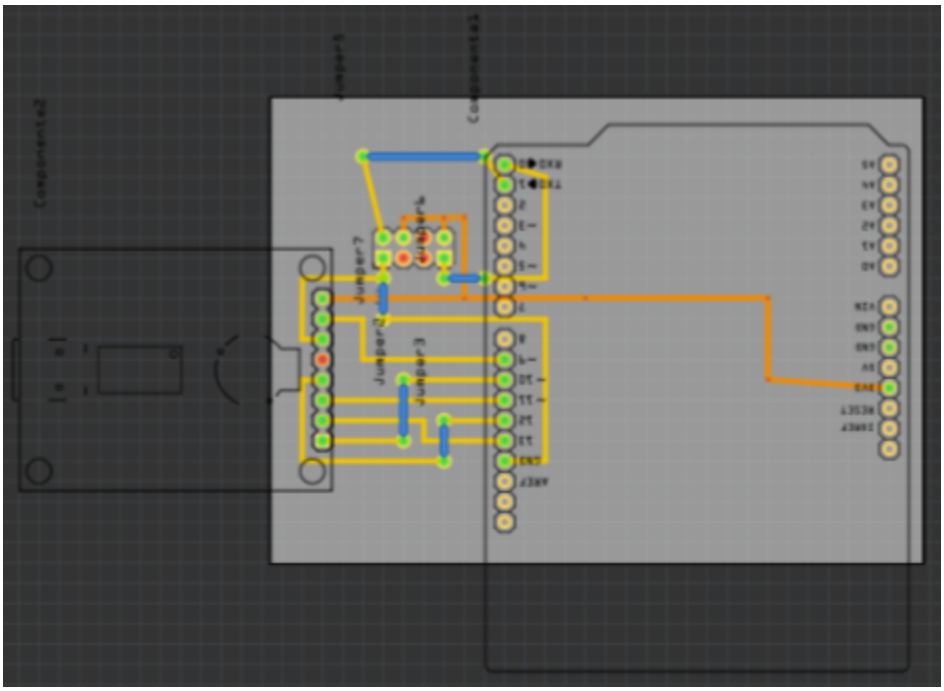


Figura 4: Esquemático para el prototipo usando Arduino UNO

Notar que el cable verde va conectado al pin 2 y el azul al 3 del Arduino UNO.

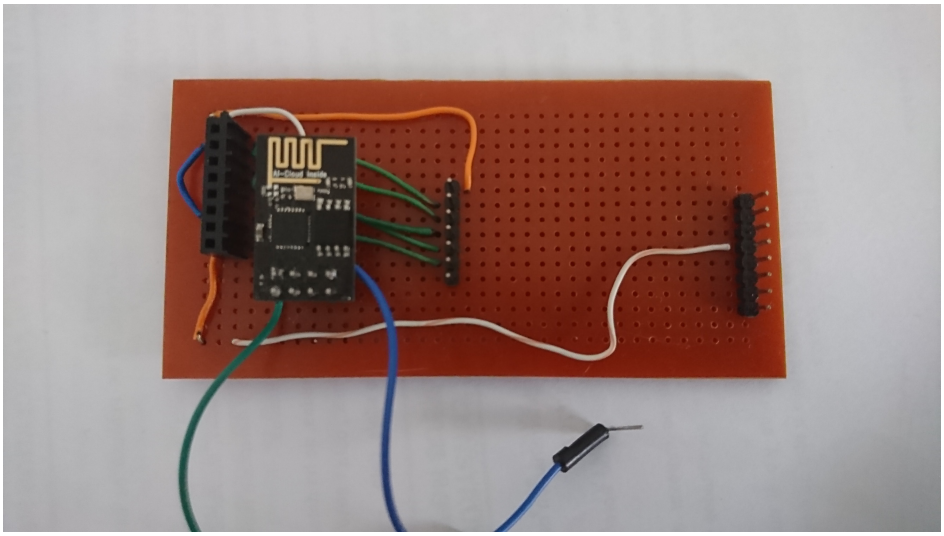


Figura 5: ESP montado sobre el prototipo.

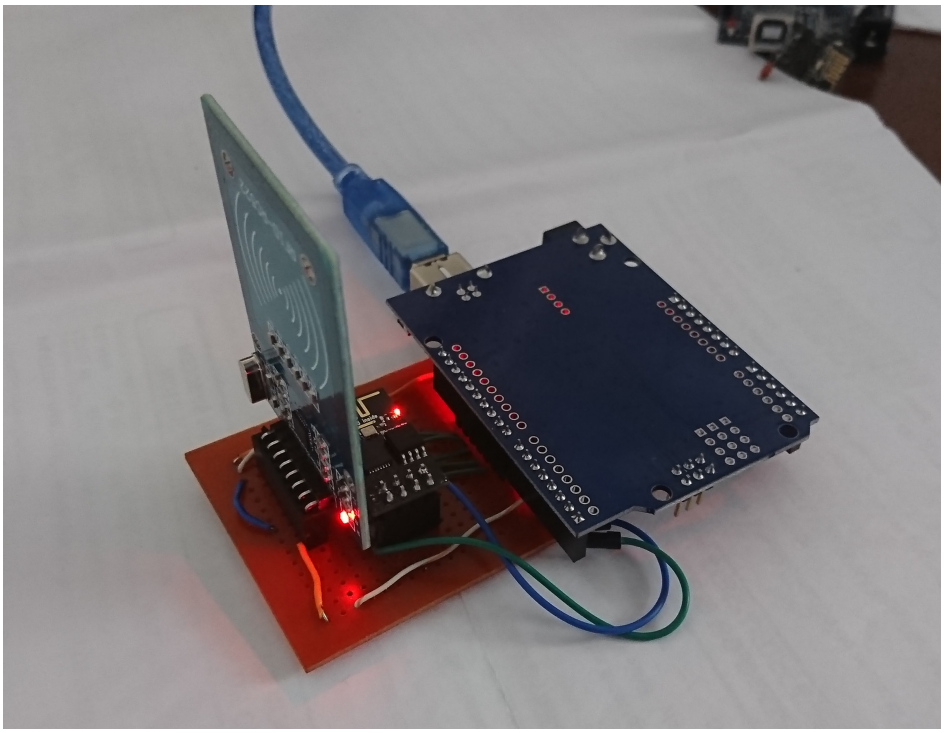


Figura 6: Foto del prototipo en funcionamiento.

4. Descripción Funcional

En esta sección se hace la distinción entre usuarios y clientes para poder dar una descripción de la funcionalidad del proyectos. Los usuarios son los que disponen de los identificadores RFID que el sistema se encarga de leer, mientras que los clientes son los que hacen uso del sistema para ver la información detectada por radiofrecuencia.

En la implementación del proyecto que hace la trazabilidad del ganado, los usuarios serían el ganado y los trabajadores rurales los clientes.

El sistema se divide en dos partes principales, la primera está compuesta por dispositivos de hardware que se encargan de la lectura por radiofrecuencia de las tarjetas que identifican a los usuarios. La segunda parte es el servidor web donde se encuentra almacenada toda la información y es el que hace de intermediario entre la información que recibe de los usuarios y los clientes que pretenden disponer de ésta.

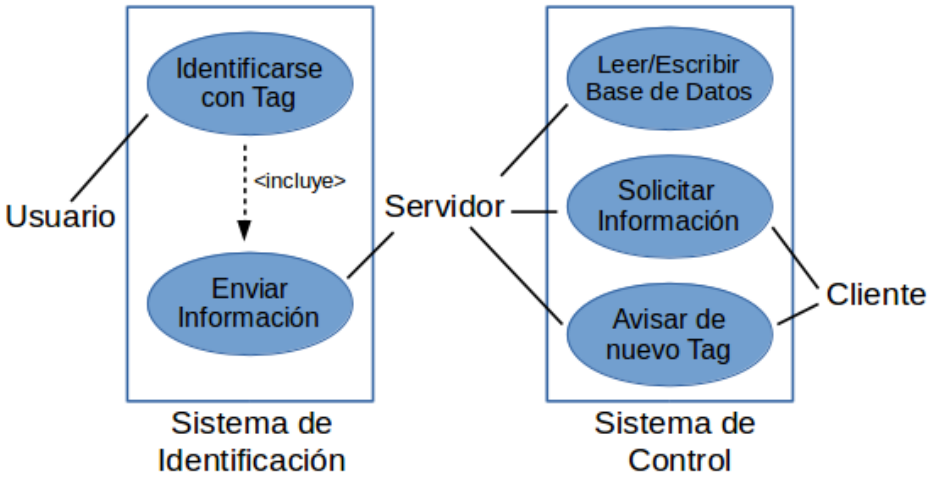


Figura 7: Casos de uso del sistema.

El diagrama de la Figura 7 muestra los casos de uso que representan al sistema dividido en sus dos partes, y la forma en la que los actores interactúan entre ellos. Se ve cómo es que un usuario puede identificarse mediante su tarjeta RFID y esta información es enviada al servidor. Al mismo tiempo, este último almacena la información en la base de datos para poder informarla a los clientes cuando estos la requieran.

En la Figura 8 se puede ver una descripción más detallada de este proceso, de los distintos componentes que lo conforman y del sentido en el que circula la información entre ellos.

Al acercar una tarjeta RFID al lector, éste la detecta y obtiene su ID. El microcontrolador Arduino lo procesa e inicia la comunicación con el servidor a través del ESP8266. Cuando el servidor recibe estos datos los persiste en la BD utilizando las funciones que provee API RFID. Al mismo tiempo, el servidor se encarga de responder las distintas peticiones que recibe de los clientes.

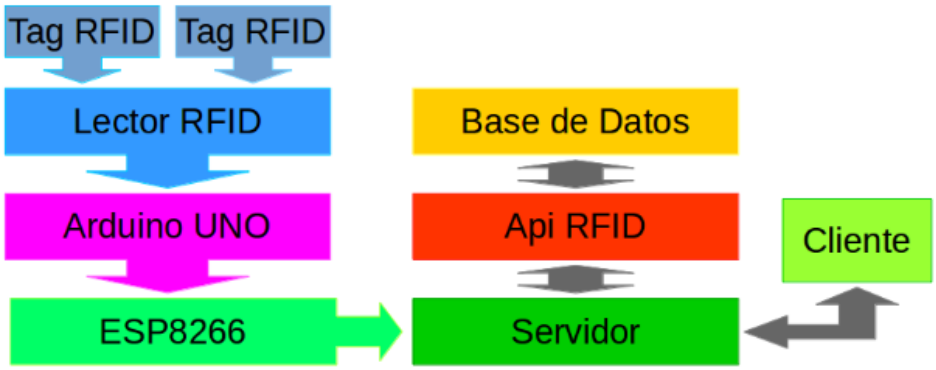


Figura 8: Esquema del flujo de la información.

Por último, la base de datos que implementamos es la que se puede ver en la Figura 9. Esta contiene tres tablas.

- **USERS:** Esta tabla almacena los datos de todos los usuarios, entre ellos se encuentra el identificador RFID llamado PICC.
- **LOGS:** Guarda todos los registros RFID detectados por el sistema. Asociándolos a un USER y un DEVICE.
- **DEVICES:** Tiene la información de los distintos dispositivos RFID conectados al sistema, ya que podría haber mas de un lector de tarjetas RFID.

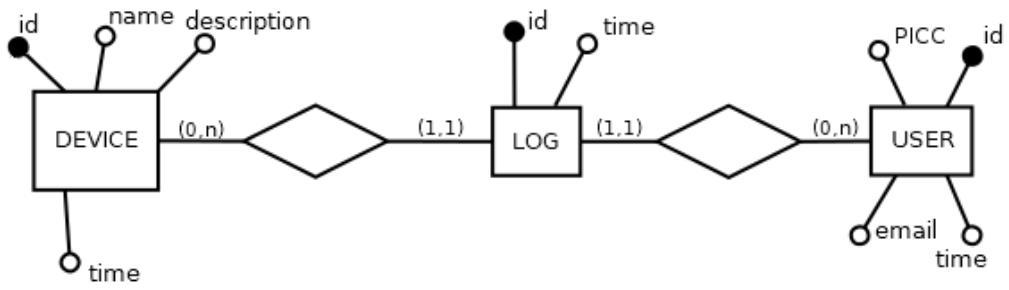


Figura 9: Modelo de la Base de Datos.

5. Descripción del Software

El software que desarrollamos se pueden subdividir en dos, uno para el sistema web y otro asociado al hardware.

5.1. Software del Sistema Web

5.1.1. El servidor web:

Este cuenta con una serie de rutas o endpoints para brindar sus servicios, estas se listan a continuación.

- /: es la ruta raíz del servidor, se encarga de renderizar un archivo HTML que muestra un listado de los registros RFID almacenados en el sistema.
- **/cow/**: muestra la información de un usuario con su lista de registros. En este caso los usuarios son vacas.
- **/newCow**: es un formulario que permite crear un nuevo usuario para guardarlo en el sistema.
- **/newCowForm**: es la ruta que procesa el formulario de un nuevo usuario y lo persiste en la BD.
- **/newLog**: por esta ruta se recibe y guarda la información que envían los distintos dispositivos RFID que forman parte del sistema. Recibe el ID que se leyó con un número para identificar al dispositivo que envía los datos.
- **/refreshData**: el objetivo de esta ruta es que los clientes, mediante AJAX, puedan solicitar al servidor por nueva información que les sea útil. En este caso se usa para preguntar si se detectó algún registro nuevo en el sistema.
- **/login**: permite a los clientes empezar a usar el sistema, guardando cierta información en la sesión de servidor.
- **/logout**: termina con la sesión del cliente.

5.1.2. API RFID:

El servidor maneja toda la información sobre los registros, IDs y usuarios a través de funciones provistas por esta API. Esto le permite abstraerse de la implementación de BD y otras cuestiones, para poder focalizarse en resolver la problemática para la que fue creado. A continuación se describen algunas de las funciones más importantes que contiene la API.

- ***RFID_getUserByPicc()***: esta función devuelve los datos de un usuario a partir del ID de una tarjeta.
- ***RFID_getAllLogs()***: devuelve todos los registros almacenados en el sistema.
- ***RFID_getLogsByUser()***: devuelve todos los registros de un determinado usuario.
- ***RFID_getLogsByDate()***: entrega todos los registros que se encuentren entre dos fechas pasadas como parámetros.

- ***RFID_addUser()***: agrega un nuevo usuario a la BD.
- ***RFID_delLogsByUser()***: elimina todos los registros de un determinado usuario.
- ***RFID_changeAdminPass()***: permite cambiar la contraseña del administrador, la cual es requerida cuando se quieren hacer cambios en la BD.

El resto de las funciones que componen la API son muy similares a las ya descritas. La documentación completa se encuentra junto al código en el repositorio.

5.2. Software Asociado al Hardware

5.2.1. Funciones asociadas al lector RFID:

Las siguientes funciones no fueron implementadas por nosotros, sino que están contenidas en la libreria MFRC522 (https://github.com/miguelbalboa/rfid)

- ***rfid.PCD_Init()***: esta función devuelve los datos de un usuario a partir del ID de una tarjeta.
- ***rfid.PICC_IsNewCardPresent()***: Devuelve true si el PICC responde a PICC_CMD_REQA
- ***rfid.PICC_ReadCardSerial()***: Devuelve true si un UID pudo ser leído.
- ***rfid.PICC_GetType(rfid.uid.sak)***: Convierte el SAK (Select Acknowledge) a un tipo PICC.
- ***rfid.PICC_HaltA()***: Fuerza al PICC en estado ACTIVE(*) a entrar en estado HALT.
- ***rfid.PCD_StopCrypto1()***: Usado para sacar al PCD de su estado autenticado. Está función debe llamarse después de comunicarse con el PICC autenticado, de lo contrario no se podrán establecer nuevas comunicaciones.

Una explicación más detallada de las funciones de la librería podrá ser encontrada en el repositorio de la misma.

5.2.2. Funciones asociadas al ESP8266:

Para trabajar con el ESP no se utilizó una librería, sino que se implementaron las funciones deseadas mediante el uso de comandos AT.

- ***setWifi()***: La configuración del módulo consta de 3 pasos: primero, establecer el modo de funcionamiento del ESP como station y accespoint(los dos en conjunto); segundo, establecer los parámetros del access point(el SSID, la contraseña, el tipo de encriptación y el canal); y tercero definir la dirección de host(192.168.4.1).
- ***httppost()***: Esta función realiza el post sobre el servidor, persistiendo el ID registrado, junto con el dispositivo que lo detectó. Para ello se realizan, mediante comandos AT, la siguiente secuencia de pasos.
    - Se define el tipo de conexión(TCP), la dirección del servidor (192.168.4.2) y el puerto(8002).
    - Se definen el número de caracteres a enviar(estos se calculan en función del cuerpo del post request).
    - Cuando se recibe la confirmación de que el envío fue correcto se cierra la conexión.

5.2.3. Funciones asociadas al Arduino:

- ***setup()***: Configura los baud rates del ESP y del Serial (utilizado para debuguear). Inicializa el bus SPI y llama a la función de inicialización de la librería MFRC522, así como también a la función setWifi() definida anteriormente.
- ***loop()***: Bucle infinito donde se consulta si hay una nueva tarjeta, y en caso de haberla realiza el post al endpoint del servidor que registra nuevos logs.

5.2.4. Pseudocódigo

```
Inicializar (baudrates);
Inicializar (MFRC522);
Inicializar (WIFI);
while true do
    if hay nueva tarjeta presente then
        Realizar el post al servidor para persistir el log;
    end
end
```

6. Guía de Instalación Completa

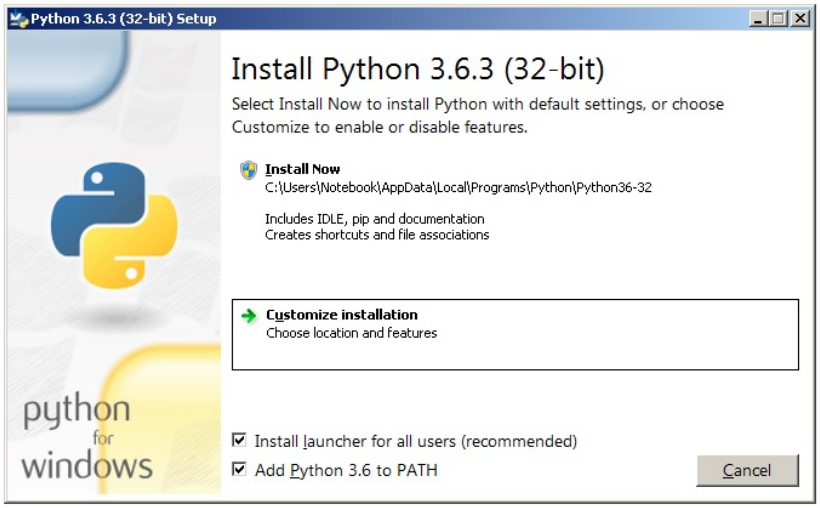
6.1. Ambiente de desarrollo

Para todo el proyecto se utilizaron herramientas de desarrollo libres, gratuitas y accesibles en distintos sistemas operativos.

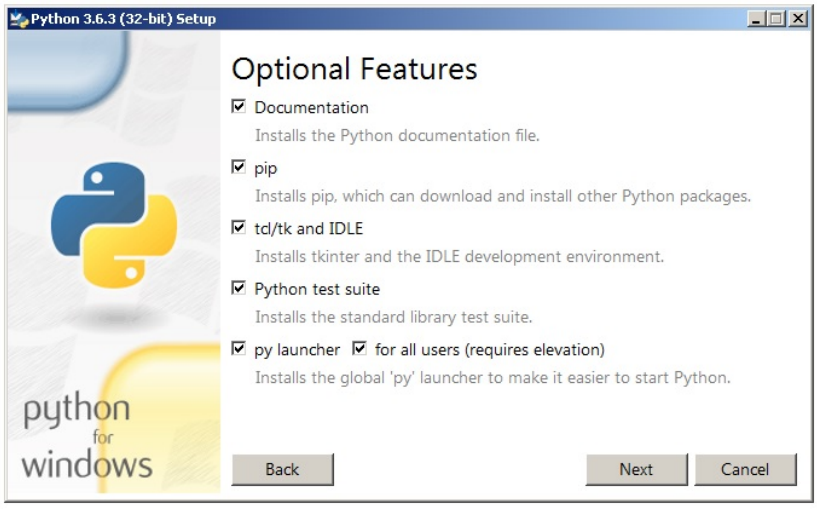
Python - Pip - Flask

Para la instalación en Windows, ingresar al siguiente enlace y descargar el software: https://www.python.org/downloads/, al finalizar la descarga ejecutar el archivo y seguir los pasos de instalación.

Una ventaja de instalar las nuevas versiones de Python es que incluyen pip.



Nota: Agregar Python en las variables de entorno del sistema. Para esto, hay dos formas, la primera tildar la opción "Add Python 3.6 to PATH". O, hacer click derecho sobre el icono de Equipo*Z*Propiedades. En la ventana que se abre buscar la opción "Configuración avanzada del sistema" situada en la esquina superior izquierda. Luego, hacer click en el botón "Variables de Entorno" y nos dirigimos a "Variables del sistema" y en Path agregamos la ruta donde se instaló Python.



Por último, para instalar Flask abrimos la consola de Windows y ejecutamos el siguiente comando: **pip install flask**

MySQL

Para administrar la base de datos usamos MySQL, en el siguiente enlace descargamos el instalador para Windows: https://dev.mysql.com/downloads/installer/.

Ejecutamos el instalador y seguimos los pasos de instalación.

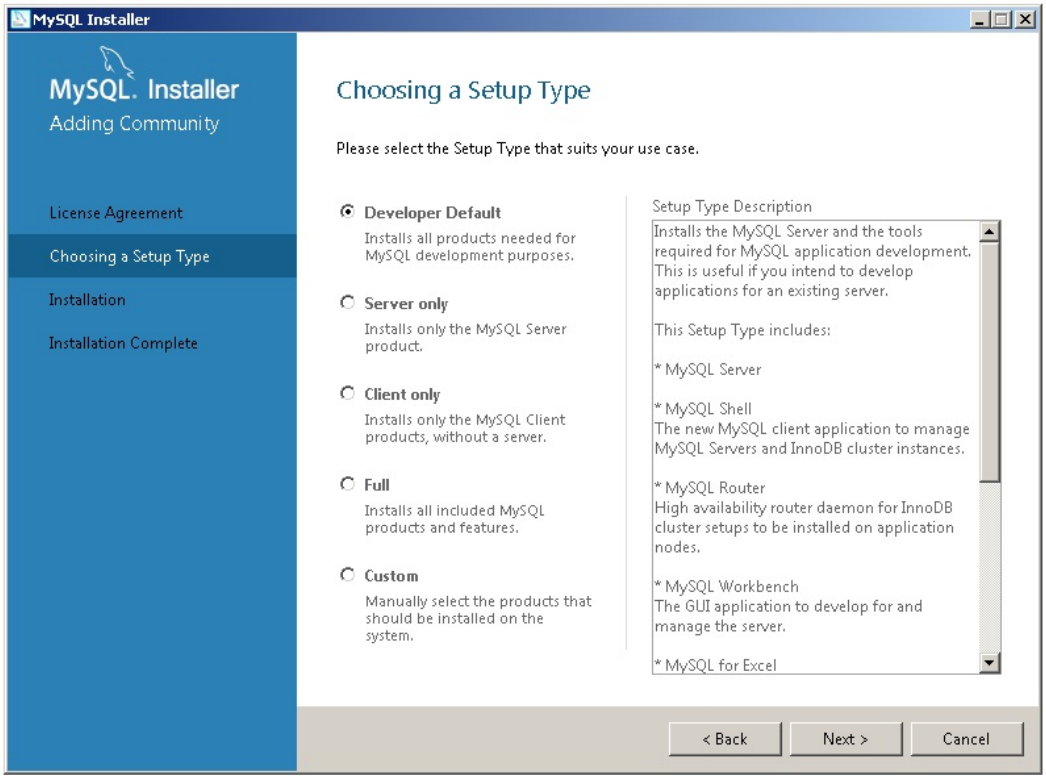


Figura 10: Instalación de MySQL.

Python - Pip - Flask - MySQL

En Linux, para instalar Python y Pip ejecutamos los siguientes comandos en consola:

sudo apt-get install python
sudo apt-get install python-pip

A partir de estas instalaciones, podemos instalar Flask: **sudo pip install Flask**

También se puede crear el archivo "requirements.txt" que contiene las dependencias necesarias para poder levantar un servidor con Flask, cuyo contenido se muestra a continuación:

Flask
Flask-mysqldb

Y luego ejecutar el siguiente comando: **pip install -r requirements.txt**

Asimismo, instalamos MySQL, para administrar la base de datos, a partir del siguiente comando:

sudo apt-get install mysql-server libmysqlclient-dev

Creamos una carpeta con los archivos requeridos para levantar un servidor Flask descargándolos del repositorio. Por último ejecutamos el comando para crear y configurar la base de datos:

mysql -u root -p RFID_BD.sql

Arduino IDE

Los programas dedicados al Arduino se desarrollaron en el IDE que esta misma plataforma provee y se hizo uso de la librería MFRC522 que permite interactuar con el lector RFID. Para monitorear lo que hace el Arduino por el puerto serie (UART) se utiliza el monitor serie incluido en el IDE de Arduino.

Instalación

Para instalar el software de Arduino, primero se debe descargar desde el siguiente enlace: https://www.arduino.cc/en/main/software el instalador.

En el caso de Windows, al finalizar la descarga ejecutar el archivo y seguir los pasos de instalación. Puede que durante la instalación se requieran instalar algunos drivers.

Figura 11: Instalación del Arduino IDE.

En el caso de Linux, luego de descargar el software correspondiente se debe extraer a una carpeta desde donde será ejecutado. Abrir la carpeta recién creada y ejecutar el script install.sh.

Tanto en Windows como en Linux:

Para el proyecto se hizo uso de la librería MFRC522 que permite interactuar con el lector RFID, la cual se puede instalar con el gestor de librerías integrado en el IDE. Para ello luego de la instalación se debe ejecutar el software e ir a: Programa > Incluir Librería > Gestionar Librerías. Allí buscar MFRC522, elegir una versión y clickear instalar.

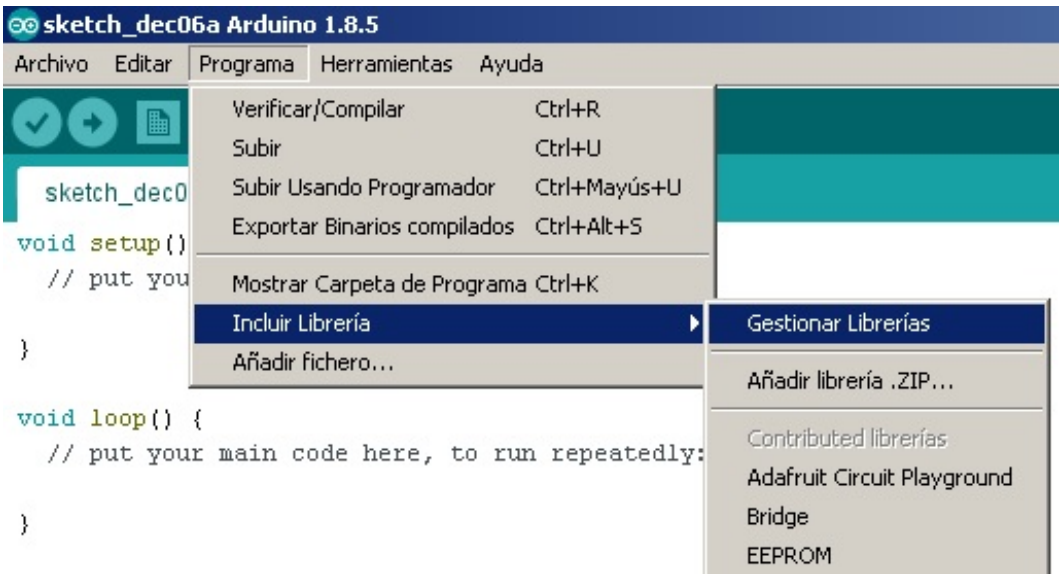


Figura 12: Instalación de la librería MFRC522.

6.2. Copia e Instalación de Código:

Clonar el repositorio donde se encuentra el proyecto completo: https://github.com/tomasdisk/Proyecto2

Para hacer funcionar el sistema solo hay que realizar los siguientes cuatro pasos:

1. Hay que cargar el programa final.ino, disponible para descargar desde GitHub: https://github.com/tomasdisk/Proyecto2/tree/master/Proyecto/Arduino/final en el Arduino UNO con el IDE arriba mencionado y mantenerlo encendido con una fuente de alimentación.

2. Ejecutar el comando: pip install -r requirements.txt

3. Crear la base de datos con el script RFID_BD.sql. Disponible en: https://github.com/tomasdisk/Proyecto2/tree/master/Proyecto/Server.

4. Iniciar el servidor por consola con el comando: python app.py.

A su vez, se adjunta el software desarrollado junto con este informe un archivo en formato .zip.

6.3. Guía de Compilación e Instalación de Ejecutable:

Para trabajar con el módulo ESP se implementó una función setWifi(), la cual configura el ESP8266, dicha función fue explicada en la sección 5.2.

A su vez se implementó una función setup(). La cual configura los baud rates del ESP y del serial (utilizado para debuguear), inicializa el bus SPI, llama a la función de inicialización de la librería MFRC522 y la función setWifi().

Por otro lado, hay que modificar en el archivo *app.py* (seccion Config DB) con el usuario y la contraseña correspondientes en cada caso, para utilizar MySql.

A. Propuesta Original Completa

A.1. Introducción

Se propone implementar una API que permita potenciar el desarrollo de futuras aplicaciones que interactúen con RFID, proporcionando la estructura base para las mismas.

Existen varios sistemas de identificación. Se decidió utilizar RFID debido a las ventajas que éste presenta, algunas de las cuales se listan a continuación:

- Ofrece una gran variedad de aplicaciones y usos.
- Ofrece la posibilidad de disponer de un mayor control en todo el proceso de distribución de bienes muebles.
- Es menos vulnerable al daño que otros sistemas ya que una etiqueta RFID se coloca de forma segura dentro de un objeto o incrustado en plástico, lo que permite al sistema ser utilizado en una variedad de ambientes hostiles.
- No requiere visión directa de lo que hay que descifrar comparado con las tecnologías de código de barras e infrarrojo. Haciendo que las lecturas sean más rápidas y precisas.
- Es portable: la transmisión de datos a través de una pequeña etiqueta (que es un transmisor portátil).
- Aunque cuesta más implementar que un sistema de código de barras, ofrece un buen retorno de la inversión en el largo plazo, ya que la RFID es significativamente más eficiente.

A.2. Objetivo

El objetivo es proveer a camadas futuras de alumnos con un sistema autocontenido que posea las rutas más comunes a la hora de diseñar una plataforma que utilice identificación por radiofrecuencia como entrada fundamental del sistema. Lateralmente, se desarrollará un sistema que haga uso de esta API para que pueda apreciarse su potencial.

Al estar el foco puesto en la implementación de la API como una piedra angular de propósito general, es irrelevante cuál sea el sistema que haga uso de ella. No obstante, se proponen las siguientes alternativas:

Sistema de trazabilidad de ganado:

Consiste en poner en las orejas de los animales tags RFID para que al pasar por una manga, un lector pueda identificar al animal y permitir ver a los trabajadores rurales la información concerniente a su calendario de vacunación, peso, edad, período de lactancia (si lo hubiera), y demás variables pertinentes.

Sistema de notificación inmobiliaria:

Se fundamenta en la necesidad de simplificar el cobro y pago de los alquileres por parte de las inmobiliarias. Un inquilino presenta una tarjeta RFID al momento de pagar, el sistema informa al operario el monto que debe cobrarle y cuando el pago es

confirmado se envía automáticamente un email de aviso al dueño del inmueble pertinente, generando un historial de pago para futuras referencias.

Sistema de validación de tiques en comedores universitarios:

Pretende resolver la congestión que se genera en las sedes más concurridas al momento de retirar el tickets para la vianda. El sistema verifica que un estudiante tenga pagada su ración de comida al momento de presentar el identificador RFID.

A.3. Dispositivos a utilizar

• **Arduino Uno R3:**

Placa de programación que se utilizará para procesar los tags leídos y hacer las peticiones GET y POST al servidor. Costo estimado $250 (por una unidad). Figura A.1a.

• **Llavero o tarjeta RFID:**

Tags RFID de 13.56 MHz. Costo estimado $50 (por cinco unidades). Figura A.1b.

• **Módulo RFID RC522 13.56 MHz:**

Lector de proximidad que utiliza el protocolo SPI. Costo estimado $100 (por una unidad). Figura A.1c.

• **ESP8266 o Ethernet shield:**

Componente para hacer los request. Costo estimado $250 (por una unidad). Figura A.1d.

• **Raspberry Pi 3 Model B (OPCIONAL):**

Se incluye como opcional porque su utilización sería en reemplazo del "**Arduino Uno R3**" y del "**ESP8266 o Ethernet shield**" por ser una computadora con WIFI integrado y que soporta el protocolo de comunicación SPI. Costo estimado $1000 (por una unidad). Figura A.1e.

A.4. Identificación de partes

E/S del controlador con el exterior, exceptuando PC:

El controlador tendrá como entrada la información que proporcione el lector RFID a través del canal SPI, y como salida hará peticiones a un servidor mediante el módulo ESP8266, conectado por RS232.

Comunicaciones con la PC:

La PC o dispositivos móviles se utilizarán para acceder a la información mostrando en pantalla(de forma responsiva) los datos asociados a la última lectura, y proporcionando formularios de alta, baja y modificación en caso de contar con permisos administrativos.

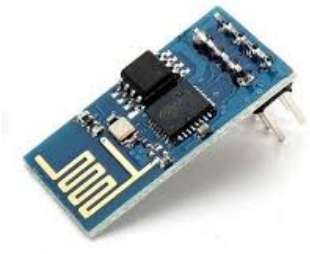
Servidor:

Consistirá en una aplicación Python desarrollada en Flask, que proveerá las rutas asociados a operaciones GET, POST, PUT y DELETE, persistiendo y leyendo desde una base de datos relacional MySQL.

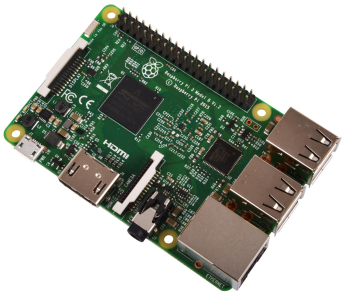
(a) Arduino Uno R3.

(b) Llavero o tarjeta RFID.

(c) Módulo RFID RC522.



(d) Módulo ESP8266.



(e) Raspberry Pi 3 Model B.

Figura A.1: Dispositivos a utilizar

B. Código del Sistema Web

B.1. app.py

Código del servidor web.

```
# Aqui se listan los imports
from flask import Flask
from flask import render_template
from flask import request
from flask import redirect
from flask import url_for
from flask import session
from flask import jsonify
from flask_mysqldb import MySQL
from datetime import datetime
import RFID_Api

app = Flask(__name__)

# Config DB
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'tomasdisk'
app.config['MYSQL_DB'] = 'RFID_BD'
app.config['MYSQL_CURSORCLASS'] = 'DictCursor'
# init DB
mysql = MySQL(app)

# Define la ruta y metodo con el que se debe llegar a este endpoint
@app.route('/')
def home():

    logs = RFID_Api.RFID_getAllLogs(mysql);

    # renderiza la pagina correspondiente con los parametros dados
    return render_template('home.html', logs=logs)

# recibe la peticion de nuevos datos por AJAX y los devuelve como JSON
@app.route('/refreshData', methods = ['GET'])
def refreshData():

    data = {
    'new' : "",
    'known': "",
    'picc': "",
    }
    logs = RFID_Api.RFID_getLogsByDate(mysql, datetime.now(),
  session['log_update']);

    if logs:
        data['new'] = "true"
        log = logs[0]

        data['picc'] = log['PICC'] # asignar picc del registro obtenido
        session['log_update'] = log['registrated_at']
        if RFID_Api.RFID_getUserByPicc(mysql, data['picc']):
            data['known'] = "true"
        else:
            data['known'] = "false"
```

```
55          return jsonify(data)
        else:
57          data['new'] = "false"
            return jsonify(data)
59
# ruta que muestra el perfil de un usuario(vaca)
61  @app.route('/cow/<string:id>', methods = ['GET'])
    def cow(id):
63
        cow = RFID_Api.RFID_getUserByPicc(mysql, id)
65      if cow:
            logs = RFID_Api.RFID_getLogsByPicc(mysql, id)
67          return render_template('cow.html', cow=cow, logs=logs)

69      return redirect(url_for('home'))

71  # ruta con formulario para cargar un nuevo usuario(vaca)
    @app.route('/newCow', methods=['GET', 'POST'])
73  def newCow():

75      data = request.values
        if 'picc' in data:
77          picc = data["picc"]
            # renderiza la pagina correspondiente con los parametros dados
79          return render_template('newCow.html', picc=picc)

81      return redirect(url_for('home'))

83  # ruta que procesa el formulario de un nuevo usuario(vaca)
    @app.route('/newCow/form', methods = ['POST'])
85  def newCowForm():

87      data = request.values
        #app.logger.info(data)
89      if 'picc' in data and 'description' in data and 'count' in data:
            if RFID_Api.RFID_addUser(mysql, data['picc'], data['description']
91      , data['count'], "admin"):
                # renderiza la pagina correspondiente
93              return redirect(url_for('cow', id=data['picc']))

95      return redirect(url_for('home'))

97  # ruta altenativa sin contenido
    @app.route('/newLog', methods = ['POST', 'GET'])
99  def newLog():

101     if request.method == 'POST':
            log = request.values
103         app.logger.info("POST\nPICC: " + log["picc"] + "\nDevice: " +
        log["device"])
105         if(RFID_Api.RFID_addLog(mysql, log["picc"], log["device"])):
                app.logger.info("Se cargo el log en la BD")
107             return render_template('ok.html')

109     if request.method == 'GET':
            log = request.values
111         app.logger.info("GET\nPICC: " + log["picc"] + "\nDevice: " +
        log["device"])
113         if(RFID_Api.RFID_addLog(mysql, log["picc"], log["device"])):
                app.logger.info("Se cargo el log en la BD")
```

```
115             return render_template('ok.html')

117     app.logger.info("Hubo un problema al cargar el log en la BD!!")
        return render_template('fail.html')
119
@app.route('/develop')
121 def develop():

123     return render_template('develop.html')

125 # ruta de logueo por session sin autentificacion
@app.route('/login')
127 def login():

129     # se guardan los datos en la session
        session['logged'] = True
131     session['log_update'] = datetime.now()

133     return redirect(url_for('home'))

135 # ruta de salida que cierra la session
@app.route('/logout')
137 def logout():

139     # se limpia la session
        session.clear()
141     session['logged'] = False

143     return redirect(url_for('home'))

145 if __name__ == '__main__':
        app.secret_key='12345'
147     # Define HOST y PUERTO para accerder
        # app.run(host='localhost', port=80)
149     app.run(port=8002, host='0.0.0.0', debug=True)
```

B.2. RFID_API.py

Código de la API RFID con funciones que manipulan la BD con la información de los registros.

```
1 from datetime import date

3 _admin_pass = "admin"

5 # la idea original es que un usuario tenga asociados varios PICCs de
# una tabla de PICCs, pero por ahora cada usuario tiene un solo PICC
7 # almacenado con sus datos (en la tabla users).
# las funciones necesitan recibir la coneccion con la BD para operar
9 # (parametro 'mysql').
# Los IDs no pueden ser numeros negativos!
11
###------------------------------------------------------------###
13 ### Access to information ###

15 # Users #

17 #ok
def RFID_getAllUsers(mysql):
```

```
19      # crea un cursor a la base de datos
        cur = mysql.connection.cursor()
21      # ejecuta la consulta que guarda los datos en la BD
        r = cur.execute("SELECT * FROM users")
23      # persiste los cambio en la DB
        mysql.connection.commit()
25
        if r > 0:
27          # obtengo los datos
            data = cur.fetchall()
29          # cierra la coneccion con la DB
            cur.close()
31          return data
        else:
33          cur.close()
            return 0
35
#ok
37  def RFID_getUser(mysql, userId):
        # crea un cursor a la base de datos
39      cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
41      r = cur.execute("SELECT * FROM users WHERE id = %s", str(userId))
        # persiste los cambio en la DB
43      mysql.connection.commit()

45      if r > 0:
            # obtengo los datos
47          data = cur.fetchone()
            # cierra la coneccion con la DB
49          cur.close()
            return data
51      else:
            cur.close()
53          return 0

55  #ok
    def RFID_getUserByPicc(mysql, picc):
57      # de momento usa el picc interno de la tabla user
        # crea un cursor a la base de datos
59      cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
61      r = cur.execute("SELECT * FROM users WHERE PICC = %s", [picc])
        # persiste los cambio en la DB
63      mysql.connection.commit()

65      if r > 0:
            # obtengo los datos
67          data = cur.fetchone()
            # cierra la coneccion con la DB
69          cur.close()
            return data
71      else:
            cur.close()
73          return 0

75  #ok
    def RFID_getLastUserLogged(mysql):
77      # busca el ultimo log
        log = RFID_getLastLog(mysql)
```

```
79      if log == 0:
            return 0
81      # crea un cursor a la base de datos
        cur = mysql.connection.cursor()
83      # ejecuta la consulta que guarda los datos en la BD
        r = cur.execute("SELECT * FROM users WHERE PICC = %s", [log["PICC"]])
85      # persiste los cambio en la DB
        mysql.connection.commit()
87
        if r > 0:
89          # obtengo los datos
            data = cur.fetchone()
91          # cierra la coneccion con la DB
            cur.close()
93          return data
        else:
95          cur.close()
            return 0
97
# PICCs # (de momento cada user tiene un PICC en su misma tabla)
99 #
# def RFID_getAllPiccs(mysql):
101 #     pass
#
103 # def RFID_getPicc(mysql, piccId):
#     pass
105 #
# def RFID_getPiccsByUser(mysql, userId):
107 #     pass
#
109 # def RFID_getLastPicc(mysql):
#     pass
111
# Devices #
113
#ok
115 def RFID_getAllDevices(mysql):
        # crea un cursor a la base de datos
117     cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
119     r = cur.execute("SELECT * FROM devs")
        # persiste los cambio en la DB
121     mysql.connection.commit()

123     if r > 0:
            # obtengo los datos
125         data = cur.fetchall()
            # cierra la coneccion con la DB
127         cur.close()
            return data
129     else:
            cur.close()
131         return 0

133 #ok
    def RFID_getDevice(mysql, deviceId):
135     # crea un cursor a la base de datos
        cur = mysql.connection.cursor()
137     # ejecuta la consulta que guarda los datos en la BD
        r = cur.execute("SELECT * FROM devs WHERE id = %s", str(deviceId))
```

```
139     # persiste los cambio en la DB
        mysql.connection.commit()
141
        if r > 0:
143         # obtengo los datos
            data = cur.fetchone()
145         # cierra la coneccion con la DB
            cur.close()
147         return data
        else:
149         cur.close()
            return 0
151
# Logs #
153
#ok
155 def RFID_getAllLogs(mysql):
        # crea un cursor a la base de datos
157     cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
159     r = cur.execute("SELECT * FROM logs")
        # persiste los cambio en la DB
161     mysql.connection.commit()

163     if r > 0:
            # obtengo los datos
165         data = cur.fetchall()
            # cierra la coneccion con la DB
167         cur.close()
            return data
169     else:
            cur.close()
171         return 0

173 #ok
    def RFID_getLog(mysql, logId):
175     # crea un cursor a la base de datos
        cur = mysql.connection.cursor()
177     # ejecuta la consulta que guarda los datos en la BD
        r = cur.execute("SELECT * FROM logs WHERE id = %s", str(logId))
179     # persiste los cambio en la DB
        mysql.connection.commit()
181
        if r > 0:
183         # obtengo los datos
            data = cur.fetchone()
185         # cierra la coneccion con la DB
            cur.close()
187         return data
        else:
189         cur.close()
            return 0
191
#ok
193 def RFID_getLogsByUser(mysql, userId):
        # crea un cursor a la base de datos
195     cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
197     r = cur.execute(
"SELECT * FROM logs WHERE PICC = (SELECT PICC FROM users WHERE id = %s)",
```

```
199     str(userId))
        # persiste los cambio en la DB
201     mysql.connection.commit()

203     if r > 0:
            # obtengo los datos
205         data = cur.fetchall()
            # cierra la coneccion con la DB
207         cur.close()
            return data
209     else:
            cur.close()
211         return 0

213 #ok
    def RFID_getLogsByPicc(mysql, picc):
215     # crea un cursor a la base de datos
        cur = mysql.connection.cursor()
217     # ejecuta la consulta que guarda los datos en la BD
        r = cur.execute("SELECT * FROM logs WHERE PICC = %s", [picc])
219     # persiste los cambio en la DB
        mysql.connection.commit()
221
        if r > 0:
223         # obtengo los datos
            data = cur.fetchall()
225         # cierra la coneccion con la DB
            cur.close()
227         return data
        else:
229         cur.close()
            return 0
231
    #ok
233 def RFID_getLastLog(mysql):
        # crea un cursor a la base de datos
235     cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
237     r = cur.execute("SELECT * FROM logs ORDER BY id DESC LIMIT 1")
        # persiste los cambio en la DB
239     mysql.connection.commit()

241     if r > 0:
            # obtengo los datos
243         data = cur.fetchone()
            # cierra la coneccion con la DB
245         cur.close()
            return data
247     else:
            cur.close()
249         return 0

251 #ok
    def RFID_getLogsByDate(mysql, beginDate, endDate = date.min):
253     # compara entre las fechas establecidas sin incluir los limites
        # crea un cursor a la base de datos
255     cur = mysql.connection.cursor()
        # ejecuta la consulta que guarda los datos en la BD
257     r = cur.execute(
    "SELECT * FROM logs WHERE %s < registrated_at AND registrated_at < %s",
```

```
259     (endDate, beginDate))
        # persiste los cambio en la DB
261     mysql.connection.commit()

263     if r > 0:
            # obtengo los datos
265         data = cur.fetchall()
            # cierra la coneccion con la DB
267         cur.close()
            return data
269     else:
            cur.close()
271         return 0

273 ###------------------------------------------------------------###
    ### Administation tools ###
275
    # Users #
277
    #ok
279 def RFID_addUser(mysql, picc, description, count, password):
        try:
281         if password == "admin":
                cur = mysql.connection.cursor()
283             # ejecuta la consulta que guarda los datos en la BD
                cur.execute(
285       "INSERT INTO users(PICC, description, count)VALUES (%s, %s, %s)",
          (picc, description, str(count)))
287             # persiste los cambio en la DB
                mysql.connection.commit()
289             # cierra la coneccion con la DB
                cur.close()
291             return 1
            return 0
293     except Exception as e:
            return 0
295
    #TODO
297 def RFID_delUser(mysql, userId, password):
        try:
299         if password == "admin":
                cur = mysql.connection.cursor()
301             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
303             # persiste los cambio en la DB
                mysql.connection.commit()
305             # cierra la coneccion con la DB
                cur.close()
307             return 1
            return 0
309     except Exception as e:
            return 0
311
    # PICCs #
313 #
    # def RFID_addPicc():
315 #     pass
    #
317 # def RFID_delPicc():
    #     pass
```

```
319
# Devices #
321
#TODO
323 def RFID_addDevice():
    try:
325         if password == "admin":
                cur = mysql.connection.cursor()
327             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
329             # persiste los cambio en la DB
                mysql.connection.commit()
331             # cierra la coneccion con la DB
                cur.close()
333             return 1
            return 0
335     except Exception as e:
            return 0
337
#TODO
339 def RFID_delDevice(mysql, deviceId, password):
    try:
341         if password == "admin":
                cur = mysql.connection.cursor()
343             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
345             # persiste los cambio en la DB
                mysql.connection.commit()
347             # cierra la coneccion con la DB
                cur.close()
349             return 1
            return 0
351     except Exception as e:
            return 0
353
# Logs #
355
#TODO
357 def RFID_delLogsByUser(mysql, userId, password):
    try:
359         if password == "admin":
                cur = mysql.connection.cursor()
361             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
363             # persiste los cambio en la DB
                mysql.connection.commit()
365             # cierra la coneccion con la DB
                cur.close()
367             return 1
            return 0
369     except Exception as e:
            return 0
371
#TODO
373 def RFID_delLogsByDate(mysql, beginDate, endDate, password):
    try:
375         if password == "admin":
                cur = mysql.connection.cursor()
377             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
```

```
379             # persiste los cambio en la DB
                mysql.connection.commit()
381             # cierra la coneccion con la DB
                cur.close()
383             return 1
            return 0
385     except Exception as e:
            return 0
387
#TODO
389 def RFID_delAllLogs(mysql, password):
    try:
391         if password == "admin":
                cur = mysql.connection.cursor()
393             # ejecuta la consulta que guarda los datos en la BD
                cur.execute()
395             # persiste los cambio en la DB
                mysql.connection.commit()
397             # cierra la coneccion con la DB
                cur.close()
399             return 1
            return 0
401     except Exception as e:
            return 0
403
# Others #
405
#ok
407 def RFID_changeAdminPass(oldPass, newPass):
    if oldPass == _admin_pass:
409         _admin_pass = newPass
            return 1
411     return 0

413 #ok
def RFID_executeQuery(mysql, query, password):
415     try:
            if password == _admin_pass:
417             # crea un cursor a la base de datos
                cur = mysql.connection.cursor()
419             # ejecuta la consulta que guarda los datos en la BD
                r = cur.execute("SELECT * FROM users")
421             # persiste los cambio en la DB
                mysql.connection.commit()
423
                if r > 0:
425                 # obtengo los datos
                    data = cur.fetchall()
427                 # cierra la coneccion con la DB
                    cur.close()
429                 return data

431             cur.close()
            return 0
433     except Exception as e:
            return 0
435
###-------------------------------------------------------------###
437 ### Devices tools ###
```

```
439 #ok
    def RFID_addLog(mysql, picc, device):
441     try:
            cur = mysql.connection.cursor()
443         # ejecuta la consulta que guarda los datos en la BD
            cur.execute("INSERT INTO logs(PICC, device) VALUES (%s, %s)",
445     (picc, str(device)))
            # persiste los cambio en la DB
447         mysql.connection.commit()
            # cierra la coneccion con la DB
449         cur.close()
            return 1
451     except Exception as e:
            return 0
453
455
    ###------------------------------------------------------------------###
457 ### Module test and develop ###
    if __name__ == '__main__':
459     pass
```

C. Código de la Placa de Desarrollo

C.1. final.ino

Sketch para subir a Arduino.

```
#include <SPI.h>
#include <MFRC522.h>
#include <SoftwareSerial.h>

//#define esp Serial1
#define SS_PIN 10
#define RST_PIN 9

SoftwareSerial esp(2,3); // RX, TX

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class

// Init array that will store new NUID
byte nuidPICC[4];


String CIPSTART  = {"AT+CIPSTART=\"TCP\",\"192.168.4.2\",8002\r\n"};
String device = "0";
String data;
String server = "192.168.4.2";
String uri = "/newLog";

byte dat [5];

void setup() {

  esp.begin(9600);

  Serial.begin(9600);

  SPI.begin(); // Init SPI bus
  rfid.PCD_Init(); // Init MFRC522

  Serial.println("Test");
  delay(2000);

  setWifi();
}


void reset() {

  esp.println("AT+RST");

  delay(1000);

  if(esp.find("OK") ) Serial.println("Module Reset");
  else Serial.println("Failed");

}

void setWifi(){
  esp.println("AT+CWMODE=3");
```

```
55    delay(1000);
57    if(esp.find("OK") ){
      Serial.println("Paso 1");
59    esp.println("AT+CWSAP=\"ESP\",\"password\",1,4");
       if(esp.find("OK") ){
61        Serial.println("Paso 2");
          esp.println("AT+CIPAP=\"192.168.4.1\"");
63          delay(500);
        if(esp.find("OK") ){
65          Serial.println("Paso 3");
            Serial.println("Creado acces point");
67          return;
       }
69     }
      }
71    Serial.println("Fallo setup");
      setWifi();
73  }

75  void loop () {

77  // Look for new cards
      if ( ! rfid.PICC_IsNewCardPresent())
79    return;

81    // Verify if the NUID has been readed
      if ( ! rfid.PICC_ReadCardSerial())
83    return;

85    Serial.print(F("PICC type: "));
     MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
87    Serial.println(rfid.PICC_GetTypeName(piccType));

89    // Check is the PICC of Classic MIFARE type
      if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
91    piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
      piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
93    Serial.println(F("Your tag is not of type MIFARE Classic."));
      return;
95    }

97    if (rfid.uid.uidByte[0] != nuidPICC[0] ||
      rfid.uid.uidByte[1] != nuidPICC[1] ||
99    rfid.uid.uidByte[2] != nuidPICC[2] ||
      rfid.uid.uidByte[3] != nuidPICC[3] ) {
101   Serial.println(F("A new card has been detected."));

103   // Store NUID into nuidPICC array
      data="";
105   for (byte i = 0; i < 4; i++) {
        nuidPICC[i] = rfid.uid.uidByte[i];
107     data+=nuidPICC[i];
      }
109
    httppost();
111   Serial.println(F("The NUID tag is:"));
      Serial.println(data);
113   }
      else Serial.println(F("Card read previously."));
```

```
115
    // Halt PICC
117 rfid.PICC_HaltA();

119 // Stop encryption on PCD
    rfid.PCD_StopCrypto1();
121 }

123 void httppost () {

125 Serial.println(CIPSTART);
    esp.println(CIPSTART);
127 delay(500);

129 if( esp.find("OK")) {

131 Serial.println("TCP connection ready");
    }
133
    String postRequest =
135
    "GET " + uri + "?picc=" + data + "&device=" + device + " HTTP/1.1\r\n" +
137 "Host: " + server + "\r\n" +
    "Connection: close\r\n" +
139 "\r\n";

141 Serial.println(postRequest);

143 //determine the number of caracters to be sent.
    String sendCmd = "AT+CIPSEND=";
145
    esp.print(sendCmd);
147
    esp.println(postRequest.length() );
149
    if(esp.find(">")) {
151 Serial.println("Sending..."); esp.print(postRequest);

153   if( esp.find("SEND OK")) {
        Serial.println("Packet sent");
155
        while (esp.available()) {
157
          String tmpResp = esp.readString();
159
          Serial.println(tmpResp);
161     }

163     // close the connection
        esp.println("AT+CIPCLOSE");
165   }
    }
167 Serial.println("fin post");
}
```