**DOMÓTICA**

Ingeniería en Telecomunicaciones



Sensor de temperatura con RFID de autenticación remota en bases de datos mediante el uso del framework php LARAVEL

**Jhosser F. Pedraza & Andrés F. Ramírez**

**Unidades tecnológicas de Santander**

**2019**

**INTRODUCCIÓN**

Este texto proporciona una guía útil básica, un ejemplo representativo para el desarrollo de sistemas domotizados centralizados. Un arreglo de este tipo garantiza escalabilidad en entornos industriales de alta densidad de sensores inalámbricos conectados a una base de datos que gestiona métodos de autenticación, procesamiento, consultas y respuestas SQL sobre servicios API. El ejemplo ejecuta en un servicio API-REST local para uno o muchos periféricos. Estos periféricos consisten en uno o muchos microcontroladores con un sensor de temperatura, humedad y un sensor de acceso mediante RFID cada uno. La lógica local implica la toma de temperatura y humedad cada que se autentica de forma remota en la base de datos el usuario autorizado.

Se empieza con una muy breve compilación teórica respecto de las herramientas a utilizar, posteriormente plantearemos el desarrollo del trabajo en dos secciones. Primero, para todo lo relacionado con las bases de datos y el framework de LARAVEL, es decir, esquemas, modelos relacionales y motores de bases de datos, luego extrapolaremos estos mismo sobre el framework php LARAVEL. Una vez resuelta esta sección cubriremos toda la parte en cuanto al script en JAVA sobre el microcontrolador en cuestión (NodeMCU 8266). Al final del documento haremos uso de análisis de datos mediante Rstudio, caracterizaremos las relaciones entre las variables o “logs” ajustaremos curvas (regresión de mínimos cuadrados) e interpolaremos o extrapolaremos algunos valores en función del tiempo.

**OBJETIVOS**

**OBJETIVO GENERAL.**

Diseñar un sistema de autenticación remota para la validación de usuarios asociados a tarjetas, los cuales muestrean parámetros ambientales (temperatura y humedad) a través de un sensor acoplado a un periférico inalámbrico para el posterior análisis de dichas variables mediante software especializado.

**OBJETIVOS ESPECÍFICOS.**

* Diseñar un modelo de base de datos con las entidades y relaciones correspondientes para tarjetas “cards” y parámetros ambientales “temps”.
* Extrapolar el diseño de bases de datos creando un servicio php API REST mediante el framework LARAVEL: migraciones, controladores, rutas y recursos. Así como la lógica de validación de usuarios mediante consultas SQL y el almacenamiento de las variables ambientales.
* Crear un script sobre IDE Arduino para el periférico inalámbrico (Node MCU ESP8266) con su respectiva lógica de conectividad con la LAN, adquisición de datos (UID, temperatura y humedad), peticiones http al API para la escritura de los mismos.
* Tomar 50 muestras en dos horas para realizar un análisis de datos, caracterizaciones, relaciones y ajustes de curvas para el par de variables (temperatura y humedad) en función del tiempo mediante el uso del software Rstudio.

**MARCO TEÓRICO**

**FRAMEWORK PHP LARAVEL.**

Laravel, el “framework PHP para artesanos” según dicta su slogan. Laravel es un framework PHP de código abierto que intenta aprovechar las ventajas de otros Frameworks y desarrollar con las últimas versiones de PHP (entre otras muchas cosas que aporta como framework).

Su filosofía es desarrollar código PHP de forma elegante y simple basado en un modelo MVC (Modelo-Vista-Controlador). En su web https://laravel.com/ encontraremos una extensa y organizada documentación que hará mucho más fácil y efectiva la labor de los desarrolladores. Este framework está en constante mantenimiento y expansión por parte de sus desarrolladores lo que asegura la continuidad y seguridad del framework con actualizaciones regulares (Sierra, 2018).

Las características más notables que aporta Laravel son las siguientes:

* **Blade:** Blade es un sistema de plantillas para crear vistas en Laravel. Este permite extender plantillas creadas y secciones en otras vistas en las cuales también tendremos accesibles las variables y con posibilidad de utilizar código PHP en ellas, además, ligado al uso de bootstrap u otro famework HTML generará resultados optimizados a los diferentes dispositivos (Móviles, Tablets, PC’s, etc..).
* **Eloquent:** Eloquent es el ORM que incluye Laravel para manejar de una forma fácil y sencilla los procesos correspondientes al manejo de bases de datos en nuestro proyecto. Transforma las consultas SQL a un sistema MVC lo que no permite procesar consultas SQL directamente y así protegernos de la inyección SQL.
* **Routing:** Laravel proporciona un sistema de organización y gestión de rutas que nos permite controlar de manera exhaustiva las rutas de nuestro sistema.
* **Middlewares:** Son una especie de controladores que se ejecutan antes y después de una petición al servidor, lo que nos permite insertar múltiples controles, validaciones o procesos en estos puntos del flujo de la aplicación.
* **Comunidad y documentación:** Un gran punto a destacar de este framework es la gran comunidad y documentación que existe, una comunidad de profesionales activa que aporta conocimiento y funcionalidades, además de testear nuevas versiones y detectar fallos del framework, lo que le da seguridad al framework. Y una documentación muy completa y de calidad pensada para los propios desarrolladores.

**SOBRE LOS SISTEMAS RFID.**

RFID son las siglas inglesas de Radio Frequency IDentification lo que en español significa Identificación por radiofrecuencia. El propósito fundamental de la tecnología RFID es identificar mediante un lector, sin contacto y a distancia, una tarjeta o etiqueta (tag) portada por una persona, un vehículo en movimiento o por ejemplo cualquier producto que se encuentra en un almacén o en una cadena de producción automatizada (NUO, 2019).

El sistema de identificación por radiofrecuencia o RFID consta de dos componentes principales, un transpondedor (etiqueta adjunto a un objeto que debe identificarse) y un transceptor también conocido como interrogador o lector.

Un lector consta de un módulo de radiofrecuencia y una antena que genera un campo electromagnético de alta frecuencia. Por otro lado, la etiqueta suele ser un dispositivo pasivo, lo que significa que no contiene una batería. En su lugar, contiene un microchip que almacena y procesa información y una antena para recibir y transmitir una señal.

Para leer la información codificada en una etiqueta, se coloca cerca del lector (no es necesario que esté dentro de la línea de visión directa del lector). Un lector genera un campo electromagnético que hace que los electrones se muevan a través de la antena de la etiqueta y, posteriormente, alimenten el chip.

El chip que se encuentra dentro de la etiqueta responde respondiendo enviando la información almacenada al lector en forma de otra señal de radio. Esto se llama retrodispersión. El lector detecta e interpreta la retrodispersión, o cambio en la onda electromagnética de RF, que luego envía los datos a una computadora o microcontrolador (Engineers, 2019).

El funcionamiento se describe de la siguiente forma: El lector realiza peticiones o preguntas por radiofrecuencia al chip que integran las etiquetas RFID, las cuales emiten una respuesta al recibir las señales del lector, permitiendo la identificación con gran seguridad y precisión en tiempo real.

El código de identificación que contienen es único y puede ser personalizado durante la fabricación de la etiqueta. Los fundamentos físicos en los que se basa la tecnología RFID, implican la aparición de varios modelos de comunicación entre los dispositivos básicos del sistema. La comunicación por radiofrecuencia, requiere la incorporación de una antena RF en cada uno de los dispositivos implicados en la comunicación cuya forma y características depende de la banda de frecuencia en la que funcionen.

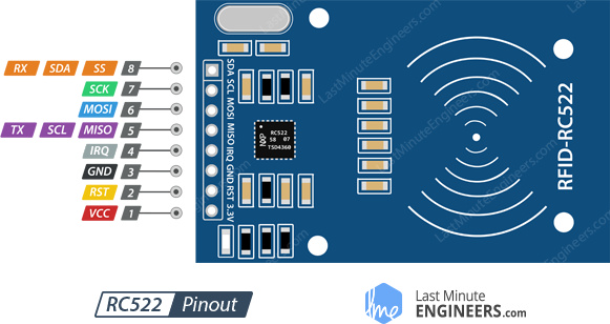
La frecuencia de 13,56Mhz es la utilizada por las modernas tarjetas de identificación, lo bueno de esta tecnología es la alta seguridad de las aplicaciones y la cantidad de información que se puede guardar en su memoria, como por ejemplo el patrón de la huella del portador de la tarjeta, también es posible incorporar en la tarjeta varias aplicaciones diferentes, como puede ser el control de accesos y monedero para máquinas de vending (NUO, 2019).

**RFID-RC522.**

El módulo RFID RC522 es una placa de identificación de tags y tarjetas (lectura y escritura) por medio del estándar ISO/IEC 14443 tipo A que funciona bajo 13.56 MHz. Este módulo es extremadamente útil para proyectos de controles de acceso dándonos una gran versatilidad en los modos de encriptación de los códigos de acceso. EL módulo puede comunicarse con un microcontrolador por medio de comunicación SPI y por un puerto UART (Morales, 2017).

La distribución de pines se define de la siguiente forma, según la figura 1.

* **GND:** Referencia negativa.
* **RST:** Reset.
* **3.3V:** Voltaje de alimentación 3.3VDC máximo
* **MOSI:** (Master Out Slave In). Por esta línea se envían datos en forma serial desde el dispositivo “maestro” a los dispositivos esclavos.
* **MISO:** (Master In Slave Out). Por esta línea los dispositivos esclavos envían datos al dispositivo maestro.
* **SCK/SCLK** (Clock): Línea de reloj, envía un tren de pulsos que sincroniza las comunicaciones entre los dispositivos.
* **SS/SDA** (Select): Línea por la que el dispositivo maestro habilita al dispositivo esclavo, indica qué dispositivo es el que tiene que enviar/recibir datos.

 *Figura.1: Distribución de pines RFID-RC522. Fuente:LastMinuteEngineers*

**SOBRE EL RFID Y EL NODEMCU ESP8266**

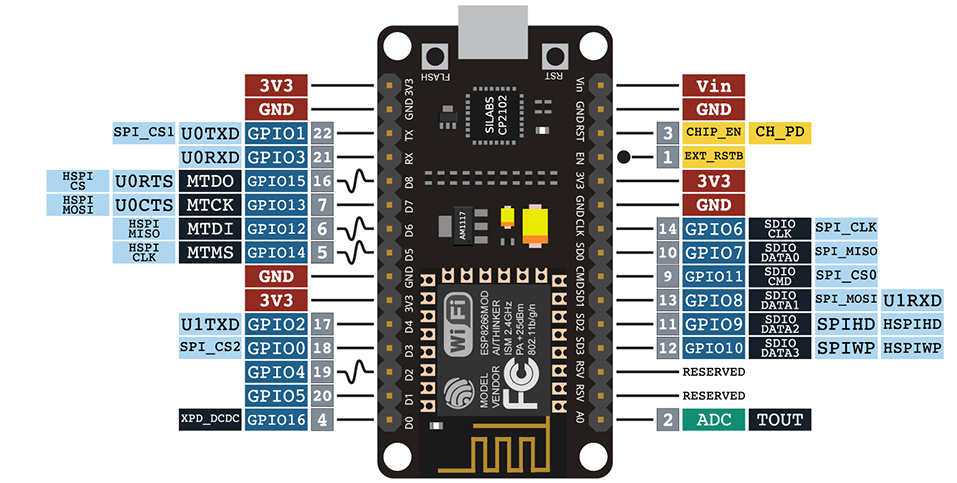
Para conectar NodeMCU con el lector de identificación de RF RC522 se utiliza la biblioteca Arduino para MFRC522 y otros módulos basados en RFID RC522. Esta biblioteca lee y escribe diferentes tipos de tarjetas de Identificación por Radiofrecuencia (RFID) en NodeMCU utilizando un lector basado en RC522 conectado a través de la interfaz de Interfaz Periférica Serial (SPI).

El NodeMCU es una tarjeta de desarrollo similar a Arduino, especialmente orientada al Internet de las cosas (IoT). Está basado en el SoC (System on Chip) ESP8266, un chip altamente integrado, diseñado para las necesidades de un mundo conectado. Integra un potente procesador con Arquitectura de 32 bits (más potente que el Arduino Due) y conectividad Wifi.

Para el desarrollo de aplicaciones se puede elegir entre los lenguajes Arduino y Lua. Al trabajar dentro del entorno Arduino podremos utilizar un lenguaje que ya conocemos y hacer uso de un IDE sencillo de utilizar, ademas de hacer uso de toda la información sobre proyectos y librerías disponibles en internet. La comunidad de usuarios de Arduino es muy activa y da soporte a plataformas como el ESP8266.

NodeMCU viene con un firmware pre-instalado el cual nos permite trabajar con el lenguaje interpretado LUA, enviandole comandos mediante el puerto serial (CP2102). Las tarjetas NodeMCU y Wemos D1 mini son las plataformas más usadas en proyectos de Internet de las cosas (IoT). No compite con Arduino, pues cubren objetivos distintos, incluso es posible programar NodeMCU desde el IDE de Arduino.

La tarjeta NodeMCU está diseñada especialmente para trabajar en protoboard. Posee un regulador de voltaje en placa que le permite alimentarse directamente del puerto USB. Los pines de entradas y salidas trabajan a 3.3V. El chip CP2102 se encarga de la comunicación USB-Serial (del Valle Hernandez, 2017).

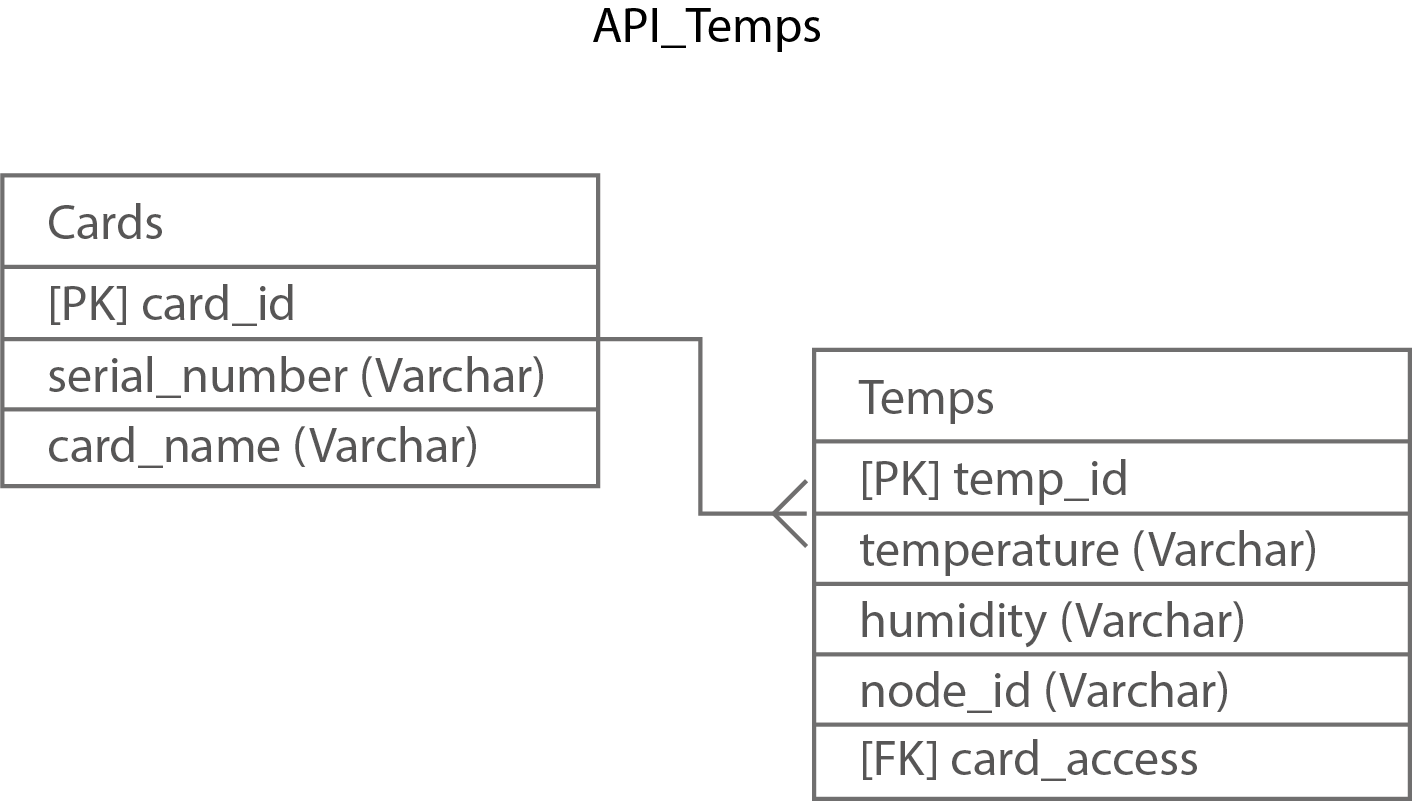
* Figura.2: Distribución de pines NodeMCU ESP8266.*

**DESARROLLO DEL PROYECTO**

**MODELO DE BASE DE DATOS: API\_Temps.**

En este modelo se considera que, para una multiplicidad de usuarios existen algunos registros de temperatura, en otras palabras, cada muestra creada estará asociado a un único usuario, sin embargo cada usuario podrá tomar tantas muestras ambientales que desee.

El diagrama entidad relación de la base de datos “API\_Temps” consiste en un par de tablas (modelos) relacionadas de uno a muchos: tarjetas y temperaturas. Las tarjetas (la tabla padre) “cards”, se compone de los siguientes campos: La llave primaria “card\_id” un entero auto incrementado secuencialmente de uno en uno, el nombre del usuario “card\_name” que será el usuario asociado a la tarjeta y el UID de la tarjeta “serial\_number”. La tabla hija (temps) se compone de los siguientes campos: La llave primaria “temp\_id”, la magnitud de la temperatura “temperatura”, naturalmente un dato del tipo “float” con longitud 2,2 (decenas en parte entera y parte flotante), humedad “humidity” con el mismo tipo de dato que las temperaturas salvo que su interpretación es porcentual, el ID del nodo “node\_id” una serie de caracteres alfanuméricos que identifica de qué periférico proviene la muestra (considerándose que existen múltiples periféricos en una zona o múltiples zonas). Finalmente quizá el campo más importante en nuestro caso según la lógica de toma de muestras: La llave foránea “card\_access” que asocia las muestras a un único usuario. La figura 3 expone el diagrama entidad relación con los campos requeridos en cada una de las tablas descritas anteriormente.

 *Figura. 3: Diagrama ER, API\_temps*.

Algunos parámetros asociados a la tabla de temperatura, por ejemplo “node\_id” pueden ser escalados a otra tabla u otro par de tablas con usuarios específicos asociados a nodos o bien asociados a nodos y a tarjetas. Sin embargo, esta aplicación es puramente didáctica, con el fin de entender de forma elemental el empleo de LARAVEL como framework php con esta aplicación en específico. Aunque queda bajo el criterio del estudiante como modelar u escalar el modelo entidad relación dependiendo de su aplicación. Esta es una de las virtudes de una red de sensores centralizada.

En el modelo, la tabla “temps” contempla un parámetro de humedad, así que, de forma generalizada se referirá a lo largo del texto a esta tabla o modelo como “muestras” o “muestras ambientales” sin especificar si se trata de temperatura o humedad (el nombre de la tabla proviene de un modelo creado con anterioridad en el anteproyecto donde se consideraba solo muestras de temperatura, por cuestiones de objetividad se incluyó posteriormente temperatura y humedad).

**EXTRAPOLANDO EL MODELO RELACIONAL EN LARAVEL.**

Antes que nada se debe elegir un motor de base de datos de preferencia, en nuestro caso implementaremos “PostgreSQL 11”. Omitiremos descripción para la creación de usuarios y probilegios asociados al motor de base de datos, sin embargo, debido a la posibilidad de que el lector desee reproducir el proyecto de forma idéntica debe considerar los parámetros aquí utilizados: El usuario asociado es “API\_user” y contraseña “98032562140”; Sobre los privilegios en las propiedades este usuario, puede iniciar sesión en el motor de base de datos. Claramente durante la creación de la base de datos “API\_temps” se debe asociar a éste usuario.

Para empezar a trabajar sobre el framework se recomienda primero, cargar el arreglo de carpetas que conforman el CRUD Laravel mediante el editor de texto de programación de su preferencia, en nuestro caso será “Atom” una vez allí se configura el archivo “.env” sobre la ruta de origen del proyecto, es decir, “c:/xampp/API\_temps”. Las siguientes líneas ajustan el archivo para asociarlo con la base de datos PostgreSQL.

|  |  |
| --- | --- |
| 9 | DB\_CONNECTION=pgsql |
| 11 | DB\_PORT=5432 |
| 12 | DB\_DATABASE=API\_temps |
| 13 | DB\_USERNAME=API\_user |
| 14 | DB\_PASSWORD=98032562140 |

*Tabla.1: Configuration “.env”.*

Creando los modelos que necesitamos (según el modelo relacional expuesto con anterioridad), acudimos a la gama de comandos del controlador “**artisan**” dentro del arreglo de carpetas de nuestro proyecto que llamada casualmente igual que la base de datos “API\_temps” en la ruta c:/xampp/API\_temps. Ejecutamos entonces en cmd las siguientes líneas.

|  |  |
| --- | --- |
| 1 | php artisan make:model Card --all |
| 2 | php artisan make:model Temp --all |

*Tabla. 2: Creación de modelos en la consola “cmd”.*

Con esto, aparecerán nuevos archivos sobre varias rutas en el arreglo de archivos de nuestro proyecto, de las cuales recurrimos a: “API\_temps/database/migrations” donde están los archivos de configuración de las tablas, “API\_temps/app” donde se encuentran los modelos “Card” y “Temp” y finalmente “API\_temps/app/Http” donde se encuentran los controladores de nuestro par de modelos.

Se empieza configurando las tablas que hemos planteado en el modelo relacional con sus respectivos atributos y nombres de columnas. En las migraciones, para la tabla de tarjetas, es decir en “API\_temps/database/mirations/YYYY\_MM\_DD\_HHMMSS\_create\_cards\_table.php” se agregan las nuevas columnas mediante las siguientes líneas.

|  |  |
| --- | --- |
| 14 | public function up() |
| 15 | { |
| 16 | Schema::create('cards', function (Blueprint $table){ |
| 17 | $table->increments('card\_id'); |
| 18 | $table->String('serial\_number')->unique(); |
| 19 | $table->String('card\_name'); |
| 20 | $table->timestamps(); |
| 21 | }); |
| 22 | } |

*Tabla. 3: Configuración de la tabla (migración) “cards”.*

De la misma forma se configuran las columnas con sus características para la tabla de temperaturas “temps” en “API\_temps/database/mirations/YYYY\_MM\_DD\_HHMMSS\_create\_temps\_table.php”.

|  |  |
| --- | --- |
| 14 | public function up() |
| 15 | { |
| 16 | Schema::create('temps', function (Blueprint $table){ |
| 17 | $table->increments('temp\_id'); |
| 18 | $table->String('temperature’); |
| 19 | $table->String(‘humidity’); |
| 20 | $table->String('node\_id'); |
| 21 | $table->String('card\_access'); |
| 22 | $table->foreign('card\_access')->references('card\_id')->on('cards'); |
| 23 | $table->timestamps(); |
| 24 | }); |
| 25 | } |

*Tabla. 4: Configuración de la tabla (migración) “temps”.*

De aquí cabe resaltar el campo “card\_access” como llave foránea, este será el campo que relacionará las muestras con los usuarios.

Al adecuar el modelo para esta tabla en “API\_temps/app/Card.php” para la clase “Card” con las características de las columnas y la función para la llave foránea se configura de la siguiente forma.

|  |  |
| --- | --- |
| 7 | class Card extends Model |
| 8 | { |
| 9 | protected $table = "cards"; |
| 10 | protected $primaryKey = "card\_id"; |
| 11 | protected $guarded = []; |
| 12 | public function temps(){ |
| 13 | return $this->hasMany('App\Temp', 'card\_access', 'card\_id'); |
| 14 | } |
| 15 | } |

*Tabla. 5: Configuración del modelo para las tarjetas “Card”.*

Donde “card\_id” ocupa la llave primaria y es aquella que se referencia en la línea 22 de la migración para temperaturas (véase la tabla 4). Haremos lo mismo para el modelo de temperaturas “Temp” en “API\_temps/app/Temp.php” como sigue.

|  |  |
| --- | --- |
| 7 | class Temp extends Model |
| 8 | { |
| 9 | protected $table = "temps"; |
| 10 | protected $primaryKey = "temp\_id"; |
| 11 | protected $guarded = []; |
| 12 | protected $hidden =['card\_access','created\_at','updated\_at']; |
| 13 | public function cards(){ |
| 14 | return $this->belongsTo('App\Card', 'card\_access', 'card\_id'); |
| 15 | } |
| 16 | } |

*Tabla. 6: Configuración del modelo para las muestras “Temp”.*

En esta etapa del proyecto se pueden ejecutar las migraciones desde cmd mediante: php artisan migrate. Sin embargo, se puede seguir estructurando el arreglo de carpetas; Esta vez se procede a definir los parámetros para el API que incluyen cambios en los controladores, rutas y recursos como también la agregación de validadores en casos de errores por parte de consultas SQL.

Para las rutas, las cuales proporcionan la dirección a la que deben apuntar los periféricos para cualquiera de las solicitudes http (GET, POST, PUT, DELETE), de forma general (líneas 19 y 20) aunque algunas de estas pueden ser rutas específicas que apuntan a funciones dentro de un controlador para procesos puntuales (línea 21); Se definen en dentro de “API\_temps/routes/api.php” de la forma.

|  |  |
| --- | --- |
| 19 | Route::APIresource('cards', 'CardController'); |
| 20 | Route::APIresource('temps', 'TempController'); |
| 21 | Route::post('cards/auth','CardController@auth')->name('card.auth'); |

*Tabla. 7: Rutas API con métodos http a los controladores “Card” y “Temp”.*

Las rutas de las líneas 19 y 20 corresponden a todos los métodos de solicitudes http asociados a sus respectivas funciones por defecto para API-REST es decir: show, store, update y delete, es importante aclararar que API no emplea las funciones por defecto “create” ni “edit”. Para visualizar las rutas que hemos creado con sus respectivos métodos, para posteriormente usar para apuntar un cliente http se ejecuta sobre cmd el comando: php artisan route:list.

Una nueva ruta específica se describe en la línea 21 llamada “card.auth” que tendrá la lógica para la autenticación de usuarios. Esta función (auth) puede ser descrita en cualquier parte dentro del controlador “CardController” en “API\_temps/app/Http/Controllers/CardController.php” de la siguiente forma.

|  |  |
| --- | --- |
| 45 | public function auth(Request $request) |
| 46 | { |
| 47 | $cards = Card::where('serial\_number', $request->serial\_number)->first(); |
| 48 | if($cards==null){ |
| 49 | return response()->json(['error'=>'usuario invalido', 'codigo'=>404], 404); |
| 50 | } |
| 51 | else{ |
| 52 | return new CardResource($cards); |
| 53 | } |
| 54 | } |

*Tabla. 8: Configuración del controlador de tarjetas “CardController” para la validación de las mismas.*

La función “auth” consiste en un “constructor de consulta de LARAVEL” que equivale a una consulta SQL tradicional de la siguiente forma: Mediante una petición POST en el periférico se envía el dato en consideración que pertenece a la tarjeta, es decir, el UID bajo el parámetro requerido (request) “serial\_number”. En el framework, se crea una variable de consulta “$cards” que busca en toda la tabla de tarjetas una coincidencia entre el “serial\_number” dispuesto en la tabla frente al “serial\_number” que llega de la petición, en caso de encontrarle toma la primera coincidencia y devuelve todos los parámetros asociados a este usuario (en la variable ya mencionada “$cards”) (línea 47), la respuesta estará expuesta en la tabla 9 mediante el recurso “CardResourse” (línea 52). Si la consulta no encuentra coincidencias con el campo requerido devolverá un campo nulo sobre la variable “$cards”, así que, mediante una lógica simple (línea 48 a 51), la validación recurre a una respuesta rápida en caso de que el usuario sea desconocido (línea 49).

En este punto se debe introducir la definición de recurso. Éstos no son más que una respuesta organizada “.json” para una solicitud http. Para utilizarse se debe crear sobre comando en cmd de la forma: “artisan make:resource CardRe”.

Luego aparecerá un nuevo archivo sobre la ruta “API\_temps/app/Http/Resources/CardResource.php” el cual importaremos al controlador en el encabezado del archivo “CardController” como: use “App\Http\Resources\CardResource;” Y configuraremos de la forma.

|  |  |
| --- | --- |
| 15 | public function toArray($request) |
| 16 | { |
| 17 | return [ |
| 18 | 'id' => $this->resource->card\_id |
| 19 | ]; |
| 20 | } |

*Tabla. 9: Respuesta de acceso permitido para la tarjeta*

Cuya lógica puede ser descrita como: se recibe una la variable “$cards” procedente de la consulta SQL con todos los parámetros asociados al usuario con el que coincide el “serial\_number”, de estos, se tomará el “card\_id” sobre el campo “id” en la respuesta “.json”.

Una forma anexa de cosolidar la autenticación de un usuario válido o inválido está en función de su código de respuesta http. De estos existen muchos a los cuales se eles atrubulle distintas características, sin embargo, nos centraremos en dos muy importantes: El código 400 y el código 200. El primero de estos corresponde a un error de un parámetro, ruta o modelo no encontrado. El segundo corresponde a cualquier forma de lógica con respuesta efectiva desde el servidor al cliente (que será indispensable dentro de la lógica del script para el periférico descrito más adelante).

En este punto se debe configurar los distintos tipos de respuestas a solicitudes http en la dirección “API\_temps/app/Exceptions/Handler.php”. Antes que nada se importan las librerías con los tipos de respuestas en el encabezado del archivo y el contenido “Handler.php” de la forma.

|  |  |
| --- | --- |
| 6 | use\Symfony\Component\HttpKernel\Exception\HttpException; |
| 7 | use\Illuminate\Database\Eloquent\ModelNotFoundException; |
| 8 | use\Illuminate\Database\QueryException; |
| 9 | use\Symfony\Component\HttpKernel\Exception\HttpException; |
| 10 | use\Illuminate\Validation\ValidationException; |
| 51 | public function render($request, Exception $exception) |
| 52 | { |
| 53 | if ($exception instanceof ValidationException) { |
| 54 | return response()->json(['error' => $exception->validator->errors(), 'codigo' => 400], 400); |
| 55 | } |
| 56 | if ($exception instanceof QueryException) { |
| 57 | return response()->json(['error' => 'Error de consulta', 'codigo' => 400], 400); |
| 58 | } |
| 59 | if ($exception instanceof ModelNotFoundException) { |
| 60 | return response()->json(['error' => 'Error en el modelo', 'codigo' => 400], 400); |
| 61 | } |
| 62 | if ($exception instanceof HttpException) { |
| 63 | return response()->json(['error' => 'Ruta no enconrada', 'codigo' => 404], 404); |
| 64 | } |
| 65 | return parent::render($request, $exception); |
| 66 | } |

*Tabla. 10: Configuración de respuestas de código http.*

En caso de que el acceso sea autorizado, es decir, se encuentre una coincidencia con el “serial\_number” dentro de la base de datos, el periférico una lógica simple (bajo una respuesta del servidor de código http 200) tomará una muestra en ese instante y la enviará como método de escritura a la base de datos con los datos requeridos por la tabla “temps”. Dicha lógica se expondrá más adelante, por ahora, nos centraremos a consolidar la lógica en cuanto se reciba esta solicitud POST del periférico.

Usando la función “store” dentro del controlador del modelo “Temp”, es decir “TempController” dentro de la ruta para los controladores (estos se crean automáticamente cuando se ejecuta el argumento --all” al momento de crear los modelos).

|  |  |
| --- | --- |
| 7 | use App\Http\Resources\TempResource; |
| 27 | public function store(Request $request) |
| 28 | { |
| 29 | $temp=Temp::create($request->all()); |
| 30 | return new TempResource($temp); |
| 31 | } |

*Tabla. 11: Función de almacenamiento de muestras.*

La función escribe una nueva muestra sobre la variable “$temp” y retorna al recurso “TempResource” con dicha variable. EL recurso (al igual que con el recurso en el controlador del modelo “Card”) se importa en el encabezado del archivo de recurso (línea 7) creado mediante la sentencia en cmd: “artisan make:resource TempResource” y configurado para una respuesta “.json” de la forma.

|  |  |
| --- | --- |
| 15 | public function toArray($request) |
| 16 | { |
| 17 | return [ |
| 18 | 'temperature: ' => $this->resource->temperature, |
| 19 | 'humidity: ' => $this->resource->humidity, |
| 20 | 'created by: '=> $this->resource->cards->card\_name |
| 21 | ]; |
| 22 | } |

*Tabla. 12: Respuesta .json a el recurso luego de registrar una muestra satisfactoriamente.*

Esta forma de validación remota permite asociarles más atributos a los usuarios que realizan registros de muestras ambientales haciéndole escalable, es decir, relacionarle con múltiples otros parámetros dentro de una base de datos. También pueden implementarse otros accionamientos sobre la base de datos o sobre el código del periférico, esto más bien depende de la aplicación del diseñador.

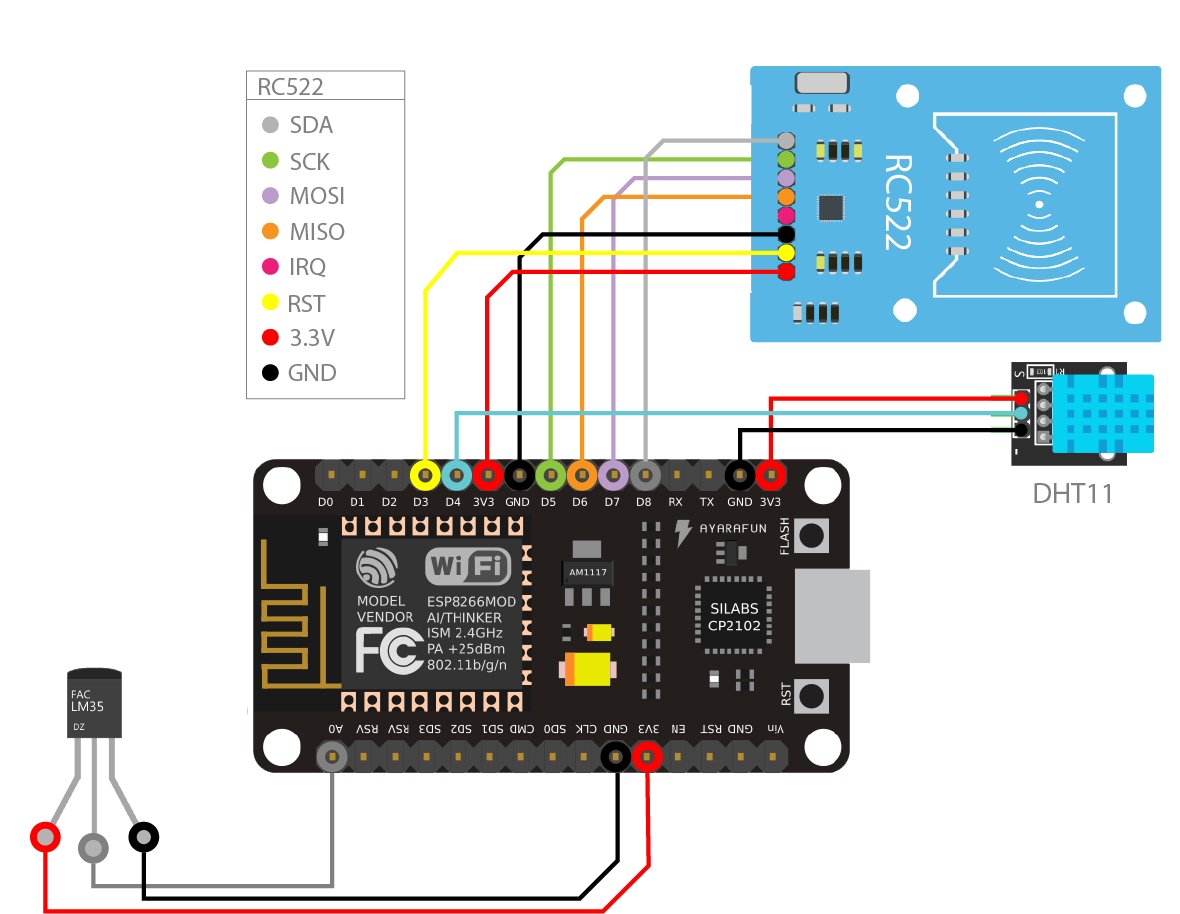
En el diseño de las migraciones (tablas), el campo “timestamps” viene por defecto, este campo es muy últil a la hora de hacer seguimiento del momento en que se ha tomado la muestra. Así que debemos ajustar la zona horaria del framework para una correcta interpretación de los datos almacenados.

En el archivo “app.php” sobre la ruta “API\_temps/config” de nuestro proyecto, se cambia la línea 68 de la siguiente forma.

|  |  |
| --- | --- |
| 68 | 'timezone' => 'America/Bogota', |

*Tabla. 13: Cambio de zona horaria en LARAVEL.*

**CREANDO EL SCRIPT EN IDE ARDUINO PARA EL PERIFERICO.**

El arreglo circuital expuesto en la figura 4 es la configuración empleada para el desarrollo de la aplicación. La lógica de la aplicación y el algoritmo que se construye sobre este periférico consiste en lo siguiente: Se considera un contexto donde se tiene solo la disponibilidad del UID en la tarjeta de acceso (ciertamente estas tarjetas pueden ser programadas con muchos otros atributos asociados al mismo, sin embargo por simplicidad solo consideramos el código único de identificación asociado a cada tarjeta “UID”) el periférico tomará lectura de esta tarjeta y enviará dicho parámetro a través de una solicitud POST bajo el argumento de “serial\_number”. Se realiza una consulta de su existencia en la base de datos y si es así, se trae devuelta el recurso expuesto en la tabla 9 (el “card\_id”). Una vez se obtenga respuesta oportuna (código 200, en caso de no existir el usuario será código 400 o bien 404) del servidor, el script procede a tomar la muestra ambiental (temperatura y humedad) en ese preciso instante de tiempo y se le asocia al usuario que se acaba de autenticar en la base de datos mediante otra solicitud POST, pero esta vez hacia la escritura de dicha muestra. De esta forma se obtiene la información de quien, cuando y desde donde se ha tomado la muestra.

*Figura. 4: Esquema del periférico (NodeMCU8266, LM35, RC522).*

Para empezar se deben por incluir las librerías previamente instaladas (omitiremos el procedimiento de descargar e instalar librerías así como el establecimiento de compatibilidad del IDE de Arduino con el NodeMCU ESP8266) que se han de emplear tanto para el cliente http, cliente WiFi y las librerías para el RFID-RC522 y el DHT11 sobre el editor de scripts de Arduino de la siguiente forma.

|  |  |
| --- | --- |
| 1 | #include <SPI.h> |
| 2 | #include <MFRC522.h> |
| 3 | #include <ESP8266WiFi.h> |
| 4 | #include <WiFiClient.h> |
| 5 | #include <ArduinoJson.h> |
| 6 | #include "DHT.h" |
| 7 | #include <ESP8266HTTPClient.h> |

*Tabla. 14 Inclusión de librerías*

Posteriormente se definen los GPIOS para la configuración esquemática de la figura 3 del entrono Node-FRID, también los parámetros para el cliente WiFi, y las variables requeridas en las tablas de la figura 1.

|  |  |
| --- | --- |
| 8 | #define RST\_PIN 0 |
| 9 | #define SS\_PIN 15 |
| 10 | #define DHTPIN 2 |
| 11 | #define DHTTYPE DHT11 |
| 12 | MFRC522 mfrc522(SS\_PIN, RST\_PIN); |
| 13 | const char \*ssid = "YOUR\_SSID"; |
| 14 | const char \*password = "YOUR\_PASSWORD”; |
| 15 | String serial\_number, temperature, humidity; |
| 16 | String node\_id = "23TE5290"; |
| 17 | byte ReadUID[4]; |

*Tabla. 15: Definición de GPIO’s variables y constantes.*

En este punto se define la función “setup” del script de la siguiente forma: Se inicializa el puerto serial con una velocidad de 115200 baudios por segundo, también se inicializa el módulo SPI y el modulo DHT11, el servicio cliente WiFi y se condicionan los parámetros de conexión a la red donde estemos trabajando.

|  |  |
| --- | --- |
| 18 | void setup() **{** |
| 19 | Serial.begin(115200); |
| 20 | SPI.begin(); |
| 21 | dht.begin(); |
| 22 | mfrc522.PCD\_Init(); |
| 23 | WiFi.begin(ssid, password); |
| 24 | Serial.println(""); |
| 25 | Serial.println("Connecting"); |
| 26 | while (WiFi.status() != WL\_CONNECTED) { |
| 27 | delay(500); |
| 28 | Serial.print("."); |
| 29 | } |
| 30 | Serial.println(""); |
| 31 | Serial.print("Connected to "); |
| 32 | Serial.println(ssid); |
| 33 | Serial.print("IP address: "); |
| 34 | Serial.println(WiFi.localIP()); |
| 35 | **}** |

*Tabla. 16: Configuración de la función “setup”.*

En la función de bucle “loop” la se define de la siguiente forma: Primero describiremos lo que repercute la lectura del UID de cada tarjeta.

|  |  |
| --- | --- |
| 36 | void loop() **{** |
| 37 | if ( ! mfrc522.PICC\_IsNewCardPresent()) |
| 38 | return; |
| 39 | if ( ! mfrc522.PICC\_ReadCardSerial()) |
| 40 | return; |
| 41 | Serial.print("UID:"); |
| 42 | for (byte i = 0; i < mfrc522.uid.size; i++) { |
| 43 | if (mfrc522.uid.uidByte[i] < 0x10){ |
| 44 | Serial.print("0"); |
| 45 | } |
| 46 | else { |
| 47 | Serial.print(""); |
| 48 | } |
| 49 | serial\_number.concat(String(mfrc522.uid.uidByte[i], HEX)); |
| 50 | ReadUID[i] = mfrc522.uid.uidByte[i]; |
| 51 | } |
| 52 | serial\_number.toUpperCase(); |
| 53 | Serial.print(serial\_number); |
| 54 | Serial.println(""); |
| 55 | delay(300); |
| 56 | mfrc522.PICC\_HaltA(); |

*Tabla. 17: Función “loop” lectura del UID*

Aquí cabe destacar el arreglo de la línea 44, para sacar del ciclo “for” el valor del UID de cada tarjeta se concadena sobre la variable “serial\_number”, posteriormente en 47 se convierten los valores a caracteres mayúsculas y finalmente se imprime en visor serial en 48 (por cuestiones de seguridad, no es conveniente imprimir este valor en el monitor serial, sin embargo, por cuestiones didácticas se ha dejado así). Una vez obtenido el UID, éste se enviará mediante una consulta a la base de datos con el método POST a la función “auth” apuntando a la ruta que hemos creado y listado "http://192.168.1.53:8000/api/cards/auth", en este caso se haempleado la dirección de la máquina que aloja la base de datos. Para definir a que ruta ip dentro del área local se debe iniciar el servicio API de LARAVEL se ejecuta el siguiente comando sobre cmd: “php artisan serve --host=192.168.1.53”. La lógica de la consulta se describe a continuación.

|  |  |
| --- | --- |
| 56 | HTTPClient http; |
| 57 | String postData; |
| 58 | postData = "serial\_number=" + serial\_number; |
| 59 | http.begin("http://192.168.1.53:8000/api/cards/auth"); |
| 60 | http.addHeader("Content-Type", "application/x-www-form-urlencoded"); |
| 61 | int httpCode = http.POST(postData); |
| 62 | String payload = http.getString() |

*Tabla. 18: Función “loop” petición POST sobre la validación del UID.*

En la línea 54 se ha concadenado al argumento de la solicitud POST el UID “serial\_number”. En la línea 57 se define una nueva variable llamada “httpCode” aquí se guardan los códigos de respuestas del servidor: 200, 400, 404, etc. Este campo es crucial para la lógica que se describe a continuación.

La validación del usuario se centra en que si la consulta tiene respuesta “OK” con código 200, es decir, si se ha encontrado un usuario asociado a la llave “serial\_number” sobre la variable “serial\_number” (la variable se ha llamado así casualmente igual que la llave primaria para la tabla “cards”) se plantea la siguiente lógica: Primero se toma el valor del ID “card\_id” del usuario asociado al UID “serial\_number” mediante la forma.

|  |  |
| --- | --- |
| 63 | StaticJsonDocument<200> datos; |
| 64 | deserializeJson(datos, payload); |
| 65 | int id = datos["data"]["id"]; |
| 66 | delay(300); |

*Tabla. 19: Función “loop” lectura del ID “id\_card” en la respuesta json*

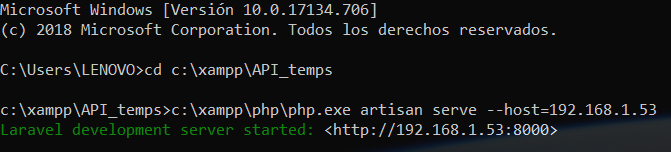
La línea 63 valida que sobre la respuesta 200 del servidor, se agrupe dicha respuesta en una forma organizada llamada “datos” y asocie la respuesta “payload” a dicho arreglo. La línea 65 define una variable “id” que tomará de la línea de caracteres organizados el “id” que proviene de la respuesta json. De esta forma se extrae el parámetro que asocia la muestra con el usuario que se ha autenticado.

|  |  |
| --- | --- |
| 67 | if (httpCode == 200){ |
| 68 | HTTPClient http; |
| 69 | Serial.println("taking environments parameters"); |
| 70 | String postData2; |
| 71 | float hum = dht.readHumidity(); |
| 72 | float temp = dht.readTemperature(); |
| 73 | if (isnan(hum) || isnan(temp)) { |
| 74 | Serial.println(F("Failed to read from DHT sensor!")); |
| 75 | return; |
| 76 | } |
| 77 | temperature = String(temp); |
| 78 | humidity = String(hum); |
| 79 | postData2 = "temperature=" + temperature + "&humidity=" + humidity + "&node\_id=" + node\_id + "&card\_access=" + id; |
| 80 | http.begin("http://192.168.1.53:8000/api/temps"); |
| 81 | http.addHeader("Content-Type", "application/x-www-form-urlencoded"); |
| 82 | String payload2 = http.getString(); |
| 83 | Serial.println(payload2); |
| 84 | http.end(); |
| 85 | serial\_number = ""; |
| 86 | } |
| 87 | else{ |
| 88 | String payload3 = http.getString(); |
| 89 | Serial.println(payload3); |
| 90 | http.end(); |
| 91 | } |
| 92 | http.end(); |
| 93 | serial\_number = ""; |
| 94 | delay(500); |
| 95 | **}** |

*Tabla. 19: Función “loop” lógica de la validación del código http y escritura de la muestra ambiental.*

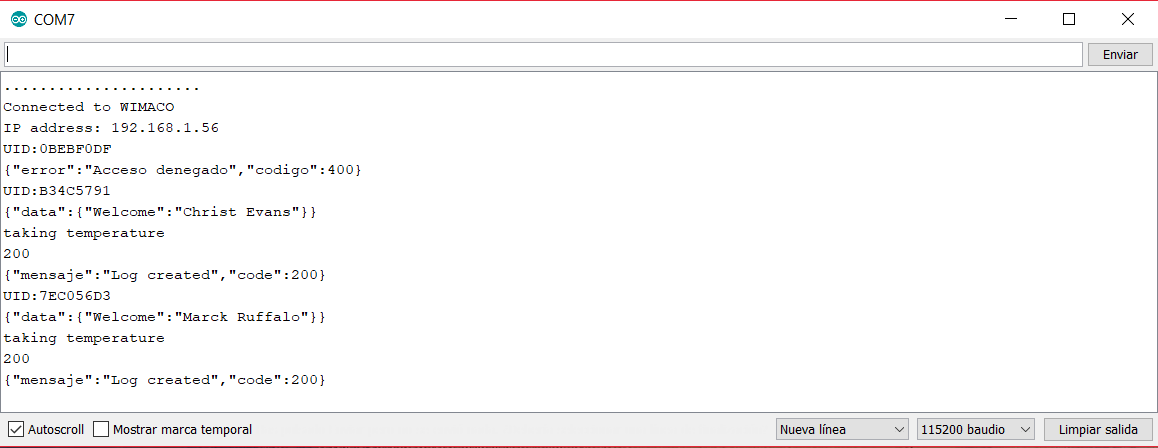
Si el código recibido es “200” entonces se ejecuta el script que toma la muestra y además envíe los datos que sugiere la tabla “temps” mediante una petición POST. Cabe resaltar la línea 79 donde se concadena los parámetros, también la línea 80 la cual es la ruta de la función “store” del TempController. Las líneas 85 y 93 limpian la cadena sobre “serial\_number” para que posteriormente se adicione el valor sobre un campo vacío y no sobre un campo ya escrito por el proceso previo, esto es para cualquiera de los dos casos, si se recibe un “200” o no. La llave “**}**” cierra la función “loop” dando por terminado el script.

Mediante este método se autentican usuarios (tantos como queramos) de forma remota. Un método simple de hacer el registro de tarjetas válidas es inicializando el servicio REST y solicitar consultas que serán rechazadas, con ello se obtiene el UID de cada tarjeta y lar registramos manualmente en PostgreSQL como se muestra en las siguientes figuras.



*Figura. 5: Inicio del servicio REST sobre la ip especificada*

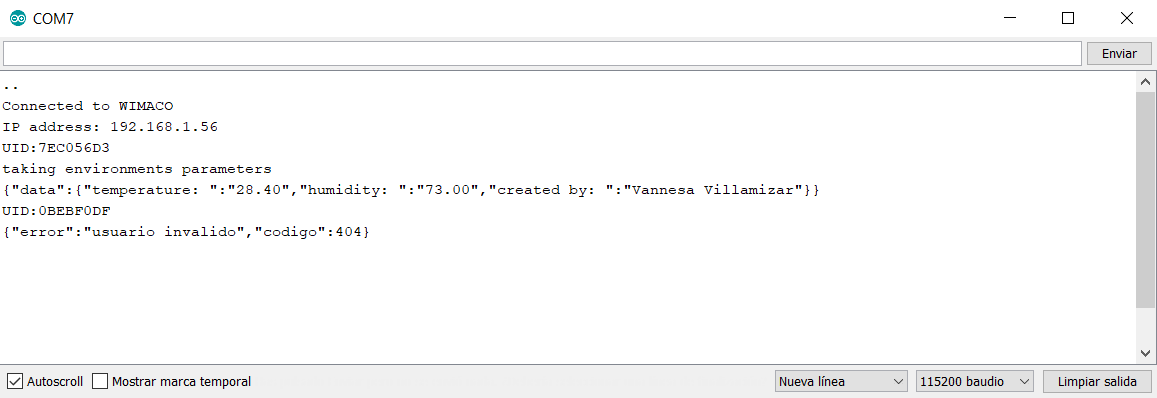
En el monitor serial de IDE Arduino veremos los acontecimientos del nodo. El dispositivo se ha conectado a la red local, ha realizado algunas lecturas UID consultando en la base de datos su existencia, se ha rechazado a “0BEBF0DF” y se han aceptado al otro par de tarjetas. Esto se debe a que el último par de tarjetas ya se encuentran en la base de datos dentro de la tabla “cards” como se muestra en la figura 5.



*Figura. 6: Log del monitor serial IDE Arduino*

*Figura. 7: Base de datos de las tarjeras: Usuarios asociados a la UID.*

De ahora en adelante se crearán las muestras ambientales si y solo sí el usuario está dentro de la base de datos. La figura 9 muestra algunos valores de muestras creados por los usuarios. La figura 8 muestra la respuesta en el monitor serie al crease una muestra ambiental por un usuario autenticado y el rechazo de uno desconocido, correspondiendo a los recursos y la lógica de respuesta configurada en el framework.

*Figura. 8: Monitor serie.*

*Figura. 9: Tabla de temperaturas*

La tabla muestra que las temperaturas se organizan desde la más antigua a la más reciente (este atributo puede permutarse), se logra identificar quien hace la lectura mediante el campo “card\_access” allí se encuentra el ID (la llave foránea) o bien el “card\_id” del usuario, de que nodo proviene la lectura “node\_id” y a qué hora y fecha fue tomada la muestra.

**ANALISIS DE DATOS SOBRE 50 MUESTRAS AMBIENTALES.**

Una vez obtenidos las muestras a analizar, se procede a realizar dicho análisis utilizando el método de regresión lineal simple implementado en la herramienta Rstudio, en ella se explora y cuantifica la relación entre una variable dependiente en este caso la humedad y una variable independiente o predictora, la temperatura; una vez establecido y desarrollado dicho análisis, el sistema me proporciona una ecuación lineal con la cual se puede lograr predecir valores de humedad en determinado rango de temperaturas.

Para llevar a cabo este proceso, en este caso, se extrajo un archivo temps11 .xlsx ( .csv .txt entre otros) de la base de datos en la cual se encuentran los datos a analizar ( time, temperature, humidity), una vez extraído el documento se procede a instalar la librería “readxl” en Rstudio, ya que por defecto no me permite leer archivos provenientes de Excel. Para ello ingresamos en “File -> Import Datased -> From Excel…” una vez seleccionado nos aparecerá una ventana emergente advirtiéndonos que debemos instalar la librería “readxl” la cual procederemos a instalar, finalizada la instalación aparecerá un recuadro en el cual seleccionaremos nuestro archivo “temps11.xlsx” tal como se ve en la figura 10.

*Imagen que contiene captura de pantalla

Descripción generada automáticamente*

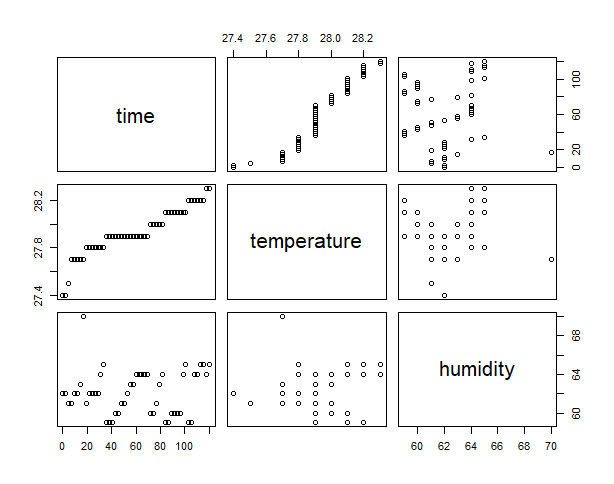
*Figura. 10: Archivo temps11.xlsx importado y vista previa del mismo en Rstudio.*

Como se puede ver en la figura 10, en la parte inferior derecha “Code Preview”, se establece el código inicial el cuan me describe la importación de la librería “readxl” en la primera línea de código, en la segunda línea, se asigna el nombre al archivo importado en este caso “domo” y la ruta de dicho archivo, precedido del comando read\_excel que me permite como su nombre lo indica leer el documento. Es importante saber cuáles serán los nombres de las variables con las cuales se va a trabajar por el resto del proceso, para ello se ingresa la tercera línea de código “names” el cual me identifica cada una de ellas y su columna respectiva, una vez establecidos estos parámetros procedemos a importar el archivo. Estos comandos aparecerán en la consola, una vez importado el archivo como se puede apreciar en la figura 11.



*Figura. 11: Comandos al importar el archivo temps11.xlsx en consola.*

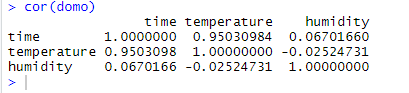
Para facilitar la identificación de una posible relación entre las variables y lograr una predicción más precisa, implementamos una matriz de diagrama de dispersión la cual representa la relación entre dos variables de forma gráfica, lo que hace más fácil visualizar e interpretar los datos, para ello ejecutamos el comando “pairs(domo)” como se representa en la figura 12.



*Figura. 12: Diagrama de dispersión de las variables.*

En la figura 12, podemos determinar que la variable “temperatura” tiene una mayor relación lineal positiva en cuanto al tiempo, dado que, en este caso, la temperatura transcurridas las 2 horas de muestreo, no presentó ningún cambio abrupto, por otro lado, si se compara con la relación tiempo-humedad se aprecia que ésta, no mantiene una relación lineal, por consiguiente se puede determinar que esta variable tiene poca correlación con respecto al tiempo al igual que la relación temperatura-humedad, en la cual se puede identificar el mismo comportamiento no lineal.

Dada la necesidad de cuantificar el grado de relación entre las variables calculamos el coeficiente de correlación de dichas variables con el comando cor(domo), la figura 13 me muestra la matriz de coeficientes de correlación en la cual se puede apreciar que la relación tiempo-temperatura hay una alta dependencia funcional entre ambas variables puesto que la correlación es fuerte y positiva aproximándose a 1; por el contrario en las relaciones temperatura-humedad y tiempo-humedad sus coeficientes de correlación están próximas a 0 lo que me indica una correlación débil y una casi parcial dependencia entre las variables.



*Figura. 13: Matriz de coeficientes de correlación.*

Predecir datos de humedad basándonos en la temperatura se logra implementando el método de mínimos cuadrados, este consiste en una línea que hace mínima la suma de los cuadrados de las desviaciones respecto a ella de todos los puntos que corresponden a la [información](https://www.monografias.com/trabajos7/sisinf/sisinf.shtml) recogida y almacenada en las variables “temperature” y “humidity” respectivamente, este método ayuda a determinar la ecuación de la recta que posteriormente funcionara como la ecuación predictora, para establecer los valores de la ecuación es indispensable identificar cual es nuestra variable dependiente en este caso “humidity” y cuál es la variable independiente “temperature”, ya identificadas se ingresa la línea de comandos que se muestra en la figura 14.

El comando para realizar la operación del modelo de mínimos cuadrados en Rstudio es lm() el segundo argumento que se encuentra dentro de este comando, se determina como data , este especifica cuál es el fichero en el que se encuentran las variables. El resultado de este proceso lo guardamos en un objeto llamado “regresión”, este objeto es una lista que contiene toda la información relevante sobre el análisis; posteriormente mediante el comando “summary” obtenemos los resultados.

Imagen que contiene captura de pantalla

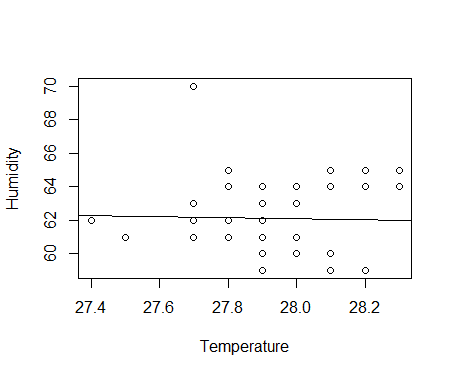
Descripción generada automáticamente

*Figura. 13: Resultados de mínimos cuadrados para Humedad y temperatura.*

La relación del porcentaje de humedad en el ambiente teniendo en cuenta la temperatura de este, está dada por los parámetros de la ecuación de la recta encontrados en la columna “Estimate” de la tabla “Coefficients”, por lo tanto, la ecuación de la recta por mínimos cuadrados estará descrita por la ecuación (1) y su representación gráfica en la figura 14.

(1)

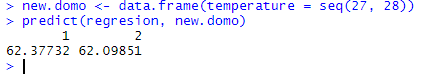
****



*Figura. 14: Representación gráfica de la recta de mínimos cuadrados*

El **coeficiente de determinación** mide la bondad del ajuste de la recta a los datos. A partir de los resultados anteriores, vemos que su valor en este caso es “Multiple R-squared= 0.0006374”.

Finalmente para lograr realizar la predicción, se debe crear un fichero en el cual agrupare las nuevas variables en este caso será “new.domo” y ejecutar el comando “predict” que me realiza la predicción en base a los resultados previamente obtenidos como se establece en la figura 15.



*Figura.15: Predicción de humedad en base a un rango de temperatura.*

Esta herramienta es realmente útil a la hora de llevar un análisis predictivo de los datos dado que a pesar de no tener un alto nivel de correlación entre estas variables se puede observar que las predicciones del factor humedad se encuentran en el rango de valores que tomó dicha variable durante la toma de muestras.

**CONCLUSIONES.**

EL hecho de que la autenticación sea remota, y se tenga la posibilidad de consultar una base de datos, le proporciona al sistema (o a la red de sensores) escalabilidad y robustez, además de reducir la carga de procesamiento y pre-procesamiento en los periféricos inalámbricos.

El modelo de base de datos resultó ser efectivo como una arquitectura escalable, la lógica de las consultas asociadas a la autenticación de los usuarios fueron implementadas satisfactoriamente, generando el paso o rechazo de aquellos usuarios autorizados o no autorizados respectivamente. Por simplicidad, consideramos en el proyecto dos modelos “cards” y “temps” (en “cards” se encuentra implícito un único nombre de usuario) relacionados de uno a muchos. Sin embargo, por practicidad este esquema relacional se puede escalar, por ejemplo a “users”, “cards” y “temps”, considerando ahora que un usuario puede tener muchas tarjetas (en caso de que este la pierda). Esto, entre otras cosas es una virtud de los modelos escalables.

Se ha considerado peticiones http para el script programado en el periférico y en las rutas del framework LARAVEL por estar familiarizados con su estructura en clientes y servidores. Sin embargo se pueden emplear otros tipos de protocolos de transferencia de datos usados habitualmente en entornos domotizados, por ejemplo, MQTT o

Los periféricos cuentan con posibilidad de pre-procesamiento de la información para establecer una cadena de caracteres compatibles con el encabezado y los requerimientos de la petición http.

El framework php LARAVEL tiene compatibilidad con múltiples motores de bases de datos. No obstante, utilizamos Postgresql debido a su facilidad para exportar (extraer datos) en formato “.csv”. En cuanto a la extrapolación en LARAVEL, la diferencia consiste arreglos configurados en el archivo .env en la ruta “c:/xampp/API\_temps”, por ejemplo el número de puerto de comunicación con el motor de base de datos.

En la elaboración del script para el periférico inalámbrico se consideró en primera instancia hacer una petición “GET” para la función de autenticación de usuarios en el API, sin embargo, una consulta de esta naturaleza expone atributos del usuario sobre la ruta web a la que apunta la petición. Considerando lo anterior, se ha decidido implementar usar una petición “POST” (como escritura) en lugar de “GET” donde los atributos no son expuestos de esta manera haciendo el servicio menos vulnerable.

La implementación de la herramienta Rstudio, facilita la interpretación de los resultados de los datos recolectados, cuantificándolos y dando la posibilidad de generar valores predictivos con base a realización de un análisis de correlación de estos, la predicción de un factor en específico puede mejorar significativamente integrando una mayor cantidad de muestras previas para establecer una mejor relación entre las variables y por consiguiente la posibilidad de establecer una efectiva toma de decisiones a futuro.

# **REFERENCIAS.**

del Valle Hernandez, L. (20 de Junio de 2017). *NodeMCU tutorial paso a paso desde cero*. Obtenido de programarfacil: https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/

Engineers, L. M. (2019). *What is RFID? How It Works? Interface RC522 RFID Module with Arduino*. Obtenido de Last Minute Engineers: https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/

Morales. (20 de Enero de 2017). *Lector RFID basado en RC522*. Obtenido de Playbyte: http://www.playbyte.es/electronica/arduino/lector-rfid-basado-en-rc522/

NUO. (2019). *¿Qué es RFID?* Obtenido de by.com: https://www.by.com.es/blog/que-es-rfid/

Sierra, K. (31 de Enero de 2018). *synergyweb*. (Fondo Europeo de Desarrollo Regional) Recuperado el 30 de Abril de 2019, de https://www.synergyweb.es/blog/laravel-desarrollo-medida/