UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de:

"INGENIERO ELECTRÓNICO"

"DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA DE CONTROL DE ACCESO BIOMÉTRICO CON CONFIGURACIÓN CENTRALIZADA DE BAJO COSTO UTILIZANDO RASPBERRY PI 3."

Autores:

Danny Ramiro Vaca Orrala

Juan Andrés Peña Coronel

Tutor:

Ing. Gino Alvarado

GUAYAQUIL, ECUADOR 2018

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

Los conceptos desarrollados en el presente proyecto de titulación, el desarrollo del tema, el análisis y respectivas conclusiones de este, corresponden exclusivamente a **Danny Ramiro Vaca Orrala** con cédula de identificación 0925010092 y **Juan Andrés Peña Coronel** con cédula de identificación 0929104628 y el patrimonio intelectual del mismo a la **Universidad Politécnica Salesiana.**

Guayaquil, febrero del 2018

(f) Danny Ramiro Vaca Orrala	(f) Juan Andrés Peña Coronel
C.I: 0925010092	C.I: 0929104628

DOCUMENTO DERECHOS DE AUTOR

Nosotros **Danny Ramiro Vaca Orrala** con cédula de identificación 0925010092 y **Juan Andrés Peña Coronel** con cédula de identificación 0929104628, manifestamos nuestra voluntad y cedemos a la **Universidad Politécnica Salesiana** la titularidad sobre los derechos del actual proyecto de titulación en virtud de que somos los autores del tema: "**DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA DE CONTROL DE ACCESO BIOMÉTRICO CON CONFIGURACIÓN CENTRALIZADA DE BAJO COSTO UTILIZANDO RASPBERRY PI 3**", mismo que ha sido realizado para la obtención del título de: **INGENIERO ELECTRÓNICO**, quedando la universidad facultada para ejercer y usar los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual del Ecuador, en nuestra condición de autores nos reservamos los derechos morales de la obra anteriormente citada.

En concordancia, suscribimos este documento en el momento que hagamos entrega del trabajo final en formato impreso y digital a la Biblioteca de la **Universidad Politécnica Salesiana**.

Guayaquil, febrero del 2018

(f) Danny Ramiro Vaca Orrala
C.I: 0925010092

(f) Juan Andrés Peña Coronel
C.I: 0929104628

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo Msc. Gino Alvarado certifico que bajo mi dirección y asesoría fue desarrollado el proyecto técnico de titulación "**Diseño e implementación de una plataforma de control de acceso biométrico con configuración centralizada de bajo costo utilizando Raspberry pi 3**", realizado por los señores **Danny Ramiro Vaca Orrala** y **Juan Andrés Peña Coronel** para la obtención del título de **Ingeniero Electrónico**.

Por medio de la presente certifico que el documento cumple con los requisitos establecidos en el Instructivo para la Estructura y Desarrollo de Trabajos de Titulación para pregrado de la Universidad Politécnica Salesiana.

En virtud de lo anterior, autorizo su presentación y aceptación como una obra auténtica y de alto valor académico.

Guayaquil, febrero del 2018

_____

(f) Gino Adrián Alvarado

DEDICATORIA

Dedico este proyecto de titulación a mis padres Ramiro y Alexandra quienes son los principales responsables de todos los éxitos y logros cosechados durante mi vida, sin su trabajo y sacrificio nada de esto sería posible, dedico también esto a mis hermanos Neil y Sandy quienes son parte importante de mi vida.

Danny Ramiro Vaca Orrala

Dedico este libro de tesis a mis padres, Benjamín Peña Campoverde y Blanquita Coronel Garcés que siempre han estado apoyándome y sacrificándose económicamente desde el primer momento que decidí este reto, y saber que he cumplido como hijo todo ese esfuerzo que plasmaron en mí, siendo un profesional de la Patria. También este éxito se lo dedico a mi hermano Rogger Peña Coronel que también forma parte de este camino largo de la Universidad, que sus palabras de motivación se haga realidad este logro.

Juan Andrés Peña Coronel

AGRADECIMIENTO

Estamos agradecidos con nuestros familiares que fueron nuestro pilar fundamental y que gracias a su apoyo moral, psicológico y económico nos han permitido llegar hasta este punto.

A cada uno de los compañeros y docentes con los que hemos tenido el gusto de compartir durante nuestra etapa universitaria.

A nuestro tutor Msc. Gino Alvarado quien siempre nos brindó su apoyo y experiencia para que podamos salir adelante con nuestro proyecto de titulación.

A nuestros revisores quienes con su sabiduría, conocimiento y experiencia ayudaron de una u otra forma a la realización de este proyecto de titulación.

Danny Ramiro Vaca Orrala

Juan Andrés Peña Coronel

AGRADECIMIENTO INSTITUCIONAL

Este agradecimiento va exclusivamente a los señores propietarios de la Lubricadora " Blanquita ", por habernos gentilmente permitido implementar nuestro proyecto en su local.

A ellos le gustó la idea de este proyecto, ya que les facilitó de llevar el control de entrada y salida de sus trabajadores para poder liquidar sus pagos semanales.

Danny Ramiro Vaca Orrala

Juan Andrés Peña Coronel

RESUMEN

El presente proyecto plantea el desarrollo de una plataforma de control de acceso biométrico cuyo objetivo es optimizar la gestión de accesos de empleados, se pudo observar que con otros sistemas la gestión no era muy eficiente debido a que los equipos estaban diseñados para trabajar de forma independiente, por lo que se decidió centralizar la configuración de accesos.

La centralización de la configuración de accesos consiste en la implementación de un API en un servidor web (Raspberry Pi 3) al cual se conectarán los dispositivos biométricos para obtener la respectiva información de los accesos y horarios de los empleados.

El servidor web dispondrá de una base de datos donde se almacenará toda la información y será gestionada desde una interfaz web la cual es visible desde cualquier navegador web lo que dotará al sistema de una facilidad de configuración.

Por su parte, los biométricos se conectarán al servidor central usando su dirección IP para obtener la información de los empleados registrados en el actual biométrico, cuando un empleado ubique su huella en el sensor biométrico el equipo realizará una petición al servidor central y enviará la información del empleado para que el servidor consulte en su base de datos y verifique si el empleado está dentro del horario permitido para permitir o denegar su acceso.

Además de esto el servidor admite el establecimiento de horarios de acceso para los empleados, es decir en la interfaz web del servidor se podrá configurar las horas en las cuales la persona tendrá permitido el acceso, esto es muy importante ya que el biométrico no solo

llevará el registro de entrada y salida de los empleados, sino que además controlará la apertura de la puerta.

El servidor central llevará un registro de todos los accesos realizados a cada biométrico en su base de datos, esto es con fines de control y estadística.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Las empresas usan control de acceso biométrico para registrar la entrada y salida de empleados de las instalaciones y poder tener un control de las horas laboradas, de los atrasos, por ende, estos dispositivos se han vuelto indispensables en todos lados.

Cuando la cantidad de empleados es grande el proceso de configuración no es tan simple ya que cada empleado debe acercarse personalmente al dispositivo para registrar su huella, cuando un mismo empleado debe tener acceso a diversos lugares donde hay un biométrico el proceso de configuración es más tedioso aún.

El proceso anteriormente descrito implica que tanto el empleado como el administrador de los dispositivos biométricos vayan personalmente al dispositivo para realizar la configuración, es por este motivo que en situaciones donde un mismo empleado debe acceder a varias zonas a la vez, se realiza al acceso mediante tarjetas magnéticas ya que estas se pueden configurar con la presencia del empleado.

EL uso de tarjetas no es el más seguro ya que el empleado puede perderla y cualquiera que la encuentre puede tener acceso a la ubicación restringida, igualmente si el empleado olvida la tarjeta ya no tiene acceso al lugar.

El presente proyecto de titulación busca ofrecer una solución a todos las situaciones anteriormente mencionadas, se va a implementar un servidor central donde se realizará toda la configuración vía web, es decir se podrá configurar desde un navegador web en un dispositivo conectado a la misma red del biométrico, desde la aplicación web se podrá agregar, editar, eliminar empleados, establecer horarios y biométricos de forma rápida, intuitiva y sencilla, toda esta configuración se realiza en el servidor y todos los

dispositivos biométricos se conectarán a este servidor para solicitar la información necesaria y poder controlar el acceso de cada empleado.

1.1. Descripción del problema

Los controles de acceso biométrico son una realidad hoy en día ya que en muchas instituciones se usan para permitir el acceso a ciertas instalaciones restringidas o simplemente para llevar un control de asistencia.

La principal ventaja de las soluciones de acceso biométrico es la seguridad, ya que la persona físicamente debe acercarse al dispositivo y ubicar su dedo en el sensor biométrico para poder acceder a algún lugar o simplemente registrar su presencia, mientras que con otras soluciones como tarjetas RFID, los empleados pueden intercambiar, perder o dañar las tarjetas, estos son importante desde un punto de vista funcional y de seguridad.

En el mercado existen diversos productos para el control de acceso mediante biométricos, los cuales están limitados por ser “stand alone (funcionan de forma independiente)”, no hay configuración centralizada, todos los cambios deben hacerse físicamente en él equipo lo que dificulta el control y estadística cuando se desea usar muchos equipos.

Si se desea instalar varios biométricos con los cuales los empleados podrán acceder a determinadas ubicaciones, es necesaria la configuración manual de cada uno de los dispositivos ya que no hay comunicación entre ellos, por ejemplo, si un empleado debe tener permitido el acceso a varios biométricos habrá que configurarlo manualmente en cada dispositivo, para una pequeña cantida de empleados el proceso puede ser

pequeño, pero para grandes cantidades de empleados el trabajo suele ser demasiado grande.

Los dispositivos más modernos tienen un sistema de accesos por horario, es decir, se pueden definir horarios para que el empleado pueda acceder a una determinada ubicación, pero como se mencionó anteriormente se debe hacer esta configuración en cada dispositivo biométrico.

En base a lo mencionado anteriormente podemos destacar 3 principales inconvenientes del sistema de control de acceso tradicional.

- Poca escalabilidad.
- Escaso control.
- Nula interoperabilidad.

1.1.1. Poca escalabilidad

Con los dispositivos biométricos más comerciales se debe configurar en cada dispositivo las huellas que tendrán permitido su acceso, si el empleado se desliga de la institución hay que eliminar la huella en el dispositivo, esto hace que se deba disponer de un personal capacitado para realizar los cambios cada vez que sean necesarios.

Este problema limita el crecimiento de la empresa, ya que si un empleado debe acceder a varios dispositivos se debe configurar uno por uno de forma manual cada dispositivo, de la misma forma si se desea eliminar el empleado de la base de datos de usuarios permitidos se debe ir a cada dispositivo a eliminarlo manualmente, esto implica un desgaste operativo enorme para solamente la gestión de accesos de empleados, lo cual a la larga resulta insostenible.

1.1.2. Escaso control

Dispositivos modernos disponen de una interfaz web para monitorización, pero estos tienen un precio más elevado de forma que si se desea instalar varios en una institución hay que realizar una inversión importante.

Los dispositivos comúnmente comerciales no disponen de interfaz de supervisión para obtener el historial de acceso cada dispositivo dispone de una base de datos interna la cual puede ser descargada mediante un USB, esto implica que si se desea hacer un control exhaustivo de los movimientos de los empleados se debes descargar la información de cada biométrico de forma manual y crear una base de datos nueva con la información consolidada de todos los biométricos para hacer el análisis.

Esto dificulta el control ya que para poder hacer el análisis se debe realizar todo un proceso de recuperación de la información.

1.1.3. Nula interoperabilidad

Estos equipos trabajan de forma independiente por lo que la información de accesos está alojada en cada uno de estos equipos, si por alguna circunstancia los equipos se descomponen la información se perderá por lo que se debe desarmar para extraer la tarjeta donde se encuentra la base de datos para poder recuperar la información, por lo que hay riesgo de pérdidas de información.

Al no disponer de supervisión web el administrador no se da cuenta de las averías hasta que llegan las quejas de los empleados al no poder tener acceso a su área de trabajo, de acuerdo con lo analizado en la sección de **Poca escalabilidad**, se debe volver a crear manualmente los registros de cada uno de los empleados, lo que implica un importante gasto de recursos humanos y logísticos.

Esta ausencia de comunicación entre los dispositivos provoca grandes problemas de escalabilidad ya que para cambios pequeños se puede llegar a requerir grandes costos operativos y sobre todo de un gran tiempo de implementación, lo que hace que los cambiso sean lentos y costosos.

1.2. Delimitación

1.2.1. Espacial

La implementación del presente proyecto de titulación será realizada en las instalaciones de la “Lubricadora Blanquita” que se encuentre ubicada en la Avenida Assad Bucaram Elmhallin entre El Oro y Maracaibo, tal como se muestra en la *Figura 1*.

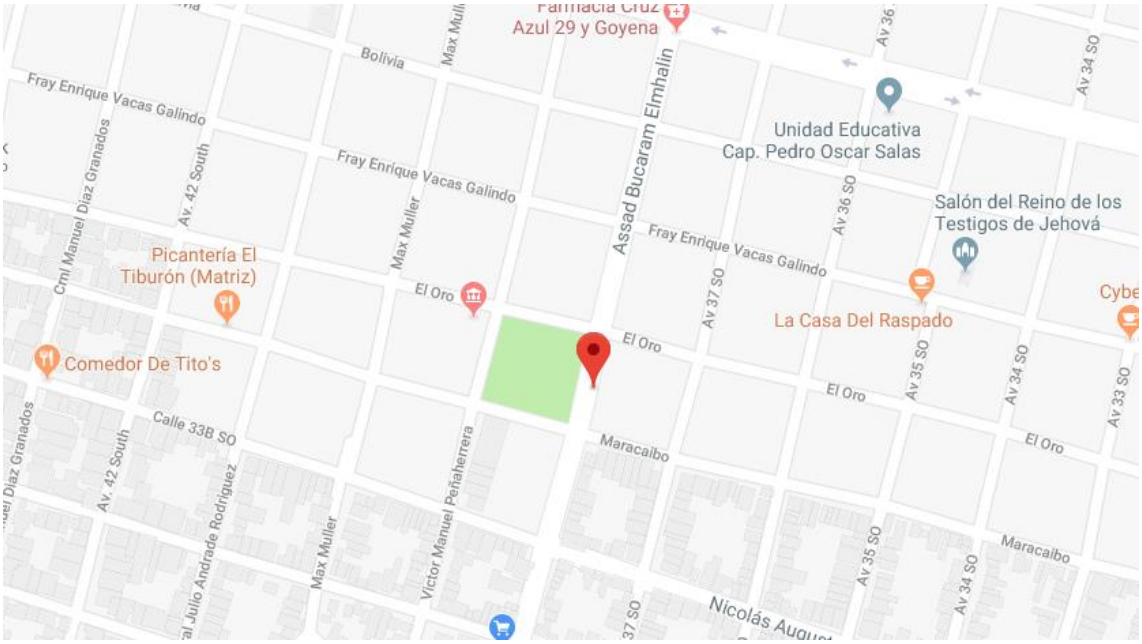


Figura 1 - Ubicación de la Lubricadora Blanquita. (Google Maps, 2018)

1.2.2. Temporal

EL presente proyecto de titulación será efectuado en el periodo de mayo del 2017 a julio del 2018.

1.2.3. Académica

El presente proyecto de titulación consiste en la creación de una plataforma de acceso biométrico de bajo costo usando Raspberry Pi, motivo por el cual es necesario tener buenos conocimientos en el área de la programación, electrónica y redes IP.

Es necesario entender cómo funcionan y poder desarrollar aplicaciones web para poder gestionar los equipos desde cualquier navegador, por lo tanto, es importante además saber usar herramientas como Laravel, Bootstrap y Vue para el desarrollo web.

1.3. Antecedentes

Hace mucho tiempo atrás se utilizaba el método de la almohadilla de tinta donde cada persona colocaba su dedo pulgar y de esa manera se registraba su huella digital.

En la actualidad, la mayoría de las personas utilizan contraseñas para poder acceder ya sea a lugar público o donde brinda servicios informáticos.

Esto proporciona sus ventajas y sus desventajas. Las ventajas sería mantener toda la información de los equipos informáticos bajo contraseñas ya que solo las personas que tengan el acceso al sistema puedan ingresar y manipular.

En cambio, las desventajas sería que la persona debe recordar las claves de acceso para cada lugar o sistema informático donde él necesite ingresar, por ende, esto ocasiona múltiples inconvenientes.

Con el uso de las tarjetas inteligentes es algo similar a las contraseñas ya que si dejamos olvidado la tarjeta no podríamos tener acceso para utilizar todos los beneficios que ellas nos brindan.

Hoy en día por lo general se utiliza las técnicas de la biometría ya que cada persona tiene sus propias características como los rasgos faciales, el patrón del iris del ojo, los rasgos de la escritura, la huella dactilar, etc. Por lo cual son únicas y difíciles poder falsificar.

El método más popular es el uso de huella digital, ya que ofrecen un alto grado de seguridad, confiabilidad y rapidez para la identificación de las personas, así como llevar el control de asistencia.

1.4. Importancia

En la actualidad, el uso del biométrico es fácil de usar y a la vez es seguro porque no se necesita ningún tipo de claves de acceso o códigos, ya que el elemento de identificación es una parte de nuestro cuerpo y no necesitamos un elemento externo como por ejemplo las llaves o tarjetas. Por lo cual, al no haber ningún elemento externo de identificación, no se necesita renovación cada cierto tiempo por caducidad, robo o pérdida.

Los lectores de huella digital cumplen estos beneficios, tales como:

- Evitar el uso de tarjetas magnéticas, ya que hay posibilidades que se estropee o extravié.
- Reconocer huellas digitales de cada persona, solo ellos pueden verificar la hora de entrada y salida.
- Evitar la suplantación de identidad, es decir, es virtualmente imposible falsificar los atributos físicos de una persona.
- Este tipo de sensores de huellas digitales no necesitan hardware ni software muy caro para su buen funcionamiento.

- Bajo costo de mantenimiento.

1.5. Alcance

El alcance del presente proyecto de titulación es:

- Elaborar un software desarrollado que permitirá llevar un registro y control de las asistencias del personal.
- Registrar los datos personales y de su huella dactilar para que quede registrado en la base de datos de cada uno de los trabajadores del establecimiento.
- Registrar la hora exacta de entrada y salida de los trabajadores.
- Indicar con un mensaje la hora actual en la que fue registrado.
- La herramienta podrá ser utilizada en varios lugares donde se necesite llevar un registro de asistencias.
- El sistema se limita a reportar un registro de asistencias, faltas, atrasos.

La solución de sistema biométrico brindada solucionó los problemas anteriormente descritos en la sección denominada **Descripción del problema**.

Se solucionó el problema de la escalabilidad mediante una arquitectura de cliente – servidor como se ve en la Figura 2.

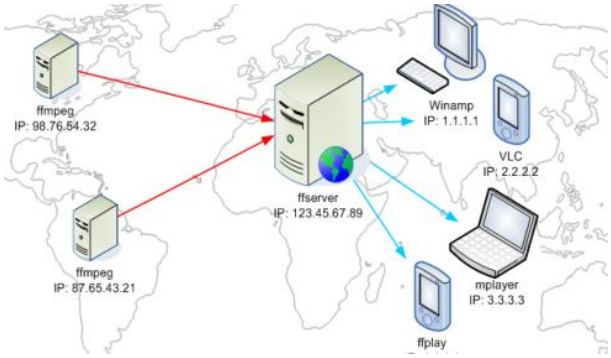


Figura 2 - Arquitectura cliente - servidor. (C.Calderón, 2014)

Con la arquitectura cliente - servidor cada biométrico solicita los datos de los empleados a un servidor centralizado que será hecho con una Raspberry Pi 3, el cual se puede apreciar en la Figura 3 y este a su vez le proporciona la información necesaria a través de un archivo en formato “JSON” permitiendo tener centralizada toda la configuración en el servidor además de permitir configurar de forma rápida los nuevos equipos ya que estos simplemente se conectan al servidor y descargan toda la información actualizada.

Figura 3 - Raspberry Pi 3 (Molloy, 2016)

Todos los equipos deberán estar conectados a una red IP mediante un cable de red o vía wifi para su gestión en remota, por razones de seguridad y eficiencia la red de dispositivos biométricos deberá ser independiente de la red de datos ya sea esto mediante el uso de VLANS, VPNS o creando una red completamente independiente, la plataforma de control biométrico funciona sobre cualquiera de estos escenarios.

El servidor central contiene la información de todos los empleados disponibles, así como la información de los accesos y horarios en una base de datos MySQL, la información será accesible mediante una elegante interfaz web la cual dispondrá de un diseño “responsive”, es decir, será completamente funcional tanto en dispositivos móviles con pantallas pequeñas y en dispositivos con pantallas más grandes como laptops o computadoras de escritorio como se puede apreciar en la *Figura 4*, esto

permitirá que los usuarios administradores puedan ver una lista de accesos con hora y fecha lo que facilita llevar un control de la información de los usuarios.

Figura 4 - Diseño responsive (Ceballos, 2013)

Al tener un servidor remoto los problemas de interoperabilidad desaparecen ya que es el servidor el que contiene la información de configuración de todos los equipos los cuales se intercambiarán mediante un archivo JSON el cual tiene un formato como el de la Figura 5, lo que significa que cada nuevo equipo que se conecte a la red descargará la información de configuración del servidor y se auto-configurará de forma automática.

```
{
  "University":
    {
     "Student":
              [
            {"idno": "134", "name": "Lakshmi", "dept": "CSE", "age": "19"},
              {"idno": "132", "name": "Karuna", "dept": "CNS", "age": "18"}
              ],
     "Faculty":
              [
              {"idno": "1301","name": "Krishna Mohan","dept": "CSE"},
              {"idno": "1404","name": "Tirupathi Rao","dept": "CSE"}
              ]
    }
}
```

Figura 5 - Estructura Json (B. Navya Rupa, 2015)

1.6. Justificación

El presente proyecto de titulación busca ofrecer una alternativa mucho más eficiente, escalable y rentable frente a los clásicos sensores biométricos.

Los sensores biométricos traen la desventaja de ser “stand alone[1]” es decir, operan de forma independiente, por lo que la información de cada equipo debe ser replicada manualmente entre todos los dispositivos, además de que la información de accesos recolectada por cada dispositivo se guarda en bases de datos independientes dentro de sí mismo, ciertos dispositivos tiene acceso remoto vía web pero estos tiene costos más elevados con lo que se puede extraer la información desde la interfaz web, pero de todas formas cada dispositivo tiene su base de datos independiente, debido a esto se complica la recolección y análisis de datos de accesos.

El actual proyecto de titulación va más allá de ser un simple sensor biométrico de los que actualmente dispone el mercado, este busca convertirse en una plataforma de control de acceso biométrico.

Como plataforma de control de acceso biométrico busca centralizar la configuración en un dispositivo central el cual dispone de una base de datos usando una arquitectura tal como se aprecia en la Figura 6, la base de datos estará instalada dentro del servidor y además de contener todos los parámetros de configuración para los dispositivos biométricos que se van a conectar a la plataforma, también guarda la información de empleados, horarios y accesos.

Esta centralización de la información permite brindar escalibilidad al sistema.

[1] Dispositivos que operan de forma independiente, es decir no se comunican ni interactúan con otros.

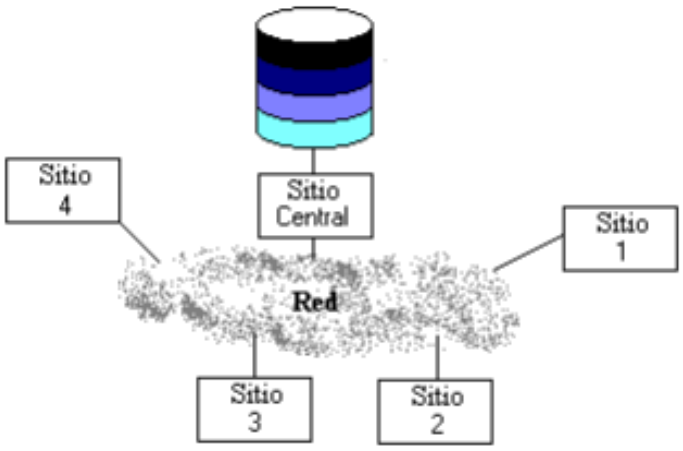


Figura 6 - Ubicación de la base de datos remota (Javier, 2001)

Cada cliente cuenta una interfaz web con la cual se configura los parámetros básicos como:

- Dirección IP del servidor
- Nombre de usuario del biométrico
- Contraseña del biométrico

Cada biométrico solicita su lista de empleados disponibles al servidor, el servidor consulta su base de datos de empleados y las devuelve el biométrico para que este sepa que usuarios pueden pasar y cuáles no, para lograr esto es necesario que el servidor tenga una IP estática de forma que los biométricos siempre sepan a qué dirección realizar las peticiones.

Una vez que el equipo se configuró con la información del servidor, este realiza el control de acceso biométrico registrando en la base de datos remota vía HTTP cada acceso de forma que toda la información repose en la base de datos del servidor principal esto hacer que la recolección de datos sea rápida y segura.

En el cliente existe un hilo asíncrono que se ejecuta cuando la Raspberry Pi se enciende el cual se encarga de gestionar la comunicación con el servidor, es el encargador de mantener al día la lista de usuarios, así como de registrar en el servidor los ingresos de usuarios que se realicen.

En el servidor no existe ningún hilo asíncrono corriendo ya que el funciona con un diseño orientado a eventos, es decir que solamente cuando se realiza la petición web desde el cliente el ejecuta su código para procesar la respuesta, si bien este proceso requiere estar a la escucha de datos entrantes por un socket, este proceso lo realiza a apache de forma automática y no requiere configuración extra por nuestra parte, por eso no se detalle ese proceso en el presente documento.

1.7. Objetivos

1.7.1. Objetivo general

Desarrollo de una plataforma de control de acceso biométrico para gestión remota de equipos biométricos.

1.7.2. Objetivos específicos

- Diseñar un equipo biométrico con Arduino y Raspberry Pi.
- Configurar la Raspberry Pi para funcionar como servidor web con el software Apache.
- Realizar una aplicación web para gestión remota del servidor central.
- Desarrollar una aplicación web para gestión local el biométrico
- Desarrollar un protocolo de comunicación entre los biométricos y la central mediante archivos JSON.

2. MARCO TEÓRICO REFERENCIAL

2.2. Arduino

Es una plataforma para el desarrollo de prototipos electrónicos open-source, la cual se basa en un hardware y software fácil de usar y de bajo costo, el cual permite que desde pequeños artistas y makers[2] hasta grandes empresas puedan desarrollar prototipos que van desde una pequeña estación climatológica hasta una impresora 3D ambas completamente controladas por un Arduino.

"Arduino es una plataforma de hardware libre, la cual está diseñada en una placa con un microcontrolador y un entorno de desarrollo abierto que facilita a los usuarios en el uso de la electrónica en proyectos multidisciplinares." (Elio, Encinas Martínez Jesús Ernesto & Ochoa Vela, 2015)

Entre las principales ventajas que nos brinda Arduino tenemos:

- Bajo costo – Las placas Arduino son la alternativa en microcontroladores más económicas del mercado.
- Multiplataforma – Se puede programar un Arduino desde cualquier sistema operativo sea Windows, Mac, GNU/Linux, etc.
- Gran comunidad – Para arduino se dispone de una comunidad inmensa de desarrolladores que comparten el código que escriben con la comunidad para que esta siga creciendo, esto hace que sea muy sencillo adquirir nuevos conocimientos en base a las experiencias de otros desarrolladores.
- Enorme ecosistema de librerías reutilizables creadaspor arduino y la comunidad.

[2] Es alguien que siente pasión y busca significado en la realización de prototipos de diversos indoles.

- Código abierto – Todo el código de Arduino, así como los esquemáticos de las placas es abierto y asequible para la comunidad lo que hace que sea fácilmente extensible.

2.3. Arduino nano

Uno de los modelos de arduinos disponibles en el mercado, este modelo destaca por su pequeño tamaño, bajo costo y buenas prestaciones.

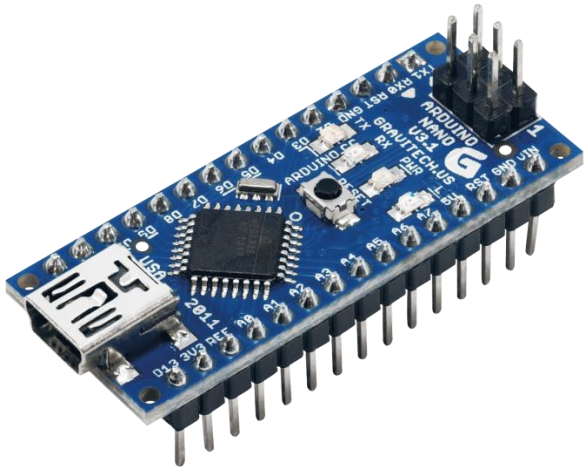


Figura 7 - Arduino Uno (Arduino, 2008)

El Arduino nano está basado en el chip ATmega328p y dispone de las mismas características que el Arduino uno, siendo este último más grande.

La arquitectura de este Arduino es AVR y trabaja a 5 voltios los cuales pueden ser suministrados a través del puerto mini-USB integrado en la placa.

Cuenta con una memoria flash de 32Kb, en donde se almacena el código de nuestro programa, de estos 32Kb disponibles 2Kb son usados por el bootloader[3].

[3] Es un software el que permite que los microcontroladores sean programados a través de su puerto serie sin la necesidad de un quemador de software externo.

2.4. Raspberry PI

Es una microcomputadora, la cual fue creada como una alternativa mucho más económica a los tradicionales computadores de escritorio para enseñar programación en países de escasos recursos debido a su bajo precio.

Pero debido a sus altas prestaciones y a su bajo precio se ha convertido en la plataforma predilecta para los “makers” alrededor del mundo.

La Raspberry Pi es de arquitectura ARM[4] por lo que puede ejecutar cualquier sistema operativo moderno que de soporte a esta arquitectura.

Actualmente el sistema operativo más usado es Raspbian, una distribución que deriva de Debian, es una de las distribuciones GNU/Linux más antiguas y robustas dentro del mundo Linux.

Para el presente proyecto de titulación es necesario usar dos Raspberry Pi, la primera será el servidor central donde se almacenará toda la información y la segunda será el núcleo de nuestro biométrico.

2.5. Raspberry PI 3

Para el servidor se utilizará una Raspberry Pi 3 *Figura 3*, debido a que es la Raspberry de mayores prestaciones, entre sus principales características destacan.

En esta Raspberry Pi 3 está almacenada toda la información de empleados, biométricos, horarios y accesos en una base de datos, además de tener instalado un

[4] Es una arquitectura de 32 bits para microcontroladores, fue desarrollada en 1983 por la empresa Acron Computers Ltd, usa un sistema de instrucciones simple lo que permite ejecutar tereas con un mínimo consumo de energía.

servidor web el cual permitirá ejecutar una aplicación de control para poder interactuar con los registros de la base de datos.

- Un procesador (CPU) QuadCore de 1.2Ghz Broadcom BCM2837 de 64 bits
- Una memoria RAM de 1Gb de capacidad.
- Una tarjeta Wireless LAN y Bluetooth Low Energy (BLE) BCM43438
- Dispone de 40 pines de uso general GPIO
- 4 puertos USB
- Salida HDMI.
- Como unidad de almacenamiento una ranura MicroSD
- Puerto de alimentación micro USB con soporte de 5 voltios hasta 2.5 amperios.

2.6. Raspberry PI Zero W

Para cada biométrico se usa una Raspberry Pi Zero W, ya que esta tiene un consumo energético, precio y tamaño menor que la Raspberry Pi 3, aunque por obvias razones su rendimiento también es inferior, pero es más que suficiente para albergar el biométrico.

La Raspberry Pi Zero será la encargada de comunicarse con el Arduino Nano para poder hacer uso del Biométrico y a su vez se comunica de forma inalámbrica con el servidor Raspberry Pi 3 para la descarga de la información de los empleados.

Esta Raspberry Pi Zero W, cuenta con un software de gestión web, pero que solo estará disponible de forma local y es accesible mediante una pantalla conectada al puerto mini HDMI de la Raspberry Pi Zero W.

Las principales características de la Raspberry Pi Zero W son:

- Un CPU SingleCore de 1Ghz
- Memoria RAM de 512Mb
- Puerto mini HDMI
- Dispone de 40 pines de uso general GPIO
- Tarjeta 802.11 b/g/n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- Como unidad de almacenamiento una ranura MicroSD
- Puerto de alimentación micro USB con soporte de 5 voltios hasta 2.5 amperios.
- 1 puerto micro USB

Figura 8 - Raspberry Pi Zero W (Peck, 2017)

2.7. Sensor biométrico

Es un sensor capaz de digitalizar las huellas dactilares de las personas, almacenarlas en su base de datos interna y compararlas para detectar coincidencias.

Dispone de un API el cual es accesible mediante comunicación serie y nos da control absoluto sobre la creación, edición, eliminación y comparación de huellas digitales.

La digitalización la realiza mediante un sensor óptico es cual es controlado por un microcontrolador que trae internamente el sensor.

Algunos modelos de biométricos tienen capacidad para almacenar hasta 100 huellas dactilares, pero estos tienen un precio más elevado, ya que hace falta una mayor cantidad y calidad de procesamiento para poder trabajar con esa cantidad de huellas.

Como el propósito del presente proyecto de titulación es realizar una plataforma de bajo costo se optó por un sensor con capacidad de hasta 25 huellas dactilares, que para la mayoría de los casos es suficiente, en caso de ser necesarios más huellas se puede cambiar por un sensor de mayor capacidad, ya que el programa diseñado si da soporte para ambos tipos de sensores.

Figura 9 - Sensor biométrico (Jimy Alexander Cortés Osorio, 2010)

2.8. Servidor web

Para que un dispositivo sea administrable vía web, es decir, a través de un navegador, es necesario que tenga instalado un software de servidor web, el cual pueda escuchar las peticiones realizadas y genere una respuesta para el mismo.

El navegador realiza peticiones HTTP/HTTPS las cuales por defecto van al puerto 80/443 respectivamente.

Cada servicio web tiene su propio puerto por el cual recibe las peticiones, los principales protocolos y sus respectivos puertos se detallan en la Tabla 1.

Tabla 1
Principales servicios web y sus puertos

Protocolo	Puerto	Descripción
21	ftp	Puerto del Protocolo de transferencia de archivos
22	ssh	Servicio de shell seguro
23	telnet	El servicio Telnet
25	smtp	Protocolo simple de transferencia de correo
80	http	Protocolo de transferencia de hipertexto para los servicios del World Wide Web
109	pop2	Protocolo Post Office versión 2
110	pop3	Protocolo Post Office versión 3
123	ntp	Protocolo de tiempo de red
143	imap	Protocolo de acceso a mensajes de Internet
161	snmp	Protocolo simple de administración de redes
443	https	Protocolo de transferencia de hipertexto seguro

Para el presente proyecto de titulación se usará el protocolo HTTP, pero perfectamente funciona con el protocolo HTTPS, ya que el navegador se encarga de realizar la petición y el servidor web se encarga de procesar la petición.

Una petición web tiene el formato de la Tabla 2

Tabla 2
Estructura de una petición web

https://	Tipo del protocolo de la petición
www.	Identificador del tipo de sitio
ups.edu.ec	Dominio o dirección solicitada
:80	Puerto al que se realiza la petición (defecto 80)
/noticias	Ruta del recurso solicitado
?articleId=10802587&version=1.0	Variables que se envía al servidor

Para el presente proyecto de titulación no se envía el identificador ya que vamos a usar un recurso local, el tendrá una dirección IP estática, tampoco se envia el puerto ya que por defecto se usa el puerto 80 que es el puerto del servicio http.

El protocolo http es un protocolo sin sesión, es decir, el servidor procesa la petición recibida y genera una respuesta e inmediatamente terminal la petición como se puede apreciar en la *Figura 10*.

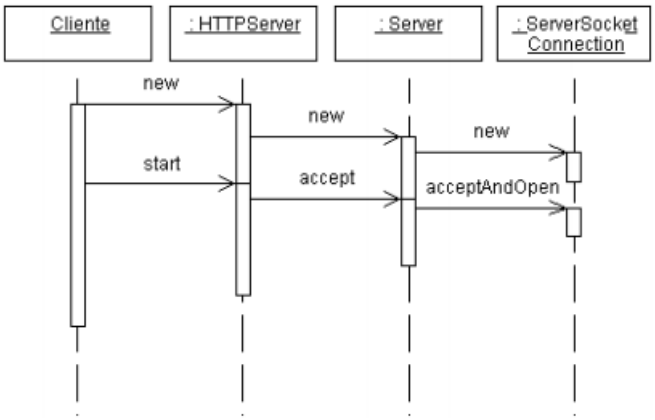


Figura 10 - Comunicación HTTP (Guillermo Diez-Andino Sancho, 2003)

En ocasiones como en las aplicaciones web es importante mantener una sesión iniciada para no pedirle credenciales en cada petición al usuario, estas situaciones se solucionan mediante cookies y variables de sesión, pero esto se realiza a nivel de programación.

2.9. Apache

El software de servidor web más usado del mercado es Apache como se puede apreciar en la *Figura 11*, para el presente proyecto de titulación se usa en concreto “apache2” el cual es la última versión del software dentro de los repositorios de software de Raspberry Pi.

Se prefiere el uso de apache frente a otras alternativas como Nginx o IIS debido a la gran cantidad de información, ya que la idea de esta plataforma de control de biométricos busca que sea fácil de implementar nuevas funcionalidades por eso se basa en un API Json, donde lo único que hace falta para comunicarse es la contraseña y el usuario del biométrico.

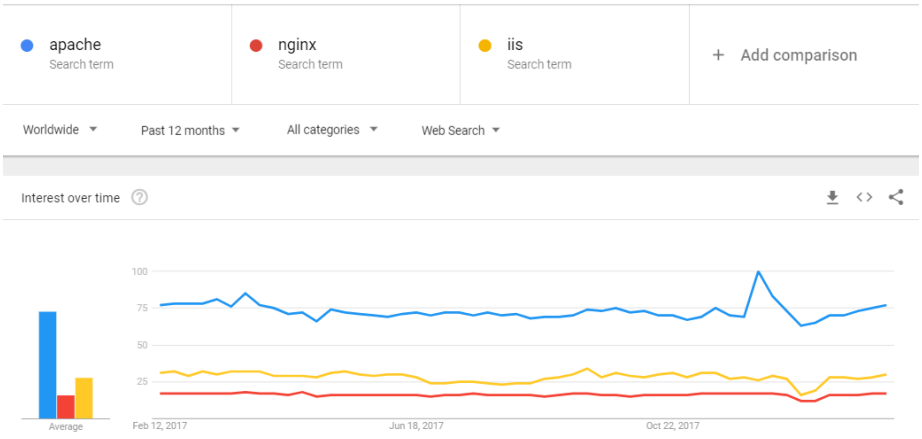


Figura 11 - Comparación entre servidores web. (Google Trends, 2018)

2.10. Base de datos

Una base de datos es un conjunto de datos almacenado de forma ordenada, organizada y relacionada para un posterior uso como se puede apreciar en la *Figura 12*.

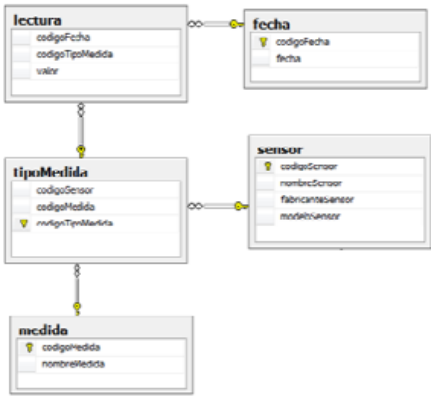


Figura 12 - Modelado de una base de datos relacional (María F. Maldonado, 2008)

Las principales características de una base de datos son:

- Integridad de los datos.
- Acceso concurrente para múltiples usuarios.
- Seguridad de acceso.
- Respaldo de datos.
- Acceso a la información mediante lenguajes de programación comunes.

2.11. Mysql

Es el gestor de bases de datos open-source más usado del mercado y es el elegido para gestionar la base de datos del servidor.

Es un gestor de bases de datos multiusuario y multihilo, permite realizar múltiples consultas desde varios usuarios a la vez, esto es importante ya que puede haber varios biométricos haciendo consultas al servidor y este debe consultar la base de datos para extraer la información.

Como se puede ver en la *Figura 13*, MySQL es el tercer gestor de bases de datos más usado del mundo, pero como SQL Server y Oracle son software propietario, es decir que se debe adquirir una licencia para poder usarlo de forma comercial quedan descartados como alternativa ya que hacen que el precio del desarrollo del proyecto aumente, adicionalmente estas alternativas son más pesadas, es decir, consumen más recursos y en cuanto a la Raspberry Pi que será nuestro servidor posiblemente no se dispongan de los recursos necesarios para gestionarlas.

La mayor parte del código está escrito en C/C++ y hace uso del lenguaje SQL para hacer las consultas a la base de datos.

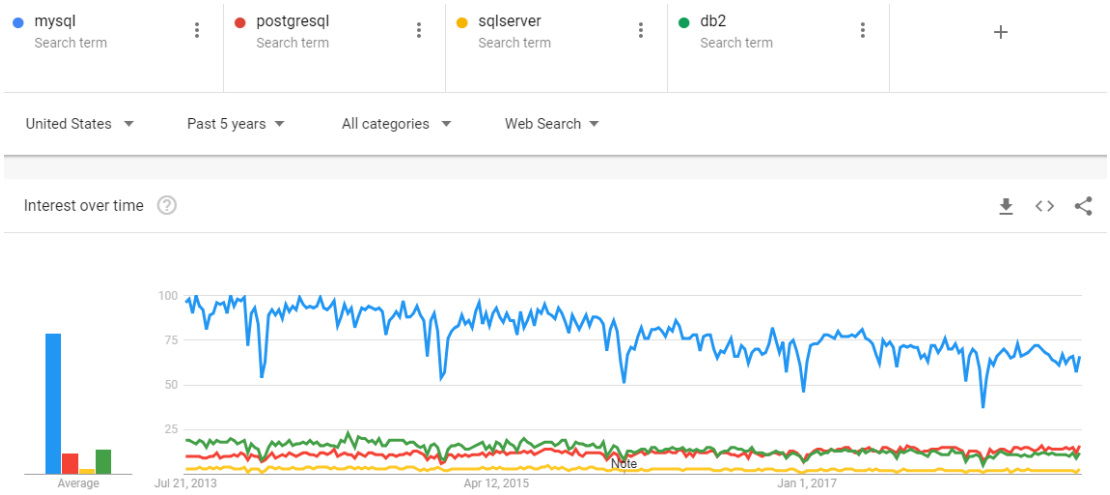


Figura 13 - Uso de los principales gestores de bases de datos (Google Trends, 2018)

2.12. Sqlite

Es un gestor de base de datos ligero, tal como lo indica el nombre “lite” fue diseñado para implementar todas las funcionalidades necesarias para una base de datos, pero haciendo un uso más eficiente de los recursos, ya que busca ser ejecutado en dispositivos de bajas o limitadas prestaciones.

Las principales características de este gestor de bases de datos son:

- Permite organizar una base de datos completa en un solo archivo “.sqlite3”.
- Permite funcionar enteramente en memoria, lo que hace que sea muy rápida.
- Tiene un footprint menor a 230Kb.
- Es totalmente auto contenido, es decir no tiene dependencias externas, lo que la hace portable, ya que simplemente con copiar un archivo se está haciendo un respaldo de toda la base de datos
- El código fuente es público y se encuentra muy bien documentado.
- Su consumo a nivel de memoria para realizar las consultas es muy pequeño en comparación con otras alternativas.

SQLite es el gestor de base de datos elegido para correr en el biométrico debido a su bajo consumo de recursos y bajos requerimientos mínimos como se puede ver la comparación con MySQL en la *Figura 14*.

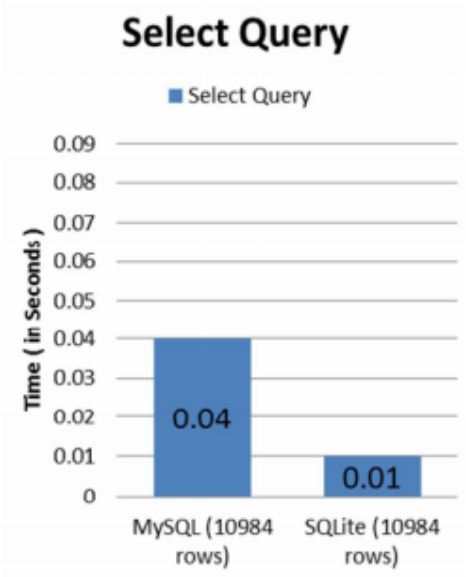


Figura 14 - Comparasión entre MySQL y SQLite (Jayesh Umre, 2014)

En él se guarda la información de configuración del biométrico como el usuario y la contraseña del biométrico, la lista de usuarios que se sincroniza con el servidor remoto, esto último es necesario para reducir el consumo de datos y aminorar la carga de trabajo en el servidor.

2.13. Programación en el servidor

Apache simplemente provee de los accesos vía web, en el servidor se debe programar una aplicación web para que el usuario pueda gestionar los biométricos, empleados, horarios y accesos.

Para programar aplicaciones web se dispone de varias alternativas del lado del servidor, entre las más conocidas tenemos:

- NodeJS
- Python
- PHP
- Ruby

La opción elegida para dar vida a nuestro servidor es PHP, ya que de igual forma que con apache y MySQL, es la alternativa más usada del mercado, esto hace que sea muy sencillo y económico conseguir personal capacitado si se desea agregar más funcionalidades a la aplicación creada en el presente proyecto de titulación.

2.14. PHP

Es un lenguaje de programación de propósito general, fue diseñado originalmente para generar páginas web dinámicas.

PHP fue uno de los primeros lenguajes de programación que aparecieron para aplicaciones web, de allí gran parte de su éxito.

En este lenguaje se programará la aplicación web del servidor, será el encargado de gestionar la base de datos y crear todas las interfaces web para que el usuario pueda administrar el biométrico.

Todas las peticiones web que reciba apache serán procesadas mienta un porgrama escrito en PHP, se crean multiples programas uno para cada posible petición, con el fin de generar una respuesta por cada recurso solicitado.

Como se puede apreciar en la *Figura 15*, PHP es el lenguaje de programación del lado del servidor más usado de la actualidad.

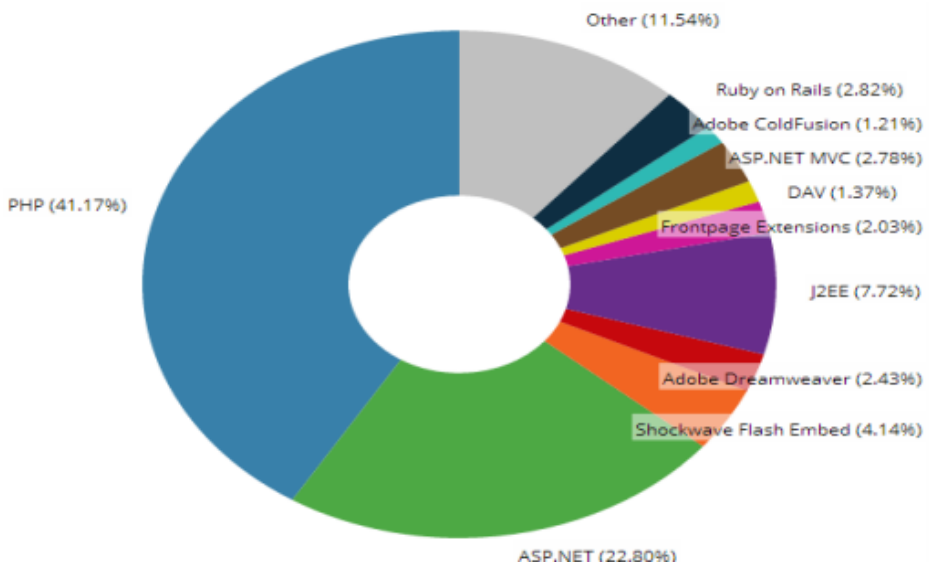


Figura 15 - Lenguaje de servidor má usados (Mishra, 2014)

Grandes proyectos como Moodle, Wordpress o Joomla lo utilizan en su backend[5].

2.15. Framework web

Un framework[6] es una estructura de proyecto compuesta por componentes personalizables, es decir es una especie de plantillas para desarrollar proyectos web.

El objetivo principal de un framework web es agilizar y dotar de una estructura escalable y reutilizable un proyecto, mediante buenas prácticas de desarrollo y patrones de diseño previamente definidos.

En todas las aplicaciones web son necesarios los siguientes componentes:

- Conector con base de datos.

[5] Es el software que se ejecuta en el servidor, no es visible al usuario por eso la terminología "back", esta capa es la responsable del acceso y manipulación de la información, así como el control de acceso a la misma.

[6] Es un esquema de trabajo definido, conceptos estandarizados y buenas prácticas previamente definidas para la creación de software, este representa la arquitectura base del software que se desea crear.

- Router para gestionar las rutas de la aplicación.
- Gestor de plantillas para crear vistas independientes y reutilizables.
- Implementación del patrón MVC.
- Entre otros.

Dependiendo del framework web elegido pueden venir integrados otros componentes los cuales buscan hacer más fácil la labor de diseño de un software web.

2.16. Laravel

Es un framework web de código abierto escrito en PHP, su filosofía es desarrollar de forma elegante y simple, fue creado en el 2011 y actualmente es el framework de PHP más usado para desarrollar aplicaciones web.

Laravel implementa el patrón de diseño MVC y además agrega muchas funcionalidades introducidas por ellos mismos, tales como:

- Services providers.
- Inyecciones de dependencias.
- Gesto de plantillas.
- Jobs.
- Rutas.

Estas características que buscan hacer más amigable el desarrollo de aplicaciones web y son parte fundamental del éxito de Laravel.

Cuando una petición llega al servidor primero debe pasar por una serie de pre-validaciones previa a la lógica de nuestra aplicación y una serie de post-validaciones antes de devolver alguna respuesta al cliente, validaciones tales como:

- Verificar inicio de sesión.
- Que la petición lleve los parámetros necesarios para trabajar.
- Guardar un registro de acciones.
- Ejecutar tareas asíncronas.

Para esto Laravel implementa middlewares[7] como se ve en la *Figura 16* los middlewares se ejecutan antes y después de la lógica de nuestra app.

Figura 16 - Funcionamiento de los middlewares en Laravel (Nguyen, 2015)

2.17. Patron de diseño MVC

El patrón de diseño MVC[8] que separa los componentes de una aplicación en 3 grandes tipos.

- Modelo
- Vista
- Controlador

[7] Es un software que se ejecuta antes del software principal, su función es pre-validar que las peticiones cumplas ciertas condiciones antes de ser atendidas por el servidor.

[8] Es un patrón de diseño que consiste en la separación de los componentes principales de la aplicación para facilitar su reutilización y simplificar su mantenimiento, se separa la base de datos (Modelo), la parte gráfica (Vista) y la lógica de negocio (Controlador).

Donde el modelo es la entidad que gestiona la base de datos, la vista es la interfaz gráfica que puede ver el usuario y el controlador es el código que se ejecuta en el servidor.

El patrón MVC se ejecuta tal como se ve en la *Figura 17*

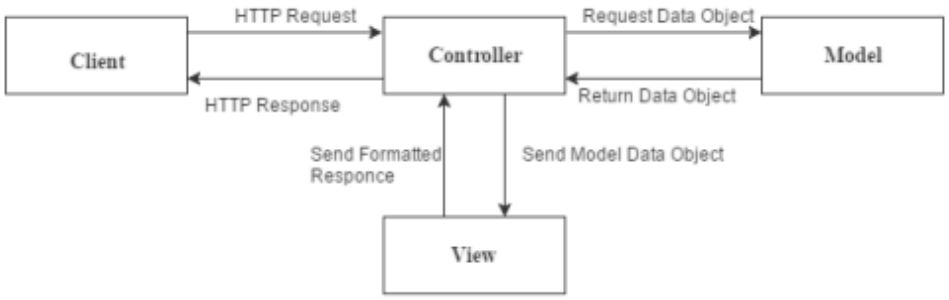


Figura 17 - Patron de diseño MVC (Nguyen, 2015)

2.18. Programación en el cliente

En el lado del cliente también se dispone de una aplicación web para gestionar el biométrico, pero esta aplicación debe tener acceso al hardware, más específicamente al puerto serie para que la Raspberry se pueda comunicar con el Arduino.

PHP no dispone de funciones para iniciar una comunicación a través del puerto serie, como es absolutamente necesaria la comunicación serie fue necesario optar por un lenguaje que si disponga de dicha funcionalidad.

Las principales características del lenguaje requeridas son:

- Lenguaje ligero y potente.
- Programación orientada a objetos.
- Fácil conexión con bases de datos.

- Disponer de un framework web de buenas prestaciones.
- Comunicación serie.

El lenguaje que mejor cumple todos los requerimientos es Python, más específicamente un framework web llamado Django, el cual está escrito en Python.

2.19. Python

Es un lenguaje de programación interpretado de uso general multiplataforma, cuya filosofía es ser un lenguaje que haga hincapié en la sintaxis, buscando siempre ser fácil de leer y entender para los desarrolladores.

Python fue el lenguaje de programación elegido para desarrollar la aplicación web del biométrico porque tiene una forma muy fácil de comunicación serial y porque existe un framework para aplicaciones web llamado Django.

Python será el encargado de gestionar la base de datos local del biométrico, capturar y responder a todas las peticiones web para que el usuario pueda interactuar con el biométrico mediante una interfaz web.

Como se puede apreciar en la *Figura 18* la sintaxis de Python es clara debido a:

- Permite colocar alias a las librerías importadas “import RPi.GPIO as GPIO”.
- Evita el uso de “{}” usando las tabulaciones para delimitar los bloques.
- Al ser un lenguaje de script, no necesita una estructura de proyecto, simplemente corre a partir de un archivo con extensión “py”.
- Requiere de pocas instrucciones para realizar grandes tareas gracias al gran ecosistema de librerías de las que dispone.

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

def setup():
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(18, GPIO.OUT)

def loop():
    GPIO.output(17, GPIO.HIGH)
    GPIO.output(18, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(17, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)

setup()
while True:
    loop()
```

Figura 18 - Programa de ejemplo en python, para encender y apagar leds

2.20. Django

Es un framework web escrito en Python lo que lo dota de la elegancia y versatilidad de Python.

Django usa el patrón de diseño MVC al igual que lo hace Laravel, Django basa su filosofía en la escritura de código reutilizable, fácil de leer el cual busca un desarrollo rápido y claro.

Las ventajas de Django son:

- Es un Framework web ligero.
- Tiene un gestor de plantillas.
- Permite utilizar todas las librerías de Python.
- Permite fácil conexión con bases de datos.

2.21. Bootstrap

Tanto Django como Laravel son framework que corren en el servidor y sirven para definir todas las funcionalidades de nuestra aplicación.

El navegador web solo puede entender 3 lenguajes:

- HTML
- CSS
- JavaScript

Cada uno de estos tiene una función específica.

El HTML sirve para definir la estructura o maqueta de nuestro proyecto, el CSS se encarga del acabado como formas, colores y ubicaciones de los elementos gráficos y finalmente Javascript se encarga de interactuar con el usuario.

Independientemente de si se usa PHP, Python u otro lenguaje de servidor todos estos deben devolver un archivo HTML el cual se enlace a los diferentes CSS y Javascript como se aprecia en la *Figura 19*.

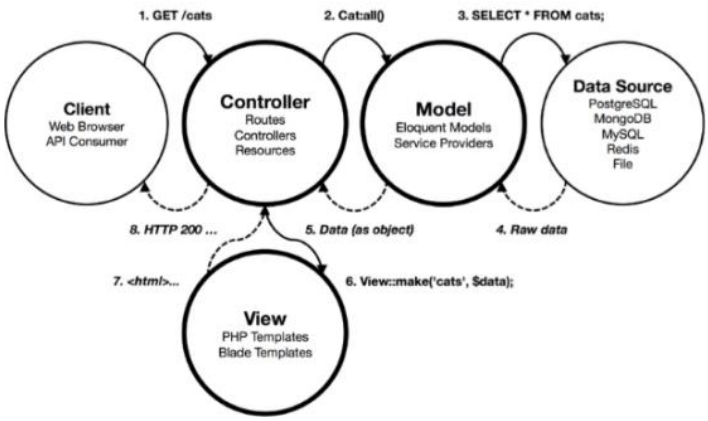


Figura 19 - Estructura simplificada de un proyecto web (Nguyen, 2015)

Para la parte del diseño se ha decidido usar Bootstrap[9], el cual es una librería de CSS y Javascript para del diseño de aplicaciones web elegantes visualmente.

2.22. API

Una API (del inglés: Application Programing Interface y en español: Interfaz de Programación de Aplicaciones) es un conjunto de clases, métodos y funciones los cuales cumplen diversas funciones y sirve para dar a los desarrolladores acceso a diferentes funciones de un sistema.

Una API es una capa de abstracción ya que les da a los desarrolladores una serie de herramientas que cumplen funcionalidades específicas y ellos no necesitan saber cómo funcionan o programarlas ellos mismos, pueden simplemente usarlas en su aplicación.

En la *Figura 20* se puede apreciar con diversos dispositivos (Android, IOS, Windows pone y Diversos navegadores) consumen un API, esta API realiza las consultas necesarias a la base de datos para generar una respuesta a cada cliente que realiza la consulta.

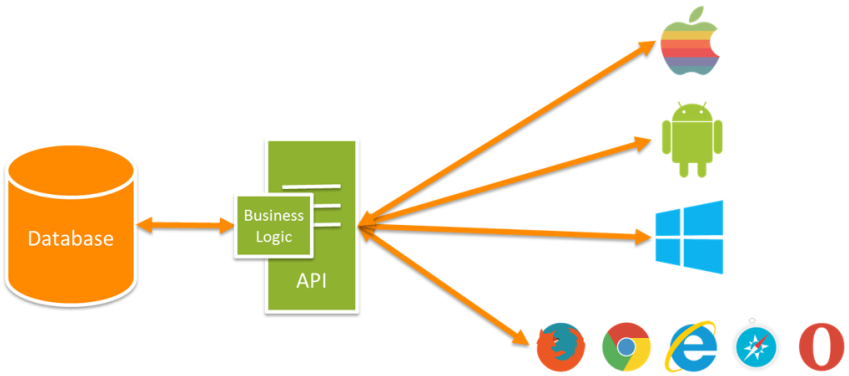


Figura 20 - Funcionamiento de una API (Mulloy, 2012)

[9] Es una librería de estilos para páginas web, las hojas de estilo son las que definen como se ve el sitio web, colores, formas, etc.

2.23. Web service

Es un conjunto de funciones, protocolos y estándares que usan las aplicaciones web para intercambiar información.

Un web service puede entenderse como una función que se ejecuta en un servidor remoto y que los clientes pueden usar “consumir” a través de internet, cuando se ejecuta esta función en el servidor remoto este retorna una respuesta al cliente, para que este continúe con su propia lógica de aplicación.

Las principales ventajas de los web services son:

- Aportan interoperabilidad entre aplicaciones ya que el servidor no necesita saber que clientes se van a conectar, que sistema operativo o lenguaje de programación ejecutan ya que la comunicación se hace vía http.
- Permite que se comuniquen diversas empresas u organizaciones alejadas remotamente.
- Se apoya en protocolos abiertos de internet por lo que no es necesario adquirir licencias para consumir un web service

2.24. DHCP

El DHCP (del inglés: Dinamic Host Configuration Protocol y en español: Protocolo de configuración dinámica de host) es el protocolo que permite que cada cliente de una red (disposivito conectado) pueda obtener una dirección IP de forma automática sin necesidad que el adminsitrdor de la red se la configure, esta IP dinámica la dará el router de un pool de dirección definidas por el administrador de la red, esto brinda mucha flexibilidad a las redes ya que se puede conectar y descoenctar dispositivos a la red de forma dinámica.

Este protocolo hace fáciles de usar la red ya que no hace falta que el administrador de la red le asigne dirección a los nuevos equipos que se conectan, en su lugar el router es el que le da una IP de la lista que tiene disponible.

2.25. ORM

Un ORM (del inglés: Object Relational Mapping y en español: Asignación Relacional de Objetos) consiste en la gestión de tablas de una base de datos mediante objetos.

SQL es el lenguaje por defecto para gestionar bases de datos, aunque cada gestor traiga pequeñas o grandes variaciones de este, en su esencia sigue siendo igual.

Representar una base de datos con objetos tiene las siguientes ventajas:

- Evitar tener que escribir SQL, ya que el objeto mismo lo genera.
- Los ORM modernos dan soporte a varias bases de datos, por lo que podemos cambiar de gestor de base de datos sin tener que reescribir “virtualmente” nada en nuestro código.
- Permite abstraer a los controladores de la lógica de la base de datos, es decir a nivel de nuestra aplicación nunca se hacen consultas a base de datos, simplemente se instancias objetos.
- Todos los ORM[10] modernos trae protección con ataques de inyección de SQL.

La única desventaja considerable del uso de un ORM es que en entornos de alta carga de trabajo las instanción y uso de objetos implica un uso mayor de memoria, cuando en su lugar podríamos escribir directamente el SQL sin necesidad de ningún objeto.

[10] Es un sistema que permite la gestión de tablas de una base de datos mediante la programación orientada a objetos.

Como se puede ver en la *Figura 21* el ORM está entre los objetos que representan las tablas de la base de datos y la base de datos, él es el encargado de que los objetos puedan modificar la base de datos.

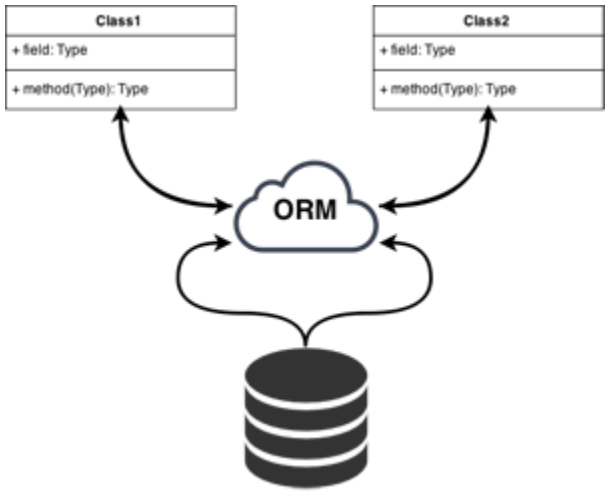


Figura 21 - Funcionamiento de un ORM (Nguyen, 2015)

Los ORM ofrecen la ventaja de abstraer la sintaxis de SQL, es decir el desarrollador no necesita aprender a escribir SQL ya que el ORM le genera el SQL en base a la estructura de objetos descrita.

Con la enorme cantidad de gestores de bases de datos que existen en la actualidad el uso de un ORM es casi obligatorio para el desarrollo web ya que permite una fácil migración entre gestores de bases de datos, además permite la reutilización de código entre proyectos, aunque estos usen un gestor de bases de datos diferentes.

Cada framework web suele straer su propio sistema de ORM, cada uno con sus ventajas y funciones propias, el ORM de Laravel se llama “Eloquent” y nos brinda una rica API para gestionar nuestras bases de datos, sin dejar de descubrir código PHP reduciendo de forma considerable el tiempo de desarrollo.

3. MARCO METODOLÓGICO

En el presente capítulo se exponen los aspectos metodológicos de la investigación y la propuesta de solución que se plantea para los problemas anteriormente descritos.

3.1. Métodología de la investigación

Para el desarrollo del presente proyecto de titulación se hizo uso de diversas metodologías adecuadas al problema planteado, los métodos se detallan a continuación.

3.1.1. Método de análisis y síntesis

El método de análisis y síntesis es un método teórico que consiste en analizar un todo complejo y dividirlo en diversas partes y componentes relativamente pequeños los cuales están relacionados entre si, el estudio individual de estas partes o componentes permite tener una idea clara del funcionamiento total.

En base a los resultados del análisis se procede a hacer una síntesis, lo que nos permite sistematizar el conocimiento, con el fin de tener claro a gran y pequeña escala el comportamiento de un sistema.

3.1.2. Investigaciónde biográfica documental

Este método consiste en la recolección de información mediante diversas fuentes tales como libros, publicaciones científicas, proyectos de titulación relacionados.

Todas estas fuentes son confiables, pero aún así cada una de las hipótesis e ideas extraidas fueron confirmadas mediante la repetición del experimento y la comparación de resultados para poder verificar su veracidad.

3.1.3. Método científico

El método científico está sustentando pos dos principales pilares que son:

La **reproducibilidad** que significa que el experimento se puede repetir por cualquier persona en cualquier parte del mundo bajo ciertas condiciones predefinidas y que el resultado será siempre el mismo.

La **refutabilidad** que implica que cualquier resultado obtenido puede ser refutado mediante otro experimento, lo que implica que los resultados no son absolutos.

Este método es importante ya que al extraer conocimientos de libros y artículos es necesario poder repetir los experimentos realizados en los mismos con el fin de poder usar sus conclusiones en nuestro proyecto investigativo.

3.1.4. Método experimental

El método experimental consiste en la realización de pruebas constantes a una hipótesis con la finalidad de verificarla o refutarla, estas pruebas se producen tanto en ambientes controlados, así como em ambientes de producción donde existen otros factores externos que pueden variar los resultados obtenidos.

Este método es importante ya que permite corroborar todos los datos y concluiones obtenidas en pasos previos.

3.2. Técnicas de investigación

Las técnicas de investigación son procedimientos metodológicos los cuales permiten implementar los métodos de investigación y nos permiten recoger información de forma inmediata.

Uno de los puntos más importantes para solucioanr un problema es tener la adecauda cantidad de información respecto al tema, para esto se utilizaron varias técncias de recolección de información.

3.2.1. La observación

La observación que es la técnica más antigua y universal que existe, nos permite obtener información directamente del objeto en cuestión, consiste en observar atentamente al fenómeno o caso de estudio, registrar toda la información importante para su posterior análisis.

3.2.2. La entrevista

La entrevista e suna técnica que consiste en el diálogo entre dos personas, el entrevistado y entrevistante, esto se hace con el fin de poder otener información de de parte del entrevistado, dicho entrevistado debe ser una persona que conozca del fenómeno.

3.3. Población y muestra

Por las características propias de nuestro proyecto no se requiere de una población, ni una muestra, ya que no se requiere el análisis de datos, sino en su lugar plantear una solución a una situación o problema.

3.4. Recolección de información

Para el presente proyecto de titulación se recopiló información de de libros yartículos científocos, así como de proyectos investigativos relacionados, todos estos se pueden encontrar en la sección PROYECTOS DE INVESTIGACIÓN VINCULADOS y REFERENCIAS BIBLIOGRÁFICAS.

3.5. Propuesta de solución

La arquitectura del presente proyecto de titulación consiste en un servidor web (Raspberry Pi 3) el cual centraliza la información tanto de estadísticas como de configuración, esto permite que los dispositivos biométricos (Raspberry Pi Zero y Arduino Nano) se conecten vía HTTP para sincronizar la información.

La comunicación será vía HTTP, el cliente subirá peticiones web y el servidor responderá con archivos JSON los cuales contiene la respuesta a la solicitud del cliente.

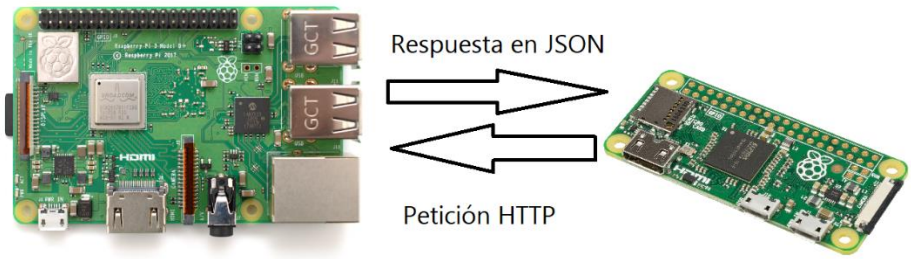


Figura 22 - Comunicación HTTP entre cliente y servidor

La raspberry Pi 3 funciona como servidor web y en su interior alberga una base de datos MySQL donde se almacenará toda la información, está base de datos es accesible gracias a una aplicación escrita en laravel la que permite que el usuario manipule desde la interfaz web, el servidor a su vez brinda un servicio web, el cual permite que los clientes autenticados sincronicen su base de datos local SQLite con la del servidor, mediante un hilo de comunicación asíncrono escrito en Python, el cliente a su vez dispone de una interfaz web para gestionar los accesos, esta aplicación está escrita en Django, la arquitectura en términos generales del sistema se muestra en la Figura 23.

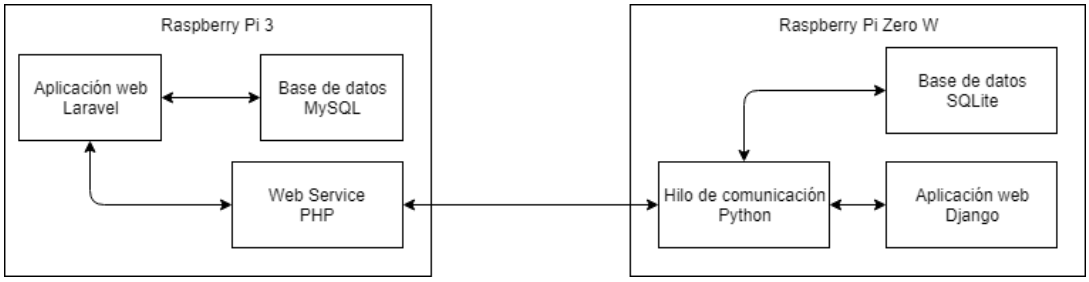


Figura 23 - Arquitectura de lo propuesta de solución

Para la digitalización de las huellas se usa un sensor biométrico el cual está conectado a la Raspberry Pi Zero mediante un Arduino Nano el cuál solomante hace las veces de una interfaz DAQ, los datos adquiridos por el Arduino son enavidos a la Raspberry Pi Zero W medianet el puerto USB, esta notifica a la Raspberry Pi 3 del ingreso del usuario para que almacene la información en la base de datos, tal como se muestra en la Figura 24.

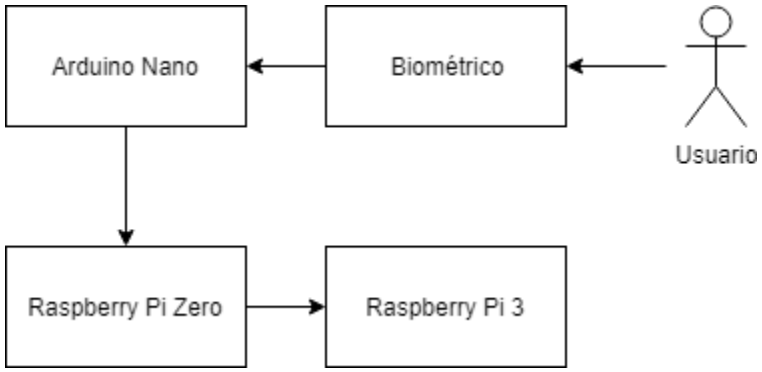


Figura 24 - Diagrama de adquisión de huellas

3.6. Arquitectura cliente – servidor

Para el presente proyecto de titulación se va a trabajar con una arquitectura cliente-servidor como se ve en la *Figura 25*, donde el servidor principal Raspberry Pi 3 centraliza toda la información y ofrece servicios web para que los clientes puedan acceder a la información.

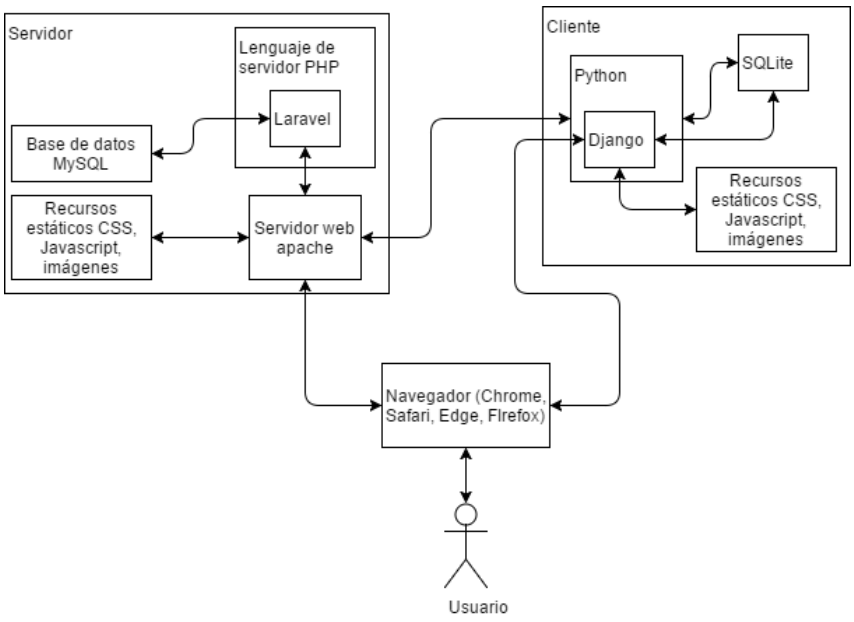


Figura 25 - Estructura del modelo cliente servidor

El servidor procesa las peticiones web mediante el software apache, el cual delega a la aplicación web escrita en PHP con el framework Laravel la responsabilidad de generar una respuesta para después devolverla al cliente, estas respuestas son o bien la página web solicitada (HTML, Css y JavaScript) o una respuesta en archivos JSON dependiendo de la URL solicitada.

El servidor envía respuestas en formato JSON cuando el cliente consume un “web service”, está es la forma de comunicación estándar entre APIs web y clientes.

El biométrico necesita la información que está alojada en el servidor y no se le debe dar acceso directo a la base de datos, por cuestiones de seguridad, por eso se usan los webs services, estos permiten tener un script el cual realice las acciones solicitadas por el cliente, de esta forma el servidor controla que acciones se realizan y siemplemente le dan al cliente el resultado de la ejecución, además de brindar seguridad hace más fácil la tarea de programación de aplicaciones del lado del cliente.

Las páginas web también disponen de recursos estáticos como las hojas de estilo CSS[11] y el código JavaScript, todos estos archivos están en la carpeta pública del servidor, para poder tener un acceso más eficiente y rápido.

La carpeta púbica no pasa por ninguna ruta o ninguna validación, lo que significa que al momento de ser solicitados son devueltos al cliente.

La aplicación hecha en Laravel tiene definidas una serie de rutas para realizar diversas acciones que realiza el servidor, estas rutas definen todas las posibles acciones que realiza el cliente.

3.7. Estructura de la red

Se ubicó como servidor central de la plataforma de control de acceso biométrico una Raspberry Pi 3 debido a bajo consumo energético, bajo precio y buen rendimiento, esta se conecta a la red mediante un cable de red o vía wifi ya que la Raspberry Pi 3 dispone de ambas opciones, siempre es más seguro que la conexión sea por cable de red, pero esto queda a la elección del usuario de acuerdo con sus necesidades.

La red no necesita tener acceso a internet por lo que con la instalación y configuración de un router es suficiente, ya que siemplemente se necesita tener conectividad entre los equipos en cuestión, ya sea cableada o inalámbrica.

En la *Figura 26* se puede apreciar la estructura que tiene la red, todos los dispositivos tanto servidor, biométrico o usuario deben estar conectados a la misma red.

[11] Del inglés Cascading Style Sheets y en español hojas de estilo en cascada son archivos que contiene la información acerca de como debe verse una página web, tantos colores, sombras, formas, etc.

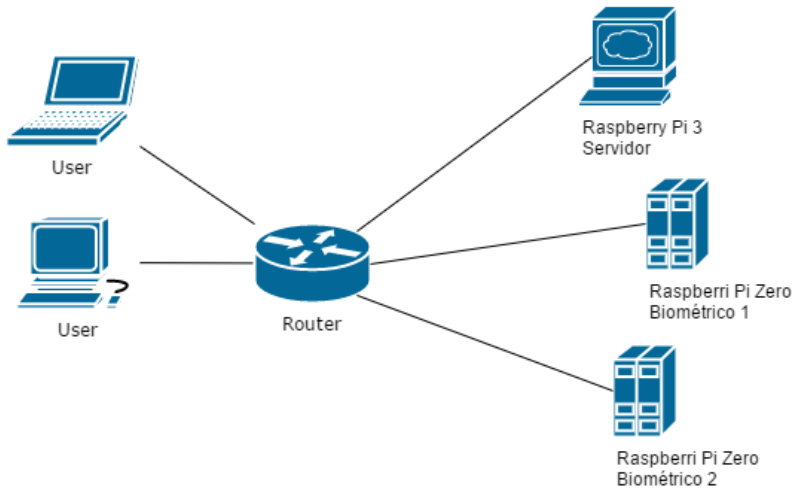


Figura 26 - Estructura de la red

Para que los usuarios puedan acceder a la interfaz web del servidor se necesita saber la dirección IP de este, de la misma forma para que los biométricos pueda buscar la información de los usuarios estos deben saber la dirección IP del servidor web.

El servidor tiene una IP estática que será “27.7.93.2” para que los demás dispositivos de la red puedan ubicarlo y comunicarse con él.

El resto de los dispositivos se reciben una IP de forma automática mediante el servicio de DHCP que brinda el router.

3.8. Servidor web

El servidor web es un software capaz de gestionar peticiones HTTP/HTTPS y se ejecuta del lado del servidor, es decir un cliente que desee realiza la petición solo necesitara de un navegador web, el cual en la actualidad es básico de todos los dispositivos.

El servidor web permite realizar peticiones unidireccionales o bidireccionales ya sean síncronas o asíncronas con el cliente, el software procesa las peticiones web y genera

una respuesta mediante un script que pude estar escrito en diversos lenguajes de programación.

Como software de servidor se va a utilizar Apache por ser el software de servidor web más utilizado en el mundo, con un índice de uso del 66% Figura 27 superando incluso a IIS de Microsoft.

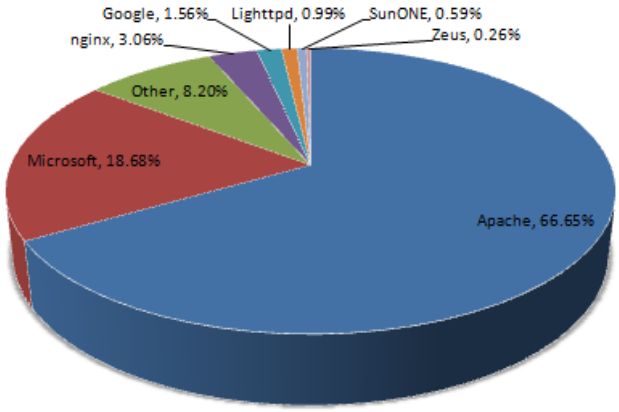


Figura 27 - Indice de uso de servidores web (Spectator, 2016)

3.9. Procesamiento de solicitudes

Apache procesa las solicitudes lidiando el con las cabeceras HTTP/HTTPS además de controlar los sockets TCP/UDP y gestionar los recursos físicos del servidor, pero Apache no es el encargado de generar las respuestas a las peticiones del cliente, para realizar esta tarea vamos a usar el lenguaje de programación PHP, el será el encargado de procesar la petición, realizar las consultas a la base de datos y devolver la respuesta al cliente

Se optó por PHP para este trabajo ya que en la actualidad es el lenguaje de programación del lado del servidor más usado Figura 28 superando inclusivo a

alternativas como el framework Ruby on Rails el cual es usado por Twitter y está escrito en el lenguaje de programación Ruby.

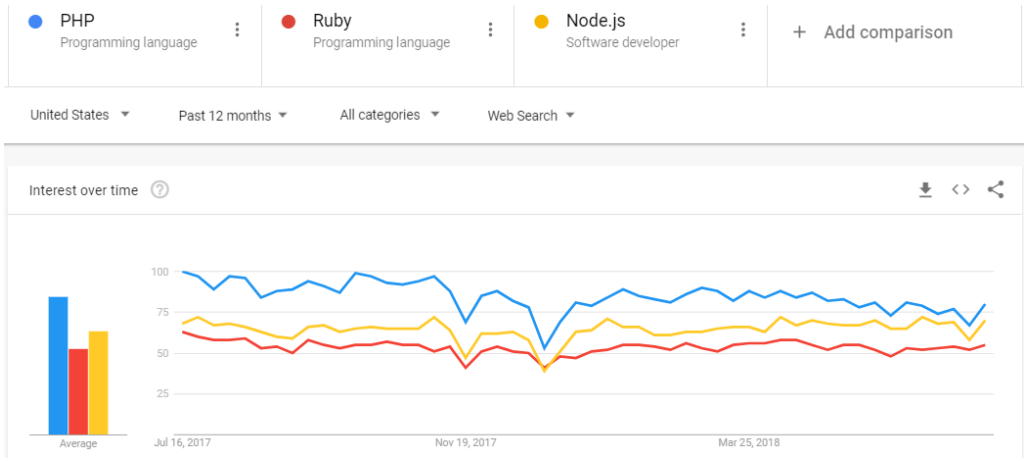


Figura 28 - Comparativo de uso de lenguajes en el servidor (Google trends, 2018)

PHP utiliza la información suministrada por el cliente para generar una respuesta acode a sus necesidades y a sus limitaciones.

De forma similar a las redes sociales de la actualidad, cada usuario inicia sesión con un usuario y una contraseña diferente, por lo que el servidor web le devuelve una página diferente a cada usuario, en la página estarán solamente sus noticias.

Hacer una aplicación web como la que se necesita para gestionar el servidor de los biométricos implica escribir una cantidad muy grande de código, en muchas ocasiones este código es el mismo que ya hemos escrito antes para proyectos anteriores, esto implica reescribir lo mismo una y otra vez, para evitar esta situación se usa un framework web, el cual ya trae parte del código más común para aplicaciones web, funciones tales como:

- Tabla de rutas.

- Conexión a la base de datos.
- Gestor de plantillas.
- Implementación del patrón MVC.
- Autenticación de usuarios-

3.10. Framework Laravel

PHP es un lenguaje muy robusto y eficiente además de ser muy flexible, gracias a todo eso es el más usado en la actualidad, pero hoy en día solo para desarrollar las bases de una aplicación web hace falta de muchos meses de desarrollo, ante esta situación surgieron muchos proyectos los cuales ofrecen bases sólidas para desarrollar aplicaciones web reduciendo de forma considerable el tiempo de desarrollo.

Existen muchas alternativas en diversos lenguajes de programación como por ejemplo Django para Python, Rails para Ruby o incluso el mismo Symfony para PHP, para el presente proyecto de titulación se usar Laravel, debido a las ventajas que ofrece las cuales se revisarán en detalle más adelante.

Un framework es el encargado de gestionar los cimientos de nuestras aplicaciones web, traen por defecto mucho código el cual siempre es el mismo en idea general entre todas las aplicaciones por lo que no es necesario escribirlo de nuevo para cada proyecto, sino que en su lugar se dispone de una plantilla de reutilizable para todos los proyectos web que se generen actualmente traen de forma común servicios de autenticación, modelos de bases de datos, tablas de rutas, gestor de plantillas para las vistas, etc.

En PHP los más destacados son Laravel, CodeIgniter y Synfoni, siendo Laravel el de mayor proyección Figura 29

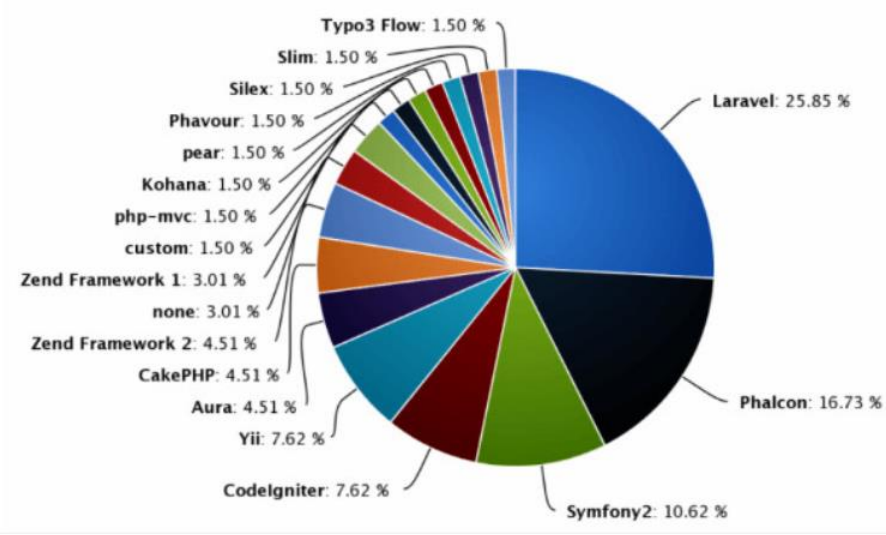


Figura 29 - Comparativa de frameworks PHP (Nguyen, 2015)

Aunque Laravel es un framework realtivamente nuevo, tiene una gran aceptación entre la comunidad de PHP debido a su facilida de uso.

3.11. Rutas o puntos de acceso

Las aplicaciones web tienen la característica de que son visibles a través de la URL mediante un navegador web, a diferencia de las aplicaciones de escritorio o móviles las cuales tiene un punto de entrada a partir del cual se ejecuta toda la aplicación.

Esto es una ventaja ya que el usuario puede acceder a ciertas funcionalidades de nuestra aplicación simplemente con la URL sin tener que pasar por toda la aplicación.

Laravel trae un archivo de rutas el cual define todos los puntos de acceso y además define que controlador será el encargado de gestionar la URL, es decir, define que acción se realiza en el servidor cuando el cliente solicite una URL.

Como se puede apreciar en la *Figura 30*, cuando un cliente (el biométrico) realizar una petición http (solicita datos al servidor), el “router” se encarga de seleccionar el

controlador correspondiente para la ruta solicitada, el controlador realiza todas las acciones definidas de la aplicación y genera una respuesta que puede ser una vista “view” o el respectivo “json” se la ruta era del API.

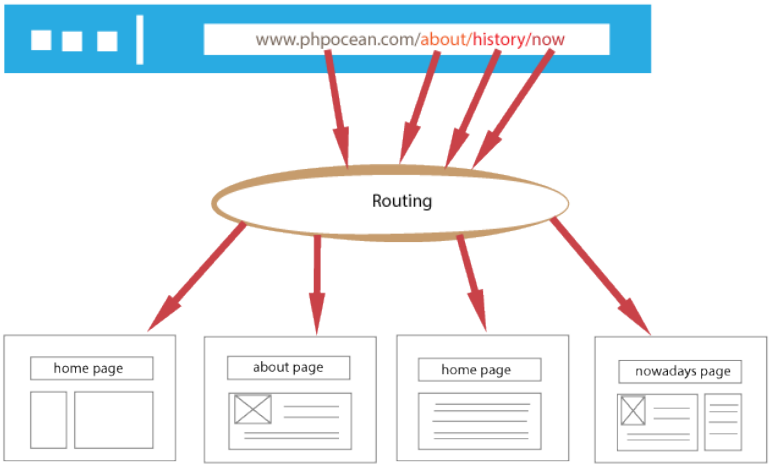


Figura 30 - Funcionamiento de las rutas en laravel (Kilicdagi, 2014)

3.12. Base de datos

La base de datos contiene toda la información de configuración de todos los dispositivos biométricos, contiene además la lista de usuarios, así como la lista de lugares a donde tiene acceso cada usuario y finalmente la lista de accesos con hora y fecha.

La base de datos busca ser la fuente de la verdad donde se almacenará toda la información.

Se usa una base de datos MySQL debido a su amplia aceptación en el mercado Figura 31, además de su modelo relacional que permite relacionar datos de varias tablas de forma muy sencilla y eficiente.

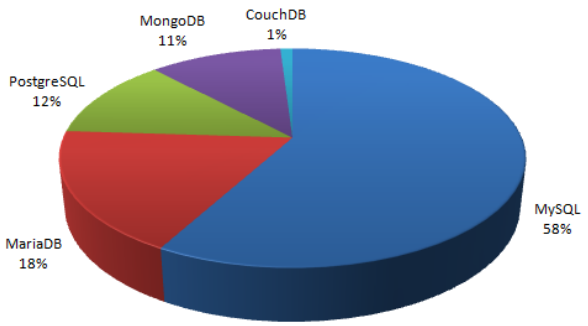


Figura 31 - Uso de bases de datos (Batini)

Las bases de datos MySQL son bases de datos relacionales, es decir se pueden guardar los datos en diferentes tablas para optimizar las consultas, pero están relacionados entre sí de ser necesario, mediante el uso de claves foráneas.

Esta base de datos se encuentra en el servidor (Raspberry Pi) y cada vez que un equipo nuevo (Biométrico compatible) se conecte a la red y sepa la dirección IP del servidor, este solicita la información al servidor para poder autenticar a los usuarios además notifica al servidor cada vez que un usuario desee acceder para que este lleve un registro de los accesos.

El acceso a la base de datos se lo puede hacer desde PHP, pero el código es un poco extenso y no es escalable, ya que un cambio en la estructura de la base de datos obliga a mover gran parte del código de la aplicación.

Ante este problema los accesos a la base de datos se realizan mediante entidades llamadas modelos los cuales son clases PHP que permiten gestionar la comunicación con la base de datos y abstraen toda la estructura interna y los comandos SQL necesario, los modelos permiten que los controladores interactúen con la base de datos a través de ellos, sin necesidad de una conexión directa con la base de datos.

3.13. Modelado de la base de datos del servidor

Laravel trae integrado un ORM el cual es el encargado de la conexión e interacción con la base de datos, el ORM de Laravel se llama Eloquent[12].

Eloquent es una interfaz para generar consultas a la base de datos nos permite usar la notación orientada a objetos de PHP para generar las líneas de SQL que se ejecuta sobre la base de datos como se puede ver en la *Figura 32*, es decir nos evitar tener que escribir SQL ya que Eloquent lo genera de forma automática.

Laravel trae de la misma forma que Eloquent un conector con la base de datos, en este caso nosotros usamos MySQL.

Eloquent usa el conector de base de datos de Laravel para ejecutar el SQL generado en la base de datos.

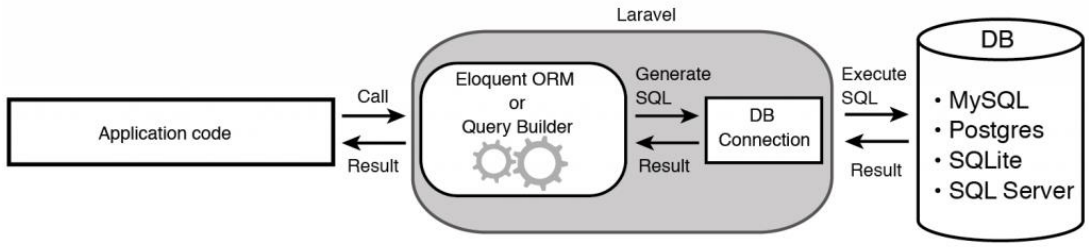


Figura 32 - Funcionamiento de un ORM (Kilicdagi, 2014)

3.13.1. Tabla de usuarios

La aplicación debe ser visible vía web, es decir, desde un navegador web, pero no esto no quiere decir que cualquier persona conectada a la web pueda gestionar el servidor, para que solo las personas autorizadas puedan gestionar el servidor la interfaz

[12] Es el ORM que viene incluido en Laravel, el permite la interacción con la base de datos mediante objetos.

web debe pedir usuario y contraseña al visitante antes de dar acceso a las funciones como se ve en la *Figura 33*, esta página se muestra cada vez que intente ingresar un usuario a la aplicación web.

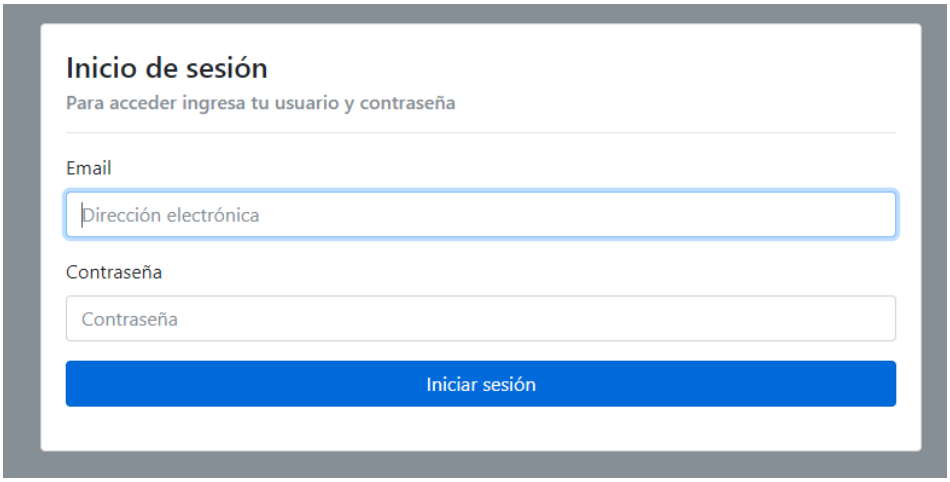


Figura 33 - Inicio de sesión del servidor

Para lograr eso es necesario tener una tabla de usuarios permitidos en la base de datos, en esta tabla se debe guardar toda la información relacionada al usuario.

La estructura de la tabla de usuarios permitidos es descrita en Tabla 3.

Tabla 3
Estructura de la tabla de usuarios

Nombre	Tipo	Clave primaria	Auto incrementable
Id	int(10)	Si	Si
Name	varchar(191)	No	No
Email	varchar(191)	No	No
Password	varchar(191)	No	No
remember_token	varchar(100)	No	No
created_at	timestamp	No	No
updated_at	timestamp	No	No
deleted_at	timestamp	No	No

Como se puede ver se almacena la información de nombre, email y contraseña del usuario, cabe resaltar que la contraseña está encriptada, lo que significa que el administrador del sistema no puede verla, lo que garantiza el acceso a la aplicación solamente por el usuario.

3.13.2. Tabla de empleados

Cada empleado que podrá acceder con su huella dactilar en el biométrico deberá estar registrado en el servidor, para esto es necesario otra tabla en la base de datos, la tabla de los empleados tiene la forma de la Tabla 4

Tabla 4
Estructura de la tabla de empleados

Nombre	Tipo	Clave primaria	Auto incrementable
Id	int(10)	Si	Si
Code	varchar(191)	No	No
Dni	varchar(191)	No	No
Name	varchar(191)	No	No
Password	varchar(191)	No	No
password_clean	varchar(191)	No	No
Lastname	timestamp	No	No
created_at	timestamp	No	No
updated_at	timestamp	No	No
deleted_at	timestamp	No	No

3.13.3. Tabla de biométricos

Los biométricos disponibles también están almacenados en la base de datos, de forma que el administrador del sistema siempre pueda borrar los que ya no sean necesarios y agregar biométricos nuevos.

La tabla de biométricos tiene la estructura definida en la Tabla 5

Tabla 5
Estructura de la tabla de biométricos

Nombre	Tipo	Clave primaria	Auto incrementable
id	int(10)	Si	Si
name	varchar(191)	No	No
username	varchar(191)	No	No
password	varchar(191)	No	No
created_at	timestamp	No	No
updated_at	timestamp	No	No
deleted_at	timestamp	No	No

3.13.4. Tabla de horarios

Los horarios de acceso son importantes para la aplicación ya que el empleado no puede acceder a cualquier hora del día, hay horarios en que ya no les permite pasar, pero como cada empleado puede tener horarios diferentes, se deben almacenar los horarios por separado, la estructura de la tabla de horarios está definida en la Tabla 6.

Tabla 6
Estructura de la table de horarios

Nombre	Tipo	Clave primaria	Auto incrementable
id	int(10)	Si	Si
name	varchar(191)	No	No
from	time	No	No
to	time	No	No
sunday	tinyint(1)	No	No
monday	tinyint(1)	No	No
tuesday	tinyint(1)	No	No
wednesday	tinyint(1)	No	No
thursday	tinyint(1)	No	No
friday	tinyint(1)	No	No
saturday	tinyint(1)	No	No
created_at	timestamp	No	No
updated_at	timestamp	No	No
deleted_at	timestamp	No	No

En esta tabla se debe guardar un horario con su nombre, hora de inicio y hora de finalización además los días en los que el horario es permitido.

3.13.5. Tabla de accesos

En las tablas anteriores se han definidos los empleados, biométricos y accesos, pero no se ha definido que empleado puede acceder a un biométrico y en que horario puede hacerlo, la tabla de accesos es la encargada de definir todas esas relaciones.

La Tabla 7 define la estructura de la tabla.

Tabla 7
Estructura de la tabla de accesos

Nombre	Tipo	Clave primaria	Auto incrementable	Clave foránea
id	int(10)	Si	Si	No
employee_id	int(10)	No	No	employees.id
biometric_id	int(10)	No	No	biometrics.id
schedule_id	int(10)	No	No	schedules.id
created_at	timestamp	No	No	No
updated_at	timestamp	No	No	No
deleted_at	timestamp	No	No	No

De esta tabla es importante resaltar que se usan campos numéricos que son claves foráneas para apuntar a valores de otras tablas, es decir, en estaba tabla no hay nombre ni contraseña del empleado, pero hay un campo “employee_id” el cual sirve para que la base de datos entienda que debe buscar en la tabla “employees” en el campo “id” el empleado con el id indicado en la tabla de accesos, estos lo logramos gracias al campo clave foránea, donde se indica en que tabla y con qué campo se debe hacer la relación.

De la forma explicada anteriormente usando la clave foránea se hace la relación para los biométricos y horarios.

3.13.6. Tabla de registros

Con las tablas anteriormente descritas el sistema tiene suficiente para saber si un empelado puede o no puede acceder, pero es importante tener un registro de los empleados que accedieron y en qué hora lo hicieron, para esto es necesario una nueva tabla donde se guarden el registro de cada empleado que accede.

La estructura de la tabla de registro de accesos está definida por Tabla 8.

Tabla 8
Estructura de la tabla de registros de accesos

Nombre	Tipo	Clave primaria	Auto incrememtable	Clave foranea
id	int(10)	Si	Si	No
employee_id	int(10)	No	No	employees.id
biometric_id	int(10)	No	No	biometrics.id
dateTime	datetime	No	No	No
created_at	timestamp	No	No	No
updated_at	timestamp	No	No	No
deleted_at	timestamp	No	No	No

3.13.7. Estructura total de la base de datos

La estructura final de la base de datos con todas sus relaciones se ve como en la *Figura 34* - Estructura de la base de datos.

Cada cuadro es una tabla de la base de datos y las líneas que las unen son las relaciones hechas con las claves foráneas.

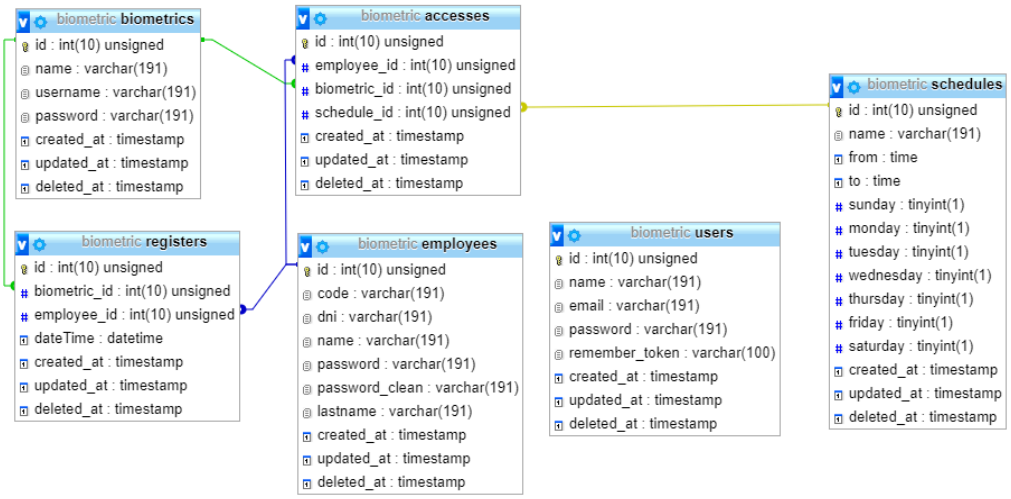


Figura 34 - Estructura de la base de datos

3.14. Blade y Bootstrap

Para el diseño de la aplicación web es importante evitar reescribir el mismo código cada vez, ya que eso hace que la aplicación no sea escalable ya que para hacer un cambio pequeño se debe modificar en todos lados de la aplicación.

En la parte visual de la aplicación pasa lo mismo, es importante separa la interfaz en “componentes” estos componentes deben ser individuales y funcionar independientemente del resto de componentes para poder reutilizarlos a lo largo de toda la aplicación.

Para este trabajo Laravel trae un gestor de plantillas llamado Blade el cual nos permite crear los componentes de forma individual, es decir en un archivo solo, de tal forma que le resto de componentes puedan usarlos simplemente con saber su ubicación como se puede ver en la *Figura 35*, tenemos componentes individuales como el encabezado llamado “header.blade.php”, la barra lateral llamada “sidebar.blade.php”, el contenido principal de la página web llamada “content.blade.php” y finalmente el pié

de página llamado “footer.blade.php”, Blade[13] los unir en une en un solo archivo el cual será enviado al cliente, pero nos permite gestionarlos por separado lo que facilita su mantenimiento y nos permite reutilizarlo entre varios archivos.

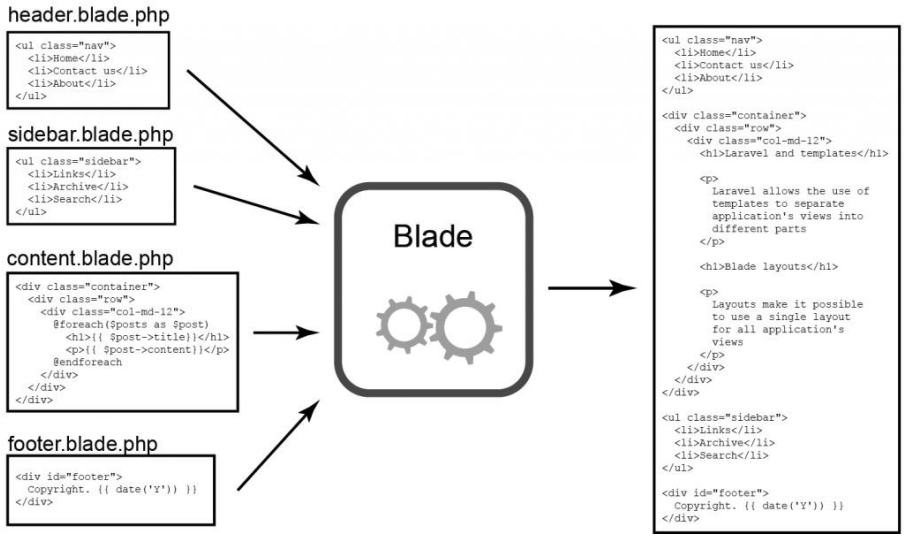


Figura 35 - Funcionamiento del gestor de plantillas Blade (Kilicdagi, 2014)

Blade permite crear los componentes individuales, pero no le da un diseño, es decir nos encontramos con componentes que no son agradables visualmente, estos componentes pueden cargar información de la base de datos e interactuar entre ellos, pero no se les defini un diseño como tal, en las aplicaciones web modenas es importante cuidar el apartado estético con el fin de que el usuario la use de forma cómoda.

Para eso usamos Bootstrap el cual es una librería de CSS la cual tiene muchos estilos definidos para que podamos usarlo en nuestra aplicación algunos de los cuales se pueden ver en la *Figura 36*.

[13] Es el gestor de plantillas de Laravel, el cual nos permite dividir la interfaz gráfica en varios componentes individuales, esto con el fin de reutilizarlos a lo largo del proyecto e incluso otros proyectos, además facilita el mantenimiento del software ya que se encuentra dividido en porciones más pequeñas y fáciles de manejar.

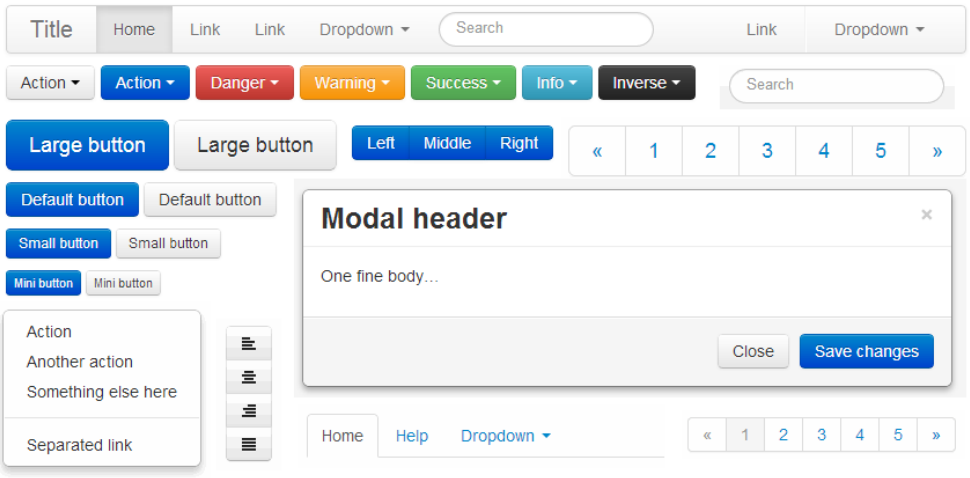


Figura 36 - Componentes por defecto de Boostrap (Fernando Bizzarro, 2016)

3.15. Interfaz web del servidor

La interfaz web de la aplicación contiene varias vistas las cuales permiten al usuario administrar los recursos de la aplicación.

Todas las vistas de la aplicación del servidor tienen en la parte superior la barra de opciones para que rápidamente podamos cambiar de vista.

Todos los listados de la aplicación disponen de 3 botones principales como se ven en la *Figura 37*.

- Botón de “exportar” para exportar los datos en un archivo de Excel.
- Botón de agregar para agregar un nuevo recurso
- Botón de buscar para realizar un filtro sobre la información

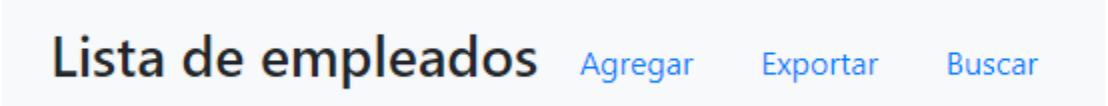


Figura 37 - Listado de acciones de los empleados.

3.15.1. Interfaz de administración

Las vistas de administración

Figura 38, muestra un listado de los recursos (empleados, biométricos, horarios, accesos y registros) registrados en el sistema, así como su información pública de cada recurso, nótese que no se debe mostrar las contraseñas, ya que esa información es privada del recurso.

Cada recurso dispone de 3 acciones principales que son

- Ver detalles “lupa”, sirve para ver las relaciones de cada recurso
- Editar “lapiz”, sirve para editar el recurso.
- Eliminar “tacho de basura” sirve para borrar el recurso.

La vista de registros no permite realizar ninguna acción, ay que los registros son permanentes y de solo lectura.

La vista de accesos no tiene botón de detalle ya que solo sirven para relacionar lso recursos previamente creados.

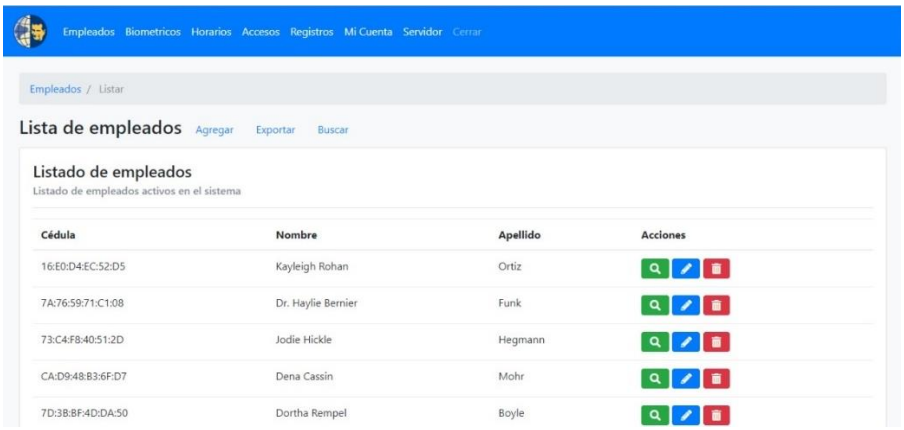


Figura 38 - Vista de administración de empleados

3.15.2. Interfaz de creación de recursos

Cada recurso dispone de una vista de creación excepto los registros ya que estos se generan automáticamente al poner el dedo en el biométrico.

Esta vista de creación es una ventana la cual nos pide la información relacionada al recurso para poder guardarla en la base de datos.

En la *Figura 39* se puede ver la vista de creación de empleados, se piden los datos necesarios para llenar los registros de la base de datos.

Figura 39 - Formulario de creación de empleados

3.15.3. Interfaz de edición de recursos

Cada registro que se hace en la base de datos puede editarse, ya que los datos se pueden actualizar o pudieron ser ingresados incorrectamente por él usuario, esta vista debe mostrar los datos previamente registrados y permitir editarlos, para que cuando el usuario pulse el botón de actualizar los datos se actualicen en el servidor.

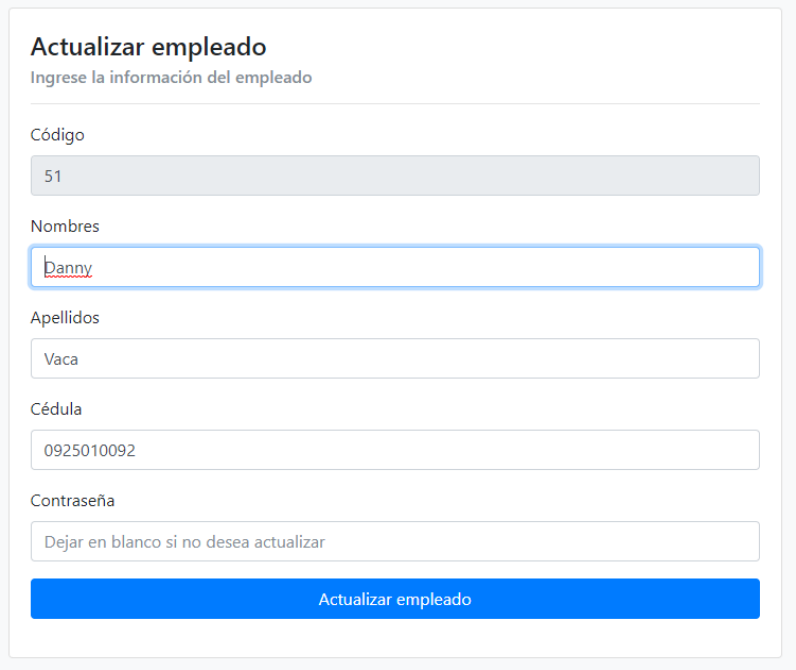


Figura 40 - Vista de edición de empleados.

3.15.4. Interfaz de eliminación de recursos

Cada recurso de la base de datos dispone de la opción de eliminar, la cual borra el registro, esto es necesario ya que cuando un empleado no trabaja más en el local debe ser eliminado su acceso, así como su información personal.

La aplicación muestra un mensaje de confirmación como el de la *Figura 41* antes de borrar el recurso.

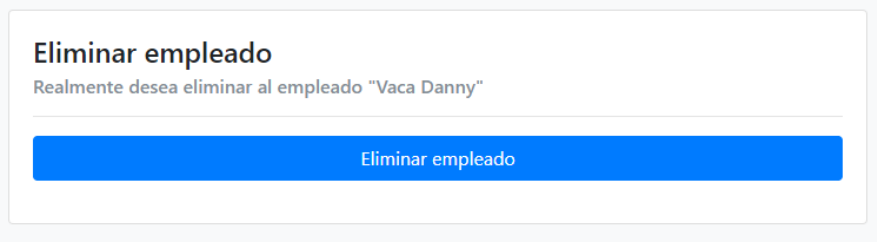


Figura 41 - Vista de eliminación de empleados.

3.16. Interfaz web del biométrico

El biométrico dispone de una interfaz web, pero esta solo es visible de forma local, es decir desde el mismo biométrico mediante la pantalla HDMI que tiene conectado no sobre la red.

El dispositivo biométrico no será visible en la red, es decir que él servidor web solo funciona de forma local ya que el no almacena información alguna, toda la información se almacena en el servidor principal el cual si dispone de una interfaz de administración web.

3.16.1. Interfaz principal

La interfaz principal *Figura 42* sirve para que el usuario pueda ver la hora de la marcación en la pantalla, pueda pulsa el botón para detectar huellas, además se cuenta con pestañas para moverse entre las demás opciones del biométrico.

Figura 42 - Ventana principal del biométrico

3.16.2. Interfaz de configuraciones

El biométrico está diseñado para que tenga una fácil configuración, por lo que dispone de una pestaña de “configuraciones” como se ve en la *Figura 43*.

Desde esta pestaña se puede configurar la dirección IP del servidor, el nombre y la contraseña del biométrico para que el servidor pueda autenticar a biométrico y así devolverle la información que solicite.

Figura 43 - Interfaz de configuración

3.16.3. Interfaz de pendientes

EL biométrico dispone de una ventana de “pendientes” como se ven la *Figura 44* donde se agregan las acciones que tiene pendiente el sistema y que deben hacerse con la confirmación del usuario, acciones principalmente como agregar un nuevo usuario.

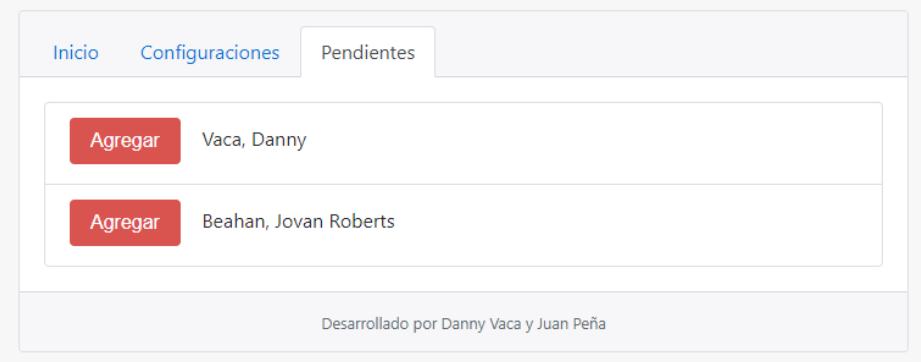


Figura 44 - Interaz de pendientes

Cuando se eliminar un empelado del servidor o simplemente se le quita el acceso a un determinado biométrico, el biométrico automáticamente lo borrar de su base de datos local, pero cuando se agrega un nuevo empleado como en el servidor no se dispone de un sensor biométrico con el fin de que el dispositivo sea lo más económico posible, se optó por pedir al usuario que registre nuevamente su huella en el biométrico cada vez que se lo agregue en un dispositivo.

Este registro de la huella en el dispositivo local implica que otra persona pueda cambiar su identificación, para solucionar este inconveniente propio de la arquitectura elegida para el desarrollo del proyecto, se optói porque cada usuario tenga una contraseña la cual la usará para verificar su identidad y así evitar que otra persona registre su huella.

Por razones de seguridad la primera vez que un usuario registra su huella en un biométrico se le pedirá la contraseña para poder verificar su identidad como se ve en la *Figura 45*.

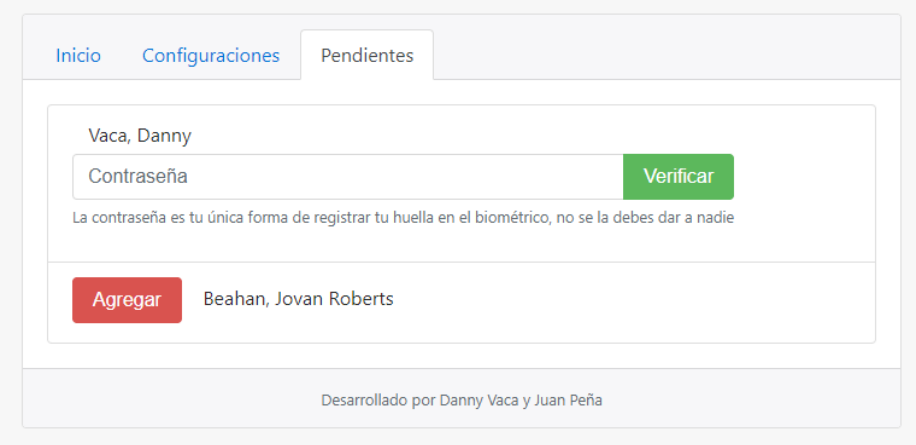


Figura 45 - Interfaz de pendientes, solicitando contraseña al empleado

3.16.4. Sección de notificaciones

El biométrico puede producir diferentes advertencias o mensajes de estado para informar su actividad al usuario, estos mensajes son desplegados en la interfaz del usuario para que el pueda leerlos al momento y tomar alguna acción de ser necesaria.

También se despliegan notificaciones cuando alguno de los datos ingresados en la sección de configuración del biométrico es incorrecto o existe alguna información que tanto el biométrico como el servidor necesiten notificar al usuario, estos mensajes informativos serán desplegados en la parte superior de la aplicación web como se ve en la *Figura 46*.

Figura 46 - Sección de notificaciones por error en las configuraciones

Estos mensajes son leidos del servidor de forma asíncrona o en su defecto el cliente los genera de forma automática y sirven para que el usuario del biométrico este informado de cualquier situación que exista en el biométrico.

Los mensajes de información no se almacenan en ninguna base de datos, se despliegan al momento de ser generados y una vez cancelados se eliminan, esto es así con el fin de no llenar la base de datos con información necesaria.

4. ANÁLISIS DE RESULTADOS

Para el presente proyecto de titulación se propuso como objetivo la realización de una plataforma de control de acceso biométrico, las fases de la elaboración del proyecto se detallan a continuación.

4.1. Diagrama de bloques del biométrico

El diagrama de bloque del esquemático es el de la *Figura 47*.

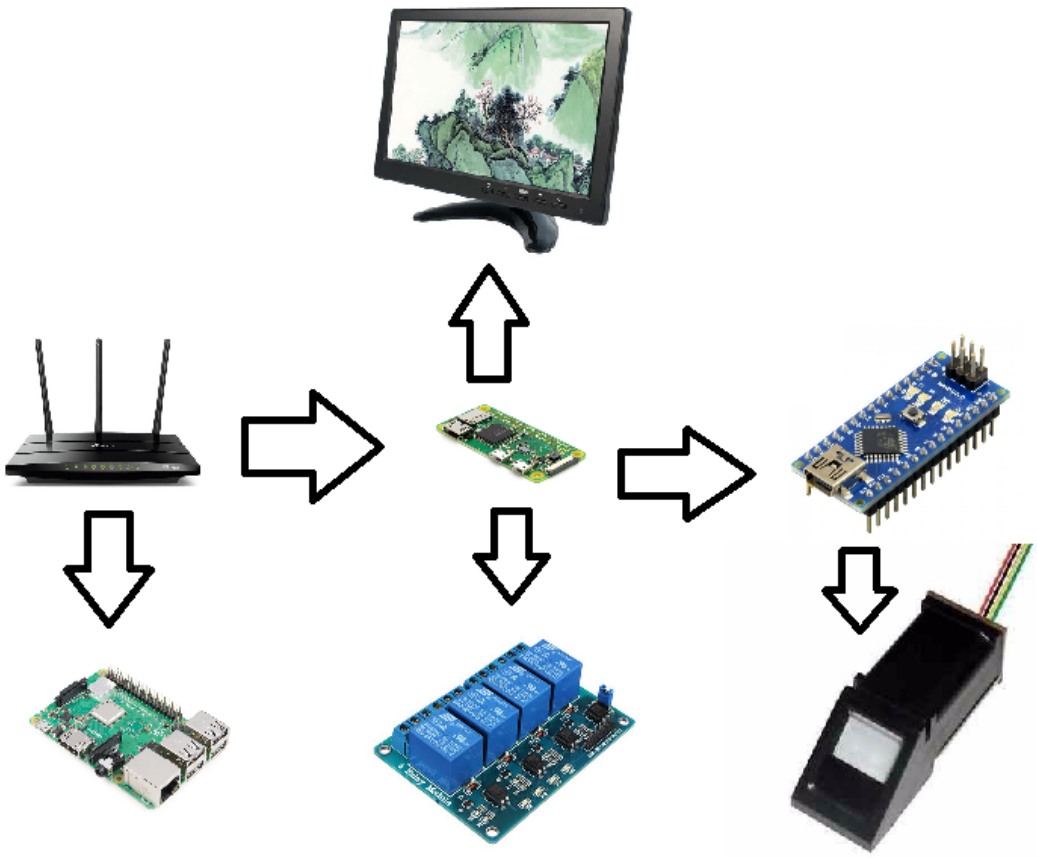


Figura 47 - Esquemático deo Biométrico

La Raspberry Pi Zero W, solo cuenta con un puerto micro USB, por lo que es necesario usar un conversor de micro USB a USB y a su vez usar Hub USB para

disponer de al menos 2 salidas UBS para poder conectar el Arduino Nano, el teclado y mouse del biométrico.

La salida de vídeo de la Raspberry Pi Zero W es un puerto mini HDMI, como la pantalla que se dispone es una pantalla HDMI fue necesario incluir un conversor de mini HDMI a HDMI.

El sensor biométrico se conectar al Arduino Nano a través del puerto serie por software que se programó para la comunicación con el biométrico, ya que el puerto serie por hardware que trae el Arduino Nano está siendo usado para la comunicación con la Raspberry.

La Raspberry Pi Zero W no cuenta con un puerto de red por lo que la única forma de conectarnos es por Wifi, o de ser necesario se puede usar un conversor de USB a Ethernet para conectarlo a la red, pero para nuestro caso con la comunicación wifi es suficiente.

4.2. Diagrama de bloques del servidor

El servidor no tiene ningún equipo extra para realizar su función solo necesita estar conectado a la red, aunque la Raspberry Pi 3 si dispone de un puerto de ethernet para conectarse a la red, durante la implementación del proyecto por razones de comodidad para realizar la instalación se optó por la conexión inalámbrica.

Siempre es recomendable usar la conexión cableada ya que nos da una capa extra de seguridad, pero en condiciones donde no es posible llegar mediante cable de red nos queda la opción inalámbrica la cual dispone de un ancho de banda menor, pero es más que suficiente para nuestros requerimientos.

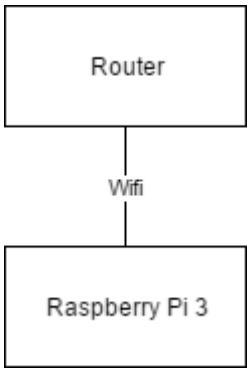


Figura 48 - Diagrama de bloques del servidor.

4.3. Diagrama de flujo del arduino

El Arduino es el encargado de recibir por el puerto serie las instrucciones de la Raspberry Pi e interactuar con el sensor biométrico.

El Arduino Nano funcionará como si de un DAQ se tratara, su trabajo es adquirir los datos del sensor biométrico y entregárselos a la Raspberry Pi Zero para que esta los pueda procesar.

El biométrico dispone de un API vía comunicación serie mediante el cual el arduino se conecta e interactúa con él, esta comunicación sea hace mediante un puerto serie por software.

El sensor biométrico se comunica vía puerto serie por lo que es necesario que el Arduino tenga un puerto serie para la comunicación con la Raspberry y otro para comunicarse con el sensor, esto se puede lograr creando un puerto serie por software para el biométrico y el puerto físico para la Raspberry Pi Zero W.

El Arduino está inactivo esperando instrucciones para ejecutar el código correspondiente como se puede apreciar en la *Figura 49*

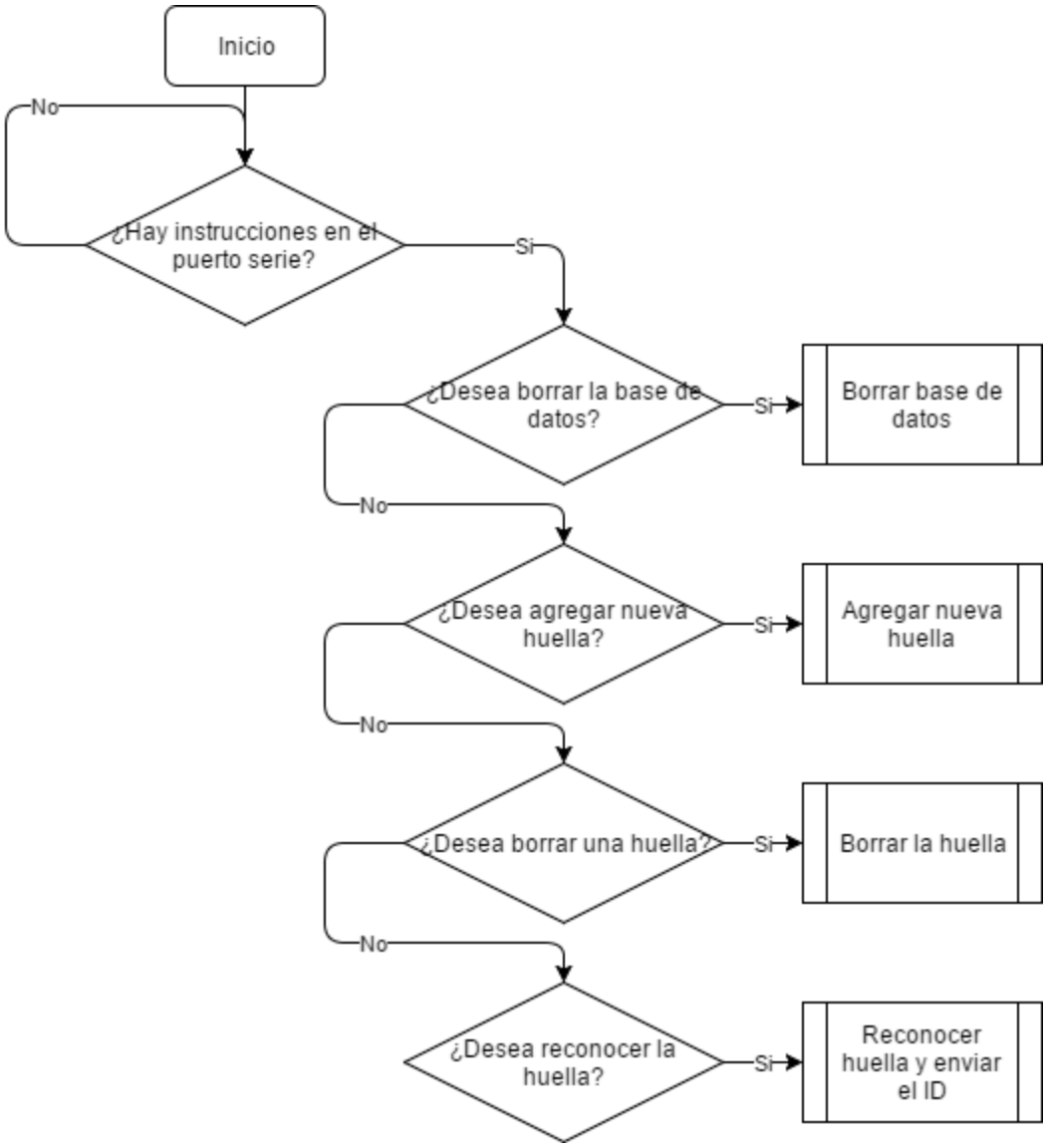


Figura 49 - Diagrama de flujo del arduino

Cada una de las acciones “Borrar base de datos”, “Agregar huella nueva”, “Borrar la huella” son acciones más grandes y complejas, por lo que requieren su propio diagrama de flujo para poder apreciar, entender y comprender su funcionamiento.

Estas son las funciones *Figura 50* son las funciones básicas del biométrico leer y borrar huellas de la base de datos local.

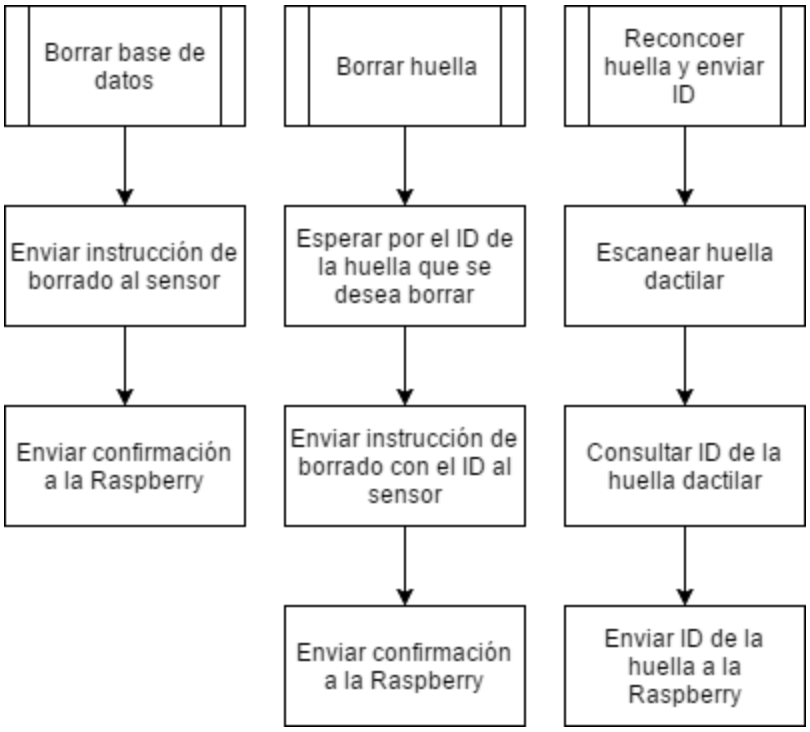


Figura 50 - Funciones de Borrado de la base de datos y reconocimeinto de huellas

La función de “agregar una nueva huella” es más compleja ya que requiere de una validación de la huella.

La huella se debe validar en varias ocasiones ya que es necesario asegurarnos que la digitalización se realizó de forma correcta.

El proceso de validación consiste en ubicar la huella dos veces para poder compararlas y verificar que una mala medición no nos deje dos huellas completamente diferentes, también se consulta la base de datos por si la huella ya fué previamente registrada, esto con el fin de evitar la duplicidad de la misma y que los datos conserven el orden y sentido que deben tener.

La función “agregar una nueva huella“que realiza el Arduino Nano se la detalla en la *Figura 51*.

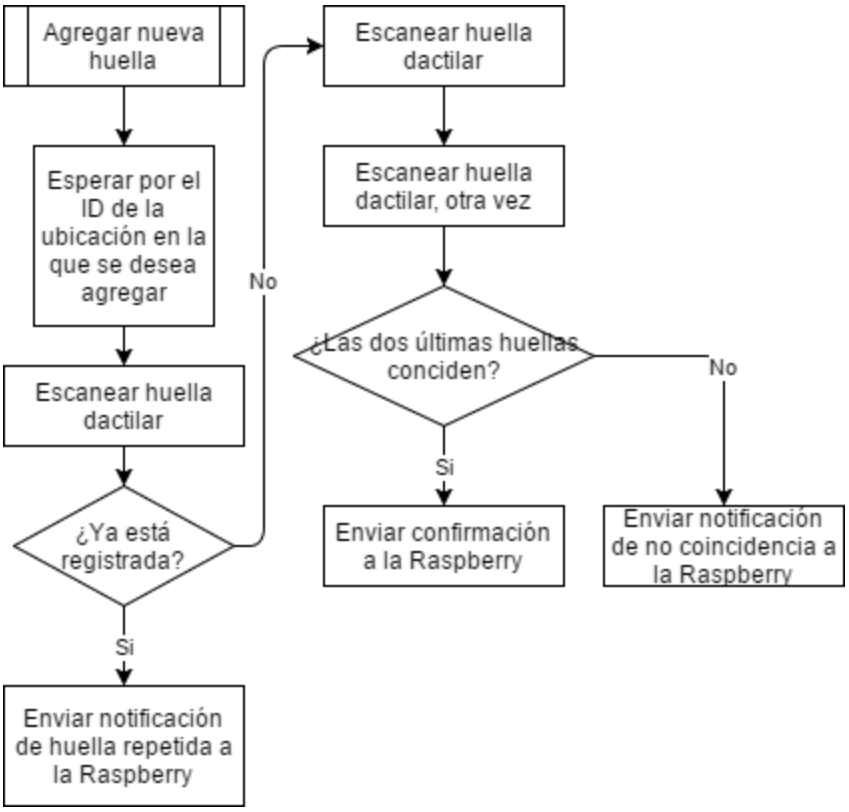


Figura 51 - Función de agregar huella nueva

4.4. Diagrama de flujo del biométrico

El biométrico tiene dos hilos en ejecución, es decir realiza dos tareas asíncronas las cuales son:

- Descarga asíncrona de empleados
- Página web principal

Cadauna de estas tareas es de suma importancia para el funcionamiento del biométrico ya que sustentan la interacción del usuario y la comunicación con el servidor.

La descarga asíncrona de empleados consiste en consultar en el servidor los empleados y actualizar la base de datos local con los cambios necesarios como se muestra en la *Figura 52*.

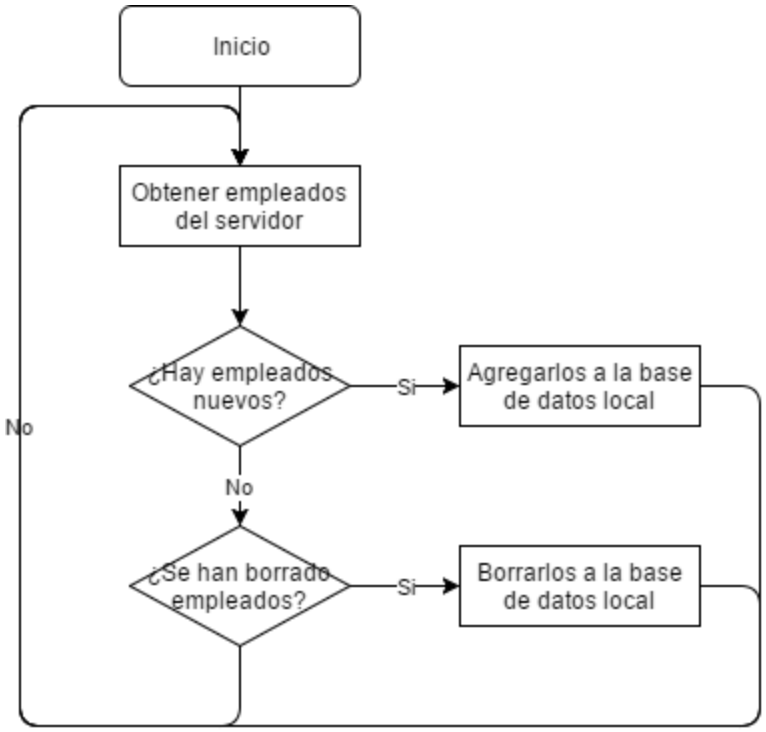


Figura 52 - Diagrama de flujo de la descarga asincrona de empleados

La interfaz web es dirigida a eventos, ya que cuando el usuario da clic en alguno de los botones se ejecutan las acciones, tal como se ve en la *Figura 53*, solo se realizan las acciones cuando los botones indicados reciben un clic.

Las acciones que provee la interfaz gráfica son la interacción con el biométrico tanto para agregar como para eliminar huellas nuevas, así como para reconocer la huella colocada.

La interfaz web es el primer punto de contacto del usuario con la aplicación por lo que debe tener el diseño lo más cuidado posible y poner a la mano todas las herramientas existentes en el sistema de forma que el usuario sepa que es lo que puede hacer y donde está la opción para realizarlo de forma clara y segura.

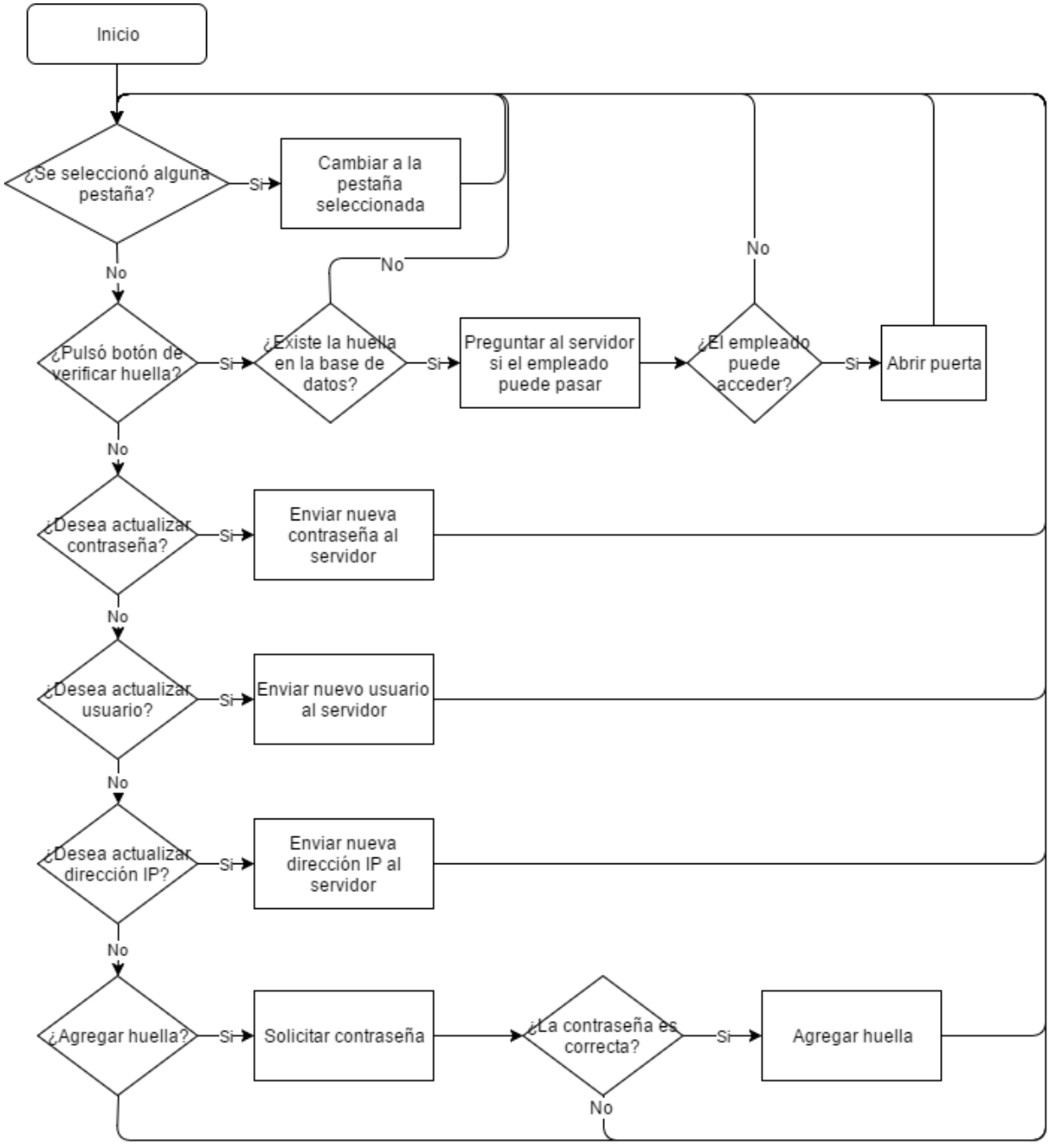


Figura 53 - Diagrama de flujo del biométrico

4.5. Implementación del proyecto

El proceso de implementación del presente proyecto de titulación se realizó en la "Lubricadora Blanquita".

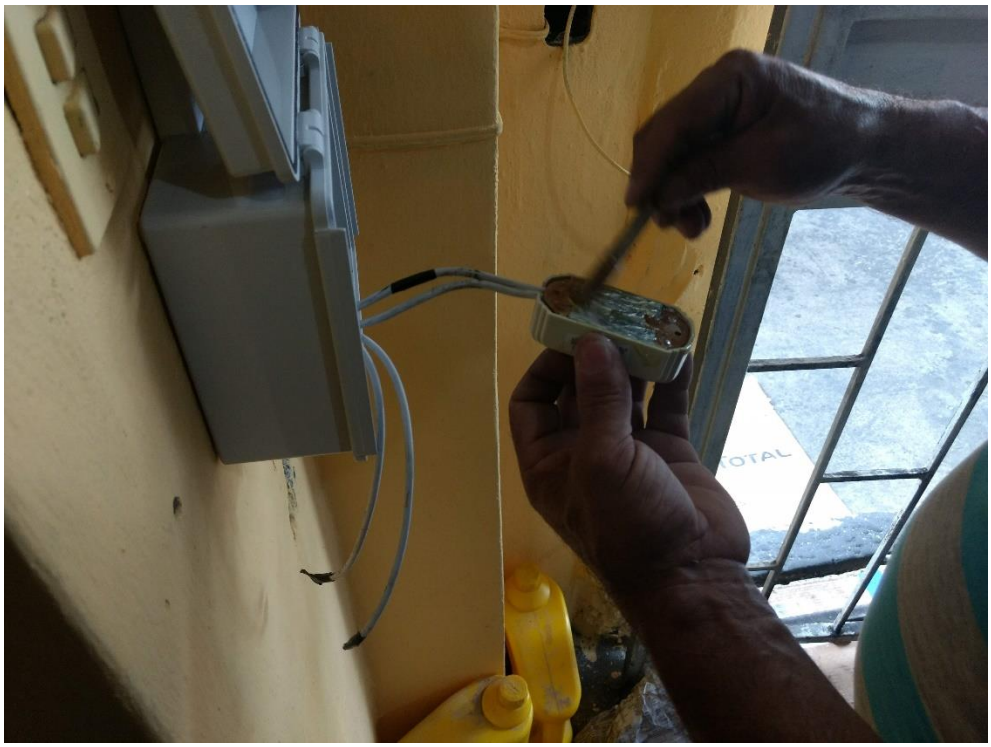


Figura 54 - Instalación de la base para el biométrico

Figura 55 - Chapa eléctrica para control de apertura

Figura 56 – Acabado final de la caja con su placa

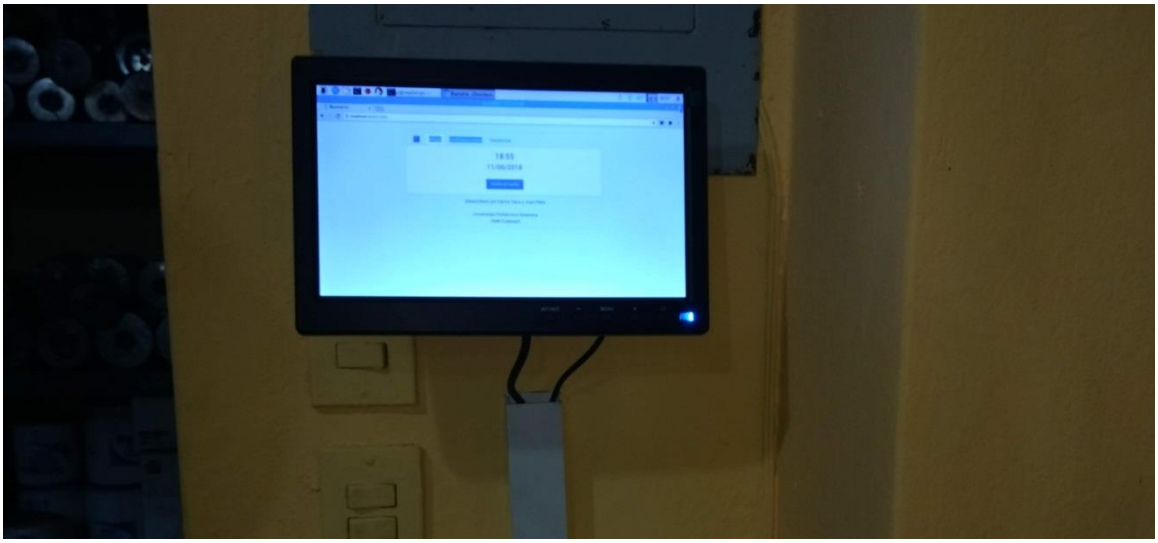


Figura 57 - Instalación de la pantalla para el biométrico

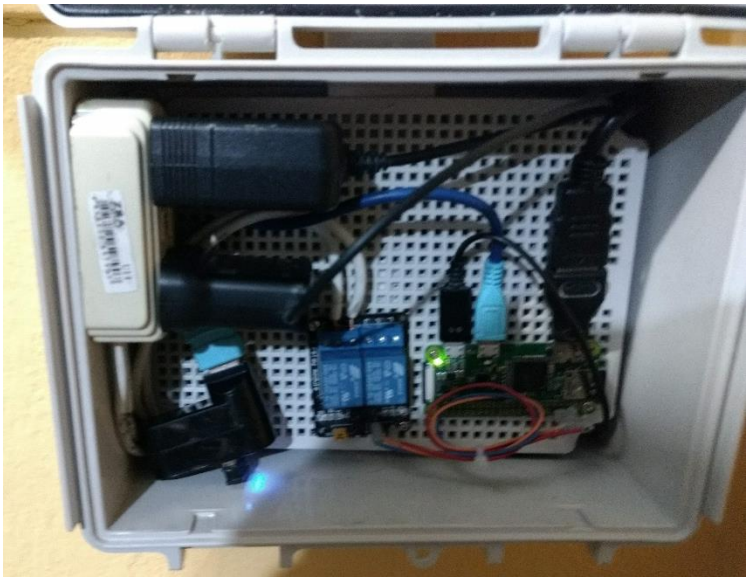


Figura 58 - Alimentación del biométrico

Figura 59 - Instalación del servidor

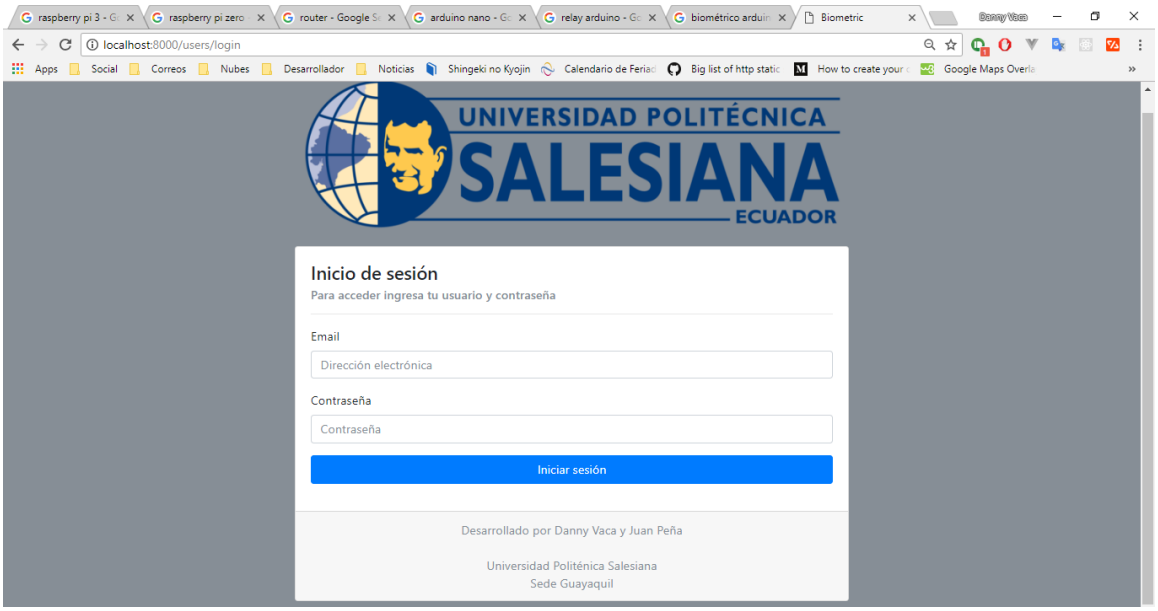


Figura 60 - Interfaz web del servidor (ventana de login), vista desde una PC

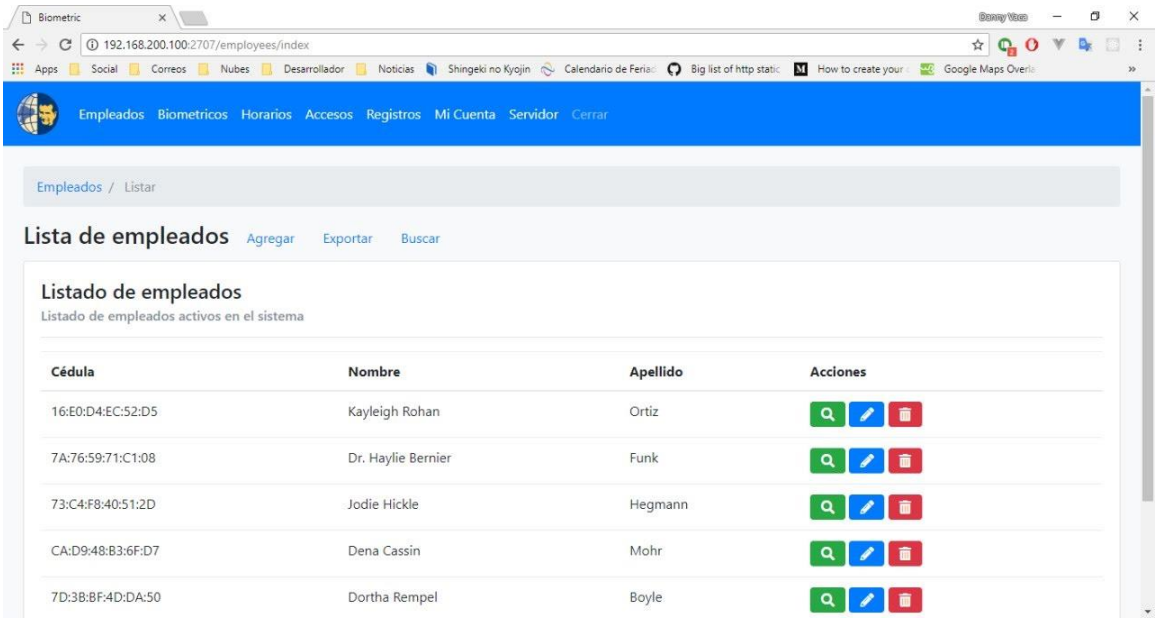


Figura 61 - Panel principal del servidor, vista desde una PC

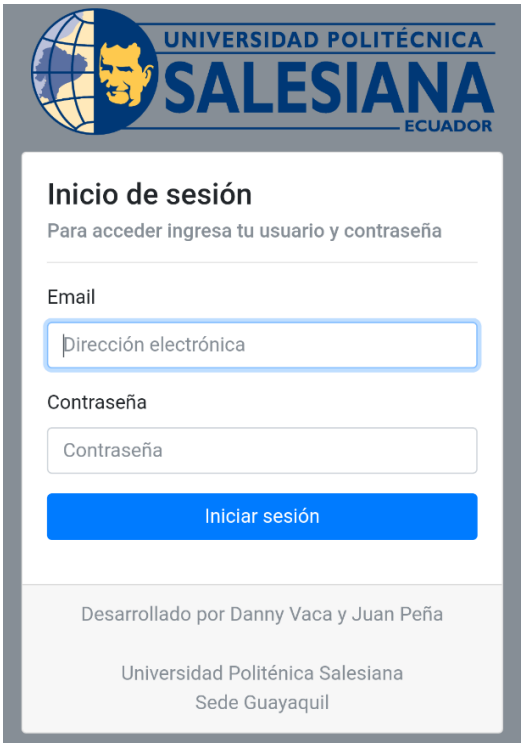


Figura 62 - Interfaz web del servidor (ventana de login), vista desde un smartphone

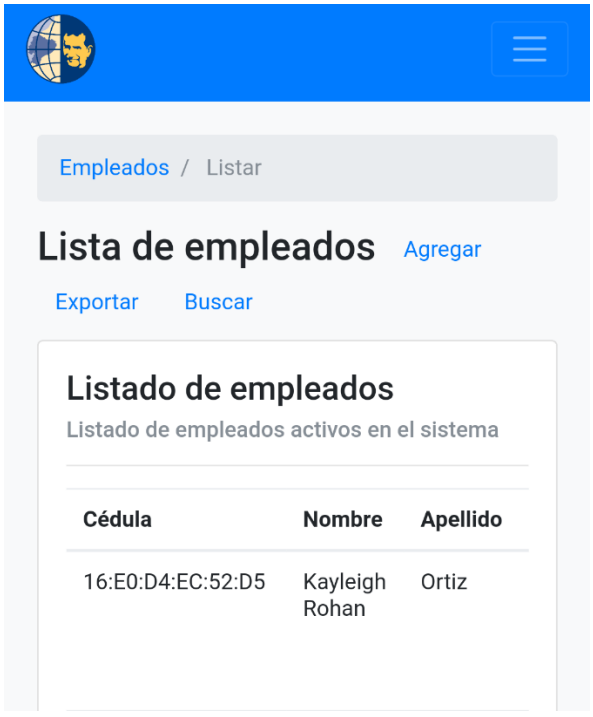


Figura 63 - Panel principal del servidor, vista desde un smartphone

CONCLUSIONES

Se pudo comprobar que la Raspberry Pi 3 es una minicomputadora con suficientes prestaciones para alojar aplicaciones web que usan bases de datos y permiten la conexión de varios clientes de forma estable y con un bajo consumo energético.

Se demostró que el uso de Arduino para acceso al hardware fue una gran elección debido a su gran cantidad de librerías y sus excelentes funcionalidades a bajo nivel, las cuales lo convierten en el complemento ideal para los proyectos desarrollados con Raspberry Pi.

La Raspberry Pi Zero W, aunque más limitada que la Raspberry Pi 3 posee suficientes capacidades para poder dar cabida a proyectos que necesiten una aplicación web, bases de datos y acceso al hardware.

El software apache que fue usado como servidor web posee un gran rendimiento y permite la múltiple conexión de varios hosts al servidor.

Las aplicaciones web ofrecen muchas ventas considerables frente a las aplicaciones nativas, pero una de las más destacables es que estas son ejecutadas por el navegador sin importar el sistema operativo que se esté usando, lo que las hace más portables.

El uso de un API basada en JSON hace que el consumo de datos de los dispositivos sea muy reducido y que el desarrollo de nuevas funcionalidades en la plataforma sea rápido y sencillo dotando a la plataforma de una alta escalabilidad.

RECOMENDACIONES

Se debe evitar ubicar el biométrico en un área húmeda o con estática ya que estos dos factores afectan el funcionamiento de los dispositivos electrónicos.

Para una mayor precisión del sensor biométrico es recomendable que se limpié el lente del sensor al dos veces al día.

Se recomienda desarrollar la buena costumbre de hacer una “backup” de la base de datos de forma semanal para evitar la pérdida de información en caso de alguna avería del servidor principal.

Se recomienda hacer una limpieza de las tablas de accesos y registros de la base de datos cada 6 meses con el fin de reducir el trabajo al servidor y mejorar su rendimiento.

No se debe manipular ningún elemento que conforma el sistema ya que se puede desconfigurar averiando el equipo.

No se deben exponer los equipos al agua ni al sol de forma directa ya que estos pueden degradar y dañar de forma permanente los equipos.

Se recomienda usar una red propia para los biométricos o usar una VLAN con el fin de asegurar la calidad del servicio en la red para los dispositivos.

Siempre que sea posible preferir dar acceso a la red a los dispositivos con el cable de red en lugar del wifi, ya que se dispone de una conexión más estable y segura.

PROYECTOS DE INVESTIGACIÓN VINCULADOS

- Diseño e implementación de un módulo didáctico de domótica por medio de láminas con la tarjeta Raspberry PI y el programa QT en la universidad politécnica salesiana sede Guayaquil, autores: Córdova Rivadeneira, Luis; Gordon Sánchez, Bryan Raúl; Tacurí Romero, Edgar Gustavo.
- Diseño e implementación de un sistema de seguridad de control local y remoto con dispositivos de vigilancia, desarrollo local y remoto con dispositivos de vigilancia, desarrollado en el software PYTHON, centralizándose en una tarjeta RASPBERRY PI, autores: Véliz Noboa, Bremnen; Campuzano Bulgarín, Andy; Cedeño Vélez, Vanessa.
- Análisis, diseño y desarrollo de una aplicación móvil para la visualización del consumo de energía eléctrica residencial utilizando las plataformas Android y Raspberry Pi, autores: Tufiño Cárdenas, Rodrigo Efraín; Licto Sánchez, Carlos Eduardo.
- Diseño e implementación de un parqueo inteligente utilizando arduino y basado en internet de las cosas (loT), autores: Cajo Diaz, Ricardo; Rosales Lindao, Leandro Nelson.
- Diseño de aplicaciones de sistemas embebidos basados en tecnología Raspberry Pi y Odroid U3, autores Martillo Ayala Daniel Humberto y Zambrano Mendoza Erly Loberty.
- Utilización de microcomputadora Raspberry Pi con capacidad de comunicación wifi, para la captura de imágenes mediante cámara y almacenamiento de información en base de datos externa, autores: Marcos Naula, Robert Llanos, Carlos Valdiviezo.

REFERENCIAS BIBLIOGRÁFICAS

Andrew W. Senior, J. H. (2004). *Guide to Biometrics.*

Anil K. Jain, A. A. (2011). *Introduction to Biometrics.*

Arduino. (2008). *Arduino Nano.* Retrieved from arduino.cc: https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf

B. Navya Rupa, G. K.-h. (2015). Test Report Generation Using JSON. *International Journal of Software Engineering and Its Applications*.

Banzi, M. (2008). *Getting Started with Arduino.*

Batini, C. (n.d.). *Conceptual Database Design, An Entity-relationship Approach.* 1992: Benjamin/Cummings.

Ben Everard, E. U. (2014). *Learning Computer Architecture with Raspberry Pi.*

Blum, C. B. (2013). *Sams Teach Yourself Python Programming for Raspberry Pi in 24 Hours.*

Boulgouris, N. V. (2009). *Biometrics: Theory, Methods, and Applications.*

Boxall, J. (2013). *Arduino Workshop: A Hands-On Introduction with 65 Projects.*

Brock Craft, J. E. (2015). *Raspberry Pi Projects For Dummies.*

C.Calderón, M. C. (2014). Desarrollo de una Aplicación Cliente/Servidor para un Wall View en base a la. *REVISTA EPN*, 7. Obtenido de https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/155/pdf

Ceballos, E. L. (2013). Diseño web adaptativo o responsivo. *Revista Digital Universitaria*.

Elio, E. M. (2015). *Microcontrolador Arduino.* Universidad Cristobal Colón.

Fernando Bizzarro, A. H. (2016). *The V-Dem Party Institutionalization Index: a new global indicator (1900-2015).*

Guillermo Diez-Andino Sancho, R. M. (2003). Desarrollo de un servidor HTTP para dispositivos móviles en J2ME. *Departamento. Ingeniería Telemática - Universidad Carlos III de Madrid*. Retrieved from http://www.it.uc3m.es/celeste/papers/ServidorHTTP.pdf

Javier, L. S. (2001). Base de Datos Distribuidas Estudio de Actualización de Réplicas. *Facultad de Informática – UNLP*. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/3872/Documento_completo__.pdf-PDFA1b.pdf?sequence=1

Jayesh Umre, K. B. (2014). Comparative performance analysis of MySQL and SQLite relational database management systems in Windows 10 environments. *International Journal of Latest Trends in Engineering and Technology*.

Jimy Alexander Cortés Osorio, F. A. (2010). *Sistemas de seguridad basados en biometría.*

Kilicdagi, A. (2014). *Laravel Design Patterns and Best Practices.*

Margolis, M. (2011). *Arduino Cookbook.*

María F. Maldonado, A. C. (2008). Implementación de un sistema Web para manejo de datos meteorológicos del Laboratorio de Energías Alternativas y Eficiencia Energética de la Escuela Laboratorio de Energías Alternativas y Eficiencia Energética de la Escuela Politécnica Nacional. Retrieved from http://bibdigital.epn.edu.ec/bitstream/15000/4923/1/PAPER%20Implementaci%C3%B3n%20de%20un%20sistema%20Web%20para%20manejo%20de%20datos%20meteorol%C3%B3gicos%20del%20Laboratorio%20de%20Energ%C3%ADas%20Alterna.pdf

Mishra, A. (2014). Critical Comparison Of PHP And ASP.NET For Web Development. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 3,*. Retrieved from http://www.ijstr.org/final-print/july2014/Critical-Comparison-Of-Php-And-Aspnet-For-Web-Development.pdf

Molloy, D. (2016). *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux.* Retrieved from https://www.element14.com/community/community/raspberry-pi?src=raspberrypi

Monk, S. (2012). *Programming the Raspberry Pi: Getting Started with Python.*

Mulloy, B. (2012). *Web API Design, Crafting Interfaces that Developers Love.* apigee.

Nguyen, Q. H. (2015). Building a web application with LARAVEL 5. *Oulu University of Applied Sciences*.

Oracle. (2011). *PHP Scalability and High Availability Database Resident Connection Pooling and Fast Application Notification.*

Peck, A. (2017). *Jumpstarting the Raspberry Pi Zero W.*

Philbin, C. A. (2013). *Adventures in Raspberry Pi.*

Purdum, J. J. (2012). *Beginning C for Arduino: Learn C Programming for the Arduino.*

Ross, A. A. (2006). *Handbook of Multibiometrics.*

Scott Trent, M. T. (2008). *Performance Comparison of PHP and JSP as Server-Side Scripting Languages.* Retrieved from https://www.researchgate.net/publication/225161349_Performance_Comparison_of_PHP_and_JSP_as_Server-Side_Scripting_Languages

Spectator, C. (2016). *Comparativa de rendimiento de la red entre distintos proveedores Cloud desde diferentes localizaciones geográficas.*

ANEXOS

ACTA DE ENTREGA-RECEPCIÓN

En la ciudad de Guayaquil a los 8 días del mes de febrero del año 2018 el señor **Danny Ramiro Vaca Orrala** y el señor **Juan Andrés Peña Coronel** ambos estudiantes de la **Universidad Politécnica Salesiana** procedieron a la instalación de una plataforma de control de acceso biométrico y a la capacitación de las personas responsables de "**Lubricadora Blanquita**".

Yo **Blanquita Isabel Coronel Garcés** declaro que se realizó en mi local "**Lubricadora Blanquita**" la instalación de la plataforma de control de acceso biométrico.

Guayaquil, febrero del 2018

_____ _____

(f) Danny Ramiro Vaca Orrala
C.I: 0925010092

(f) Juan Andrés Peña Coronel
C.I: 0929104628

_____

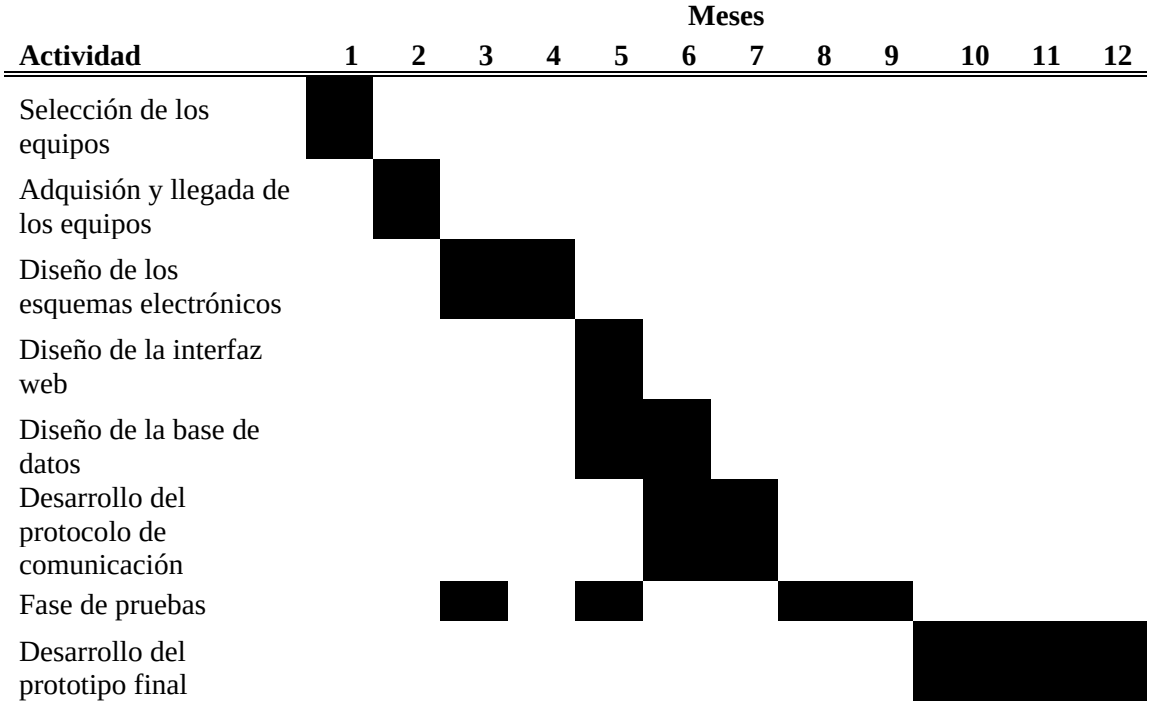
(f) Blanquita Isabel Coronel Garcés
C.I: 0907707723

PRESUPUESTO

Tabla 9 - Presupuesto para implemetación

Descripción	Cantidad	Precio	Subtotal
Fuente	2	$ 10.00	$ 20.00
Tarjeta SD	2	$ 10.00	$ 20.00
Raspberry Pi 3 Case	1	$ 20.00	$ 20.00
Raspberry Pi 3	1	$ 90.00	$ 90.00
Raspberry Pi Zero case	1	$ 15.00	$ 15.00
Raspberry Pi Zero	1	$ 50.00	$ 50.00
Hub USB	1	$ 15.00	$ 15.00
Plantilla web	1	$150.00	$ 150.00
Mini teclado inalámbrico	1	$ 15.00	$ 15.00
Shield de relay	1	$ 15.00	$ 15.00
Canaleta	1	$ 10.00	$ 10.00
Cable de red	2	$ 10.00	$ 20.00
Arduino Nano	1	$ 13.00	$ 13.00
Lector biométrico (integrado)	1	$ 40.00	$ 40.00
Caja contenedora del prototipo	1	$ 70.00	$ 70.00
Pantalla HDMI con base para pared	1	$120.00	$ 120.00
Router	1	$ 35.00	$ 35.00
Mini teclado bluetooth	1	$ 15.00	$ 15.00
Gastos varios	1	$200.00	$ 200.00
Cable HDMI	1	$ 12.00	$ 12.00
Curso de diseño web con bootstrap	1	$200.00	$ 200.00
Curso de desarrollo en laravel	1	$250.00	$ 250.00
Total			**$1,395.00**

CRONOGRAMA DE ACTIVIDADES



CÓDIGO DEL CLIENTE (RASPBERRY PI ZERO)

COMUNICACIÓN SERIAL

Una de las ventajas de usar Python en el backend de la Raspberry Pi Zero es la facilidad de acceso al hardware, el código fuente para realizar la comunicación serial es:

```
import serial


class Serial:
    def __init__(self, port, baud_rate=9600):
        self.port = port
        self.baud_rate = baud_rate
        self.port = serial.Serial(port=port, baudrate=baud_rate)


    def read_decoded(self):
        character = self.port.read()
        print(character)
        if character == '':
            return ''
        else:
            return character.decode('utf-8')


    def read(self, initializer='', finisher=''):
        if initializer in '':
            return self.read_decoded()


        character = self.read_decoded()
        while character not in initializer:
            character = self.read_decoded()
```

```
        data = initializer

        if finisher in '':
            return self.read_decoded()

        while character not in finisher:
            character = self.read_decoded()
            data = data + character
        return data

    def write(self,data):
        self.port.write(str(data).encode('utf-8'))

    def close(self):
        print('close')
        self.port.close()

if __name__=='__main__':
    arduino = Serial(port='com3')
    data = arduino.read(initializer='{', finisher='}')
    print(data)
    data = arduino.read(initializer='{', finisher='}')
    print(data)
```

Una vez exista una comunicación serie, sobre esta se realiza la comunicación con el Arduino para que este a su vez se comunique con el biométrico.

EL script que gestiona la comunicación con el biométrico sobre el puerto serie es:

```
from serial_tools import Serial
from json import loads as json_decoder
from serial import SerialException


class FingerprintNotFoundException(Exception):
    pass


class Fingerprint:
    levels = {
        'MAX_LEVEL':'99'
    }

    actions = {
        'CLEAR_DATABASE': '0',
        'ADD_NEW_FINGER': '1',
        'REMOVE_PLACED_FINGER': '2',
        'REMOVE_FINGER_BY_ID': '3',
        'GET_PLACED_FINGER_ID': '4',
        'HAS_FINGER': '6',
    }

    responses = {
        '0x01':'Verificando sensor',
```

```
    '0x02':'Sensor listo',
    '0x03':'Sensor no encontrado',
    '0x04':'Obteniendo imagen',
    '0x05':'Imagen capturada',
    '0x06':'Error de comunicación',
    '0x07':'Error de image',
    '0x08':'Error desconocido',
    '0x09':'Tomando imagen',
    '0x10':'Imagen convertida',
    '0x11':'Imagen en malas condiciones',
    '0x12':'Error de función',
    '0x13':'Imagen inválida',
    '0x14':'Por favor ubique el dedo',
    '0x15':'Por favor quite el dedo',
    '0x16':'Creando modelo',
    '0x17':'Huella encontrada',
    '0x18':'Huellas no coinciden',
    '0x19':'Modelo agregado',
    '0x20':'Huella agregada',
    '0x21':'Ubicación inválida',
    '0x22':'Error escribiendo en el sensor',
    '0x23':'Error buscando',
    '0x24':'Huella encontrada',
    '0x25':'Huella no encontrada',
    '0x26':'Eliminar modelo',
    '0x27':'Huella eliminada',

    #clear database
```

```
    '0x28':'Borrando la base de datos',
    '0x29':'Base de datos borrada',

    #add new finger
    '0x30':'Verificando si ya está registrada',
    '0x31': 'Huella ya registrada',
    '0x32': 'Error digitalizando la huella',
    '0x33': 'Huellas inválidas',
    '0x34': 'Error guandando los modelos',
    '0x35': 'Huella registrada',
    '0x36': 'Registrando nueva huella',

    #remove finger
    '0x37':'Eliminando huella',
    '0x38':'Huella no eliminada',

    #searcg finger
    '0x39':'Huella desconocida',
    '0x40': 'Coincidencia encontrada',

    #check finger
    '0x41':'Huella detectada',
    '0x42': 'Huella no detectada'
}

ports = (
    '/dev/ttyACM0',
    '/dev/ttyACM1',
```

```
        '/dev/ttyUSB0',
        '/dev/ttyUSB1',
        'com1',
        'com2',
        'com3',
        'com4',
        'com5'
    )
    def instance():
        for port in Fingerprint.ports:
            try:
                fingerprint=Fingerprint(port=port)
                fingerprint.wait_for_sensor()
                return fingerprint
            except SerialException as error:
                pass
            except OSError as error:
                pass
        raise SerialException('Serial device not found')

    def remove_finger_at(position):
        try:
            fingerprint=Fingerprint.instance()
            fingerprint.remove_finger(position)
            fingerprint.close()
            return None
        except SerialException as error:
            return ('Sensor no encontrado','No se pudo establecer conexión con el sensor')
```

```
        except JSONDecodeError as error:
            return ('Respuesta inválida','El sensor envió una respuesta que no pudo ser procesada')
        except KeyError as error:
            return ('Respuesta inválida','La respuesta recibida por el sensor no tiene un formato correcto')


    def print_by_level(self,json):
        if json['level'] in self.levels['MAX_LEVEL']:
            print(self.responses[json['code']])
        else:
            print('\t', self.responses[json['code']])


    def send_position(self, position):
        if position < 10:
            print('position', '00'+str(position))
            self.arduino.write('00'+str(position))
        elif 100 > position > 10:
                print('position', '0' + str(position))
                self.arduino.write('0'+str(position))
        elif 1000 > position > 100:
            print('position', str(position))
            self.arduino.write(str(position))


    def __init__(self,port='COM3', baud_rate=9600):
        self.arduino = Serial(port=port, baud_rate=baud_rate)


    def wait_for_sensor(self):
        while True:
```

```
        data = self.arduino.read(initializer='{', finisher='}')
        if data not in '':
            json = json_decoder(data)
            self.print_by_level(json)
            if json['code'] in '0x01':
                while True:
                    data = self.arduino.read(initializer='{', finisher='}')
                    if data not in '':
                        json = json_decoder(data)
                        self.print_by_level(json)
                        if json['code'] in '0x02':
                            return


    def clear_database(self):
        self.arduino.write(self.actions['CLEAR_DATABASE'])
        while True:
            data = self.arduino.read(initializer='{', finisher='}')
            if data not in '':
                json = json_decoder(data)
                self.print_by_level(json)
                if json['code'] in '0x29':
                    return


    def add_new_finger(self, position):
        self.arduino.write(self.actions['ADD_NEW_FINGER'])
        self.send_position(position)
        while True:
            data = self.arduino.read(initializer='{', finisher='}')
```

```
            if data not in '':
                json = json_decoder(data)
                self.print_by_level(json)
                if json['code'] in '0x31':
                    return 0
                elif json['code'] in '0x35':
                    return position
                if json['code'] in ['0x32', '0x33', '0x34']:
                    return -1


    def remove_finger(self, position):
        self.arduino.write(self.actions['REMOVE_FINGER_BY_ID'])
        self.send_position(position)
        while True:
            data = self.arduino.read(initializer='{', finisher='}')
            if data not in '':
                json = json_decoder(data)
                self.print_by_level(json)
                if json['code'] in ['0x38','0x27']:
                    return


    def get_id(self):
        self.arduino.write(self.actions['GET_PLACED_FINGER_ID'])
        while True:
            data = self.arduino.read(initializer='{', finisher='}')
            if data not in '':
                json = json_decoder(data)
                self.print_by_level(json)
```

```
            if json['code'] in '0x39':
                return -1
            if json['code'] in '0x40':
                return json['id']


    def has_finger_placed(self):
        self.arduino.write(self.actions['HAS_FINGER'])
        while True:
            data = self.arduino.read(initializer='{', finisher='}')
            if data not in '':
                json = json_decoder(data)
                self.print_by_level(json)
                if json['code'] in '0x41':
                    return True
                if json['code'] in '0x42':
                    return False


    def close(self):
        self.arduino.close()


if __name__=='__main__':
    fingerprint=Fingerprint(port='com3')
    fingerprint.wait_for_sensor()
```

CONSUMIR WEB SERVICES

Desde el cliente ese necesario consumir los web services del servidor para poder tener la información de los empleados, estos web services son consumidos vía peticiones http, las cuales deben ser realizadas desde Python, el script que gestiona las peticiones web es:

```
from urllib import request
from json import loads as json_decoder
from database_tools import Setting


class InvalidBiometricPasswordException(Exception):
    pass


class InvalidBiometricNameException(Exception):
    pass


class HttpApi:
    @staticmethod
    def can_pass(employee):
        base_url = Setting.by_name(name='ip_address').value
        biometric_name = Setting.by_name(name='biometric_name').value
        full_url = base_url + '/api/biometrics/' + biometric_name + '/employees/' +
employee.code + '/check'
        print(full_url)


        response = request.urlopen(full_url)
        response_data = response.read().decode("utf-8")
        response_json = json_decoder(response_data)
```

```
        print(response_json)
        return response_json['code']=='0x04'


    def __init__(self):
        self.server_employees = None


    def get_employees(self):
        base_url = Setting.by_name(name='ip_address').value
        biometric_name = Setting.by_name(name='biometric_name').value
        biometric_password = Setting.by_name(name='biometric_password').value


        full_url = base_url + '/api/biometrics/' + biometric_name +
'/employees/index?password='+biometric_password
        print(full_url)
        response = request.urlopen(full_url)
        response_data = response.read().decode("utf-8")
        response_json = json_decoder(response_data)
        if 'code' in response_json:
            if response_json['code'] == '0x01':
                raise InvalidBiometricNameException()
            if response_json['code'] == '0x06':
                raise InvalidBiometricPasswordException()


        self.server_employees = response_json
        return self.server_employees


    def exists_on_server(self, local_employee):
        local_employee_is_on_server = False
        for server_employee in self.server_employees:
```

```
            if server_employee['code'] == local_employee.code:
                local_employee_is_on_server = True
        return local_employee_is_on_server
```

CONEXIÓN CON LA BASE DE DATOS

Es importante tener una base de datos local de los empleados para poder registrar las huellas en el sensor biométrico, en el cliente usamos SQLite ya que es una alternativa más ligera, el script que gestiona la conexión con la base de datos es:

```
import sqlite3
from json import dumps as json_encoder


class Database:
    def __init__(self,database_path):
        self.connection = sqlite3.connect(database_path)
        self.cursor = self.connection.cursor()


    def query(self, query):
        response = None
        if isinstance(query,list) or isinstance(query,tuple):
            for current_query in query:
                response = self.cursor.execute(current_query)
        else:
            response = self.cursor.execute(query)
        self.connection.commit()
```

```
        return response


database_path = 'db.sqlite3'


class Notification:


    @staticmethod

    def by_code(code):

        database = Database(database_path)

        notifications = []

        for notification in database.query('select code,message,rowid as id from
notifications where code='+code):

            notifications.append(Notification(code=notification[0], message=notification[1],
id=notification[2]))

        coincidences=[]

        for notification in Notification.all():

            if notification.code==code:

                coincidences.append(notification)

        return coincidences


    @staticmethod

    def all():

        database = Database(database_path)

        notifications = []

        for notification in database.query('select code,message,rowid as id from
notifications'):

            notifications.append(Notification(code=notification[0], message=notification[1],
id=notification[2]))

        return notifications
```

```
    def __init__(self,code,message,id=None):
        self.code=code
        self.message=message
        self.id=id
        self.database = Database(database_path)
        if self.id is None:
            self.database.query(
                'insert into notifications (code,message) values ('' + self.code + ''," +
self.message + '')')
            for id in self.database.query('select last_insert_rowid()'):
                self.id = id[0]


    def delete(self):
        self.database.query('delete from notifications where rowid='+str(self.id))


    def to_dict(self):
        return {'code':self.code,'message':self.message}


    def __str__(self):
        return str({'code':self.code, 'message': self.message})


class Action:
    @staticmethod
    def by_employee(employee_id):
        database = Database(database_path)
        actions = []
        for action in database.query('select employee_id,action,rowid as id from actions
where employee_id='+str(employee_id)):
            actions.append(Action(employee_id=action[0], action=action[1], id=action[2]))
```

```
        return actions


    @staticmethod
    def all():
        database = Database(database_path)
        actions = []
        for action in database.query('select employee_id,action,rowid as id from actions'):
            actions.append(Action(employee_id=action[0], action=action[1], id=action[2]))
        return actions


    def __init__(self, employee_id, action, id=None):
        self.employee_id = employee_id
        self.action = action
        self.id = id
        self.database = Database(database_path)
        if self.id is None:
            self.database.query(
                'insert into actions (employee_id,action) values (''' + str(self.employee_id) + ''','''
+ self.action + ''')')
            for id in self.database.query('select last_insert_rowid()'):
                self.id = id[0]


    def employee(self):
        return Employee.find(self.employee_id)


    def delete(self):
        self.database.query('delete from actions where rowid='+str(self.id))


    def __str__(self):
```

```
        return str({'employee_id': self.employee_id, 'action': self.action})


    def to_dict(self):
        return {'action': self.action, 'employee':self.employee().to_dict()}


class Setting:
    @staticmethod
    def by_name(name):
        database = Database(database_path)
        for setting in database.query('select name,value,rowid as id from settings where name="' + name + '"'):
            return Setting(name=setting[0], value=setting[1], id=setting[2])


    @staticmethod
    def all():
        database = Database(database_path)
        settings = []
        for setting in database.query('select name,value,rowid as id from settings'):
            settings.append(Setting(name=setting[0], value=setting[1], id=setting[2]))
        return settings


    def find(id):
        database = Database(database_path)
        for setting in database.query('select name,value,rowid as id from settings where id=' + str(id)):
            return Setting(name=setting[0], value=setting[1], id=setting[2])


    def __init__(self, name, value, id=None):
        self.id = id
```

```
        self.name = name
        self.value = value
        self.database = Database(database_path)
        if self.id is None:
            self.database.query('insert into settings (name,value) values ("' + self.name + '","' + self.value + '")')
            for id in self.database.query('select last_insert_rowid()'):
                self.id = id[0]

    def save(self):
        self.database.query('update settings set value="' + self.value + '" where rowid="' + str(self.id) + '"')

    def delete(self):
        self.database.query('delete from settings where rowid='+str(self.id))

    def __str__(self):
        return str({'name': self.name, 'value': self.value})

class Employee:
    @staticmethod
    def exists(code):
        return not (Employee.by_code(code) is None)

    @staticmethod
    def by_code(code):
        database = Database(database_path)
        for employee in database.query('select code,name,lastname,password,rowid as id from employees where code=' + str(code)):
            return Employee(code=employee[0], name=employee[1], lastname=employee[2],
```

```
password=employee[3], id=employee[4])


    @staticmethod

    def all():

        database = Database(database_path)

        employees = []

        for employee in database.query('select code,name,lastname,password,rowid as id
from employees'):

            employees.append(Employee(code=employee[0], name=employee[1],
lastname=employee[2], password=employee[3], id=employee[4]))

        return employees


    def find(id):

        database = Database(database_path)

        for employee in database.query('select code,name,lastname,password,rowid as id
from employees where id='+str(id)):

            return Employee(code=employee[0], name=employee[1], lastname=employee[2],
password=employee[3], id=employee[4])


    def __init__(self, code, name, lastname, password, id=None):

        self.code = code

        self.name = name

        self.lastname = lastname

        self.password = password

        self.id = id

        self.database = Database(database_path)

        if self.id is None:

            self.database.query('insert into employees (code,name,lastname,password)
values ('"'+self.code+'"','"'+self.name+'"','"'+self.lastname+'"','" + self.password + '")')

            for id in self.database.query('select last_insert_rowid()'):
```

```
            self.id = id[0]


    def save(self):
        self.database.query('update employees set code="' + self.code + '",name="' +
self.name + '",lastname="' + self.lastname + '",password="' + self.password + '" where
rowid="' + str(self.id) + '"')


    def delete(self):
        self.database.query('delete from employees where rowid='+str(self.id))


    def __str__(self):
        return str({'name': self.name, 'lastname': self.lastname, 'code': self.code, 'id':
self.id})


    def to_dict(self):
        return {'name': self.name, 'lastname': self.lastname, 'code': self.code, 'id': self.id,
'password':self.password}


def create_tables():
    database = Database(database_path)
    delete_tables = (
        'drop table if exists employees',
        'drop table if exists actions',
        'drop table if exists settings',
        'drop table if exists notifications',
    )
    database.query(delete_tables)
    create_tables = (
        'create table employees (code text,name text,lastname text,password text)',
        'create table actions (employee_id unsigned integer,action text)',
```

```
        'create table settings (name text,value text)',
        'create table notifications (code,message)',
    )
    database.query(create_tables)

if __name__ == '__main__':
    create_tables()
    #Setting(name='ip_address', value='http://27.7.93.2')
    Setting(name='ip_address', value='http://127.0.0.1')
    Setting(name='biometric_name', value='main')
    Setting(name='biometric_password', value='secret')
```

En el servidor hay un script que se ejecuta de forma recurrente para actualiza la información del servidor, actualizar la información de la base de datos, borrar las huellas del biométrico cunado un empleado sea eliminado del servidor, el script que se ejecuta de forma recurrente es:

```
#!/usr/bin/env python
import os
import sys

from threading import Thread

from database_tools import Setting
from database_tools import Action
from database_tools import Employee
```

```
from database_tools import Notification


from urllib.error import HTTPError

from urllib.error import URLError

from http_api import InvalidBiometricNameException

from http_api import InvalidBiometricPasswordException

from http_api import HttpApi


from logger_tools import logger

from logger_tools import my_print


from serial import SerialException

from json.decoder import JSONDecodeError


from time import sleep


from fingerprint_tools import Fingerprint


if __name__ == "__main__":


    def async_server_connection():

        while True:

            try:

                server_employees = app.get_employees()

                for server_employee in server_employees:

                    if not Employee.exists(code=server_employee['code']):

                        my_print('Nuevo empleado "{lastname}, {name}" agregado en el
servidor'.format(name=server_employee['name'],
lastname=server_employee['lastname']))
```

```
                    logger.debug('New employee "{lastname}, {name}" add in
server'.format(name=server_employee['name'], lastname=server_employee['lastname']))



                    my_print('Agregar empleado a base de datos local')

                    logger.debug('\tAdd employee on local database - %s', server_employee)

                    new_employee = Employee(code=server_employee['code'],
name=server_employee['name'], lastname=server_employee['lastname'],
password=server_employee['password_clean'])



                    my_print('Agregar acción pendiente para el empleado')

                    logger.debug('\tAdd pending action "add" to employee')

                    Action(employee_id=new_employee.id, action='add')

                else:

                    employee = Employee.by_code(server_employee['code'])



                    name_have_changed=employee.name != server_employee['name']

                    lastname_have_changed=employee.lastname !=
server_employee['lastname']

                    password_have_changed=employee.password !=
server_employee['password_clean']

                    if name_have_changed or lastname_have_changed or
password_have_changed:

                        my_print('Actualiza cambios en el empleado
{employee}'.format(employee=employee))

                        logger.debug('Must update employee
{employee}'.format(employee=employee))



                        employee.name = server_employee['name']

                        employee.lastname = server_employee['lastname']

                        employee.password= server_employee['password_clean']

                        employee.save()
```

```
            for local_employee in Employee.all():
                if not app.exists_on_server(local_employee=local_employee):
                    my_print('Empleado eliminado del servidor "{lastname},
{name}"'.format(name=local_employee.name, lastname=local_employee.lastname))
                    logger.debug('Employee deleted from server')

                    my_print('Eliminar empleado de la base de datos local')
                    logger.debug('\tDelete from local database - %s', local_employee)
                    local_employee.delete()

                    my_print('Eliminar huella del empleado')
                    logger.debug('\tDelete from biometric database - %s', local_employee)
                    response=Fingerprint.remove_finger_at(int(local_employee.code))
                    if response is not None:
                        return send_error(response[0],response[1])

                    my_print('Eliminar acciones pendientes del empleado')
                    logger.debug('Employee has pending actions, delete each one')
                    actions = Action.by_employee(local_employee.id)
                    if len(actions) > 0:
                        for action in actions:
                            my_print('Eliminar acción pendiente - %s', action)
                            logger.debug('Delete pending action - %s', action)
                            action.delete()

            for notification in Notification.all():
                notification.delete()
        except InvalidBiometricNameException as error:
            biometric_name = Setting.by_name(name='biometric_name').value
```

```
            my_print('No existe un biométrico con el nombre
"{name}"'.format(name=biometric_name))

            logger.error('InvalidBiometricNameException: No biometric named - %s',
biometric_name, exc_info=True)

            if len(Notification.by_code('0x01'))==0:

                Notification(code='0x01', message='No biometric named
{name}'.format(name=biometric_name))

        except InvalidBiometricPasswordException as error:

            my_print('Contraseña del biométrico incorrecta')

            logger.error('InvalidBiometricPasswordException: Invalid biometric password',
exc_info=True)

            if len(Notification.by_code('0x02'))==0:

                Notification(code='0x02', message='Invalid password')

        except HTTPError as error:

            my_print('Error en el servidor')

            logger.error('HTTPError: Error on server - %s', error, exc_info=True)

            if len(Notification.by_code('0x03'))==0:

                Notification(code='0x03', message='HTTP Error')

        except KeyError as error:

            my_print('Respuesta con formato erróneo')

            logger.error('KeyError - Error decoding json - %s', error, exc_info=True)

            if len(Notification.by_code('0x04'))==0:

                Notification(code='0x04', message='Error decoding response from server')

        except URLError as error:

            my_print('Dirección IP inválida')

            logger.error('URLError - Invalid server IP address - %s', error, exc_info=True)

            if len(Notification.by_code('0x05'))==0:

                Notification(code='0x05', message='Invalid server address')

        except ValueError as error:

            my_print('Formato de la url inválido')
```

```
            logger.error('ValueError - Invalid value - %s', error, exc_info=True)
            if len(Notification.by_code('0x06'))==0:
                Notification(code='0x06', message='Invalid URL format')
        except ConnectionResetError as error:
            my_print('Conexión rechazada')
            logger.error('ConnectionResetError - Conexión rechazada - %s', error,
exc_info=True)
            if len(Notification.by_code('0x07'))==0:
                Notification(code='0x07', message='Connection refused')
        sleep(5)


my_print('Start server connection')
logger.info('Start server connection')
app = HttpApi()


my_print('Start background task')
logger.info('Start background task')
thread = Thread(target=async_server_connection)
thread.setDaemon(daemonic=True)
thread.start()
thread.join()
```

URLS

Todas las aplicaciones web son accesibles desde el navegador, en el navegador se accede mediante las URL, que son las direcciones del recurso al que queremos acceder

del servidor, en Django se dispone de un archivo donde se deben registrar todas las URLs de la aplicación, el script es:

```
#from django.urls import path
from django.conf.urls import url


from . import views


urlpatterns = [
    url(r'^index', views.index, name='index'),
    url(r'^add', views.add, name='add'),
    url(r'^actions', views.actions, name='actions'),
    url(r'^notifications', views.notifications, name='notifications'),
    url(r'^verify', views.verify, name='verify'),
    url(r'^shutdown', views.shutdown, name='shutdown'),
    url(r'^update/ip', views.update_ip, name='update_ip'),
    url(r'^update/name', views.update_name, name='update_name'),
    url(r'^update/password', views.update_password, name='update_password'),
    url(r'^update/datetime', views.update_datetime, name='update_datetime'),
]
```

CÓDIGO PRINCIPAL

Todas las URLs anteriormente mencionadas son procesadas por diversas funciones almacenadas en el script “views.py” el cual contiene:

```
debug=True
```

```
from django.shortcuts import render
from django.http import JsonResponse
from serial import SerialException
from json.decoder import JSONDecodeError
from database_tools import Setting
from database_tools import Action
from database_tools import Employee
from database_tools import Notification
from http_api import HttpApi
from time import sleep
from logger_tools import logger
from logger_tools import my_print
from fingerprint_tools import Fingerprint
from urllib import request
#from gpiozero import LED
from os import system

def send_error(title,subtitle):
    return JsonResponse({'title':title,'subtitle':subtitle,'type':'alert-danger'})

def send_warming(title,subtitle):
    return JsonResponse({'title':title,'subtitle':subtitle,'type':'alert-warming'})

def send_sucess(title,subtitle):
    return JsonResponse({'title':title,'subtitle':subtitle,'type':'alert-success'})

def index(request):
    ip_address = Setting.by_name('ip_address')
```

```
    biometric_name = Setting.by_name('biometric_name')
    address = ip_address.value.split('http://')
    if len(address)==2:
        ip_address.value=address[1]
    context={
        'ip_address':ip_address.value,
        'biometric_name':biometric_name.value
    }
    return render(request,'index.html',context)


def add(request):
    try:
        logger.debug('Adding new fingerprint')
        if not debug:
            fingerprint=Fingerprint.instance()
            position = fingerprint.add_new_finger(int(request.GET['code']))
            fingerprint.close()
        else:
            position=1
        if position == -1:
            logger.debug('\tError register fingerprint')
            return send_error('Huella no registrada', 'Error registrando la huella, para más
información revise el log de error')
        elif position == 0:
            logger.debug('\tAlready exists employee')
            return send_error('Huella ya registrada', 'Ya existe un usuario con esa huella')
        else:
            logger.debug('\tFinger registered')
            employee = Employee.by_code(request.GET['code'])
```

```
            for action in Action.by_employee(employee.id):
                logger.debug('\tDeleting all pending actions')
                action.delete()
            return send_sucess('Huella registrada', 'La huella fue registrada correctamente')
    except SerialException as error:
        return send_error('Sensor no encontrado','No se pudo establecer conexión con el sensor')
    except JSONDecodeError as error:
        return send_error('Respuesta inválida','El sensor envió una respuesta que no pudo ser procesada')
    except KeyError as error:
        return send_error('Respuesta inválida','La respuesta recibida por el sensor no tiene un formato correcto')


def actions(request):
    logger.debug('Loading biometric pending actions')
    actions = Action.all()
    actions_list = []
    for action in actions:
        actions_list.append(action.to_dict())

    return JsonResponse({
        'actions':actions_list
    })


def notifications(request):
    logger.debug('Loading biometric notifications')
    notifications = Notification.all()
    notifications_list = []
```

```
    for notification in notifications:
        notifications_list.append(notification.to_dict())
    return JsonResponse({
        'notifications':notifications_list,
    })


def verify(request):
    try:
        logger.debug('Verifiying finger')
        if not debug:
            fingerprint=Fingerprint.instance()
            id=fingerprint.get_id()
            fingerprint.close()
        else:
            id=-1
        if id==-1:
            logger.debug('\tFinger not found')
            return send_error('Empleado no registrado','La huella ingresada no coincide con
la de ningún empleado')
        else:
            logger.debug('\tFinger found')
            employee=Employee.by_code(str(id))
            if HttpApi.can_pass(employee):
                logger.debug('\tEmployee out of range')
                if not debug:
                    relay=LED(14)
                    relay.off()
                    sleep(1)
                    relay.on()
```

```
            return send_sucess('Empleado registrado','Bienvenido
'"+employee.lastname+', '+employee.name+"")

        else:

            logger.debug('\tWelcome employee')

            return send_warming('Empleado registrado','Empleado
'"+employee.lastname+', '+employee.name+"' está fuera de su horario permitido')


    except SerialException as error:

        return send_error('Sensor no encontrado','No se pudo establecer conexión con el
sensor')

    except JSONDecodeError as error:

        return error('Respuesta inválida','El sensor envió una respuesta que no pudo ser
procesada')

    except KeyError as error:

        return send_error('Respuesta inválida','La respuesta recibida por el sensor no tiene
un formato correcto')


def update_ip(request):

    try:

        logger.debug('Updating biometric ip')

        if not debug:

            fingerprint=Fingerprint.instance()

            fingerprint.clear_database()

            fingerprint.close()


        logger.debug('\tDeleting actions')

        for action in Action.all():

            action.delete()


        logger.debug('\tDeleting employees')
```

```
        for employee in Employee.all():
            employee.delete()


        logger.debug('\tUpdating ip')
        setting = Setting.by_name('ip_address')
        setting.value = 'http://' + request.GET['ip']
        setting.save()


        return send_sucess('IP actualizada', 'La IP fue actualizada correctamente')
    except SerialException as error:
        return send_error('Sensor no encontrado','No se pudo establecer conexión con el sensor')
    except JSONDecodeError as error:
        return error('Respuesta inválida','El sensor envió una respuesta que no pudo ser procesada')
    except KeyError as error:
        return send_error('Respuesta inválida','La respuesta recibida por el sensor no tiene un formato correcto')


def update_password(request):
    try:
        logger.debug('Updating biometric password')
        if not debug:
            fingerprint=Fingerprint.instance()
            fingerprint.clear_database()
            fingerprint.close()



        logger.debug('\tDeleting actions')
```

```
        for action in Action.all():
            action.delete()


        logger.debug('\tDeleting employees')
        for employee in Employee.all():
            employee.delete()


        logger.debug('\tUpdating password')
        setting = Setting.by_name('biometric_password')
        setting.value = request.GET['password']
        setting.save()


        return send_sucess('Contraseña actualizada', 'La contraseña del biométrico fue actualizada correctamente')
    except SerialException as error:
        return send_error('Sensor no encontrado','No se pudo establecer conexión con el sensor')
    except JSONDecodeError as error:
        return error('Respuesta inválida','El sensor envió una respuesta que no pudo ser procesada')
    except KeyError as error:
        return send_error('Respuesta inválida','La respuesta recibida por el sensor no tiene un formato correcto')


def update_name(request):
    try:
        logger.debug('Updating biometric name')
        if not debug:
            fingerprint=Fingerprint.instance()
            fingerprint.clear_database()
```

```
            fingerprint.close()



        logger.debug('\tDeleting actions')
        for action in Action.all():
            action.delete()


        logger.debug('\tDeleting employees')
        for employee in Employee.all():
            employee.delete()


        logger.debug('\tUpdating name')
        setting = Setting.by_name('biometric_name')
        setting.value = request.GET['name']
        setting.save()


        return send_sucess('Nombre actualizado', 'El nombre del biométrico fue
actualizado correctamente')
    except SerialException as error:
        return send_error('Sensor no encontrado','No se pudo establecer conexión con el
sensor')
    except JSONDecodeError as error:
        return error('Respuesta inválida','El sensor envió una respuesta que no pudo ser
procesada')
    except KeyError as error:
        return send_error('Respuesta inválida','La respuesta recibida por el sensor no tiene
un formato correcto')


def update_datetime(request):
    logger.debug('Updating biometric datetime')
```

```
    system('sudo date -s "'+request.GET['datetime']+'"')
    return send_sucess('Fecha actualizada','La fecha del servidor fue actualizada correctamente')


def shutdown(request):
    logger.debug('Shutting down')
    system('sudo shutdown -h now')
    return send_sucess('Apagando Raspberry','La raspberry recibió la señal de apagado correctamente')
```

CÓDIGO DEL SERVIDOR (RASPBERY PI 3)

RUTAS DE LA APLICACIÓN

El servidor corre una aplicación web escrita en PHP usando el framework web Laravel, todas las aplicaciones web tiene una lista de rutas o “end-points” que son los puntos de entrada de la aplicación, es decir, solo se ejecuta el código definido en los “end-points”.

Tabla 9
Tabla de endpoints de la aplicación del servidor

Método HTTP	Ruta	Acción
GET	users/login	UsersController@showLogin
POST	users/login	UsersController@login
GET	/	UsersController@home
GET	users/logout	UsersController@logout
GET	users/edit	UsersController@edit
PATCH	users/{user}/update	UsersController@update
GET	server/edit	ServerController@edit
PATCH	server/update	ServerController@update
POST	server/shutdown	ServerController@shutdown
GET	employees/index	EmployeesController@index

GET	employee/{employee}/show	EmployeesController@show
GET	employees/create	EmployeesController@create
PUT	employees/store	EmployeesController@store
GET	employee/{employee}/edit	EmployeesController@edit
PATCH	employee/{employee}/update	EmployeesController@update
GET	employee/{employee}/delete	EmployeesController@delete
DELETE	employee/{employee}/destroy	EmployeesController@destroy
GET	employees/export	EmployeesController@export
GET	biometrics/index	BiometricsController@index
GET	biometrics/{biometric}/show	BiometricsController@show
GET	biometrics/create	BiometricsController@create
PUT	biometrics/store	BiometricsController@store
GET	biometrics/{biometric}/edit	BiometricsController@edit
PATCH	biometrics/{biometric}/update	BiometricsController@update
GET	biometrics/{biometric}/delete	BiometricsController@delete
DELETE	biometrics/{biometric}/destroy	BiometricsController@destroy
GET	biometrics/export	BiometricsController@export
GET	schedules/index	SchedulesController@index
GET	schedules/{schedule}/show	SchedulesController@show
GET	schedules/create	SchedulesController@create
PUT	schedules/store	SchedulesController@store
GET	schedules/{schedule}/edit	SchedulesController@edit
PATCH	schedules/{schedule}/update	SchedulesController@update
GET	schedules/{schedule}/delete	SchedulesController@delete
DELETE	schedules/{schedule}/destroy	SchedulesController@destroy
GET	schedules/export	SchedulesController@export
GET	accesses/index	AccessesController@index
GET	accesses/{access}/show	AccessesController@show
GET	accesses/create	AccessesController@create
PUT	accesses/store	AccessesController@store
GET	accesses/{access}/edit	AccessesController@edit
PATCH	accesses/{access}/update	AccessesController@update
GET	accesses/{access}/delete	AccessesController@delete
DELETE	accesses/{access}/destroy	AccessesController@destro

		y
GET	accesses/export	AccessesController@export
GET	registers/index	RegistersController@index
GET	registers/export	RegistersController@export
GET	api/biometrics/{biometric}/employees/index	Api\BiometricsController@employeesIndex
GET	api/biometrics/{biometric}/employees/{employee}/check	Api\BiometricsController@employeesCheck
GET	/datetime	Api\BiometricsController@datetime

CONTROL DE USO DE LOS RECURSOS

Para cada recuso de la aplicación se dispuso de un controlador, el cual es una clase PHP que gestiona el recurso.

El controlador es el encargado de realizar todas las acciones que se describen en las rutas o “endpoints”.

Para la gestión de usuarios con acceso a la aplicación disponemos de un controlador llamado UsersController.php

```
<?php
namespace App\Http\Controllers;


use App\Http\Requests\LoginUserRequest;
use App\Http\Requests\UpdateUserRequest;
use App\Libraries\Redirector\Redirector;
use App\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
```

```
class UsersController extends Controller
{
    function home()
    {
        return redirect(__('urls.employees.index'));
    }
    function showLogin()
    {
        return view('guests.login');
    }
    function login(LoginUserRequest $request)
    {
        $data=$request->only(['email','password']);
        if(Auth::attempt($data))
        {
            return redirect(__('urls.home'));
        }
        return Redirector::error('Credenciales inválidas',['Las credenciales enviadas no son correctas']);
    }
    function logout()
    {
        Auth::logout();
        return redirect(__('urls.users.login'));
    }
    function edit()
    {
        return view('admin.accounts.edit',[
            'user'=>Auth::user(),
```

```
            /*'date'=>\Carbon\Carbon::now(),*/
        ]);
    }
    function update(User $user,UpdateUserRequest $request)
    {
        $user->name=$request->name;
        $user->email=$request->email;
        if(isset($request->password))
        {
            if($request->password != $request->passwordVerify)
            {
                return Redirector::error('Las contraseñas no coinciden',['No ingresó la misma contraseña dos veces']);
            }
            $user->password=bcrypt($request->password);
        }
        $user->save();
        return Redirector::success('Cuenta actualizada',['La cuenta fue actualizada correctamente']);
    }
}
```

De la misma forma cada biométrico dispone de un controlador llamado BiometricsController.php.

```
<?php
```

```php
namespace App\Http\Controllers;


use App;

use Maatwebsite\Excel\Facades\Excel;

use App\Biometric;

use App\Http\Requests\StoreBiometricRequest;

use App\Http\Requests\UpdateBiometricRequest;

use App\Libraries\Redirector\Redirector;

use Illuminate\Http\Request;


class BiometricsController extends Controller

{

    //MESSAGES

    static function biometricCreatedMessage(Biometric $biometric)

    {

        return Redirector::success('Biométrico creado',['El biométrico "'.$biometric->name.'" fue creado exitosamente.'],__('urls.biometrics.index'));

    }

    static function biometricUpdateMessage(Biometric $biometric)

    {

        return Redirector::success('Biométrico actualizado',['El biométrico "'.$biometric->name.'" fue actualizado exitosamente.'],__('urls.biometrics.index'));

    }

    static function biometricDestroyMessage(Biometric $biometric)

    {

        return Redirector::success('Biométrico eliminado',['El biométrico "'.$biometric->name.'" fue eliminado exitosamente.'],__('urls.biometrics.index'));

    }
```

```
    static function biometricCanNotBeDestroyedMessage(Biometric $biometric)
    {
        return Redirector::error('Biométrico no eliminado',['El biométrico "'.$biometric->name.'" no puede ser eliminado porque ya tiene accesos asignados.'],__('urls.biometrics.index'));
    }

    //ACTIONS
    function index(Request $request)
    {
        if(isset($request->searchField))
        {
            $biometrics=Biometric::whereRaw('lower('.$request->searchField.') like "%'.$request->searchValue.'%"')->orderBy('id','desc')->paginate(5);
        }
        else
        {
            $biometrics=Biometric::orderBy('id','desc')->paginate(5);
        }
        return view('admin.biometrics.index',[
            'biometrics'=>$biometrics,
        ]);
    }
    function show(Biometric $biometric)
    {
        return view('admin.biometrics.show',[
            'biometric'=>$biometric,
            'accesses'=>$biometric->accesses()->paginate(5),
        ]);
```

```
}
function create()
{
    return view('admin.biometrics.create');
}
function store(StoreBiometricRequest $request)
{
    $biometric=Biometric::create([
        'name'=>$request->name,
        'username'=>$request->username,
        'password'=>$request->password,
    ]);
    return self::biometricCreatedMessage($biometric);
}
function edit(Biometric $biometric)
{
    return view('admin.biometrics.edit',[
        'biometric'=>$biometric,
    ]);
}
function update(Biometric $biometric,UpdateBiometricRequest $request)
{
    $biometric->name=$request->name;
    $biometric->username=$request->username;
    if(isset($request->password))
    {
        $biometric->password=bcrypt($request->password);
    }
```

```
        $biometric->save();
        return self::biometricUpdateMessage($biometric);
    }
    function delete(Biometric $biometric)
    {
        return view('admin.biometrics.delete',[
            'biometric'=>$biometric,
        ]);
    }
    function destroy(Biometric $biometric)
    {
        if($biometric->canBeDeleted())
        {
            $biometric->delete();
            return self::biometricDestroyMessage($biometric);
        }
        return self::biometricCanNotBeDestroyedMessage($biometric);
    }
    function export()
    {
        $excel=App::make('excel');
        Excel::create('Biometric - Biométricos', function($excel) {
            $excel->setTitle('Biometric - Biométricos');
            $excel->setCreator('Danny Vaca y Juan Peña')->setCompany('Biometric');
            $excel->setDescription('Lista de biométricos registrados');
            $excel->sheet('Biométricos',function($sheet){
                $biometrics=[];
                foreach(Biometric::all() as $biometric)
```

```
            {
                $biometrics[]=[
                    'Nombre'=>$biometric->name,
                    'Usuario'=>$biometric->username,
                ];
            }
            $sheet->fromArray($biometrics);
        });
    })->download('xlsx');
  }
}
```

Para cada uno de los horarios de acceso se tiene el siguiente controlador llamado SchedulesController.php.

```
<?php

namespace App\Http\Controllers;

use App;
use Maatwebsite\Excel\Facades\Excel;
use App\Http\Requests\StoreScheduleRequest;
use App\Http\Requests\UpdateScheduleRequest;
use App\Libraries\Redirector\Redirector;
use App\Schedule;
use Illuminate\Http\Request;
```

```
class SchedulesController extends Controller
{
    private static $DAYS_NAMES=[
        'domingo'=>'sunday',
        'lunes'=>'monday',
        'martes'=>'tuesday',
        'miércoles'=>'wednesday',
        'jueves'=>'thursday',
        'viernes'=>'friday',
        'sábado'=>'saturday',
    ];

    //MESSAGES
    static function scheduleCreatedMessage(Schedule $schedule)
    {
        return Redirector::success('Horario creado',['El horario "'.$schedule->name.'" fue creado exitosamente.'],__('urls.schedules.index'));
    }
    static function scheduleUpdateMessage(Schedule $schedule)
    {
        return Redirector::success('Horario actualizado',['El horario "'.$schedule->name.'" fue actualizado exitosamente.'],__('urls.schedules.index'));
    }
    static function scheduleDestroyMessage(Schedule $schedule)
    {
        return Redirector::success('Horario eliminado',['El horario "'.$schedule->name.'" fue eliminado exitosamente.'],__('urls.schedules.index'));
    }
    static function scheduleCanNotBeDestroyedMessage(Schedule $schedule)
```

```
    {
        return Redirector::error('Horario no eliminado',['El horario "'.$schedule->name.'" no puede ser eliminado porque ya tiene accesos asignados.'],__('urls.schedules.index'));
    }

    //ACTIONS
    function index(Request $request)
    {
        if(isset($request->searchField))
        {
            switch($request->searchField)
            {
                case 'from':
                    $schedules=Schedule::where('from','>',$request->searchValue)->orderBy('id','desc')->paginate(5);
                    break;
                case 'days':
                    $day=self::$DAYS_NAMES[strtolower($request->searchValue)];
                    $schedules=Schedule::where($day,'1')->orderBy('id','desc')->paginate(5);
                    break;
                case 'to':
                    $schedules=Schedule::where('to','<',$request->searchValue)->orderBy('id','desc')->paginate(5);
                    break;
                default:
                    $schedules=Schedule::whereRaw('lower("'.$request->searchField.'") like "%'.$request->searchValue.'%"')->orderBy('id','desc')->paginate(5);
                    break;
            }
        }
```

```
        else
        {
            $schedules=Schedule::orderBy('id','desc')->paginate(5);
        }
        return view('admin.schedules.index',[
            'schedules'=>$schedules,
        ]);
    }
    function show(Schedule $schedule)
    {
        return view('admin.schedules.show',[
            'schedule'=>$schedule,
            'accesses'=>$schedule->accesses()->paginate(5),
        ]);
    }
    function create()
    {
        return view('admin.schedules.create');
    }
    function store(StoreScheduleRequest $request)
    {
        $schedule=Schedule::create([
            'name'=>$request->name,
            'from'=>$request->from,
            'to'=>$request->to,
            'sunday'=>(isset($request->sunday) and $request->sunday==true)?true:false,
            'monday'=>(isset($request->monday) and $request->monday==true)?true:false,
            'tuesday'=>(isset($request->tuesday) and $request->tuesday==true)?true:false,
            'wednesday'=>(isset($request->wednesday) and $request-
```

```
>wednesday==true)?true:false,

            'thursday'=>(isset($request->thursday) and $request-
>thursday==true)?true:false,

            'friday'=>(isset($request->friday) and $request->friday==true)?true:false,

            'saturday'=>(isset($request->saturday) and $request-
>saturday==true)?true:false,

        ]);

        return self::scheduleCreatedMessage($schedule);

    }

    function edit(Schedule $schedule)

    {

        return view('admin.schedules.edit',[

            'schedule'=>$schedule,

        ]);

    }

    function update(Schedule $schedule, UpdateScheduleRequest $request)

    {

        $schedule->name=$request->name;

        $schedule->from=$request->from;

        $schedule->to=$request->to;

        $schedule->sunday=(isset($request->sunday) and $request-
>sunday==true)?true:false;

        $schedule->monday=(isset($request->monday) and $request-
>monday==true)?true:false;

        $schedule->tuesday=(isset($request->tuesday) and $request-
>tuesday==true)?true:false;

        $schedule->wednesday=(isset($request->wednesday) and $request-
>wednesday==true)?true:false;

        $schedule->thursday=(isset($request->thursday) and $request-
>thursday==true)?true:false;

        $schedule->friday=(isset($request->friday) and $request->friday==true)?true:false;

        $schedule->saturday=(isset($request->saturday) and $request-
```

```
>saturday==true)?true:false;
        $schedule->save();
        return self::scheduleUpdateMessage($schedule);
    }
    function delete(Schedule $schedule)
    {
        return view('admin.schedules.delete',[
            'schedule'=>$schedule,
        ]);
    }
    function destroy(Schedule $schedule)
    {
        if($schedule->canBeDeleted())
        {
            $schedule->delete();
            return self::scheduleDestroyMessage($schedule);
        }
        return self::scheduleCanNotBeDestroyedMessage($schedule);
    }
    function export()
    {
        $excel=App::make('excel');
        Excel::create('Biometric - Horarios', function($excel) {
            $excel->setTitle('Biometric - Horarios');
            $excel->setCreator('Danny Vaca y Juan Peña')->setCompany('Biometric');
            $excel->setDescription('Lista de horarios registrados');
            $excel->sheet('Horarios',function($sheet){
                $schedules=[];
```

```
            foreach(Schedule::all() as $schedule)
            {
                $schedules[]=[
                    'Nombre'=>$schedule->name,
                    'Desde'=>$schedule->from,
                    'Hasta'=>$schedule->to,
                    'Domingo'=>$schedule->sunday,
                    'Lunes'=>$schedule->monday,
                    'Martes'=>$schedule->tuesday,
                    'Miércoles'=>$schedule->wednesday,
                    'Jueves'=>$schedule->thursday,
                    'Viernes'=>$schedule->friday,
                    'Sábado'=>$schedule->saturday,
                ];
            }
            $sheet->fromArray($schedules);
        });
    })->download('xlsx');
  }
}
```

Cada relación entre “horario” y “empleado” se llama “acceso” y dispone de su propio controlador llamado AccessesController.php.

```
<?php
```

```
namespace App\Http\Controllers;


use App;
use Maatwebsite\Excel\Facades\Excel;
use App\Access;
use App\Biometric;
use App\Employee;
use App\Http\Requests\StoreAccessRequest;
use App\Http\Requests\UpdateAccessRequest;
use App\Libraries\Redirector\Redirector;
use App\Schedule;
use Illuminate\Http\Request;


class AccessesController extends Controller
{
    //MESSAGES
    static function accessCreatedMessage(Access $access, $lastUrl)
    {
        $lastUrl=($lastUrl!='')?$lastUrl:__('urls.accesses.index');
        return Redirector::success('Acceso creado',['El acceso fue creado
exitosamente.'],$lastUrl);
    }
    static function accessUpdateMessage(Access $access, $lastUrl)
    {
        $lastUrl=($lastUrl!='')?$lastUrl:__('urls.accesses.index');
        return Redirector::success('Acceso actualizado',['El acceso fue actualizado
exitosamente.'],$lastUrl);
    }
    static function accessDestroyMessage(Access $access, $lastUrl)
```

```
    {
        $lastUrl=($lastUrl!='')?$lastUrl:__('urls.accesses.index');
        return Redirector::success('Acceso eliminado',['El acceso fue eliminado
exitosamente.'],$lastUrl);
    }

    //ACTIONS
    function index(Request $request)
    {
        if(isset($request->searchField))
        {
            $accesses=Access::orderBy('accesses.id','desc');
            switch($request->searchField)
            {
                case 'employee':
                    $accesses->join('employees','employees.id','=','accesses.employee_id');
                    $accesses->where('employees.name','like','%'.$request->searchValue.'%');
                    break;
                case 'biometric':
                    $accesses->join('biometrics','biometrics.id','=','accesses.biometric_id');
                    $accesses->where('biometrics.name','like','%'.$request->searchValue.'%');
                    break;
                case 'schedule':
                    $accesses->join('schedules','schedules.id','=','accesses.schedule_id');
                    $accesses->where('schedules.name','like','%'.$request->searchValue.'%');
                    break;
            }
            $accesses=$accesses->paginate(5);
        }
```

```
        else
        {
            $accesses=Access::orderBy('id','desc')->paginate(5);
        }
        return view('admin.accesses.index',[
            'accesses'=>$accesses
        ]);
    }
    function create()
    {
        return view('admin.accesses.create',[
            'employees'=>Employee::orderBy('id','desc')->get(),
            'biometrics'=>Biometric::orderBy('id','desc')->get(),
            'schedules'=>Schedule::orderBy('id','desc')->get(),
            'lastUrl'=>redirect()->back()->getTargetUrl(),
        ]);
    }
    function store(StoreAccessRequest $request)
    {
        $access=Access::create([
            'employee_id'=>$request->employee_id,
            'biometric_id'=>$request->biometric_id,
            'schedule_id'=>$request->schedule_id
        ]);
        return self::accessCreatedMessage($access,$request->lastUrl);
    }
    function edit(Access $access)
    {
```

```
        return view('admin.accesses.edit',[
            'employees'=>Employee::orderBy('id','desc')->get(),
            'biometrics'=>Biometric::orderBy('id','desc')->get(),
            'schedules'=>Schedule::orderBy('id','desc')->get(),
            'access'=>$access,
            'lastUrl'=>redirect()->back()->getTargetUrl(),
        ]);
    }
    function update(Access $access,UpdateAccessRequest $request)
    {
        $access->employee_id=$request->employee_id;
        $access->biometric_id=$request->biometric_id;
        $access->schedule_id=$request->schedule_id;
        $access->save();
        return self::accessUpdateMessage($access,$request->lastUrl);
    }
    function delete(Access $access)
    {
        return view('admin.accesses.delete',[
            'access'=>$access,
            'lastUrl'=>redirect()->back()->getTargetUrl(),
        ]);
    }
    function destroy(Access $access,Request $request)
    {
        $access->delete();
        return self::accessDestroyMessage($access,$request->lastUrl);
    }
```

```
    function export()
    {
        $excel=App::make('excel');
        Excel::create('Biometric - Accessos', function($excel) {
            $excel->setTitle('Biometric - Accessos');
            $excel->setCreator('Danny Vaca y Juan Peña')->setCompany('Biometric');
            $excel->setDescription('Lista de accesos registrados');
            $excel->sheet('Accessos',function($sheet){
                $accesses=[];
                foreach(Access::all() as $access)
                {
                    $accesses[]=[
                        'Empleado'=>$access->employee->lastname.', '.$access->employee->name,
                        'Biométrico'=>$access->biometric->name,
                        'Horario'=>$access->schedule->name,
                    ];
                }
                $sheet->fromArray($accesses);
            });
        })->download('xlsx');
    }
}
```

Cada vez que un empleado accede a una ubicación se genera un registro del acceso, estos registros disponen del siguiente controlador llamado RegistersController.php.

```php
<?php

namespace App\Http\Controllers;

use App;
use Maatwebsite\Excel\Facades\Excel;
use App\Register;
use Illuminate\Http\Request;

class RegistersController extends Controller
{
    function index(Request $request)
    {
        if(isset($request->searchField))
        {
            $registers=Register::orderBy('registers.id','desc');
            switch($request->searchField)
            {
                case 'employee':
                    $registers->join('employees','employees.id','=','registers.employee_id');
                    $registers->where('employees.name','like','%'.$request->searchValue.'%');
                    break;
                case 'biometric':
                    $registers->join('biometrics','biometrics.id','=','registers.biometric_id');
                    $registers->where('biometrics.name','like','%'.$request->searchValue.'%');
                    break;
            }
            $registers=$registers->paginate(5);
```

```
        }
        else
        {
            $registers=Register::orderBy('id','desc')->paginate(10);
        }

        return view('admin.registers.index',[
            'registers'=>$registers,
        ]);
    }
    function export()
    {
        $excel=App::make('excel');
        Excel::create('Biometric - Registros', function($excel) {
            $excel->setTitle('Biometric - Registros');
            $excel->setCreator('Danny Vaca y Juan Peña')->setCompany('Biometric');
            $excel->setDescription('Lista de registros registrados');
            $excel->sheet('Registros',function($sheet){
                $registers=[];
                foreach(Register::all() as $register)
                {
                    $registers[]=[
                        'Biométrico'=>$register->biometric->name,
                        'Empleado'=>$register->employee->lastname.', '.$register->employee-
>name,
                        'Fecha'=>$register->created_at  ,
                    ];
                }
                $sheet->fromArray($registers);
```

```
            });
        })->download('xlsx');
    }
}
```

Las configuraciones del servidor tienen su propio controlador llamado ServerController.php.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Libraries\Redirector\Redirector;

class ServerController extends Controller
{
    function edit()
    {
        return view('admin.server.edit',[
            'datetime'=>\Carbon\Carbon::now(),
        ]);
    }
    function update(Request $request)
    {
        exec("sudo date -s '{$request->date}'");
        return Redirector::success('Servidor actualizado',['El servidor fue actualizado
```

```
exitosamente.'],__('urls.server.edit'));
    }
    function shutdown(Request $request)
    {
        exec("sudo shutdown -h now");
        return Redirector::success('Apagando servidor',['EL servidor recibió la orden de apagar exitosamente.'],__('urls.server.edit'));
    }
}
```

Para gestionar las conexiones de otros biométricos para descargar la información de los empleados se creó un controlador llamado BiometricsController.php, pero este está ubicado en la carpeta api del directorio de controladores.

```
<?php

namespace App\Http\Controllers\Api;

use App\Biometric;
use App\Employee;
use App\Register;
use Carbon\Carbon;
use Illuminate\Support\Facades\Hash;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class BiometricsController extends Controller
{
```

```
static private $isDebugging = false;
static function debug($data)
{
    if(self::$isDebugging)
    {
        echo $data.PHP_EOL;
    }
}
//MESSAGES
function biometricNotFoundMessage()
{
    return response()->json(['code'=>'0x01','message'=>'Biometric not found']);
}
function employeeNotFoundMessage()
{
    return response()->json(['code'=>'0x02','message'=>'Employee not found']);
}
function employeeCanNotAccess()
{
    return response()->json(['code'=>'0x03','message'=>'Access not allowed']);
}
function employeeCanAccess()
{
    return response()->json(['code'=>'0x04','message'=>'Access allowed']);
}
function missingPassword()
{
    return response()->json(['code'=>'0x05','message'=>'Missing password']);
```

```
    }
    function invalidPassword()
    {
        return response()->json(['code'=>'0x06','message'=>'Invalid password']);
    }

    function employeesIndex($biometricName,Request $request)
    {
        $biometric=Biometric::where('username',$biometricName)->first();
        if($biometric==null)
        {
            return self::biometricNotFoundMessage();
        }
        if(!isset($request->password))
        {
            return self::missingPassword();
        }
        if(!Hash::check($request->password,$biometric->password))
        {
            return self::invalidPassword();
        }
        if($biometric==null)
        {
            return self::biometricNotFoundMessage();
        }
        $employees=[];
        foreach($biometric->accesses as $access)
        {
            $employees[$access->employee->name]=$access->employee-
```

```
>only(['code','lastname','name','password','password_clean']);
        }
        return response()->json(array_values($employees));
    }
    function employeesCheck($biometricName,$employeeCode)
    {
        $biometric=Biometric::where('username',$biometricName)->first();
        if($biometric==null)
        {
            return self::biometricNotFoundMessage();
        }

        $employee=Employee::where('code',$employeeCode)->first();
        if($employee==null)
        {
            return self::employeeNotFoundMessage();
        }

        $now=Carbon::now();
        $now->addDay();
    self::debug('Now: '.$now);

        $accesses=$employee->accesses()->where('biometric_id',$biometric->id)->get();
        foreach($accesses as $access)
        {
            $isToday=false;
            $isToday=($now->dayOfWeek==0 and $access->schedule-
>sunday)?true:$isToday;
            $isToday=($now->dayOfWeek==1 and $access->schedule-
```

```
>monday)?true:$isToday;

        $isToday=($now->dayOfWeek==2 and $access->schedule->tuesday)?true:$isToday;

        $isToday=($now->dayOfWeek==3 and $access->schedule->wednesday)?true:$isToday;

        $isToday=($now->dayOfWeek==4 and $access->schedule->thursday)?true:$isToday;

        $isToday=($now->dayOfWeek==5 and $access->schedule->friday)?true:$isToday;

        $isToday=($now->dayOfWeek==6 and $access->schedule->saturday)?true:$isToday;

        self::debug($isToday?'today':'not today');

        if($isToday)

        {

            $from=Carbon::parse($access->schedule->from);

            $from->addDay();

            $to=Carbon::parse($access->schedule->to);

            $to->addDay();

            self::debug('From: '.$from);

            self::debug('To: '.$to);

            if($now->greaterThanOrEqualTo($from) and $now->lessThanOrEqualTo($to))

            {

                Register::create([

                    'biometric_id'=>$biometric->id,

                    'employee_id'=>$employee->id,

                    'dateTime'=>$now,

                ]);

                return self::employeeCanAccess();

            }

        }
```

```
        }
        return self::employeeCanNotAccess();
    }
    function datetime()
    {
        header("Access-Control-Allow-Origin: *");
        return response()->json(Carbon::now());
    }
}
```

ESTRUCTURA DE LA BASE DE DATOS

Cada tabla de la base de datos es creada a partir de una clase PHP, estas clases especiales se llaman migrationes.

La tabla de usuarios es creada por la migración llamada CreateUsersTable

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    public function up()
```

```
    {
        Schema::create('users', function (Blueprint $table){
            $table->engine='InnoDB';
            $table->increments('id');
            $table->string('name');
            $table->string('email');
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

La tabla de los empleados esta definida por la migración CreateEmployeesTable

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
```

```
class CreateEmployeesTable extends Migration
{
    public function up()
    {
        Schema::create('employees', function (Blueprint $table) {
            $table->engine='InnoDB';
            $table->increments('id');
            $table->string('code');
            $table->string('dni');
            $table->string('name');
            $table->string('password');
            $table->string('password_clean');
            $table->string('lastname');
            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down()
    {
        Schema::dropIfExists('employees');
    }
}
```

La tabla de biométricos es creada con la migración CreateBiometricsTable

```
<?php
```

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateBiometricsTable extends Migration
{
    public function up()
    {
        Schema::create('biometrics', function (Blueprint $table) {
            $table->engine='InnoDB';
            $table->increments('id');
            $table->string('name');
            $table->string('username');
            $table->string('password');
            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down()
    {
        Schema::dropIfExists('biometrics');
    }
}
```

La tabla de horarios es creada con la migración CreateSchedulesTable

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateSchedulesTable extends Migration
{
    public function up()
    {
        Schema::create('schedules', function (Blueprint $table) {
            $table->engine='InnoDB';
            $table->increments('id');
            $table->string('name');
            $table->time('from');
            $table->time('to');
            $table->boolean('sunday');
            $table->boolean('monday');
            $table->boolean('tuesday');
            $table->boolean('wednesday');
            $table->boolean('thursday');
            $table->boolean('friday');
            $table->boolean('saturday');
            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down()
```

```
    {
        Schema::dropIfExists('schedules');
    }
}
```

La tabla de accesos es creada con la migración CreateAccessesTable

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAccessesTable extends Migration
{
    public function up()
    {
        Schema::create('accesses', function (Blueprint $table) {
            $table->engine='InnoDB';
            $table->increments('id');

            $table->integer('employee_id')->unsigned();
            $table->foreign('employee_id')->references('id')->on('employees');

            $table->integer('biometric_id')->unsigned();
            $table->foreign('biometric_id')->references('id')->on('biometrics');
```

```
            $table->integer('schedule_id')->unsigned();
            $table->foreign('schedule_id')->references('id')->on('schedules');

            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down()
    {
        Schema::dropIfExists('accesses');
    }
}
```

Finalmente, la tabla de registros es creada con la migración CreateRegistersTable

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateRegistersTable extends Migration
{

    public function up()
    {
```

```
        Schema::create('registers', function (Blueprint $table) {
            $table->engine='InnoDB';
            $table->increments('id');
            $table->integer('biometric_id')->unsigned();
            $table->foreign('biometric_id')->references('id')->on('biometrics');
            $table->integer('employee_id')->unsigned();
            $table->foreign('employee_id')->references('id')->on('employees');
            $table->dateTime('dateTime');
            $table->timestamps();
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('register');
    }
}
```

ACCESO A LA BASE DE DATOS

El acceso a la base de datos es de vital importancia para el presente proyecto de titulación, para poder consultar cada una de las tablas de la base de datos, se creó un modelo por cada tabla, el modelo solo tiene conocimiento del contenido de cada tabla y funciona como capa de abstracción para las consultas.

La tabla de usuarios es representada por el modelo User.php.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use SoftDeletes;
    use Notifiable;
    protected $fillable = [
        'name', 'email', 'password',
    ];
    protected $hidden = [
        'password', 'remember_token',
    ];
}
```

La tabla de empleados es representada por el modelo Employee.php.

```php
<?php

namespace App;
```

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;


class Employee extends Model
{
    use SoftDeletes;
    protected $fillable = [
        'code','dni','name', 'lastname','password','password_clean'
    ];
    function accesses()
    {
        return $this->hasMany(Access::class);
    }
    function registers()
    {
        return $this->hasMany(Register::class);
    }
    function canBeDeleted()
    {
        return $this->accesses()->count()==0 and $this->registers()->count()==0;
    }
}
```

La tabla de biométricos es representada por el modelo Biometric.php

```
<?php
```

```php
namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Biometric extends Model
{
    use SoftDeletes;
    protected $fillable = [
        'name', 'username','password'
    ];
    function accesses()
    {
        return $this->hasMany(Access::class);
    }
    function registers()
    {
        return $this->hasMany(Register::class);
    }
    function canBeDeleted()
    {
        return $this->accesses()->count()==0 and $this->registers()->count()==0;
    }
}
```

La tabla de horarios es representada por el modelo Schedules.php.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Schedule extends Model
{
    use SoftDeletes;
    protected $fillable = [
        'name', 'from', 'to', 'sunday', 'monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday'
    ];
    function accesses()
    {
        return $this->hasMany(Access::class);
    }
    function canBeDeleted()
    {
        return $this->accesses()->count()==0;
    }
}
```

La tabla de accesos es representada por el modelo Access.php.

```php
<?php
```

```php
namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Access extends Model
{
    use SoftDeletes;
    protected $fillable = [
        'employee_id','biometric_id','schedule_id'
    ];
    function employee()
    {
        return $this->belongsTo(Employee::class);
    }
    function biometric()
    {
        return $this->belongsTo(Biometric::class);
    }
    function schedule()
    {
        return $this->belongsTo(Schedule::class);
    }
}
```

La tabla de registros es representada por el modelo Registers.php

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Register extends Model
{
    protected $fillable=[
        'biometric_id','employee_id','dateTime'
    ];
    function employee()
    {
        return $this->belongsTo(Employee::class);
    }
    function biometric()
    {
        return $this->belongsTo(Biometric::class);
    }
}
```

INTERFAZ GRÁFICA

La interfaz gráfica de la aplicación está escrita enteramente en HTML, CSS y Javascript, pero se usa el gestor de plantillas de Laravel llamado Blade para crear componentes independientes y reutilizables.

Todas las vistas parten de una plantilla básica la cual contiene la estructura general de todos los proyectos, esta se llamada basic.blade.php está en la carpeta templates.

```
<html lang="es">
<head>
    <!-- metas -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- .metas -->

    <title>{{$title or env('APP_NAME')}}</title>

    <!-- style -->
    <link rel="stylesheet" type="text/css"
href="{{asset('plugins/bootstrap/css/bootstrap.css')}}">
    <link rel="stylesheet" type="text/css" href="{{asset('plugins/font-awesome/web-fonts-
with-css/css/fontawesome-all.css')}}">
    <!-- .style -->

    <!-- scripts -->
    <script type="text/javascript" src="{{asset('plugins/jquery/js/jquery.js')}}"></script>
    <script type="text/javascript" src="{{asset('plugins/popper/js/popper.js')}}"></script>
    <script type="text/javascript"
src="{{asset('plugins/bootstrap/js/bootstrap.js')}}"></script>
    <!-- .scripts -->
</head>
<body class="{{$class or ''}}">
    <!-- content -->
    {{$slot}}
```

```
    <!-- .content -->
</body>
</html>
```

Si tiene de forma separada los componentes independientes y reutilizables como modale, formularios, etc.

Cada vista dispone de un formulario de búsqueda, ese formulario de búsqueda es el mismo apra todas las vistas y es el archivo search.blade.php de la carpeta partials.

```
<form method="get" id="search-form" style="display: none">
    <div class="container">
        <select name="searchField">
            @foreach($fields as $field)
                <option value="{{$field['value']}}">
                    {{$field['label']}}
                </option>
            @endforeach
        </select>
        <div class="row">
            <div class="col-md-12">
                <div class="input-group" id="adv-search">
                    <input type="text" class="form-control" name="searchValue"
placeholder="Buscar">
                    <div class="input-group-btn">
                        <div class="btn-group" role="group">
```

```
                        <button type="submit" class="btn btn-primary">
                            <i class="fa fa-search"></i>
                        </button>
                        <button type="button" class="btn btn-danger"
onclick="document.getElementById('search-form').style.display='none'">
                            <i class="fa fa-times"></i>
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</form>
```

La barra de navegación de la aplicación está en el archivo navbar.blade.php de la carpeta partials.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
    <a class="navbar-brand" href="#">
        MyBiometric
    </a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
```

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
            <a class="nav-link" href="@lang('urls.employees.index')">
                Empleados
            </a>
        </li>
        <li class="nav-item active">
            <a class="nav-link" href="@lang('urls.biometrics.index')">
                Biometricos
            </a>
        </li>
        <li class="nav-item active">
            <a class="nav-link" href="@lang('urls.schedules.index')">
                Horarios
            </a>
        </li>
        <li class="nav-item active">
            <a class="nav-link" href="@lang('urls.accesses.index')">
                Accesos
            </a>
        </li>
        <li class="nav-item active">
            <a class="nav-link" href="@lang('urls.registers.index')">
                Registros
            </a>
        </li>
```

```
            <li class="nav-item active">
                <a class="nav-link" href="@lang('urls.users.edit')">
                    Mi Cuenta
                </a>
            </li>
            <li class="nav-item active">
                <a class="nav-link" href="@lang('urls.server.edit')">
                    Servidor
                </a>
            </li>

            <li class="nav-item">
                <a class="nav-link" href="@lang('urls.users.logout')">
                    Cerrar
                </a>
            </li>
        </ul>
    </div>
</nav>
```

Los mensajes de confirmación de las acciones realizadas son manejadas por el componenete flashed.blade.php.

```
@if($errors->any())
    @component('partials.alert')
        @slot('type','danger')
```

```
        <h4 class="alert-heading">
            @if($errors->count()==1)
                Error
            @else
                Errores
            @endif
        </h4>
        <ul>
            @foreach($errors->all() as $error)
                <li>
                    {{$error}}
                </li>
            @endforeach
        </ul>
    @endcomponent
@endif

@if(session('redirector-data'))
    @component('partials.alert')
        @slot('type',session('redirector-data')['type'])
        <h4 class="alert-heading">
            {{session('redirector-data')['title']}}
        </h4>
        <ul>
            @foreach(session('redirector-data')['messages'] as $message)
                <li>
                    {{$message}}
                </li>
```

```
            @endforeach
        </ul>
    @endcomponent
@endif
```

Todos los elementos en forma de cartas al estilo de “material design” de Google están hechas con el archivo card.blade.php de la capeta partials.

```
<div class="card">
    @if(isset($header))
        <div class="card-header">
            {{$header}}
        </div>
    @endif

    <div class="card-body">
        @if(isset($title))
            <h4 class="card-title">
                {{$title}}
            </h4>
        @endif
        @if(isset($subtitle))
            <h6 class="card-subtitle mb-2 text-muted">
                {{$subtitle}}
            </h6>
        @endif
```

```
        @if(isset($title) or isset($subtitle))
            <hr>
        @endif

        <div class="card-text">
            {{$content}}
        </div>
    </div>

    @if(isset($footer))
        <div class="card-footer text-muted">
            {{$footer}}
        </div>
    @endif
</div>
```

Todos los mensajes de alerta son gestionados por el archivo alert.blade.php de la carpeta partials.

```
<div class="alert alert-{{$type or 'primary'}} alert-dismissible fade show" role="alert">
    {{$slot}}
    <br>
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">
            &times;
```

```
        </span>
    </button>
</div>
```

Todas las cuadrículas de la aplicación son gestionadas por el archivo row.blade.php de la carpeta partials/grid

```
<div class="container">
    <div class="row {{$class or ''}}">
        {{$slot}}
    </div>
</div>
```

Todos los botones de formularios de la aplicación son gestionados por button.blade.php de la carpeta partials/forms

```
<button type="submit" class="btn {{isset($color)?$color:'btn-primary'}} {{(isset($large) and $large==true)?'btn-block':''}}">
    {{$text}}
</button>
```

Los formularios para recolección de la información son gestionados por el archivo form.blade.php de la carpeta partials/forms.

```
<form action="{{$action}}" method="{{isset($method)?(($method=='get' or $method=='GET')?'GET':'POST'):'POST'}}">
    {{csrf_field()}}
    @if(isset($method))
        @if($method=='put' or $method=='PUT')
            {{method_field('PUT')}}
        @elseif($method=='patch' or $method=='PATCH')
            {{method_field('PATCH')}}
        @elseif($method=='delete' or $method=='DELETE')
            {{method_field('DELETE')}}
        @elseif($method=='post' or $method=='POST')
        @endif
    @endif
    {{$slot}}
</form>
```

Todos los campos de entrada de información que usa el formulario son hechos con le archivo input.blade.php de la carpeta partials/forms.

```
@if($type=='checkbox')
    <div class="form-check">
        <label class="form-check-label">
            <input type="checkbox" class="form-check-input" name="{{$name}}" {{(isset($disabled) and $disabled==true)?'disabled':''}} {{(isset($autofocus) and $autofocus==true)?'autofocus':''}} {{(isset($required) and $required==true)?'required':''}} {{(isset($value) and $value==true)?'checked':''}}>
            {{$label}}
        </label>
```

```
    </div>
@elseif($type=='list')
    <div class="form-group">
        <label for="{{$name}}">
            {{$label}}
        </label>
        <select class="form-control" id="{{$name}}" name="{{$name}}">
            @foreach($items as $item)
                <option value="{{$item->id}}" {{(isset($value) and $value==$item->id)?'selected':''}}>
                    {{isset($item->lastname)?$item->lastname.', ':''}}{{$item->name}}
                </option>
            @endforeach
        </select>
    </div>
@else
    <div class="form-group">
        <label for="{{$name}}">
            {{$label}}
        </label>
        <input class="form-control" type="{{$type or 'text'}}" name="{{$name}}" id="{{$name}}" placeholder="{{$placeholder or ''}}" {{(isset($required) and $required==true)?'required':''}} {{(isset($autofocus) and $autofocus==true)?'autofocus':''}} @isset($value) value="{{$value}}" @endif {{(isset($disabled) and $disabled==true)?'disabled':''}}>
        @if(isset($help))
            <small id="{{$help}}-help" class="form-text text-muted">
                {{$help}}
            </small>
        @endif
```

```
    </div>
@endif
```

El formulario de inicio de sesión de la aplicación es gestionado por el archivo login.blade.php de la carpeta guests

```
@component('templates.basic')
    @slot('class','bg-secondary')
    @component('partials.grid.row')
        @slot('class','justify-content-center')
        <div class="col-sm-8">
            <br>
            @include('partials.flashed')
            @component('partials.card')
                @slot('title','Inicio de sesión')
                @slot('subtitle','Para acceder ingresa tu usuario y contraseña')
                @slot('content')
                    @component('partials.forms.form')
                        @slot('action',__('urls.users.login'))
                        @component('partials.forms.input')
                            @slot('label','Email')
                            @slot('type','email')
                            @slot('name','email')
                            @slot('autofocus',true)
                            @slot('icon','envelope')
                            @slot('value',old('email'))
```

```
                    @slot('placeholder','Dirección electrónica')
                    @slot('required',true)
                @endcomponent
                @component('partials.forms.input')
                    @slot('label','Contraseña')
                    @slot('type','password')
                    @slot('name','password')
                    @slot('icon','lock')
                    @slot('value',old('password'))
                    @slot('placeholder','Contraseña')
                    @slot('required',true)
                @endcomponent
                @component('partials.forms.button')
                    @slot('text','Iniciar sesión')
                    @slot('large',true)
                @endcomponent
            @endcomponent
        @endslot
    @endcomponent
  </div>
 @endcomponent
@endcomponent
```