Universitat Oberta de Catalunya

uoc.edu

Securización API REST con Json para domótica

José Manuel López Castillo
Postgrado ciberseguridad en redes y sistemas

Javier Parra Arnau
Profesor responsable de la asignatura.

Junio del 2020

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Título del trabajo:	*Securización API REST con json para domótica*
Nombre del autor:	José Manuel López Castillo
Nombre del consultor/a:	*Javier Parra Arnau*
Nombre del PRA:	*Javier Parra Arnau*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Postgrado Ciberseguridad Redes y Sistemas*
Área del Trabajo Final:	*Proyecto final de postgrado*
Idioma del trabajo:	*Castellano*
Palabras clave	*API REST, token, doble validación.*

La finalidad del proyecto es crear un software seguro utilizando la arquitectura API REST y json para ofrecer servicio domótico de la manera más segura posible.

Para que el sistema sea seguro se requiere doble autentificación en el sistema además de incluir tokens en cada una de sus funciones.

La domótica está ampliamente extendida en el mundo de las empresas y particulares y a veces la seguridad pasa un poco desapercibida dando más importancia a la funcionalidad.

Muchos de estos elementos domóticos están las 24 horas del día activos, por lo que las empresas o la persona particular puede tener un problema de seguridad al compartir la misma red.

Añadiendo más seguridad a dichos elementos se securiza el entorno y hace estar más seguros en caso de un ciberataque.

Abstract :

The target of the project is to create secure software using the API REST and json architecture to offer home automation service in the most secure way possible.

For the system to be secure, double authentication is required in the system in addition to including tokens in each of its functions.

Home automation is widely spread in the world of companies and individuals and sometimes security goes a little bit unnoticed giving more importance to functionality.

Many of these home automation elements are active 24 hours a day, so companies or individuals can have a security problem when sharing the same network.

Adding more security to these elements secures the environment and makes it safer in the event of a cyber attack.

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 3
1.3 Enfoque y método seguido 4
1.4 Planificación del proyecto 5
1.5 Breve sumario de productos obtenidos 6
1.6 Breve descripción de los otros capítulos de la memoria 7
2. Funcionamiento software y arquitectura de la aplicación 7
2.1 Servidor 7
2.1.1 Validación 7
2.1.2 Tokens en las funciones 9
2.2 Esquema global de la arquitectura 10
2.2.1 Securización servidor 11
3. Especificación de métodos y funciones API REST Python del servidor 12
3.1 InitSession (string user, string password) 12
3.2 SegundaValidacion (string usuario, int pin) 13
3.3 RefreshToken (string oldToken) 14
3.4 CloseSession (string token) 15
3.5 GetAmbiente (fecha datetime, string token) 14
3.6 GetEstadoSensor(int sensorId, string token) 15
3.7 ActivarSalida (int sensorId, activar bool, string token) 15
4.Estructura de base de datos 17
4.1 Relaciones tablas 17
5. Diseño interfaz gráfica del cliente **18**
5.1 Certificado 18
5.1 Formulario de inicio o login 18
7. Conclusiones **20**
8. Glosario **22**
9. Anexos 23
9.1 Manual de usuario 23
9.2 Manual de instalación 24
9.2.1 Servidor 24
9.2.2 Cliente 24
9.2.3 Creación de certificados servidor y cliente 25
9.3 Características de servidor 25
9.3.1 Sistema operativo 25
9.3.2 Instalación Raspian 26
9.4 Instalación Base de datos 26
9.4.1 Script para la base de datos ‘Domotic’ 26
9.5 Pruebas Json para tester de API REST Flask 28
9.6 Pruebas sénsorica 28
9.7 Ejecución script de inicio 28
9.8 Ejecución de IPTABLES 29
9.9 Fotos de la sensorica y accesorios utilizados 30
9.10 Funcionamiento de los componentes y del sistema 32

10. Bibliografía.. 32

Lista de figuras

Figura 1. Evolución de los dispositivos LoT……1

Figura 2. Diagrama de doble validación..……18

Figura 3. Arquitectura del sistema……..…19

Figura 4. Raspberry Pi 4……..31

Figura 5. Relé……..…….……..…31

Figura 6. Electroválvula..……..31

Figura 7. Sensor……..……..32

Figura 8. Flotador……..…32

1. Introducción

1.1 Contexto y justificación del Trabajo

Actualmente se vive en un mundo hiperconectado donde el Internet de las cosas Internet of Things (IoT) y/o la domótica tienen su espacio en el día a día en la vida de las personas y las empresas.

Móviles, tablets, TV con conectividad a Internet, lavadoras dotadas de wifi, sistemas de calefacción con control remoto son algunos de los ejemplos.

Normalmente estos dispositivos se centran más en ofrecer funcionalidad dejando en un segundo plano la seguridad.

El hecho de que estos elementos estén conectados a través del Internet expone a las redes domésticas o a las empresas de vulnerabilidades que pueden ser aprovechadas por ciberdelincuentes.

La tendencia es exponencial en el crecimiento de estos dispositivos año tras año (ver Figura 1).

Alguno de estos dispositivos puede poner en riesgo la seguridad sin que el usuario o la empresa, sea consciente de que, por otro lado, se está creando 'una puerta de entrada' para posibles acciones maliciosas. El robo de datos, Ransomware, BotNets entre otros.

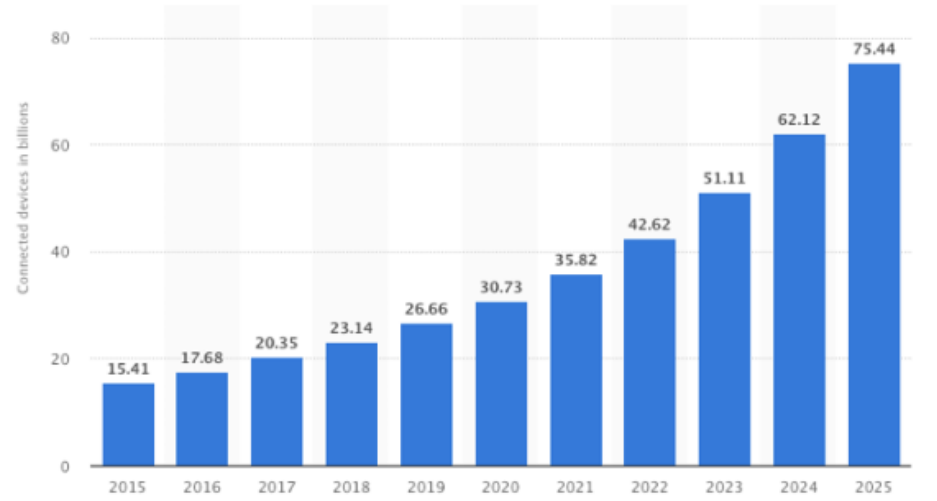


Figura 1.

En los entornos rurales la domótica poco a poco también va ganando terreno, ya es que es ideal para la automatización de la alimentación en granjas y también en la agricultura como el control de riego, sensórica etc.

En este tipo de entorno normalmente la domótica tarda más en aplicarse. Normalmente se realiza todo de manera manual (como se ha realizado durante mucho tiempo) o bien hay mecanismos manuales que hacen la funcionalidad requerida sin ofrecer nada más.

Es decir, son entornos normalmente no informatizados y en general es un sector lento en la transformación digital.

Algunas de empresas de domótica que trabajan en entornos rurales ofrecen una gran variedad de servicios y soluciones a medida, pero normalmente es de carácter industrial.

Permiten la posibilidad de configurar granjas totalmente domotizadas con sensores específicos según el tipo animal, pero están más enfocadas a la productividad y también dejan en un segundo plano la seguridad o bien lo delegan todo a empresas externas.

También hay que destacar que dichos servicios no están al alcance de todos los bolsillos y mucho menos a particulares o pequeños autónomos del sector.

En muchos ámbitos de la domótica se hace uso de las API REST [3] como métodos de autentificación ya que ofrece una gran versatilidad. Dentro de las API REST existen multitud de sistemas diferentes, quedando 4 o 5 como los más extendidos.[1]

Por ejemplo, es muy corriente sistemas domóticos basados autentificación básica HTTP, es decir el usuario y contraseña están codificados en Base64.

El problema de este método es que no requiere de cookies, ni ID de sesión y como la información viaja en el encabezado HTTP, no hay ningún sistema seguridad de tipo ‘handshake’.

Dicho sistema de autentificación es vulnerable ya que un atacante podría interceptar los datos a través de un sniffer, decodificar la contraseña y acceder al sistema.

También mencionar la autentificación por token (autenticación de portador) que va un poco más allá el sistema anterior comentado.

Ofrece una autentificación de tipo token que básicamente lo que hace es dar acceso al ‘portador’ de ese token, pero hay que tener en cuenta que si el token no se renueva o si un atacante conoce ese token podría suplantar la identidad del usuario.

Hay que destacar que, si las comunicaciones no son cifradas, con un sniffer se podría interceptar el token y acceder al sistema.

Las claves API es otro recurso muy utilizado en el mundo industrial, pero no es aconsejable como un sistema robusto en seguridad.

El principal problema de este sistema es que muchas veces estas claves API viajan como parte de la URL, lo cual hace que sea relativamente fácil de identificar la clave para un usuario con intenciones maliciosas.

También existe la opción OAuth 2.0 que proporciona flujos de autorización específicos para aplicaciones de web o de escritorio.

En este sistema se utiliza tokens específicos para acceder a diferentes recursos que tiene asignado un usuario. También es más compleja su implementación y mantenimiento y su uso no está muy extendido en el mundo de la domótica.

Dicho sistema es ampliamente utilizado por ejemplo por Microsoft, Google, Facebook etc.

El enfoque de este proyecto se centra en la creación de un software servidor-cliente que utilice API REST con json en las comunicaciones y genere una doble validación para acceder al sistema, además de utilizar un token válido para cada funcionalidad.

Con la doble validación en el inicio de sesión, evita que un usuario no autorizado, aunque disponga de usuario / contraseña correcta del sistema, no pueda acceder si no ha recibido vía mail el pin de 4 dígitos generado aleatoriamente por el servidor.

También los métodos que están implementados en el servidor necesitan de un token asociado al usuario que ha iniciado sesión de manera correcta.

Se trata de asegurar que los sistemas interconectados no puedan ser manipulados por terceros o bien no se pueda acceder si no se tiene éxito en la doble validación además del uso de tokens en cada una de sus funciones. A todo esto, se le añade la caducidad de la sesión del usuario por inactividad.

1.2 Objetivos del Trabajo

El objetivo principal al que se quiere llegar una vez finalizado el proyecto, es crear un software securizado que esté funcionando en un microordenador Raspberry Pi y poder ser utilizado en un entorno de producción donde haya elementos de dómotica.

Se clasifican en dos objetivos:

Servidor

- Funcionamiento en un microrordenador (Raspberry Pi) [2]. La portabilidad es un elemento clave ya que permite que se pueda utilizar en diversos entornos.
- Validación doble para entrar al sistema: usuario / contraseña más pin recibido por mail.
- Utilización de tokens en las funciones implementadas en el servidor.
- Comunicaciones cifradas entre cliente – servidor uso de certificado electrónico.
- Conectividad con la sensórica con el servidor.
- Securización sistema operativo mediante IPTABLES.
- Conectividad para administración del servidor con SSH securizado.
- Escalabilidad para próximas ampliaciones.
- Almacenamiento de claves cifradas en base de datos.
- Arquitectura API REST con json para el intercambio de datos.

Para comprobar todas las funcionalidades del servidor se realiza un software que correspondería con la parte cliente.

Cliente

- Formulario en login, usuario y contraseña. Primera validación.
- Formulario de PIN. Pin recibido por mail. Segunda validación
- Formulario principal donde muestre los datos del sensor y activación salida una vez validado en el sistema.

1.3 Enfoque y método seguido

Para poder realizar el proyecto se ha desglosado en fases:

Fase de investigación y requisitos
En esta fase se busca información de cuáles son las arquitecturas más utilizadas en la domótica y cuáles ofrecen la mejor alternativa tanto desde un punto de vista de seguridad como funcional.

El primer paso para poder desarrollar el producto es buscar información acerca de la arquitectura utilizada para el envío de datos cuando el cliente vaya a consultarlos. Se utiliza json como formato de texto sencillo para dicho empeño.

También hay una fase de conocimiento de sensórica para Raspberry Pi.

Fase de desarrollo
Una vez se tiene claro el sistema de envío datos y la manera en que el usuario debe logarse, se desarrolla la parte servidora con todas las funcionalidades, además del software cliente para darle un contexto al proyecto.

Fase de pruebas
Cuando ya se dispone de las primeras versiones del software, se realizan pruebas para verificar que el sistema hace lo que se pretende, además de la identificación de errores.

Puesta en producción
Una vez estén validadas las pruebas en la fase de pruebas se pasa a producción.

1.4 Planificación del proyecto

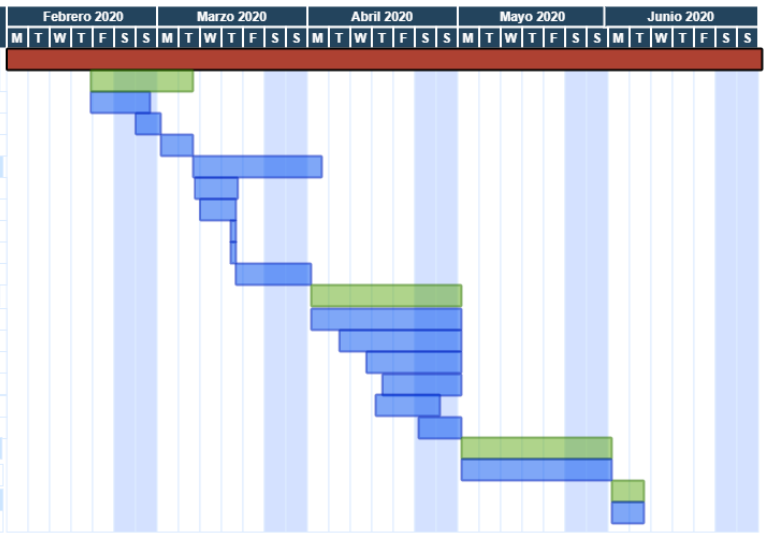
Se proyecto se distribuye en unos 4 meses aproximadamente. La planificación queda ordenada de la siguiente manera:

TAREAS	Inicio	Fin
PAC1	19.02.20	11.03.20
Proceso de investigación	19.02.20	25.02.20
Definición de tareas	25.02.20	01.03.20
Planificación del proyecto	02.03.20	11.03.20
PAC2	12.03.20	05.04.20
Análisis de requisitos	12.03.20	14.03.20
Especificación funciones del servidor	14.03.20	23.03.20
Estructura y creación de base de datos	20.03.20	23.03.20
Casos de uso	23.03.20	28.03.20
Diseño arquitectura del sistema	24.03.20	05.04.20
PAC3	06.04.20	03.05.20
Desarrollo código fuente servidor	06.04.20	03.05.20
Pruebas realizadas servidor	17.05.20	03.05.20
Desarrollo código fuente cliente	15.04.20	03.05.20
Pruebas cliente	20.04.20	03.05.20
Securización sistema operativo	12.04.20	20.05.20
Primer borrador de la memoria resultante	30.04.20	03.05.20
PAC4 - MEMORIA	04.05.20	03.06.20
Memoria resultante del proyecto	04.05.20	03.06.20
PRESENTACIÓN	04.06.20	11.06.20
Crear y preparar presentación		

Tabla de descripción de tareas

Diagrama de Gantt

	Nombre Tarea	Inicio	Fin
1	Todo el proyecto	19.02.20	11.06.20
2	**PAC1**	19.02.20	11.03.20
3	Proceso de investigación	19.02.20	25.02.20
4	Definición de tareas	25.02.20	01.03.20
5	Planificación proyecto	02.03.20	05.04.20
6	**PAC2**	12.03.20	05.04.20
7	Análisis de requisitos	12.03.20	14.03.20
8	Especificación funciones del servidor	14.03.20	23.03.20
9	Estructura base de datos	20.03.20	23.03.20
10	Casos de uso	23.03.20	28.03.20
11	Diseño arquitectura del sistema	24.03.20	05.04.20
12	**PAC3**	06.04.20	03.05.20
13	Desarrollo código fuente servidor	06.04.20	03.05.20
14	Pruebas realizadas servidor	17.03.20	03.05.20
15	Desarrollo código fuente cliente	15.04.20	03.05.20
16	Pruebas cliente	20.04.20	03.05.20
17	Securización sistema operativo	12.04.20	20.05.20
18	Primer borrador de la memoria resultante	30.04.20	03.05.20
18	**PAC4 - Memoria**	04.05.20	03.06.20
19	Memoria resultante del proyecto	04.05.20	03.06.20
20	**Presentación**	04.06.20	11.06.20
21	Preparar la memoria	04.06.20	11.06.20



1.5 Breve sumario de productos obtenidos

A continuación, se describe de manera muy resumida los productos obtenidos.

Servidor

- ➔ Funcionamiento correcto de la aplicación. Doble validación y control de funciones por token.
- ➔ Comunicación cifrada entre servidor y cliente.
- ➔ Lectura y activación de elementos conectados.

Cliente

- ➔ Aplicación que compruebe la doble validación que requiere el servidor.
- ➔ Ejecución correcta de las funcionalidades utilizando el token que generado en la sesión.

Securización sistema operativo

- ➔ Protección mediante reglas IPTABLES. Solo el tráfico estrictamente necesario.
- ➔ Administración a través de SSH securizado.

1.6 Breve descripción de los otros capítulos de la memoria

Funcionamiento de la aplicación y arquitectura
Muestra el funcionamiento de la aplicación securizada del servidor y la arquitectura de todo el sistema. Ver capítulo 2

Especificación de métodos y funciones API REST del servidor
Se describe los parámetros de entrada de cada método. Ver capítulo 3

Estructura de base de datos
Se muestra un esquema de la base de datos. Ver capítulo 4.

Diseño interfaz gráfica del cliente
Muestra el diseño de la parte cliente. Los ormularios que se utilizan. Ver capítulo 5.

2. Funcionamiento software y arquitectura de la aplicación

2.1 Servidor

La aplicación está desarrollada en Python y utiliza Flask como servidor web y json para el intercambio de datos.

2.1.1 Validación

Requisito previo.

Los usuarios han de dados de alta en la base de datos con la clave cifrada, es decir se almacena la contraseña en MD5.

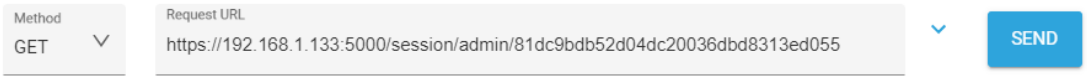
IdUsuario	Usuario	Password	Activo	Mail
1	admin	81dc9bdb52d04dc20036dbd8313ed055	1	jlopezcas@uoc.edu

Ejemplo de la tabla TUsuario con la contraseña cifrada.

Funcionalidades de la validación

1. Verificación de usuario en base de datos.
2. Comprobación de la contraseña en MD5
3. Comprobación que no exista previamente un token asociado a una sesión con el mismo usuario.

Una vez el usuario inicia sesión en el formulario, el servidor llama al método ‘InitSession’ y le pasa como parámetro el usuario y la contraseña cifrada que recoge del formulario.



Muestra de datos enviados con el software ARC

El software genera un ‘Operation Result’ que es un identificador de cómo ha ido, si devuelve 0 ha ido bien (ver Especificación de métodos y funciones, capítulo 3).


```
{
  -"Resultado": [Array[1]
     -0:  {
         "Message": "login",
         "OperationCode": 0
      }
   ],
}
```


Resultado de OperationCode

Si el resultado es positivo, genera un número aleatorio de 4 cifras y se lo envía al mail del usuario. (cada usuario tiene su mail que está almacenado en un campo de la base de datos).

El PIN generado es: 7365

Ejemplo del contenido del mail

El usuario recibe el mail con el pin y lo introduce en el formulario de PIN.

En ese momento se llama a la función ‘SegundaValidacion’ que será la encargada de verificar y generar el token correspondiente.

Si es correcto el sistema generará un token asociado a ese usuario y lo almacena en base de datos.


```
{
  -"Resultado": [Array[1]
     -0:  {
         "Message": "pin",
         "OperationCode": 0,
         "token": "89ad9b7d4aa1468ba9715579a1d3a4d0"
      }
   ],
}
```


Token generado en una sesión

A partir del momento que el usuario autorizado dispone de token, ya puede entrar en el sistema.

En la base de datos se almacena en su tabla correspondiente los siguientes valores.

IdSession	IdUsuario	Token	Fecha	Ip
544	1	f55e4629666a42b2810f1d8c26257e7e	2020-06-03 19:12:01	192.168.1.158

El diagrama de la doble validación quedaría de la siguiente manera:

Figura 2. Diagrama de doble validación

2.1.2 Tokens en las funciones

Todas las funciones que se consultan desde fuera del mismo servidor van con token asociado a la sesión.

Temperatura y humedad
Por ejemplo, cuando se solicita la temperatura y humedad actual los parámetros son la fecha del momento y el token.

```
@app.route('/ambiente/<fecha>/<token>', methods=['GET'])
```

Ejemplo de llamada en Flask

Activación de relé
Por ejemplo, el usuario haga clic en ‘Activar salida’, lo que hace internamente el servidor es con el token de ese usuario verificar si existe en base de datos, si existe realizará la orden de lo contrario no será posible.

```
@app.route('/salidas/<sensorId>/<activar>/<token>', methods=['GET'])
```

Por defecto una sesión dura 20 minutos, si el servidor detecta actividad de ese token, actualiza en base de datos el valor del tiempo actual y vuelve a esperar 20 minutos de inactividad.
De esta manera los **tokens no pueden ser reutilizados.**

2.2 Esquema global de la arquitectura

Para proteger el servidor, éste lee después de cada reinicio, reglas IPTABLES donde se filtra todos los paquetes menos los que son para SSH y para servicio que corre en el servidor.

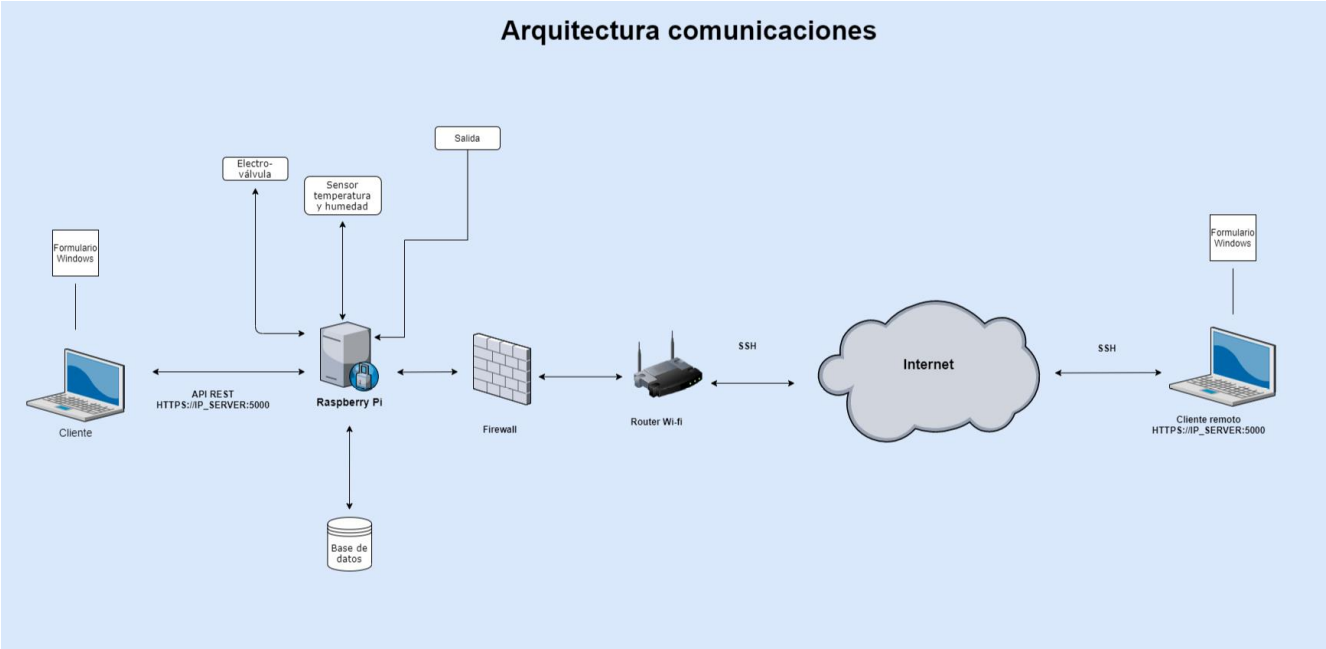


Figura3. Esquema de la arquitectura

2.2.1 Securización servidor

Características IPTABLES

- ➔ Descartar cualquier paquete TCP que no se ha iniciado con el Flag SYN activo.
- ➔ Descartar cualquier paquete fragmentado.
- ➔ Prevenir ataques de DoS eliminando paquetes sin ningún flag tcp activado (null packets).
- ➔ Eliminar cualquier paquete inválido que no pueda ser identificado.
- ➔ Eliminar cualquier paquete con todos los flags TPC activados.
- ➔ Ping limitado a 5 mínutos.
- ➔ Solo deja pasar el tráfico por el puerto 5000 (Flask) y el puerto 899 (SSH).
- ➔ Limitación de conexiones por minuto para prevenir ataque DoS

Características SSH

- ➔ Cambio de Puerto por defecto (22) al 899.
- ➔ No permitir conectarse como root.
- ➔ Número máximo de intentos 3.
- ➔ Tiempo máximo de login limitado a 60 segundos.
- ➔ No permitir login con password, se utiliza clave pública -privada

Ejecución de HTTPS y certificados

A la hora de trabajar con API REST y json es fundamental cifrar las conexiones, ya que, con un sniffer de red, se podría ver por ejemplo el token que tiene asignado el usuario. Para solucionar este tema se generan certificados en el servidor y cliente.

```
app.run(ssl_context=('cert.pem', 'key.pem'),debug=False, port=5000, host='0.0.0.0')
```

Creación de un certificado autofirmado en el servidor con **openssl**, posteriormente dicho certificado debe ser utilizado por el cliente para garantizar la confidencialidad.

3. Especificación de métodos y funciones API REST Python del servidor

Especificación métodos y funciones API REST implementados en el servidor.

3.1 InitSession (string user, string password)

Función para iniciar sesión.

InitSession (string user, string password)		
Parámetro	**Tipo**	**Descripción**
user	string	El nombre del usuario
password	string	El password del usuario cifrado
Valor de retorno		
Un objeto del tipo [OperationResultSession] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
OperationResultCode	int	Un entero con el resultado de la operación. 0=Ok. <> 0 Error. Ver tabla debajo
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista
Token	string	Cadena de texto alfanumérica que identifica la sesión. Deberá usarse en todas las operaciones siguientes. El token tiene una duración limitada en el tiempo. Cuando el token caduque hay que renovar sesión con esta función.

Códigos de error de OperationResultCode	
Valor	**Descripción**
0	Ok. Operación realizada correctamente
1	Usuario o password incorrectos
2	La ip desde donde se está conectando no tiene permiso para hacer login.
-1	Ocurrió algún error inesperado

3.2 SegundaValidacion (string usuario, int pin)

Función que se encarga de generar el token.

RefreshToken(string oldToken)		
Parámetro	**Tipo**	**Descripción**
Usuario	string	El valor del token actual. Se supone que antes de que haya caducado.
Pin	Int	Pin recibido por mail.
Valor de retorno Un objeto del tipo [OperationResultSession] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
OperationResult Code	int	Un entero con el resultado de la operación. 0=Ok. <>0 Error. Ver tabla debajo
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista
Token	string	Cadena de texto alfanumérica que identifica la sesión. Deberá usarse en todas las operaciones siguientes. El token tiene una duración limitada en el tiempo por inactividad.

Códigos de error de OperationResultCode	
Valor	Descripción
0	Ok. Operación realizada correctamente
1	Token incorrecto o caducado. No existe ninguna sesión con el token proporcionado
2	La ip desde donde se está conectando no tiene permiso para hacer login.
-1	Ocurrió algún error inesperado

3.3 RefreshToken (string oldToken)

Función que se encarga de refrescar el token.

RefreshToken(string oldToken)		
Parámetro	**Tipo**	**Descripción**
oldToken	string	El valor del token actual. Se supone que antes de que haya caducado.
Valor de retorno Un objeto del tipo [OperationResultSession] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
OperationResultCode	int	Un entero con el resultado de la operación. 0=Ok. <>0 Error. Ver tabla debajo
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista
Token	string	Cadena de texto alfanumérica que identifica el token refrescado. Deberá usarse en todas las operaciones siguientes.

Códigos de error de OperationResultCode	
Valor	Descripción
0	Ok. Operación realizada correctamente
1	Token incorrecto o caducado. No existe ninguna sesión con el token proporcionado
2	La ip desde donde se está conectando no tiene permiso para hacer login.
-1	Ocurrió algún error inesperado

3.5 GetAmbiente (fecha datetime, string token)

Ofrece la temperatura actual.

GetAmbiente(fecha datetime, string token)		
fecha	Datetime	Fecha y hora en el momento de la petición.
token	string	El token asociado a la sesión actual
Valor de retorno		
Un objeto del tipo [OperationResultAmbiente] compuesto el siguiente campo		
Parámetro	**Tipo**	**Descripción**
Temperatura	float	Un decimal con el valor de la temperatura actual.
Humedad	float	Un decimal con el valor de la humedad actual.

Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista.

Códigos de error de OperationResultAmbiente	
Valor	**Descripción**
0	Ok. Operación realizada correctamente.
1	El usuario no dispone de permisos.
2	No existe la sesión con el token especificado o bien la sesión ha caducado.
-1	Ocurrió algún error inesperado.

3.6 GetEstadoSensor(int sensorId, string token)

Función que devuelve el estado del sensor pasado como parámetro

GetEstadoSensor(int sensorId, string token)		
Parámetro	**Tipo**	**Descripción**
sensorId	int	El sensor asociado a la su Id.
token	string	El token asociado a la sesión actual
Valor de retorno Un objeto del tipo [OperationResultSensor] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
Estado	int	Descripción del estado (0 NO Ok , 1 Ok)
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista.

Códigos de error de OperationResultSensor	
Valor	**Descripción**
0	Ok. Operación realizada correctamente.
1	El usuario no dispone de permisos.
2	No existe la sesión con el token especificado o bien la sesión ha caducado.
-1	Ocurrió algún error inesperado.

3.4 CloseSession (string token)

Cierra la sesión especificada por el token proporcionado. Si no se llama a este método, la sesión se cerrará automáticamente cuando caduque el token.

CloseSession(string token)		
Parámetro	**Tipo**	**Descripción**
token	string	El token asociado a la sesión actual
Valor de retorno Un objeto del tipo [OperationResultSession] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
OperationResultCode	int	Un entero con el resultado de la operación. 0=Ok. <> 0 Error. Ver tabla debajo.
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista
Token	string	Cadena de texto alfanumérica que identifica la sesión. Deberá usarse en todas las operaciones siguientes. El token tiene una duración limitada en el tiempo. Cuando el token caduque hay que renovar sesión con esta función.

Códigos de error de OperationResultCode	
Valor	**Descripción**
0	Ok. Operación realizada correctamente.
1	El usuario no dispone de permisos.
2	No existe la sesión con el token especificado o bien la sesión ha caducado.
-1	Ocurrió algún error inesperado.

3.7 ActivarSalida (int sensorId, activar bool, string token)

Función que activa o desactiva una salida.

GetEstadoSensor(int sensorId, string token)		
Parámetro	**Tipo**	**Descripción**
sensorId	int	El sensor asociado a la su Id.
activar	bool	Activa o desactiva la salida asociada al sensorId.
token	string	El token asociado a la sesión actual.
Valor de retorno		
Un objeto del tipo [OperationResultSalida] compuesto por los siguientes campos:		
Parámetro	**Tipo**	**Descripción**
Estado	string	Descripción del estado.
Message	string	Un mensaje que describe el error en caso de que haya ocurrido alguno y proporciona información adicional en caso de que exista.

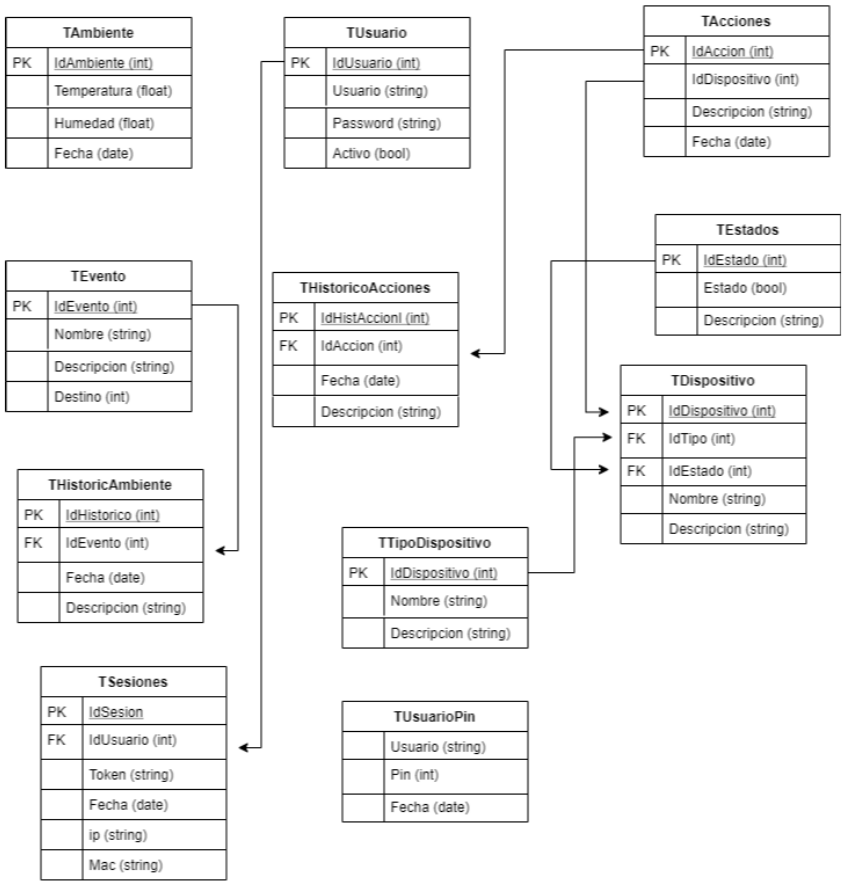
Códigos de error de OperationResultSalida	
Valor	**Descripción**
0	Ok. Operación realizada correctamente.
1	El usuario no dispone de permisos.
2	No existe la sesión con el token especificado o bien la sesión ha caducado.
-1	Ocurrió algún error inesperado.

4.Estructura de base de datos

La base de datos estará en el mismo servidor. Debe de ser una base de datos relacional.

4.1 Relaciones tablas

Relación de tablas de la base de datos DOMOTIC



5. Diseño interfaz gráfica del cliente

La parte cliente está compuesta de tres formularios: uno inicial que pide usuario y contraseña, otro que se muestra para la introducción del pin recibido vía mail y el tercero sería el formulario principal.

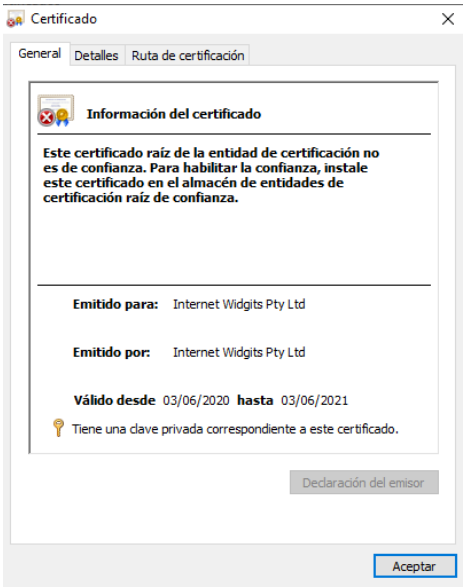
La aplicación funciona bajo Windows debe funcionar con el certificado correspondiente.

5.1 Certificado

Para que el cliente pueda ser ejecutado de manera correcta, se necesita un certificado.

En el directorio correspondiente hay un fichero de tipo pfx para que pueda funcionar la aplicación.

En este caso el certificado es auto firmado (ver capítulos anexos – creación de certificados servidor y cliente).



5.1 Formulario de inicio o login

Una vez se ejecuta el cliente, muestra un formulario con usuario y contraseña y carga el certificado.

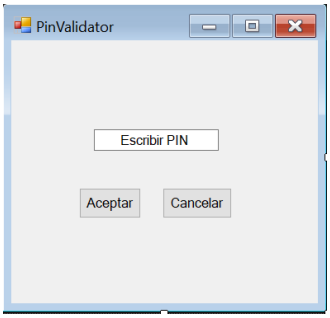
Posteriormente si el usuario y la contraseña es correcta, mostrará el formulario de pin.

Cuando el usuario introduce las credenciales válidas, automáticamente mostrará el formulario de validación de pin:

Previamente se ha de recibir un mail con el pin válido.

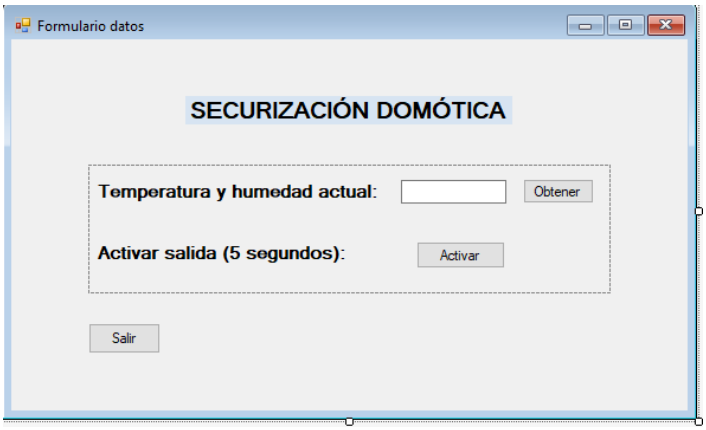
El PIN generado es: 9444

Ejemplo de pin recibido por mail



Formulario de validación de pin

Luego ya se puede acceder al formulario principal.



Funcionamiento

Cuando se hace clic en el botón ‘Obtener’ o ‘Activar’ internamente el servidor se conecta con el sensor y realiza dos tareas.

1. Comprueba si el token está autorizado
2. Refresca el valor de caducidad de la sesión.

El cliente ya puede interactuar gracias al token que le ha asignado el servidor.

7. Conclusiones

Las conclusiones que llego con este proyecto es que la seguridad perfecta no existe a la hora de proteger un sistema. Siempre que se apueste por un método de autentificación, éste llevará implícito algún punto débil.

También es importante tener presente la información que ofrece el servicio. No es lo mismo securizar datos de un sensor que los datos de un banco.

Luego es importante destacar que hay muchas maneras de autentificarse en una aplicación.

A priori se han conseguido bastantes objetivos planteados desde un principio del proyecto, pero no todos por falta de tiempo.

Para darle un poco más de contexto al proyecto e intentar también que sea lo más pragmático posible se le han añadido elementos de sensórica para simular un entorno de producción, el problema es que son pocos los elementos añadidos por falta de tiempo.

Remarcar que al añadir elementos de sensórica ha complicado bastante más el proyecto e incluso se pierde o se puede llegar a perder el enfoque inicial que en este caso es de seguridad.

Al querer añadir elementos domóticos por desconocimiento de electrónica, se ha tenido que invertir tiempo en el estudio de su funcionamiento. También Python ha supuesto un reto ya que desconocía el lenguaje.

Ha sido bastante complicado seguir la planificación porqué no he tenido una visión clara del objetivo hasta que no he ido comprendiendo exactamente a donde quería llegar.

En una primera fase se inició el proyecto con demasiada ambición, luego se tuvo que ir reduciendo funcionalidades para poder cuadrarlo lo más posible.

En la última fase se ha tenido que introducir más elementos relacionados con la seguridad, para darle más valor al proyecto.

El resultado general es satisfactorio, pero han faltado pruebas de rendimiento (10 o más usuarios conectados al servidor) y poder ampliar el sistema y poder interactuar más con los elementos conectados.

8. Glosario

Internet of Things (IoT): El internet de las cosas es un concepto que se refiere a una interconexión digital de objetos cotidianos con internet. Es, en definitiva, la conexión de internet más con objetos que con personas. También se suele conocer como internet de todas las cosas.

API REST: Es una interfaz de programación de aplicaciones que se apoya en la arquitectura REST para el desarrollo de aplicaciones en red. Aprovechando el lenguaje HTML, permite que cualquier empresa cree aplicaciones web sin problemas, aunque siempre en base a las restricciones que supone.

Ransomware: Un ransomware, o "secuestro de datos" en español, es un tipo de programa dañino que restringe el acceso a determinadas partes o archivos del sistema operativo infectado y pide un rescate a cambio de quitar esta restricción

Botnet: Botnet es un término que hace referencia a un conjunto o red de robots informáticos o bots, que se ejecutan de manera autónoma y automática. El artífice de la botnet puede controlar todos los ordenadores/servidores infectados de forma remota.

Base 64: Es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia que puede ser representada usando únicamente los caracteres imprimibles de ASCII.

Cookie: El anglicismo cookie, usado también galleta o galleta informática, es un término que hace referencia a una pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del navegador.

Token: Identificador único en un sistema o proceso.

Handshake: el proceso de intercambio de información privada. Cuando un usuario entra en una página web para intercambiar información y ésta es a su vez privada, con SSL o protocolo https, se ejecuta un proceso llamado popularmente como **Handshake**, o apretón de manos.

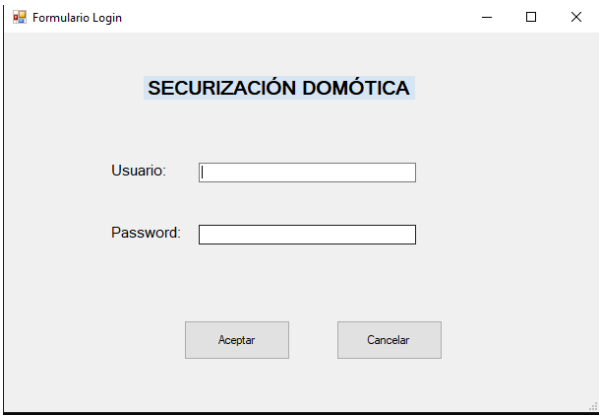
OAuth: Open Authorization es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Se trata de un protocolo propuesto por Blaine Cook y Chris Messina, que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.

Json: JSON (acrónimo de **JavaScript Object Notation**, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos.

9. Anexos

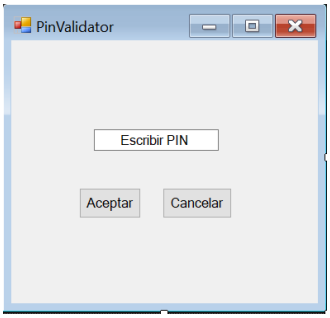
9.1 Manual de usuario

Ejecutar el fichero: domotic.exe ubicado en c:\Domotic

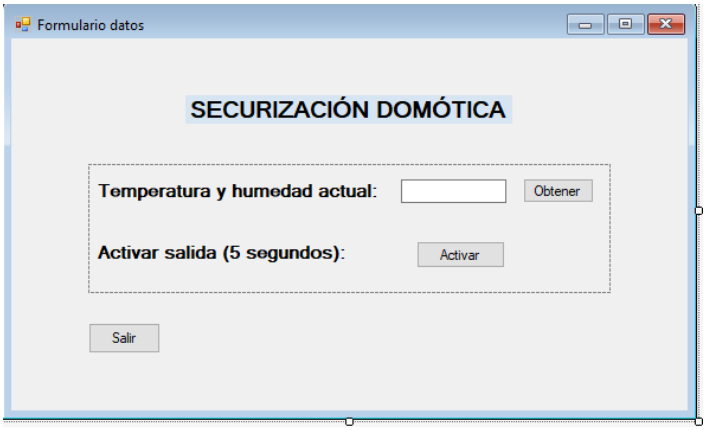


Mostrará un formulario donde pide usuario y contraseña

Posteriormente se mostrará el formulario de validación de pin.



Si todo ha ido correctamente, se accede al formulario principal:



Temperatura y humedad actual: lectura de los valores en tiempo real

Activar salida (5 segundos): Activa el relé y deja pasar el agua que hay en el depósito.

9.2 Manual de instalación

9.2.1 Servidor

Requisitos: Python y MariaDB instalados

1. Instalar Python en el sistema y MariaDB
2. Ejecutar script SQL
3. Iniciar Mysql como root
4. Crear el usuario sql:
    a. CREATE USER 'nombre_usuario'@'localhost' IDENTIFIED BY 'contrasena';
    b. GRANT ALL PRIVILEGES ON * . * TO 'nombre_usuario'@'localhost';
    c. FLUSH PRIVILEGES;
5. Editar 'Domotic.py' para la conexión de la base de datos.

```
db_connect = create_engine('mysql+pymysql://usuario_bbdd:contraseña@localhost:3306/DOMOTIC')
```

6. Edición del servidor de mail. En el proyecto se utiliza el servidor smtp de Google.

```
mailDestino = query_select(mail)
# create message object instance
msg = MIMEMultipart()

message = "El PIN generado es: %d" % (numeroEntero)

# setup the parameters of the message
password = "password"
msg['From'] = "email de gmail"
msg['To'] = mailDestino
msg['Subject'] = "Autentificacion DOMOTIC."
# add in the message body
msg.attach(MIMEText(message, 'plain'))
#create server
server = smtplib.SMTP('smtp.gmail.com: 587')
server.starttls()
# Login Credentials for sending the mail
server.login(msg['From'], password)
# send the message via the server.
server.sendmail(msg['From'], msg['To'], msg.as_string())
server.quit()
```

7. Crear un hash md5 con el password que se vaya a utilizar
8. Editar la tabla TUsuarios y poner usuario, password cifrado activo = 1 y el mail
9. Iniciar terminal y ejecutar: **python carpeta_donde_esta_el_fichero/domotic.py**

9.2.2 Cliente

Nota: La aplicación es para Windows

1. Crear la carpeta en c:\Domotic.
2. Copiar los binarios dentro de la carpeta.
3. El fichero domotic.pfx debe estar dentro de la carpeta creada anteriormente.

9.2.3 Creación de certificados servidor y cliente

1. Abrir terminal en Linux y como root ejecutar el siguiente comando:

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days
365
```

Generará dos ficheros: **cert.pem** y **key.pem** válidos 365 días.

Dichos ficheros deben estar en la misma carpeta que el fichero Domotic.py

Una vez están los ficheros en el servidor se han de convertir de .pem a .pfx

Para poder trabajar con el cliente, se debe generar un fichero pfx para que el cliente lo utilice.

```
openssl pkcs12 -inkey ruta_fichero_cert.pem -in ruta_fichero_key.pem -export
-out ruta_fichero_generado.pfx
```

Posteriormente copiar el fichero generado *.pfx al directorio c:\Domotic

9.3 Características de servidor

El proyecto se basa en un microordenador, ya que está enfocado para un entorno rural. Debe ser un servidor que sea lo suficientemente potente para poder ejecutar tareas no pesadas y sobre todo que sea portable.

Normalmente el servidor a de colocarse en algún lugar cerca de la sensórica y debidamente protegido contra condiciones adversas.

9.3.1 Sistema operativo

Se utiliza Raspbian por ser el software oficial de Raspberry Pi que está basada en Debian además de ser la más optimizada.

Hay bastantes distribuciones que funcionan para Raspberry Pi. Entre ellas: Kano OS, Flint OS, pipaOS etc. Algunas de ellas más ligeras y con menos partes de escritorio también hay que destacar que existe una versión de Kali Linux para hacer pruebas de intrusión.

9.3.2 Instalación Raspian

Instalación de la última versión del sistema operativo Raspian y se graba mediante la herramienta proporcionada en la web oficial (Raspberry Pi Imager).

Dicha herramienta se utiliza para poder grabar la imagen previamente descargada y genera la imagen en la tarjeta micro sd.

Es aconsejable utilizar el mismo software proporcionado por Raspian.

9.4 Instalación Base de datos

Se instala la base de datos MariaDb versión mysql Versión 15.1 Distrib 10.3.22-MariaDB, para debian-linux-gnueabihf (armv8l). MariaDb cumple con perfectamente con los requisitos por ser una base de datos relacional y es gratuita.

- ➔ Instalación Base de datos MariaDb a través del comando y como root (apt-get install mariadb-server mariadb-client)
- ➔ Crear base de datos 'DOMOTIC'
- ➔ Crear usuario de conexión a base de datos.
- ➔ Crear tablas correspondientes (ver apartado 4. Estructura de base de datos)

9.4.1 Script para la base de datos 'Domotic'

Ejecutar fichero SQL en servidor que contenga la base de datos.

Ver apartado Manual de instalación para permisos de usuario sobre la base de datos.

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!50503 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

-- Volcando estructura de base de datos para DOMOTIC
DROP DATABASE IF EXISTS `DOMOTIC`;
CREATE DATABASE IF NOT EXISTS `DOMOTIC` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
USE `DOMOTIC`;

-- Volcando estructura para tabla DOMOTIC.TAmbiente
DROP TABLE IF EXISTS `TAmbiente`;
CREATE TABLE IF NOT EXISTS `TAmbiente` (
  `IdAmbiente` int(11) NOT NULL AUTO_INCREMENT,
  `Temperatura` float DEFAULT NULL,
  `Humedad` float DEFAULT NULL,
  `Fecha` datetime DEFAULT NULL,
  PRIMARY KEY (`IdAmbiente`),
  KEY `IdAmbiente` (`IdAmbiente`)
) ENGINE=InnoDB AUTO_INCREMENT=19846 DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.TAmbiente: ~113 rows (aproximadamente)
/*!40000 ALTER TABLE `TAmbiente` DISABLE KEYS */;
/*!40000 ALTER TABLE `TAmbiente` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.TEstados
DROP TABLE IF EXISTS `TEstados`;
CREATE TABLE IF NOT EXISTS `TEstados` (
  `Sensor` tinyint(4) NOT NULL DEFAULT 0,
  `Descripcion` text DEFAULT NULL,
  `Estado` tinyint(1) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.TEstados: ~0 rows (aproximadamente)
```

```
/*!40000 ALTER TABLE `TEstados` DISABLE KEYS */;
/*!40000 ALTER TABLE `TEstados` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.TEvento
DROP TABLE IF EXISTS `TEvento`;
CREATE TABLE IF NOT EXISTS `TEvento` (
  `IdEvento` int(11) NOT NULL,
  `Nombre` text DEFAULT NULL,
  `Descripcion` text DEFAULT NULL,
  PRIMARY KEY (`IdEvento`),
  KEY `IdEvento` (`IdEvento`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;


-- Volcando estructura para tabla DOMOTIC.THistoricoAcciones
DROP TABLE IF EXISTS `THistoricoAcciones`;
CREATE TABLE IF NOT EXISTS `THistoricoAcciones` (
  `IdHistAccion` int(11) NOT NULL,
  `IdAccion` int(11) NOT NULL,
  PRIMARY KEY (`IdHistAccion`),
  KEY `IdHistAccion` (`IdHistAccion`),
  KEY `IdAccionFK` (`IdAccion`),
  CONSTRAINT `IdAccionFK` FOREIGN KEY (`IdAccion`) REFERENCES `TAcciones` (`IdAccion`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.THistoricoAcciones: ~0 rows (aproximadamente)
/*!40000 ALTER TABLE `THistoricoAcciones` DISABLE KEYS */;
/*!40000 ALTER TABLE `THistoricoAcciones` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.THistoricoEventos
DROP TABLE IF EXISTS `THistoricoEventos`;
CREATE TABLE IF NOT EXISTS `THistoricoEventos` (
  `IdHistorico` int(11) NOT NULL AUTO_INCREMENT,
  `Fecha` datetime DEFAULT NULL,
  `Descripcion` text DEFAULT NULL,
  PRIMARY KEY (`IdHistorico`),
  KEY `IdHistorico` (`IdHistorico`)
) ENGINE=InnoDB AUTO_INCREMENT=3538 DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.THistoricoEventos: ~270 rows (aproximadamente)
/*!40000 ALTER TABLE `THistoricoEventos` DISABLE KEYS */;
/*!40000 ALTER TABLE `THistoricoEventos` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.TSesiones
DROP TABLE IF EXISTS `TSesiones`;
CREATE TABLE IF NOT EXISTS `TSesiones` (
  `IdSession` int(11) NOT NULL AUTO_INCREMENT,
  `IdUsuario` int(11) NOT NULL,
  `Token` tinytext NOT NULL,
  `Fecha` datetime NOT NULL,
  `Ip` text NOT NULL,
  `Mac` text NOT NULL,
  PRIMARY KEY (`IdSession`),
  KEY `IdSession` (`IdSession`),
  KEY `IdUsuarioFK` (`IdUsuario`),
  CONSTRAINT `IdUsuarioFK` FOREIGN KEY (`IdUsuario`) REFERENCES `TUsuario` (`IdUsuario`)
) ENGINE=InnoDB AUTO_INCREMENT=521 DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.TSesiones: ~0 rows (aproximadamente)
/*!40000 ALTER TABLE `TSesiones` DISABLE KEYS */;
/*!40000 ALTER TABLE `TSesiones` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.TTipoDispositivo
DROP TABLE IF EXISTS `TTipoDispositivo`;
CREATE TABLE IF NOT EXISTS `TTipoDispositivo` (
  `IdDispositivo` int(11) NOT NULL,
  `Nombre` text DEFAULT NULL,
  `Descripcion` text DEFAULT NULL,
  PRIMARY KEY (`IdDispositivo`),
  KEY `IdDispositivo` (`IdDispositivo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Volcando estructura para tabla DOMOTIC.TUsuario
DROP TABLE IF EXISTS `TUsuario`;
CREATE TABLE IF NOT EXISTS `TUsuario` (
  `IdUsuario` int(11) NOT NULL,
```

```
  `Usuario` tinytext DEFAULT NULL,
  `Password` tinytext DEFAULT NULL,
  `Activo` int(1) DEFAULT NULL,
  `Mail` tinytext DEFAULT NULL,
  PRIMARY KEY (`IdUsuario`),
  KEY `IdUsuario` (`IdUsuario`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.TUsuario: ~2 rows (aproximadamente)
/*!40000 ALTER TABLE `TUsuario` DISABLE KEYS */;
/*!40000 ALTER TABLE `TUsuario` ENABLE KEYS */;

-- Volcando estructura para tabla DOMOTIC.TUsuarioPin
DROP TABLE IF EXISTS `TUsuarioPin`;
CREATE TABLE IF NOT EXISTS `TUsuarioPin` (
  `Usuario` tinytext DEFAULT NULL,
  `Pin` int(4) DEFAULT NULL,
  `Fecha` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Volcando datos para la tabla DOMOTIC.TUsuarioPin: ~4 rows (aproximadamente)
/*!40000 ALTER TABLE `TUsuarioPin` DISABLE KEYS */;
/*!40000 ALTER TABLE `TUsuarioPin` ENABLE KEYS */;

/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
@OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

9.5 Pruebas Json para tester de API REST Flask

Pruebas con ARC Advanced para probar Json (versión 15.0.0 beta) generado por el servidor.

Dicho software trabaja con htttps que es fundamental para el proyecto.

Es un software gratuito que permite efectuar operaciones GET, PUT, POST, DELETE etc. Json es un formato muy utilizado hoy y ofrece una gran versatilidad. Es rápido ya que se trabaja realmente con texto plano para el intercambio de información.

9.6 Pruebas sénsorica

Instalación de la librería Adafruit_DHT en Python para poder interactuar con el sensor de temperatura y humedad DTH11. [4]

Instalación de librerías GPIO para activación de relé. Método que le pasa la posición dentro del GPIO y activa / desactiva salida (relé) [5]

9.7 Ejecución script de inicio

Se crea una tarea en con Crontab y se configura la directiva **@reboot** para que se ejecute siempre después de cada reinicio.

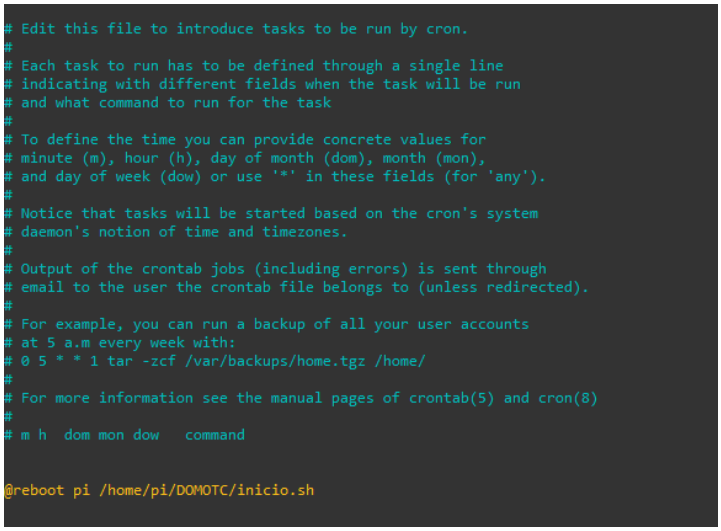
Inicio.sh. al programa principal. Este script se ejecuta al iniciar el sistema y debe tener permisos de ejecución previamente configurados. Se utiliza la directiva **@reboot** en Cron para que se ejecute siempre después de cada reinicio.

```
#!/bin/bash
DATE=$(date +'%F %H:%M:%S')
DIR=/home/pi/DOMOTIC
echo "La hora actual es $DATE" > $DIR/fichero.txt
python /home/pi/DOMOTIC/Domotic.py
```

Crontrab

Para crear la tarea basta con ejecutar un terminal y como root escribir el siguiente comando.

crontab -e y añadir la línea: **@reboot pi /home/pi/DOMOTIC/inicio.sh**



9.8 Ejecución de IPTABLES

A continuación, se ve el contenido de las reglas IPTABLES que se aplica en el servidor.

Se dejan abiertos los puertos: 899 (SSH) y 5000 (Flask)

```
## Script protección servidor DOMOTIC
## Tráfico de salida para la interfaz local
echo "*** Aplicando Reglas de Firewall..."
## FLUSH de reglas
/sbin/iptables -F
/sbin/iptables -X
/sbin/iptables -Z
/sbin/iptables -t nat -F
## Politica por defecto
/sbin/iptables -P INPUT DROP
/sbin/iptables -P OUTPUT DROP
/sbin/iptables -P FORWARD DROP
echo 1 > /proc/sys/net/ipv4/ip_forward
/sbin/iptables -t nat -P PREROUTING ACCEPT
```

```
/sbin/iptables -t nat -P POSTROUTING ACCEPT
## Filtrar
/sbin/iptables -A INPUT -i lo -j ACCEPT
/sbin/iptables -A OUTPUT -o lo -j ACCEPT
## Permitir tráfico established (o de retorno)
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
## Ping (pero evitando DoS)
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 5/minute --limit-burst 20 -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

###### PREVENCION DE ATAQUES ####
## Eliminar cualquier paquete TCP que no se ha iniciado con el Flag SYN activo
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
## Eliminar paquetes fragmentados
iptables -A INPUT -f -j DROP
# Eliminar paquetes con todos los flags tcp activados (XMAS packets).
iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP
## Prevenir Ataque DOS eliminando paquetes sin ningun flag tcp activado (NULL packets)
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
## Eliminar cualquier paquete inválido que no pueda ser identificado
iptables -A INPUT -m state --state INVALID -j DROP

####### TRAFICO DE SALIDA ####
## Permitir tráfico de salida y retorno por la interface pública
iptables -I OUTPUT -o wlan0 -d 0.0.0.0/0 -j ACCEPT
#iptables -I INPUT -i wlan0 -m state --state ESTABLISHED,RELATED -j ACCEPT
## Permitir trafico de salida y retorno por la interface local
iptables -I OUTPUT -o wlan0 -d 0.0.0.0/0 -j ACCEPT
###### DROP from INDIVIDUALS ######
iptables -A INPUT -s 191.96.249.0/24 -j DROP
###### ACCEPT from INDIVIDUALS ########
iptables -A INPUT -s 192.168.1.0/24 -p tcp --dport 899 -m state --state NEW,ESTABLISHED -j ACCEPT
###### ACCEPT from ANY ####
## FLASK- HTTP
iptables -A INPUT -p tcp --dport 5000 -j ACCEPT
## Prevenir ataques DoS
##-Limit 25/minutos : Limita a sólo 25 conexiones por minuto.
##-Limit-burst 100: Indica que el valor de limit/minute será forzado sólo después del número de conexiones en este nivel
iptables -A INPUT -p tcp --dport 5000 -m limit --limit 25/minute --limit-burst 100 -j ACCEPT

########## Log dropped packets #####
#iptables -N LOGGING
#iptables -A INPUT -j LOGGING

#iptables -A LOGGING -m limit --limit 2/min -j LOG --log-prefix "IPTables Paquetes Dropped: " --log-level 7
#iptables -A LOGGING -j DROP
```

9.9 Fotos de la sensorica y accesorios utilizados

Servidor

Raspberry Pi 4 Modelo 4 B. Se opta por este modelo por ser el más potente que hay actualmente. Necesita Fuente de alimentación propia.

Figura 4.

Salida
Relé KY-019 5V – 5v de entrada y 220v de salida.

Figura 5.

Electroválvula que funciona a 220v. Cada electroválvula es independiente lo cual sirve para servir a dar suministro a 3 zonas independientes.

En el caso del proyecto se centra en una sola, dejando la posibilidad de ampliación.

Figura 6.

Sensor
Sensor DTH11 para lecturas de temperatura y humedad. Funciona a 3,3v y se alimenta desde la misma placa Raspberry Pi.

Figura 7.

Interruptor flotador
Control de agua en depósito.

Figura 8.

9.10 Funcionamiento de los componentes y del sistema

El proyecto está centrado en seguridad, pero dispone de otros elementos domóticos para darle un poco de contexto.

Se dispone de un deposito de agua conectado a la eletrovávula.

Por otro lado, el interruptor flotador en encuentra dentro de un bebedero, cuando el flotador baja, automáticamente se activa el relé y abre la electroválvula dejando caer el agua.

Como la electroválvula es de 3 vías, con la segunda vía se conecta a través de un relé.

El sensor de temperatura y humedad determina solo los valores ambientales a nivel informativo.

10. Bibliografía

1. Most Used REST API Authentication Methods (mayo 2020)
   https://blog.restcase.com/4-most-used-rest-api-authentication-methods/

2. Raspberry Pi 4 Model B+ (abril-mayo 2020)
   https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

3. API REST (febrero 2020)

https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

4. Sensor DTH 11
   http://www.geekbotelectronics.com/dht11-sensor-de-humedad-y-temperatura/

5. Relé
   https://relequick.com/como-conectar-un-rele/