

Informe técnico de la práctica PEC2

Alberto Estévez Casarrubios, Álvaro Barragán Codesal, Eduardo Garzo Jiménez ,
Nicolae Alexandru Molnar

Campus Universitario, Ctra. Madrid-Barcelona km, 33, 600, 28805 Alcalá de Henares
eduardo.garzo@edu.uah.es, alberto.estevezc@edu.uah.es , nicolae.molnar@edu.uah.es,
alvaro.barragan@edu.uah.es.

Resumen. En el presente documento hablaremos sobre la implementación práctica de un sistema de seguridad compuesto de sensores de proximidad, luz y cámaras, para cualquier tipo de viviendas. Esto se conseguirá mediante el envío de los datos recopilados desde la capa de percepción, a una capa de procesamiento de los mismos, en la que se agruparán según de donde hayan sido obtenidos, pudiendo luego tratarlos con distintas funciones para poder guardarlos, borrarlos o cambiarlos en una base de datos, en la que almacenar esos datos asociados a los diferentes usuarios del sistema. De manera que estos usuarios puedan ver toda su información mediante el uso de una página web o aplicación, conectadas al servidor principal.

Palabras clave: Servidor, MQTT, Página Web, Android, Servlets.

Campus Universitario, Ctra. Madrid-Barcelona km, 33, 600, 28805 Alcalá de Henares
eduardo.garzo@edu.uah.es, alberto.estevezc@edu.uah.es , nicolae.molnar@edu.uah.es,
alvaro.barragan@edu.uah.es.

1

1. Introducción	2
2. Objetivo del proyecto	3
3. Arquitectura del proyecto	3
3.1. Capa de percepción	4
3.1.1. Sensores	5
3.1.2. ESP32	6
3.1.4. Cámara	7
3.2. Capa de transporte	7
3.2.1 Wi-Fi	7
3.2.2 MQTT	9
3.3. Capa de procesado	9
3.3.1. Broker	9
3.3.1.1 Instalación	9
3.3.1.2 Estructura de etiquetas elegidas	11
3.3.2. Apache Tomcat	12
3.3.2.1 Instalación	12
3.3.2.2 Configuración	13
3.3.2.3 Implementación	16

3.3.2.3.1 Paquete Database	18
3.3.2.3.2 Paquete logic	19
3.3.2.3.3 Paquete Database	21
3.3.2.3.4 Paquete Servlets	24
3.3.3. Base de datos	26
3.3.3.1 Instalación	26
3.3.3.2 Configuración	27
3.3.3.3 Diseño:	30
3.4. Capa de aplicación	31
3.4.1. Aplicación móvil	32
3.4.1.1. Interfaz de usuario	32
3.4.1.2. Lógica de negocio	35
3.4.2. Aplicación Web	37
3.4.2.1 Login	37
3.4.2.1 Register	38
3.4.2.1 Dashboard	38
3.4.2.1 Gallery	39
3.4.2.1 Streaming	41
3.4.2.1 Contact	42
3.4.2.1 About	42
3.4.2.1 Settings	43
3.4.2.1 Admin	44
4. Conclusiones	44
5. Bibliografía	45
Anexo I – Manual de instalación	45
Anexo II – Manual de usuario de la aplicación Móvil	45
Anexo III – Manual de Usuario de la aplicación Web	55

1. Introducción

Este trabajo se ha realizado con el objetivo de aportar una mayor seguridad en una parte fundamental de nuestras vidas como puede ser el hogar. Para lograr esta meta hemos creado un sistema informático compuesto por una arquitectura de 4 capas: capa de percepción, capa de transporte, capa de procesamiento y la capa de aplicación.

- En la capa de percepción procederemos a obtener una serie de datos mediante los sensores o bien imágenes haciendo uso de la cámara, los cuales enviaremos a las siguientes capas.

- La capa de transporte hace uso del wifi y mqtt para poder enviar estos datos recopilados en la capa de percepción.
- Seguidamente en la capa de procesamiento una vez obtenidos estos datos, los trataremos según la información que queramos obtener de cada uno de ellos, ya bien sea para poder para guardarlos asociándose a los diferentes usuarios poseedores del sistema, o bien enviarlos a la capa de aplicación para ver las diferentes imágenes u horas de activación de los sensores.
- En la capa de aplicación dispondremos de una aplicación móvil con la que cambiar los datos de usuario, ver las imágenes guardadas y ver el streaming de nuestra cámara. Y además hacemos uso de nuestra propia página web para mostrar la información a los clientes de forma más detallada que en la aplicación y con funcionalidades extra como filtros de imágenes, registro y página de administrador.

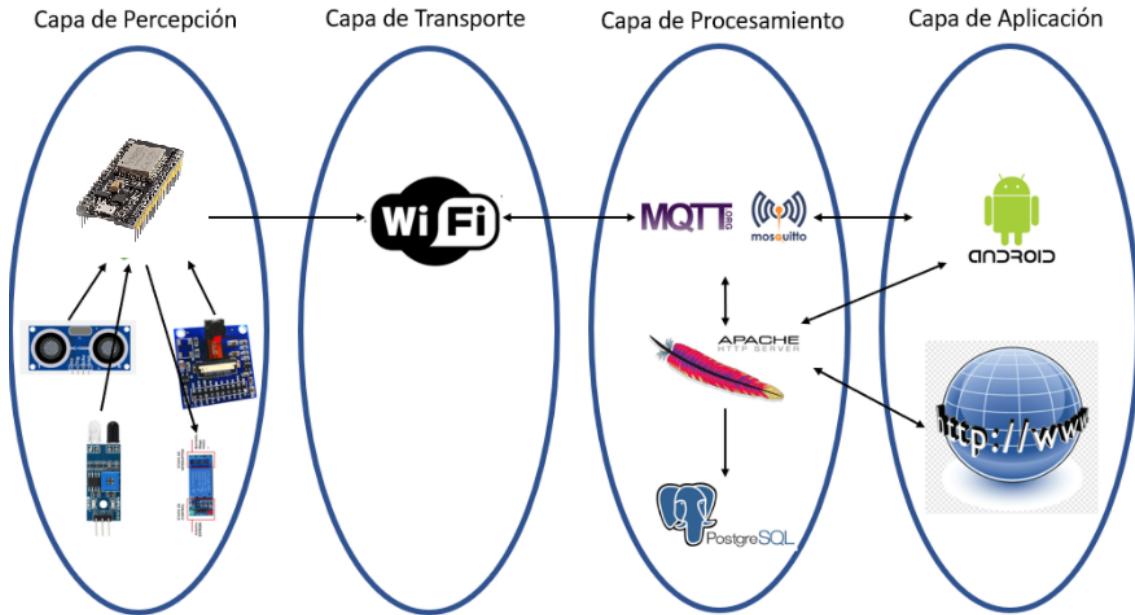
2. Objetivo del proyecto

Como objetivo principal de este proyecto hemos desarrollado un sistema de seguridad para viviendas que consta de una mirilla inteligente y un conjunto de sensores para posibles intrusiones en un perímetro determinado. Además del componente hardware también hemos logrado una interconexión de esta primera capa de percepción a las demás capas haciendo uso de comunicaciones Wi-fi y envío de datos mediante MQTT. Todo esto con el objetivo de conseguir un sistema completo, que conste de:

- Una base de datos en la que guardamos la información recogida por los sensores, así como de los usuarios/clientes de nuestro sistema.
- Un servidor con el que tratamos los diferentes datos recogidos, no sólo para guardarlos en la base de datos, sino también para poder transformarlos y prepararlos para su posterior envío.
- Una aplicación Android que ofrece a nuestros clientes de manera que les sea más sencillo acceder a sus datos, imágenes guardadas e imagen en directo.
- Una página Web con la que ofrecer además de un servicio de visión de los datos más detallado, una forma intuitiva de registrarse en nuestro sistema y poder filtrar la cantidad de información recibida.

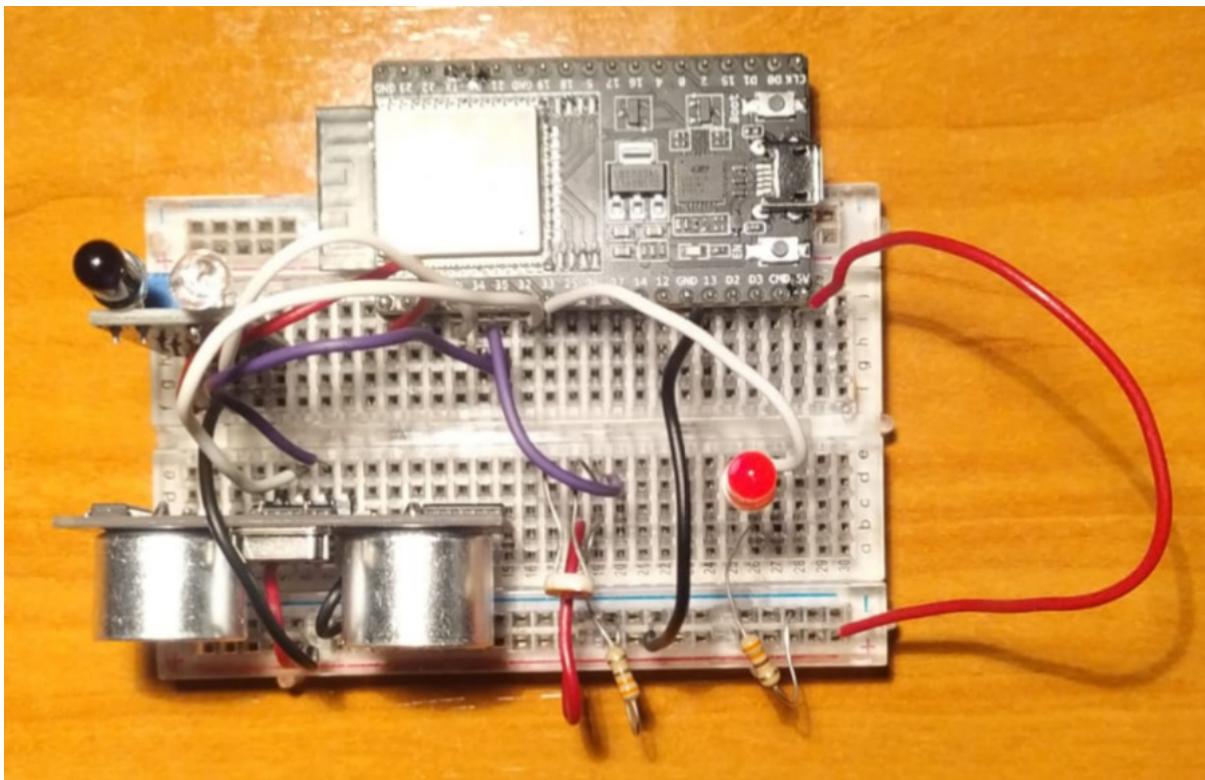
3. Arquitectura del proyecto

La arquitectura de nuestro proyecto sigue la misma estructura ideada en la fase anterior de la práctica:



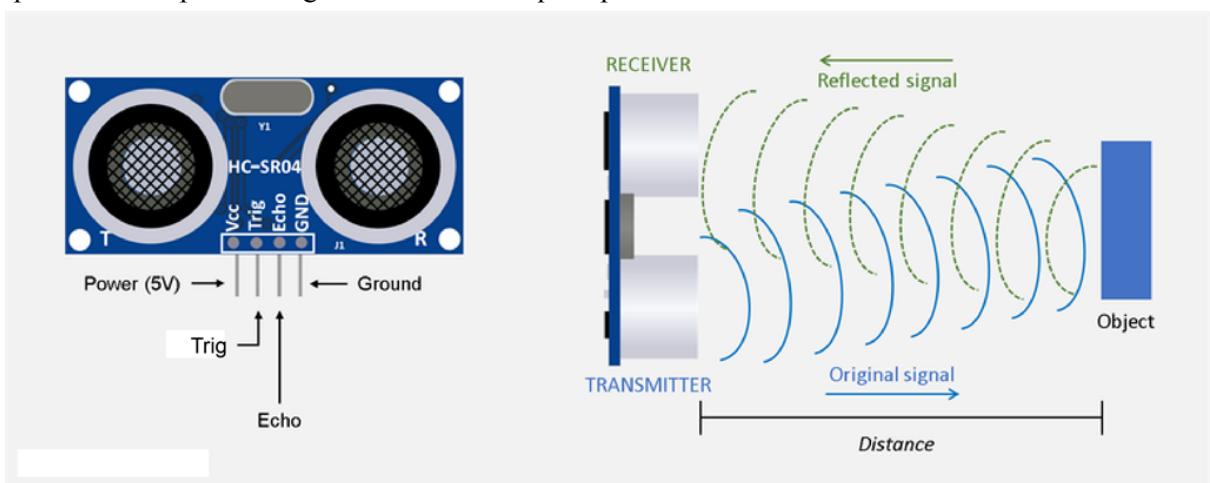
3.1. Capa de percepción

En esta capa buscamos percibir señales de activación o alertas gracias a los sensores que explicaremos a continuación y capturar imágenes por cámara para luego enviarlas. Primero mostraremos el diseño de nuestro sistema hardware, que elementos lo componen, cómo interaccionan estos elementos entre sí y qué conseguimos con los mismos para pasar la información en las sucesivas capas. Los elementos que conforman esta capa son: sensores (LDR, de proximidad y de movimiento), así como una placa ESP32 con módulo de Wi-fi integrado, la propia cámara del ordenador para recibir y capturar las imágenes, luces led para la simulación de una bombilla con la que podamos ver en caso de que no sea posible la visibilidad por la cámara y obviamente un protoboard para interconectar todos estos componentes. El montaje final de todos los componentes queda de la siguiente forma:



3.1.1. Sensores

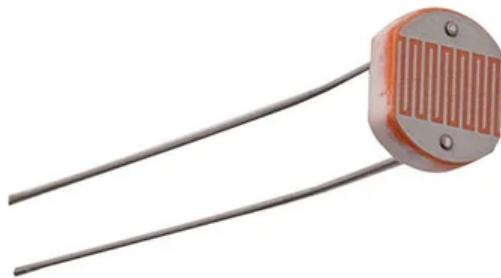
- Sensor de proximidad: que utilizamos para medir la distancia al objeto más cercano frente al sensor. La medida se obtiene por medio de ondas de ultrasonido, el módulo tiene un emisor y un receptor con los que en función del tiempo que tarda la onda en llegar al receptor podemos calcular la distancia al objeto. Este sensor nos sirve para detectar si hay un objeto/persona frente a nuestra mirilla para luego añadir información a los avisos en la capa de aplicación, ayudando a la cámara en la detección de objetos/personas cerca de la misma o incluso aportando ese ‘plus’ de seguridad en caso de que tapen la cámara.



- Sensor de movimiento: nos sirve también para comprobar si hay objetos cerca de nuestra “mirilla inteligente” y también actuaría como añadido en una implementación de nuestro sistema a mayor escala, pudiendo instalarlo en diferentes ventanas para controlar así las posibles entradas a la vivienda. En nuestro caso solo hemos hecho uso de uno para exemplificar y demostrar su funcionamiento.

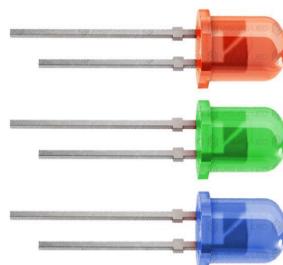


- Sensor de luz (LDR): este sensor es una adición a la funcionalidad límite de nuestra cámara, ya que su objetivo es detectar la cantidad de luz que recibimos del exterior, ya que en el caso de que la misma fuera insuficiente para poder capturar alguna imagen con la cámara, entonces nos avisará de que está demasiado oscuro. De esta forma si es el caso de que estuviera demasiado oscuro y además se activaran cualquiera de los sensores de movimiento o proximidad, de manera que luego podemos ya en la programación de la placa hacer que se encendiera un foco de luz (en nuestro caso el led) para iluminar el espacio y así tener la posibilidad de capturar imágenes.



3.1.2. Led

Las luces led nos sirven simplemente para hacer una simulación de lo que sería un foco de luz con el que luego añadiremos esa visibilidad extra a la cámara anteriormente mencionada.



3.1.2. ESP32

Esta es la placa que hemos usado para interconectar todos los componentes de nuestra capa de percepción, además tiene la ventaja de disponer de módulos de comunicación inalámbrica de serie, haciendo mucho más fácil el uso de mecanismos de comunicación como el Wi-fi que luego usamos en la capa de transporte. El propio ESP32 se conecta al Wi-fi por medio de la función “init_wifi()” y al

MQTT mediante la función “init_mqtt()” con la que nos suscribimos a todos los topics que le indicamos.

- Puertos que usamos para conectar los sensores a la placa:
 - Sensor de proximidad: se conecta a los pines: echo, que envía la onda de ultrasonido que rebota contra el objeto más cercano y trigger, que recibe la onda y calcula la distancia mediante la función “get_distance”. Cuyos números de pin son 34 y 32 respectivamente.
 - Sensor de iluminación: usa el pin 30, threshold que usamos como umbral para que en el momento en el que la luz detectada sea menor que el umbral envíe la señal al LDR, para que pueda encender la luz led.
 - Sensor de movimiento: con el que detectamos el movimiento que se produzca cerca de la cámara independientemente de si ese objeto se ha quedado parado o bien solo ha pasado por delante durante un momento. En resumen podría usarse como el sensor de proximidad, pero resultaría más discreto en lugares como ventanas para detectar ese movimiento.

3.1.4. Cámara

Nuestra idea inicial era una cámara que grababa imágenes y las mostraba por medio de una dirección IP. Sin embargo, para simplificar el proceso de implementación hemos optado en nuestro prototipo por hacer uso de la cámara que está integrada directamente en el portátil, con esta cámara integrada no podemos acceder a sus datos ni por medio de la web ni por medio de la IP, de manera que hemos programado un script en Python que nos permita ir cogiendo frames de la cámara del propio ordenador. Así mismo hemos logrado que además de capturar imágenes y prepararlas con el servidor (que era la idea inicial), sea también capaz de procesar las mismas haciendo uso de una Inteligencia Artificial que se encarga de reconocer rostros y manos. También hay que añadir que en función de una serie de configuraciones que establecemos en el servidor la cámara enviará imágenes de streaming constantemente, imágenes que tengan un rostro o una mano, reconocidas, en su interior cada 5 segundos y una etiqueta de la imagen en la que se clasifica la misma ya bien sea porque haya reconocido solo una mano, una cara, o ambas al mismo tiempo.

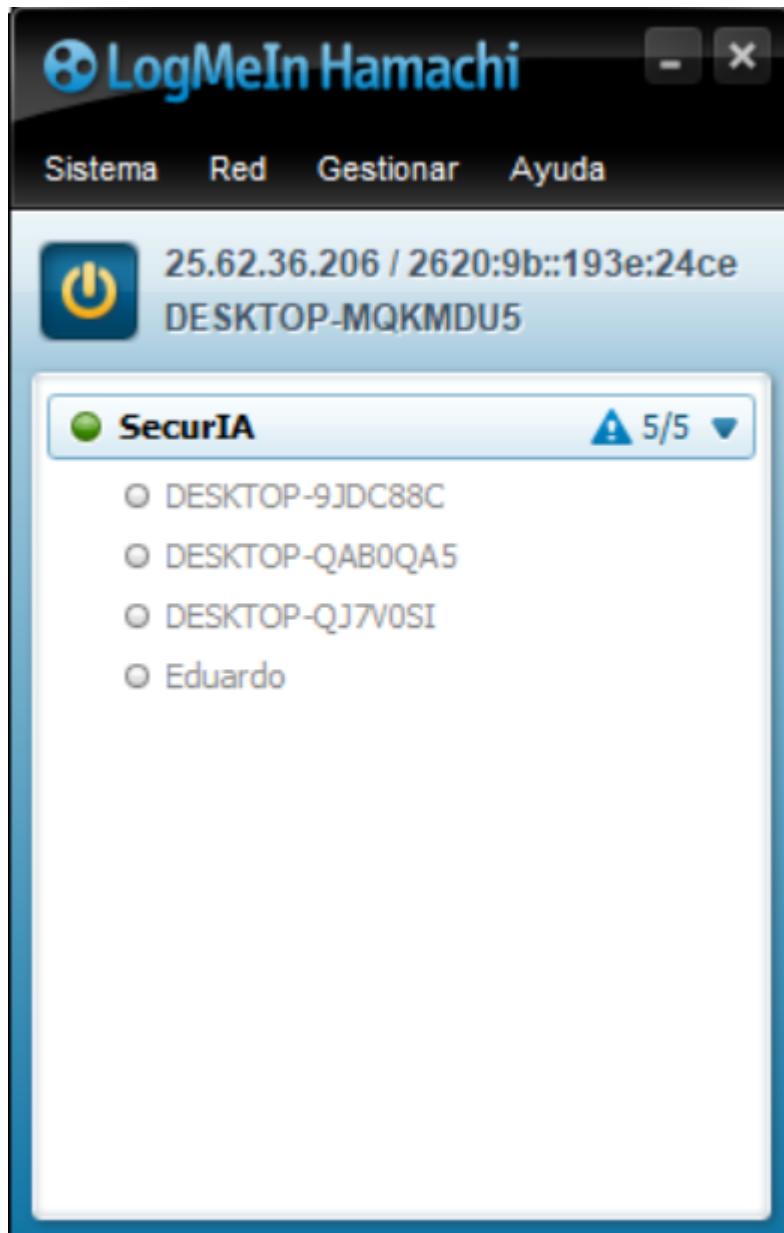
3.2. Capa de transporte

3.2.1 Wi-Fi

Haremos uso de la tecnología red Wi-Fi para respaldar toda nuestra capa de transporte. Toda la información transmitida desde la capa de percepción procesamiento, y la información intercambiada entre capa de aplicación y servidor se harán por medio de Wi-Fi.

Para ser capaces de trabajar juntos a pesar de no encontrarnos en la misma red local, hemos utilizado la VPN *LogMeIn Hamachi* que nos permite simular una red *local* para todos los componentes del sistema. Sin embargo, existen algunos dispositivos, como la capa de

percepción, a los que es complicado o imposible conectar a dicha VPN. Estos dispositivos deben necesariamente encontrarse en la misma red local que el servidor.



Hemos decidido utilizar la tecnología Wi-Fi sobre Ethernet ya que, al tratarse de un sistema escalable en el que el mismo servidor puede dar servicio a varias mirillas, la necesidad del cable sería una limitación importante y la diferencia entre tasas de transferencia no presenta ningún inconveniente.

Además, se planea que el proyecto sea escalable y, en lugar de un hogar, pueda dar servicio a varios hogares, saliéndose de la red local. Por tanto, el uso de Wi-Fi es un requisito imprescindible.

3.2.2 MQTT

Respaldados por la tecnología Wi-Fi, utilizaremos el protocolo MQTT para realizar el intercambio de información entre los miembros de la red. Este protocolo consiste en un esquema Publicador-Suscriptor en el que interviene un Servidor o Bróker.

La información se transmite por temas, *topics*, a los que cada cliente se suscribe para recibir información de manera pasiva, transmitida por el publicador.

Se ha elegido este protocolo de comunicación sobre TCP o UDP ya que nos ofrece las ventajas de ambos protocolos y una gran flexibilidad y facilidad de implementación. Además, el esquema Publicador-Suscriptor encaja a la perfección con los objetivos del proyecto y su organización.

3.3. Capa de procesado

En la capa de procesado se encuentran involucrados el Bróker mencionado, el servidor Apache Tomcat y la base de datos PostgreSQL. Cada uno de los mencionados, explicados a continuación, recibirá datos que procesará o almacenará para su posterior uso en el sistema.

Estos datos serán enviados tanto por la capa de percepción como por la capa de aplicación, como por ejemplo medidas y peticiones. Tras su procesado servirán para actualizar elementos de la base de datos o transmitir nueva información a la capa de aplicación.

En el prototipo presentado, como ya se ha mencionado, utilizamos una cámara integrada en el portátil, por tanto en la entrega codificada el procesado de imágenes no se hará en esta capa. Sin embargo, en un implementación real dicha cámara sería un módulo cámara que envíe información al servidor por MQTT, información que será procesada y almacenada en esta capa.

3.3.1. Broker

3.3.1.1 Instalación

El primer paso a realizar fue la configuración del Bróker, para lo cual instalamos [mosquitto](#) para Windows 10. Tras su instalación configuramos la variable de entorno *Path* para poder acceder a los binarios desde cualquier *cwd*.

Editar variable de entorno

X

C:\Program Files\Oculus\Support\oculus-runtime
C:\Program Files (x86)\VMware\VMware Player\bin\
C:\Program Files\Java\jdk1.8.0_311\bin
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\Users\Alex\OneDrive - Universidad de Alcalá\Documentos\Unive...
C:\Program Files\Java\jre1.8.0_311\bin
C:\Program Files (x86)\Graphviz\bin
C:\Program Files\mosquitto
C:\Program Files\PostgreSQL\14\bin

Nuevo

Modificar

Examinar...

Eliminar

Subir

Bajar

Editar texto...

Aceptar

Cancelar

Una vez preparado el programa Mosquitto, estamos listos para lanzar el servidor. Sin embargo, por defecto, en Windows, se lanza en modo *local only*, esto significa que los accesos desde fuera del dispositivo anfitrión están prohibidos. Para solucionar este problema modificamos el archivo *mosquitto.conf* añadiendo *allow_anonymous=true* y *listener 5555*.

Esta configuración nos permite acceder desde dispositivos como el ESP32 y provoca que, siempre que utilicen el puerto 5555, cualquier dispositivo de nuestra red pueda acceder al Bróker.

Finalmente lanzamos el servidor, ejecutándose como servicio para aceptar las conexiones al puerto especificado:

```

Usage: mosquitto [-c config_file] [-d] [-h] [-p port]

-c : specify the broker config file.
-d : put the broker into the background after starting.
-h : display this help.
-p : start the broker listening on the specified port.
    Not recommended in conjunction with the -c option.
-v : verbose mode - enable all logging types. This overrides
    any logging options given in the config file.

See https://mosquitto.org/ for more information.

C:\Users\Alex>mosquitto -p 5555 -c "C:\Program Files\mosquitto\mosquitto.conf" -d_

```

Para realizar pruebas durante el desarrollo se han utilizado los binarios mosquitto_sub y mosquitto_pub, pero no han resultado en una implementación posterior por lo que esta es la única mención que reciben.

Por último, explicar que MQTT permite 3 niveles de calidad de servicio, *QOS*. Nivel 0, los mensajes se envían a modo de UDP, sin comprobaciones, sin seguridad de llegada; Nivel 1, los mensajes llegan al destinatario con seguridad, pero puede haber mensajes repetidos; Nivel 2, los mensajes llegan al destinatario a modo de TCP, sin errores y sin repetidos. Haremos uso de cada uno de los niveles según los beneficios de cada uno.

3.3.1.2 Estructura de etiquetas elegidas

Para el desarrollo del proyecto se han necesitado *topics* específicos para cada capa y dispositivo. A continuación explicaremos cuáles son los *topics* utilizados en cada capa y la razón de su uso:

En la capa de percepción tiene como raíz el topic “/sensor/”, donde, según la estructura de nuestra base de datos, podemos encontrar los siguientes *topics*: “/sensor/proximity/{system_id}”, *topic* que transmite las notificaciones de detección de objetos frente a la mirilla; “/sensor/movement/{system_id}/{sensor_id}”, *topic* que transmite las notificaciones de detección de movimientos en los distintos sensores, *sensor_id*, asociados al sistema, *system_id*.

En el caso la cámara vamos a diferenciar los dos casos mencionados anteriormente: en el caso de utilizar la cámara del portátil necesitaremos los topics “/sensor/camera/{camera_id}/Streaming”, “/sensor/camera/{camera_id}/Label”, “/sensor/camera/{camera_id}/Image” y “/sensor/camera/{camera_id}/canStream”, *topics* que nos permiten transmitir la información ya procesada desde la capa de percepción; en el caso de utilizar un módulo cámara no haríamos uso ningún *topic* ya que el módulo que planeamos implementar actúa como cámara IP, de la que podemos obtener las imágenes necesarias por medio de peticiones HTTP.

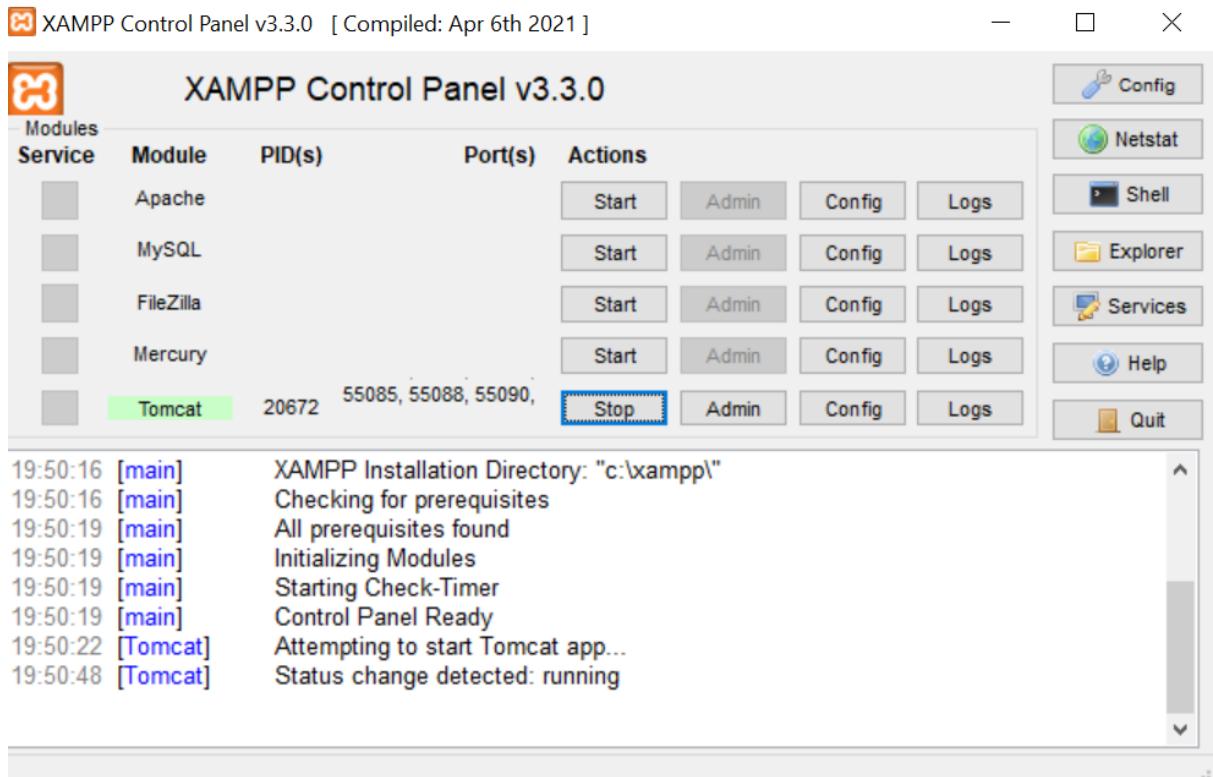
En cuanto a los *topics* relativos a la capa de aplicación, los utilizamos como herramientas de comunicación asíncrona entre el servidor y los dispositivos móviles. El único *topic* utilizado

es “`/android/notifications/{email}`”, un topic que, como se explicará más adelante, nos permite el envío de notificaciones a android sin necesidad de esperas activas por parte de la app.

3.3.2. Apache Tomcat

3.3.2.1 Instalación

La instalación de apache tomcat también se ha realizado en Windows 10. Para instalar este servidor hemos decidido hacerlo instalando el panel de control XAMPP:



Este panel de control nos ofrece, además de otros módulos que no hemos necesitado en la implementación del proyecto, un módulo para Apache Tomcat. Este módulo es fácilmente configurable y proporciona muchas comodidades para su ejecución y mantenimiento.

Para instalar este panel de control, hemos accedido a www.apachefriends.org/es y descargado el instalador para Windows:



3.3.2.2 Configuración

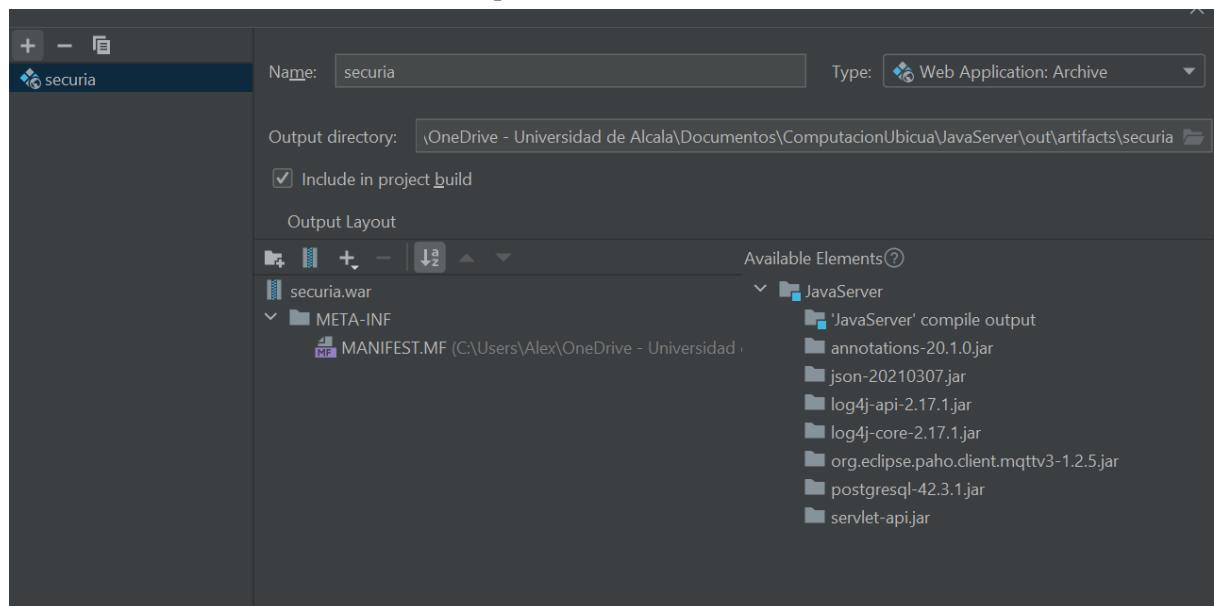
Para la configuración del servidor hemos accedido al directorio raíz del mismo, en nuestro caso “*C:\xampp\tomcat\webapps*”:

Nombre		Fecha de modificación	Tipo
do	docs	15/11/2021 22:50	Carpeta de archivos
Jniversida	examples	15/11/2021 22:50	Carpeta de archivos
os	host-manager	15/11/2021 22:50	Carpeta de archivos
	manager	15/11/2021 22:50	Carpeta de archivos
	securia	13/01/2022 11:51	Carpeta de archivos
	securia	05/01/2022 12:30	Archivo WinRAR
	securia.war	26/12/2021 15:57	Archivo WAR

Como podemos apreciar en la imagen, para configurar el servidor hemos necesitado añadir el fichero *securia.war*, para ello hemos accedido a “<http://localhost:8080/manager/html>” y utilizado la siguiente opción:



Para obtener el *.war* mencionado hemos tenido que crear un archivo *MANIFEST.MF* y acceder a la herramienta de creación de Aplicaciones Web de nuestro IDE, IntelliJ:



Por último hemos configurado el archivo *web.xml* contenido en */securia/WEB-INF/* para configurar los distintos servlets de la aplicación y sus mapeados, además de establecer como página de bienvenida el archivo *login.jsp*:

```

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1"
    metadata-complete="true">

    <description>
        SecurIA Web Application
    </description>
    <display-name>SecurIA</display-name>

    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>

    <listener>
        <listener-class>logic.ProjectInit</listener-class>
    </listener>

```

Además de lo mencionado hemos definido a la clase *ProjectInit*, contenida en el paquete *logic*, como la clase *listener* encargada de inicializar y destruir el proceso y los distintos servlets utilizados. Cómo extra, esta clase será la encargada de establecer y cerrar las conexiones con el Bróker.

```

@WebListener
public class ProjectInit implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        // init the project
        Log.log.info("Initializing the server...");
        Logic.streams = new HashMap<>();
        Logic.mqttBroker = new MQTTBroker( clientId: "SecurIA Central Broker", ip: "localhost", port: 5555, qos: 2);
        Log.logmqtt.info("MQTT Broker created");
        Logic.mqttSubscriber = new MQTTSubscriber(Logic.mqttBroker);
        Log.logmqtt.info("MQTT Subscriber created");
        Logic.mqttSubscriber.searchTopicsToSubscribe();

    }

    @Override
    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        Logic.mqttSubscriber.disconnectFromBroker();
        Log.log.info("Server service stopped.");
    }
}

```

A continuación, en el archivo *web.xml* se inicializan y mapean los servlets ofrecidos por el servidor:

```

<servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>servlets.LoginServlet</servlet-class>
</servlet>

```

```

<servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>

```

Aquí se presentan dos ejemplos de definición y mapeo, el resto se pueden observar en `web.xml`. Los servlets ofrecidos en el servidor deben encontrarse dentro del directorio `/securia/WEB-INF/classes/`, dentro de sus respectivos paquetes. Además es necesario que cualquier librería externa esté contenida como archivo *JAR* en el directorio `/securia/WEB-INF/lib/` para que puedan ser utilizadas por el servidor.

« tomcat > webapps > securia > WEB-INF >			
	Nombre	Fecha de modificación	Tipo
	classes	14/01/2022 0:48	Carpeta de archivos
	lib	03/01/2022 23:05	Carpeta de archivos
	web	14/01/2022 21:20	Documento XML

Para introducir los servlets en el servidor, debemos compilarlo en nuestro IDE respectivo, y copiar los `.class` y, si se dispone de log, el archivo `.xml` del log dentro de este directorio. Cada servlet debe estar en una carpeta con el nombre de su paquete.

« webapps > securia > WEB-INF > classes >			
	Nombre	Fecha de modificación	Tipo
	database	14/01/2022 0:48	Carpeta de archivos
	logic	14/01/2022 0:48	Carpeta de archivos
	mqtt	14/01/2022 0:48	Carpeta de archivos
	servlets	14/01/2022 0:48	Carpeta de archivos
	log4j	02/01/2022 19:44	Documento XML

Dentro del directorio `/securia/` podremos encontrar los archivos que definen la página web hosteada por Tomcat:

« xampp > tomcat > webapps > securia

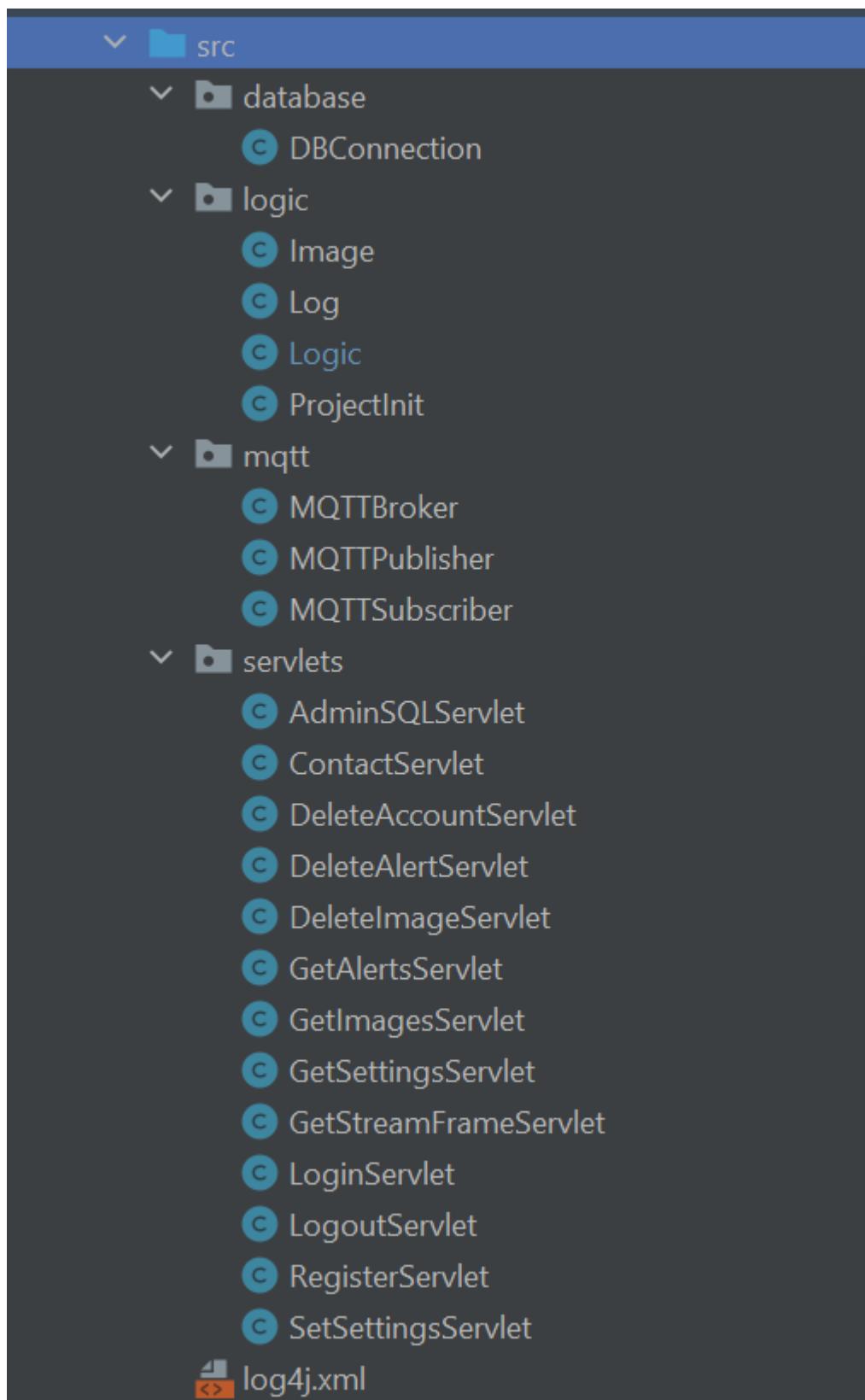
Buscar en securia

Nombre	Fecha de modificación	Tipo
css	26/12/2021 15:58	Carpeta de archivos
Images	14/01/2022 16:21	Carpeta de archivos
META-INF	26/12/2021 15:58	Carpeta de archivos
WEB-INF	26/12/2021 16:10	Carpeta de archivos
about	14/01/2022 8:42	Archivo de origen Jav...
admin	14/01/2022 8:03	Archivo de origen Jav...
Avatar	16/11/2021 13:44	Archivo JPG
contact	14/01/2022 8:42	Archivo de origen Jav...
dashboard	14/01/2022 8:41	Archivo de origen Jav...
error	26/12/2021 16:22	Archivo de origen Jav...
gallery	14/01/2022 10:11	Archivo de origen Jav...
login	11/01/2022 11:06	Archivo de origen Jav...
logo	12/11/2021 13:46	Archivo JPG
register	11/01/2022 10:09	Archivo de origen Jav...
settings	13/01/2022 19:07	Archivo de origen Jav...
stream_img	07/01/2022 16:26	Archivo de origen Jav...
streaming	14/01/2022 11:58	Archivo de origen Jav...
termsOfPrivacy	11/01/2022 11:04	Documento de texto
termsOfService	11/01/2022 11:03	Documento de texto

Las páginas de estilo estarán contenidas en la carpeta `/securia/css/`, las imágenes de la página en la carpeta `/securia/Images/` y el `MANIFEST.MF` mencionado anteriormente en la carpeta `/securia/META-INF/`. Por último, los archivos relativos a términos de privacidad, servicio y `jsp` se mantienen en el directorio raíz `/securia/`, de esta manera las redirecciones a los mismos se facilitan teniendo que añadir el nombre del archivo (o servlet que lo inicia) como `"/securia/{file}"`.

3.3.2.3 Implementación

Para la implementación de los elementos mencionados se ha utilizado el IDE de programación IntelliJ y el lenguaje de programación Java. Hemos creado el proyecto llamado `JavaServer`, que tiene la siguiente estructura:



Podemos observar que el proyecto se divide en 4 paquetes principales: *database*, *logic*, *mqtt* y *servlets*. Además de estos existe un archivo *log4j.xml* donde configuramos el sistema de Log de nuestro servidor. Este proyecto representa al

servidor, que conectará con el Bróker MQTT, la base de datos y la capa de aplicación, por medio de servlets.

3.3.2.3.1 Paquete Database

Este servlet será la única vía de acceder a la información contenida en la base de datos, de esta forma centralizamos las peticiones y actualizaciones para poder gestionarlas a todas por igual y para evitar fallos de integridad debido a un mal uso de la base de datos por parte de dispositivos no autorizados. Para ello hemos implementado la clase DBConnection que contiene la información de la base de datos y que recibe como parámetros del constructor las credenciales del proceso que quiere operar sobre la misma:

```
public class DBConnection {

    // Database information
    private String dbName = "SecurIA";
    private String userName;
    private String password;
    private String hostName = "localhost";
    private String portNumber = "5432";

    // Database status
    private Connection connection;

    public DBConnection(String username, String password) {...}
}
```

Esta clase será la única clase en la que se realizarán consultas SQL, los servlets utilizarán métodos creados en la sección *SQL Calls*, delimitada por un comentario. En esta sección podemos observar funciones que nos permiten obtener, insertar y actualizar datos de la base de datos:

```

/* ===== SQL Calls ===== */
public String login(String email, String password) throws SQLException {...}

public boolean register(String username, String first_name, String email, String password, St
public boolean insertContact(String name, String email, String phone, String company, String
public void delete_account(String email) {...}

public HashMap<String, Boolean> getSensors() {...}

public ArrayList<String> getCameras() {...}

public HashMap<String, String> getSettings(String email) {...}

public boolean setSettings(String email, String password, String firstname, String surname, S
public int get_image_id() {...}

public ArrayList<Integer> get_system_ids(){...}

```

Estos son algunos ejemplos de llamadas realizadas por los servlets, que siempre tienen la misma estructura:

```

public int get_image_id() {
    // Prepare SQL call
    int image_id = 0;

    try (Statement csmt = connection.createStatement()) {
        // Get the image_id
        ResultSet res = csmt.executeQuery( sql: "SELECT MAX(image_id) FROM public.image;" );
        if (res.next()) {
            image_id = res.getInt( columnLabel: "max" ) + 1;
        }
    }

    } catch (SQLException e) {...}
    return image_id;
}

```

Se prepara el *Statement*, se lanza la consulta y se analiza el resultado. En ocasiones, cuando las operaciones involucran inserciones o modificaciones en varias tablas hacemos uso de *Transactions*, que nos permite confirmar modificaciones, *commit*, o cancelarlas, *callback*.

3.3.2.3.2 Paquete logic

A continuación, en el paquete logic tenemos las clases que almacenan datos y funciones relativas a la lógica de negocio. La clase *Image* representa a un frame capturado por la cámara y que guarda sus distintos atributos:

```
public class Image {  
    private String path;  
    private String label;  
    private Timestamp timestamp;
```

A continuación tenemos la clase estática *Log* que ofrece la herramienta de utilizar logs para mostrar información o errores en el servidor una vez iniciado, ya que al no ejecutarlo desde nuestro usuario no podemos acceder a la salida estándar:

```
public class Log {  
    public static Logger log = LogManager.getLogger("Log");  
    public static Logger logmqtt = LogManager.getLogger("LogMQTT");  
    public static Logger logdb = LogManager.getLogger("LogDB");  
}
```

Aquí podemos observar un ejemplo de la utilización de la clase *Log* para registrar cambios en el sistema:

```
Log.logmqtt.info("Created MQTT Broker with QoS: "+qos+" and broker: "+broker+" and clientId: "+clientId);
```

El archivo *ProjectInit* ya ha sido explicado por lo que explicaremos la clase *Logic*. Esta es una clase estática que contiene información necesaria para la implementación y gestión del servidor. Los métodos reutilizables, de gestión de servidor y los atributos estáticos del sistema como las rutas de los directorios de interés se almacenan en esta clase. Además, para aportar velocidad de transmisión a la transmisión en directo, guardamos el último frame recibido de cada cámara en un *HashMap* cuya clave es el nombre de la cámara. De esta manera no tenemos que hacer lecturas a la base de datos y el tiempo que estas conllevan.

```

public class Logic {
    public static String imagePath = "/seguridad/Images/";
    public static String dbBin = "C:\\Program Files\\PostgreSQL\\14\\bin\\";
    public static String dbPath = "C:\\Program Files\\PostgreSQL\\14\\data\\";
    public static String tomcatPath = "C:\\xampp\\tomcat\\webapps";
    public static MQTTBroker mqttBroker;
    public static MQTTSubscriber mqttSubscriber;
    public static HashMap<String, String> streams;

    public static String formatDate(String date) {...}

    public static String login(String email, String password) throws SQLException {...}

    public static void delete_account(String email) throws SQLException {
        // Obtain DB connection
        DBConnection db = new DBConnection( username: "postgres", password: "123456");

        db.obtainConnection();

        // Delete account from db
        db.delete_account(email);
    }
}

```

3.3.2.3.3 Paquete Database

Para continuar, el paquete *mqtt* contiene las clases que usa nuestro servidor para comunicarse con el Bróker MQTT. Entre ellas existe una clase *MQTTBroker* que contiene la información de conexión al Bróker, la clase *MQTTSubscriber*, que contiene métodos que permiten la suscripción a *topics* y el manejo de los mensajes recibidos, y la clase *MQTTPublisher*, que permite publicar, dado un *MQTTBróker*, un mensaje en un *topic* del Bróker.

Este es un esquema de la clase *MQTTBroker*:

```

public class MQTTBroker
{
    private int qos;
    private String broker;
    private String clientId;

    public MQTTBroker(String clientId, String ip, int port, int qos)
    {
        this.qos = qos;
        this.broker = "tcp://"+ip+":"+port;
        this.clientId = clientId;
        Log.logmqtt.info("Created MQTT Broker with QoS: "+qos+" and broker: "+broker+" and clientId: "+clientId);
    }
}

```

Dentro de la clase *MQTTSubscriber* tenemos dos atributos esenciales, una instancia *MQTTBroker* y una instancia *MQTTClient*, que nos permiten conectarnos al Bróker y realizar las operaciones pertinentes. Entre ellas, caben destacar el método ***searchTopicsToSubscribe***, que recoge la información de todos los sensores del

sistema y se crea Strings con los topics de cada uno. Después llama **subscribeTopics**, que se suscribe a cada uno de los topics recopilados. Una vez suscrito a un *topic*, cuando se reciba un mensaje, este activará el método **messageArrived** donde, en función del *topic* que active la función realizaremos una serie de operaciones para añadir alertas, actualizar frames de streaming o almacenar imagenes en la base de datos:

```
public void messageArrived(String path, MqttMessage mqttMessage) throws Exception {
    String[] topics = path.split(regex: "/");
    //Log.logmqtt.error("Message arrived: "+mqttMessage.toString());

    String topic = topics[2];

    // Handle the message arrival and treat data accordingly
    switch (topic) {
        case "movement":
            int sensor_id = Integer.parseInt(topics[4]);
            // Check if user allows notifications
            int system_id = Integer.parseInt(topics[3]);
            if (Logic.allows_notifications(system_id)) {
                // If allowed, store notification in the database
                Logic.add_alert(system_id, title: "Movement detected" , description: "Movement detected in Log.logmqtt.info("Notified movement detection in sensor "+sensor_id);
            }
            break;

        case "proximity":
            system_id = Integer.parseInt(topics[3]);
            // Check if user allows notifications and the camera didn't detect any person
            if (Logic.allows_notifications(system_id) && !Logic.person_detected(system_id)) {
                // If allowed, store notification in the database
                Logic.add_alert(system_id, title: "Possible attack detected!" , description: "We strongly believe yo Log.logmqtt.info("Notified possible attack detected in system "+system_id);
            }

            break;
    }
}
```

```

case "camera":
    if (topics[4].equals("Image")) {
        // Decode base64 image payload
        String image_str = mqttMessage.toString();
        system_id = Logic.get_system_id(topics[3]);
        byte[] image_bytes = Base64.getDecoder().decode(image_str);
        Log.logmqtt.info("Image received with size: " + image_bytes.length);

        if (Logic.capture_photos(system_id)) {
            String image_path = Logic.add_image(Logic.imagePath, image_bytes, extension: "jpg", topics[3]);
            Log.logmqtt.info("Image saved in: " + image_path);
        }

        // Check if user allows notifications
        if (Logic.allows_notifications(system_id)) {
            // If allowed, store notification in the database
            Logic.add_alert(system_id, title: "Image received", description: "There is a person at your door");
            Log.logmqtt.info("Notified "+system_id+" owner for received image");
        }
    }

    }else if (topics[4].equals("Streaming")) {
        String image_str = mqttMessage.toString();
        Logic.streams.put(topics[3], image_str);
    }else if (topics[4].equals("Label")) {
        String label_str = mqttMessage.toString();
        Log.logmqtt.info("Label received: "+label_str);
        Logic.set_label(topics[3], label_str);
    }
    break;
default:
    Log.logmqtt.error("Topic not recognized: "+topic);
    break;
}

```

Por último, cuando el servidor se desconecta del Bróker, la función ***connectionLost*** detecta cuando se ha perdido la conexión y reconecta el servidor al bróker. De esta manera cuando publicamos, aunque nos desconectemos como suscriptores, podemos reconectarnos de manera automática.

En el caso de la clase *MQTTPublisher*, sólo existe el siguiente método publish:

```

public static void publish(MQTTBroker broker, String topic, String message) {
    MemoryPersistence persistence = new MemoryPersistence();
    try {
        // Connect to MQTT Broker
        MqttClient client = new MqttClient(broker.getBroker(), broker.getClientId(), persistence);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(true);
        client.connect(options);
        Log.logmqtt.info("Publisher connected to MQTT Broker");

        // Prepare message to publish
        MqttMessage mqttMessage = new MqttMessage(message.getBytes());
        mqttMessage.setQos(broker.getQos());

        // Publish message and disconnect
        client.publish(topic, mqttMessage);
        Log.logmqtt.info("Published message to topic: " + topic);
        client.disconnect();
        Log.logmqtt.info("Disconnected from MQTT Broker");
    } catch (MqttException e) {
        Log.logmqtt.error("Error while publishing message to topic: " + topic+. Cause: "+e.getCause());
    }
}

```

Este método obtendrá una conexión con el cliente, publicará el mensaje proporcionado en el topic proporcionado por parámetro y se desconectará del cliente.

3.3.2.3.4 Paquete Servlets

Por último, el paquete `servlets` contiene la implementación de cada uno de los servlets ofrecidos por el servidor. Estos servlets, en lo general, ofrecen sus métodos `doGet` y `doPost`. En el caso de las aplicaciones móviles, estas envían una petición al servlet que este responde con una respuesta de tipo `JSON`, que contiene la información a transmitir:

```

protected void doGet(HttpServletRequest request, @NotNull HttpServletResponse response) th
    String email = request.getParameter("email");
    String password = request.getParameter("password");

    try {
        String username = Logic.login(email, password);

        // Create response JSON
        JSONObject json = new JSONObject();
        json.put("successful_login", !username.equals("") && !username.equals("admin"));

        // Send JSON response
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(json.toString());
    } catch (SQLException e){
        Log.log.error("Error connecting to database: " + e.getMessage());
    }
}

```

Este es un servlet de login que valida la información recibida por parámetro y contesta con un booleano para indicar a la aplicación que permita el inicio de sesión o no. En el caso de la aplicación web, hacemos uso de las sesiones JSP y redirecciones a secciones de nuestra misma web:

```
String email = request.getParameter("email");
String password = request.getParameter("password");

try{
    String username = Logic.login(email, password);

    // If the user is valid, set the session attribute
    if (!username.equals("")) {
        // Redirect to the home page
        HttpSession session = request.getSession();
        session.setAttribute("email", email);
        if (username.equals("admin")) {
            // Redirect to the admin page
            response.sendRedirect("/securia/admin.jsp");
        } else {
            response.sendRedirect("/securia/dashboard.jsp");
        }
        Log.log.info("Login successful for user: " + username);
    } else{
        // If the user is not valid, redirect to the login page
        request.getSession().setAttribute("error", "Invalid email or password");
        response.sendRedirect("/securia/");
        Log.log.info("Login failed for user: " + username);
    }
} catch (SQLException e){
```

En algunos casos, como la generación del registro de notificaciones, la página web solicita respuestas JSON, por lo que nuestro sistema no actúa igual en todos los casos. En los casos en los que utiliza redirecciones se basa en el sistema de sesiones de JSP para enviarle a cada sesión la información necesaria con el método `setAttribute`. De esta manera nuestra página web tendrá unos atributos u otros en función del usuario que inició la sesión.

Entre la lista de servlets de nuestro sistema podemos diferenciar servlets utilizados para obtener información, actualizarla o insertar nueva información en el servidor o en la base de datos a través del mismo.

Aquí tenemos un listado de servlets y una breve descripción del objetivo de cada uno:

- **AdminSQLServlet**: Servlet que permite al administrador ejecutar sentencias SQL y por tanto operar en la base de datos desde cualquier dispositivo con acceso a la página de administrador.
- **ContactServlet**: Servlet utilizado para introducir una nueva solicitud de contacto en la base de datos al que hay que pasarle como parámetros el nombre, email, número de teléfono, compañía y mensaje del usuario solicitante.
- **DeleteAccountServlet**: Servlet que, pasándole la dirección de correo electrónico de un usuario, elimina al usuario asociado a dicho correo electrónico de la base de datos
- **DeleteImageServlet**: Servlet que pasándole el path de una imagen, borra dicha imagen de la base de datos.

- **GetImagesServlet**: Servlet que, pasándole el email de un usuario, devuelve los paths, labels y fechas de las imágenes asociadas al usuario.
- **GetSettingsServlet**: Servlet que, pasándole el email de un usuario y si se solicita desde una aplicación web o Android, devuelve la configuración del usuario
- **GetStreamFrameServlet**: Servlet que, pasándole el email de un usuario devuelve un stream de lo que está grabando la cámara
- **LoginServlet**: Servlet que recibe email y contraseña por parámetro y valida el inicio de sesión de un usuario, redirigiendo en aplicaciones web y devolviendo un booleano en aplicaciones android.
- **LogoutServlet**: Servlet que ayuda a la gestión de sesiones de *JSP*, vaciando la sesión actual y, por tanto, cerrando sesión en el sistema.
- **RegisterServlet**: Servlet que, pasándole un nombre de usuario, un email, dos contraseñas iguales, el primer nombre del usuario, su apellido, su número de teléfono y su fecha de nacimiento, crea un usuario en la base de datos con esos datos.
- **SetSettingsServlet**: Servlet que, pasandole un email, una contraseña, el primer nombre del usuario, su apellido, su número de teléfono, su fecha de nacimiento y si quiere el usuario que el sistema capture fotos, capture videos o pueda transmitir lo que graba su cámara, actualiza los ajustes de usuario a los recibidos.

3.3.3. Base de datos

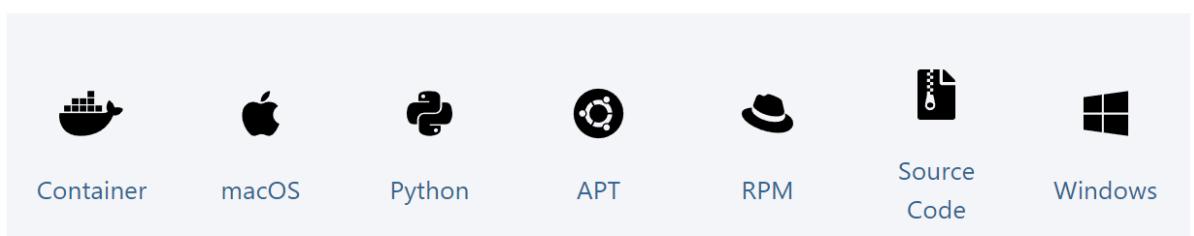
Como gestor de la base de datos hemos elegido a PostgreSQL por nuestra familiaridad con el mismo y su facilidad para lanzar, crear y gestionar transacciones por consola.

3.3.3.1 Instalación

Para la instalación de la base de datos hemos descargado el gestor de bases de datos www.pgadmin.org, la versión para Windows:

pgAdmin 4

pgAdmin 4 is a complete rewrite of pgAdmin, built using Python and Javascript/jQuery. A desktop runtime written in NW.js allows it to run standalone for individual users, or the web application code may be deployed directly on a web server for use by one or more users through their web browser. The software has the look and feels of a desktop application whatever the runtime environment is, and vastly improves on pgAdmin III with updated user interface elements, multi-user/web deployment options, dashboards, and a more modern design.



En nuestro caso hemos descargado el gestor con una interfaz web llamada *pgAdmin* y la versión de PostgreSQL instalada es la 14. Al instalar el programa pgAdmin 4 tendremos instalada la versión de PostgreSQL mencionada y el directorio binario estará añadido al path, en caso contrario, debemos añadirlo de la siguiente forma:

Editar variable de entorno

X

C:\Program Files\Oculus\Support\oculus-runtime
C:\Program Files (x86)\VMware\VMware Player\bin\
C:\Program Files\Java\jdk1.8.0_311\bin
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\Users\Alex\OneDrive - Universidad de Alcalá\Documentos\Unive...
C:\Program Files\Java\jre1.8.0_311\bin
C:\Program Files (x86)\Graphviz\bin
C:\Program Files\mosquitto
C:\Program Files\PostgreSQL\14\bin

Nuevo

Modificar

Examinar...

Eliminar

Subir

Bajar

Editar texto...

Aceptar

Cancelar

3.3.3.2 Configuración

Una vez instalado el gestor de bases de datos debemos crear la base de datos abriendo el programa pgAdmin y crear una base de datos con los siguientes atributos:



Create - Database



General Definition Security Parameters Advanced SQL

Database

SecurIA

Owner

👤 postgres



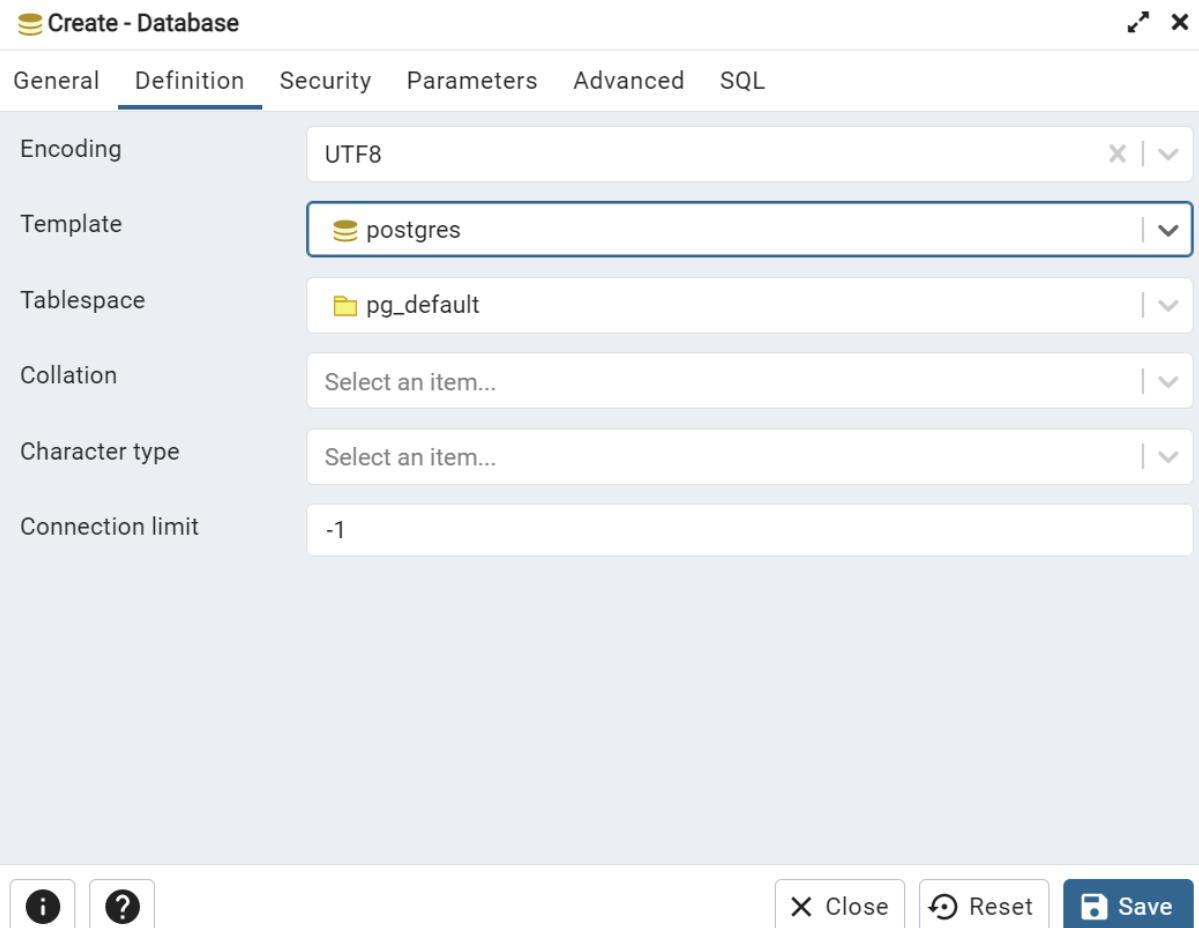
Comment



X Close

↻ Reset

💾 Save



Una vez creada la base de datos, abrimos una *Query Tool* y ejecutamos los siguientes scripts almacenados en la carpeta *DataBase*:

The screenshot shows the pgAdmin Query Tool. The 'Generated.SQL' tab is selected. The query editor displays the following SQL code:

```
1 /*  
2 Created: 26/11/2021  
3 Modified: 13/01/2022  
4 Model: PostgreSQL 11  
5 Database: PostgreSQL 11  
6 */  
7  
8 -- Create tables section --
```

Below the editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'.

Abrimos el archivo *Generated.SQL* desde pgAdmin y ejecutamos la consulta. Esto creará cada una de las tablas de la base datos:

- > alert
- > camera
- > camera_event
- > contact_request
- > image
- > sensor_event
- > sensor_type
- > system
- > user

Por último, ejecutamos el script *insert.SQL* el cual realizará una carga inicial de datos en la base de datos:

```

1 /*
2 Created: 26/11/2021
3 Modified: 13/01/2021
4 Model: PostgreSQL 11
5 Database: PostgreSQL 11
6 */
7
8 -- Table Client --

```

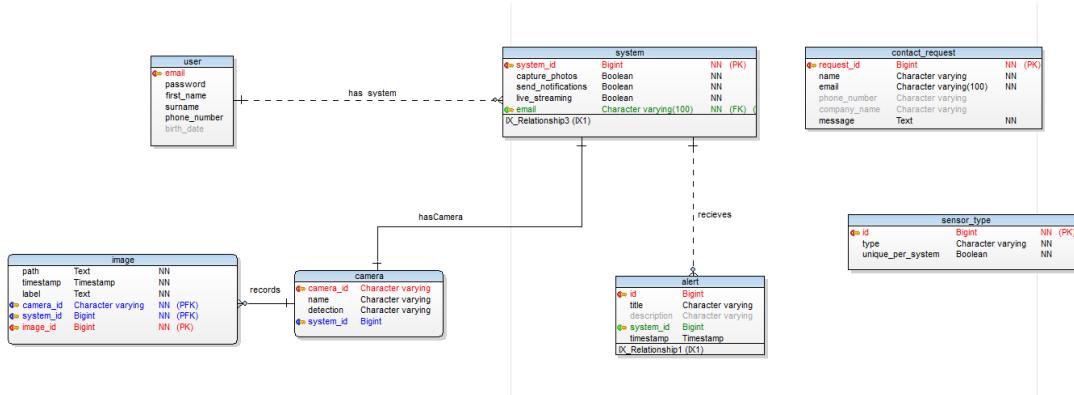
Query Editor Query History

Data Output Explain Messages Notifications

Esta carga inicial contiene el usuario de administrador y un usuario para cada miembro de nuestro de los cuales sólo el usuario con email 'nicolae.molnar@edu.uah.es' tiene asociado un sistema, es decir un set de sensores que representan la mirilla.

3.3.3.3 Diseño:

Para el diseño de la base de datos hemos utilizado la herramienta [TOAD DATA MODELER](#), con la que hemos creado el siguiente modelo E/R:



En dicho modelo podemos observar que cada usuario puede tener uno o más sistemas. Un sistema es un conjunto de sensores, de los cuales solo se encuentran definidas las cámaras ya que la aportación del resto de sensores a la base de datos se refleja en forma de alertas. Un usuario contiene información personal del cliente y el email es la clave primaria, una clave que lo identifica y que no se puede repetir.

Tal y como hemos descrito el sistema, contiene tres *booleanos* que habilitan o deshabilitan la captura de imágenes, la transmisión en vivo y el registro de alertas. Además, un sistema solo contiene una cámara, por lo que cada mirilla se corresponderá con un sistema nuevo, de esta manera aseguramos la escalabilidad del proyecto, permitiendo replicar el esquema sistema-sensores-alertas n veces para cada usuario.

Una cámara contiene un identificador, un nombre y una etiqueta de detección. Como ya se ha mencionado la IA que procesa las imágenes devuelve etiquetas "face" y "hand" o la combinación de estas y cuando una imagen nueva se almacena, hereda la etiqueta que tiene la cámara en ese momento.

Una imagen, por su parte, contiene un identificador, una ruta al archivo guardado en disco, una marca de tiempo de cuándo fue capturada y la etiqueta heredada de la cámara.

Una alerta contiene un identificador, un título, una descripción y una marca de tiempo de cuándo saltó la alerta. La tabla *alert* funciona como un registro de las alertas del sistema que aún no han sido tratadas, es decir, cuando el usuario lo deseé, eliminará las alertas que necesite de esta tabla, dejando solo las de su interés o las más recientes.

La tabla *contact_request* contiene todas las solicitudes de contacto de la página web dirigidas al administrador del proyecto. Esta tabla es independiente del resto del sistema.

La tabla *sensor_type*, actúa como registro persistente de los tipos de sensores admitidos en el sistema. En caso de, al escalar el proyecto, cambiar o añadir algún sensor, el sistema actuará de manera consecuente en función de la información de esta tabla.

3.4. Capa de aplicación

Nuestra capa de aplicación consta de:

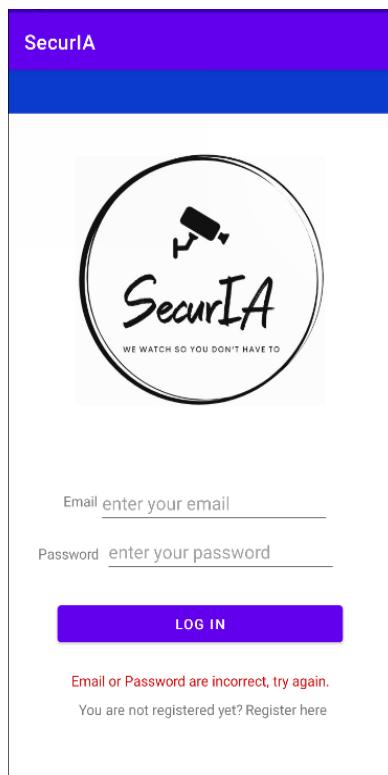
- Una aplicación móvil
- Una página web

3.4.1. Aplicación móvil

3.4.1.1. Interfaz de usuario

La interfaz de usuario se basaría en la interacción del mismo con las diferentes actividades que creamos para las funcionalidades que ofrecemos desde nuestra aplicación, que son las siguientes:

- “*login_activity*”: actividad mediante la cual el usuario podrá acceder a la aplicación, más en concreto a la actividad main, a través de un email y contraseña. Si la información proporcionada es válida, se le concederá el inicio de sesión a su perfil con sus correspondientes configuraciones. De otro modo, si la información no es correcta (el email , contraseña o ambos), aparecerá un mensaje de error advirtiendo de este fallo.

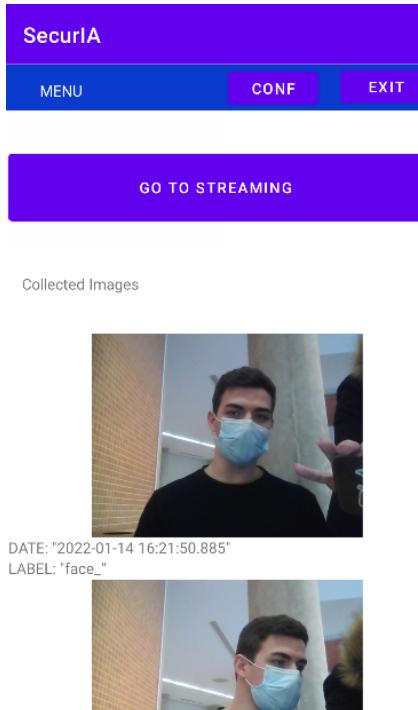


Interfaz de Login con mensaje de error de inicio de sesión.

Por otra parte, en esta actividad también disponemos de la opción de registrarnos en el caso de que todavía no tengamos nuestro propio usuario y contraseña. Al final de la actividad saldrá un mensaje de información donde podremos leer: You are not registered yet? Register here. Si pulsamos en este Register here nos llevará a la página web, directamente a la página de registro.

- “*main_activity*”: esta actividad es el menú principal de la aplicación. Es la primera ventana que nos aparece tras iniciar sesión. La función principal de esta actividad es servir de biblioteca de imágenes, las cuales estarán ordenadas por el momento de su captura. Esta biblioteca o galería de imágenes se muestra en la parte inferior de la actividad ofreciendo

todas las imágenes que son capturadas por la cámara, es decir, cuando detecta una cara, mano o ambas.



Interfaz del menú donde se muestran las imágenes y sus etiquetas correspondientes.

A parte de esta galería, en esta actividad tendremos ciertos botones que nos llevarán a otras actividades. Estos son el botón *CONF*, que nos lleva a la actividad de configuración o *config_activity*, el botón *GO TO STREAMING*, que nos llevará a la actividad de visualizar un video en vivo o *stream_activity*, y por último el botón *EXIT*, el cual nos permitirá salir de nuestra sesión devolviéndonos a la actividad del login.

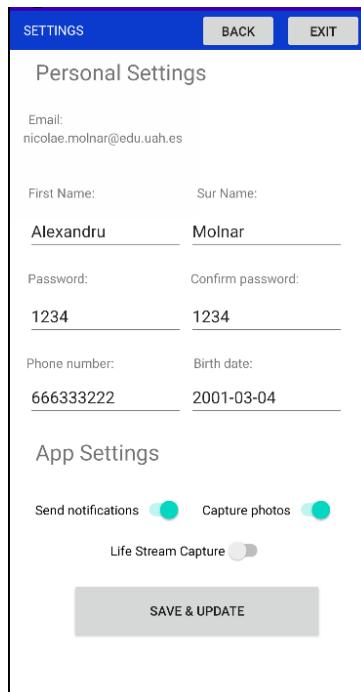
Debemos mencionar que cada actividad tiene su correspondiente archivo .xml donde montamos la interfaz de dicha actividad. Sin embargo, en este caso haremos uso de dos archivos .xml. Esto es necesario para poder mostrar todas las imágenes disponibles en la base de datos que tenga asociadas ese usuario. En primer lugar, tenemos el archivo principal *main_activity.xml* donde se mostrarán todos los botones mencionados anteriormente y un ScrollView en el cual insertamos una ImageView junto con sus correspondientes TextView que serán el timestamp y label (que indicará que ha detectado: hand_, face_, face_hand_) declarados en el segundo archivo *item.xml*. La idea es crear ese layout item tantas veces como imágenes tengamos que mostrar.

- “***config_activity***”: es la actividad de configuración, donde un usuario podrá cambiar todos sus datos excepto el email. En cuanto a los datos a cambiar podemos encontrarnos tres tipos de errores al intentar hacer estos cambios. Vamos a mencionarlos por prioridad, ya que solo aparecerá uno a la vez aunque tengamos los tres, de forma que el más prioritario saldrá siempre aunque se den los otros dos: algún campo está vacío, ya que ninguno puede estarlo, si

el campo contraseña y el de confirmar contraseña no son iguales y por último comprobar que la fecha de nacimiento es válida, consideramos que es válida si es anterior al día actual.

Además desde esta actividad se podrá habilitar o deshabilitar el capturar fotos, capturar stream y enviar notificaciones.

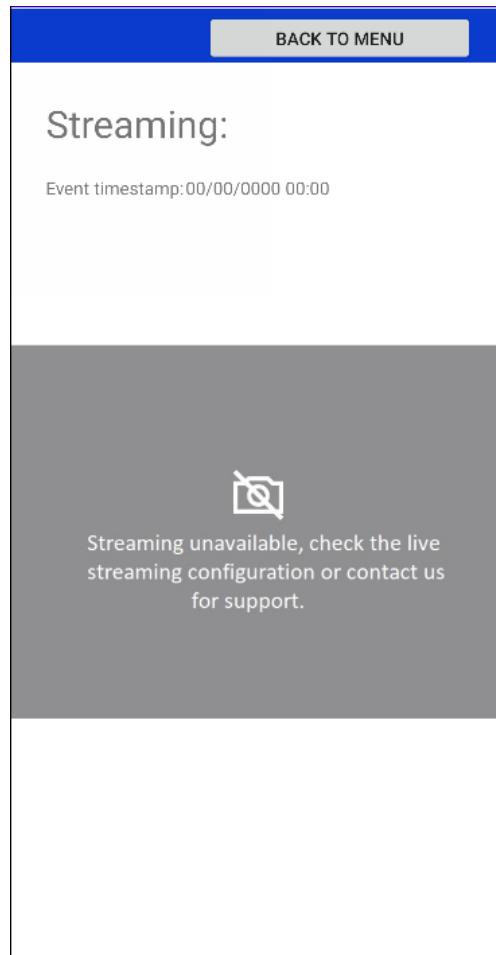
En esta actividad también podemos encontrar ciertos botones: el principal y más importante es *SAVE AND UPDATE* el cual guarda los cambios que están en los cuadros de texto en ese momento, si no ha ocurrido ningún error, modificándose en la base de datos. Para terminar tenemos dos botones en la parte superior derecha *BACK* y *EXIT*, el primero vuelve al menú y el segundo sale de la sesión devolviendo al usuario a la actividad del login.



Interfaz de configuración de datos personales y app.

- “**stream_activity**”: esta actividad solo tiene una función principal, la cual es retransmitir en vivo todo lo que capture la mirilla en ese momento. Junto a la retransmisión podemos ver la hora actual.

Podemos volver al menú principal pulsando nuestro botón *BACK TO MENU*.



Interfaz del Streaming donde vemos que la opción de streaming está desactivada.

3.4.1.2. Lógica de negocio

El funcionamiento interno de nuestra aplicación se basa también en las mismas cuatro clases de actividades clasificadas en el apartado de interfaz de usuario, y una clase denominada “Hilo” con la que crear nuestro canal de notificaciones:

- “**LoginActivity**”: clase que representa el funcionamiento y comunicación necesarios para realizar el login de un usuario en la aplicación, primero deberá leer los 2 parámetros que el usuario introduzca de email y su correspondiente contraseña. Seguidamente también tenemos declarada una clase interna, llamada “GetXMLTask” (esta subclase es usada en las diferentes actividades de nuestra aplicación, y por lo tanto también declarada en las mismas, debido a que cada una tiene su propia forma de comportarse) con la cual podemos establecer la conexión con el servidor y obtener las diferentes salidas del mismo, ya sean mensajes o bien los objetos JSON que vamos a tratar durante toda la ejecución de la aplicación. Así mismo declaramos las funciones para actuar en caso de que el usuario pulse un botón, que en esta actividad serían:
 - btnlogin: que envía al server el email y contraseña que el usuario haya introducido, por medio de la dirección URL, en la cual definimos el servlet que vamos a usar en este caso el servlet de Login, y pasando esta URL a la propia clase de

“GetXMLTask”, la cual la usará para establecer la conexión, y en caso de que este usuario esté registrado en la base de datos, comenzaremos un Intent en el que enviaremos al usuario a MainActivity.

- btnregister: para los casos en los que el usuario no este registrado lo direccionamos directamente a la pagina web, donde se abrirá la pestaña de la misma en la que el usuario debe introducir sus datos.
- “**MainActivity**”: clase del menú principal de la aplicación, en ella recibimos los datos del email de la actividad anterior, el cual vamos a pasar a ConfigActivity y StreamActivity. También en esta actividad procedemos a mostrar todas las capturas que tengamos guardadas del usuario en la base de datos, obteniendo las mismas haciendo uso del servlet de GetImagesServlet, el cual nos devolverá un JSON con las mismas. Además en la subclase de “GetXMLTask” es donde se produce la conversión de las imágenes recibidas como Strings, a imágenes en la aplicación mediante el uso de la función “StringToBitmap” con la que realizamos la transformación de un String a una imagen, las cuales vamos introduciendo en los sucesivos ImageView según la cantidad de fotos guardadas.
Incluimos en esta clase la implementación de las notificaciones de la aplicación que recibimos de MQTT mediante una clase llamada “**Hilo**” el cuál ejecutamos nada más empezar la actividad de Main, que crea el canal de notificaciones y se suscribe al MQTT mediante varias funciones:
 - En este hilo recibimos como parámetros tanto la actividad de Main, como el propio email del usuario, que usamos para pasarlo al topic al que nos vamos a suscribir, en este caso notifications. Una vez creado el cliente, se conecta a la dirección y puerto de MQTT y hace uso de las siguientes funciones:
 - setCallback: dentro del cual vamos a leer cuerpo del mensaje del topic según la notificación que recibamos.
 - createNotificationChannel y createNotification: que nos sirven para crear el canal de notificaciones en las versiones más actualizadas de android, y para crear la notificación en sí, respectivamente.
 - suscripcionTopics: con la que nos suscribimos al topic de notifications con el email y con Quality of Service de nivel 2 para evitar la pérdida de mensajes y el reenvío de los mismos.

Así mismo en la actividad Main también disponemos de varias funciones de botones:

- btnConfig: con el cual iniciaremos la actividad de configuración de datos del usuario.
- btnExit: con el cual saldríamos de la sesión del usuario volviendo a iniciar así la actividad del Login.
- btnStream: con el cual iniciaremos la actividad que nos permite ver la imagen en directo de la cámara.
- “**ConfigActivity**”: en esta actividad es donde procedemos a realizar la conexión con el servlet, mediante la subclase “GetXMLTask” haciendo uso del servlet GetSettingsServlet con el cual obtenemos todos los datos asociados al usuario, pudiendo así mostrarlos nada más entrar en los EditText que tiene el usuario para poder cambiarlos si así lo desea. Seguidamente el usuario puede cambiar estos campos (menos el de correo) para después hacer uso de la función del botón “saveUpdate” con la que volvemos a conectarnos al servidor pero esta vez haciendo uso del servlet “SetSettingsServlet” para guardar estos cambios en la base de datos.

Además disponemos de varios botones como:

- btnExit: que tiene la misma funcionalidad que en el main.
 - btnBack: que nos sirve para volver al menú principal de la aplicación.
 - captureFotos, lifeStream, sendNotifications: con los cuales podemos cambiar, al guardar la configuración, las opciones de envío y recogida de datos del usuario.
- “**StreamActivity**”: en esta actividad es donde a partir del servlet “GetStreamFrameServlet”, al cual llegamos haciendo la conexión con el URL, creamos un hilo de manera continuada para que nos muestre la imagen en directo que recibe la cámara. Esto lo hacemos creando un hilo con el cual vamos cambiando constantemente nuestra ImageView por el último frame que llega desde el servidor. La conversión de los JSON en formato String a formato de Imagen se hace con la misma función “StringToBitmap” que usamos en “MainActivity”. Así dentro de la ejecución del hilo hacemos uso de la función cambiaFrame, declarada en esta actividad de Stream con la que abrimos el JSON que recibimos del servlet y vamos actualizando el ImageView con la hora actual de stream.

3.4.2. Aplicación Web

En esta sección explicaremos el desarrollo de la página web. Entre los lenguajes utilizados hemos utilizado el lenguaje de marcado *HTML*, junto con el lenguaje de estilo *CSS*, además hemos utilizado los lenguajes *JSP* y *JavaScript* para programar los scriptlets que le proporcionan el dinamismo a nuestra página web. A continuación describiremos y mostraremos un ejemplo de los componentes de nuestra página web:

3.4.2.1 Login

En esta página observamos un header que contiene direcciones a las páginas de contacto e información, y un botón que nos redirigirá a una página de registro. En su interior tenemos un formulario que el usuario llenará y al pulsar el botón *Login*, se activará el servlet *login* que comprobará dichas credenciales y lo redirigirá a su dashboard o mostrará un mensaje de error si las credenciales no son válidas.





SecurIA Login

Sign into your account

Email address

Password

Login

3.4.2.1 Register

Al seleccionar *Register*, se nos redirigirá a esta página web en la que se nos presenta un formulario a llenar, donde todos los campos menos *Birth Date* son obligatorios. Cuando el usuario los rellene y pulse el botón *Register* estos datos se enviarán al servlet *register* y este nos redirigirá al login en caso exitoso o de nuevo al register mostrando un mensaje de error. Por último tenemos un botón *Login* que nos permite cancelar el registro y acceder al login de nuevo.

The screenshot shows a registration form titled "Sign up". On the right side, there is a circular logo with a surveillance camera icon and the text "SecurIA" in a stylized font, with the tagline "WE WATCH SO YOU DON'T HAVE TO" below it. The form itself has the following fields:

- Your Name
- Your Email
- Password
- Repeat your password
- First Name
- Surname
- Phone
- Birth Date (input field with placeholder "dd/mm/aaaa")
- A checkbox labeled "I agree all statements in [Terms of service](#)"

At the bottom of the form is a "Register" button.

3.4.2.1 Dashboard

La página de dashboard es una página de bienvenida en la que se muestra un registro de las notificaciones recientes o no eliminadas del usuario que inició sesión. Para mostrar dichas notificaciones, eliminarlas 1 por 1 o eliminarlas todas hacemos uso de tres funciones javascript que piden los datos al servlet *get_alerts*, eliminan una alerta y llaman a la función de eliminar alerta para todas las alertas, respectivamente.

Además, a la izquierda tenemos una cabecera donde el botón logout cerrará la sesión actual y nos devolverá a la página de login. El resto de nombres son enlaces a las distintas secciones de la página web.

La función que recoge y pinta las notificaciones se ejecuta cada segundo y actualiza la lista de notificaciones en tiempo real.

Welcome to SecurIA.com

Your notifications

Possible attack detected! We strongly believe your camera is being covered!! Event timestamp: 2022-01-14 23:26:31.131 x

Movement detected Movement detected in sensor 1 Event timestamp: 2022-01-14 23:26:26.768 x

Possible attack detected! We strongly believe your camera is being covered!! Event timestamp: 2022-01-14 23:26:26.192 x

Movement detected Movement detected in sensor 1 Event timestamp: 2022-01-14 23:26:21.255 x

Possible attack detected! We strongly believe your camera is being covered!! Event timestamp: 2022-01-14 23:26:20.547 x

Image received There is a person at your door. Event timestamp: 2022-01-14 23:26:17.336 x

Possible attack detected! We strongly believe your camera is being covered!! Event timestamp: 2022-01-14 23:26:15.441 x

Image received There is a person at your door. Event timestamp: 2022-01-14 23:26:10.042 x

Clear all

nicolae.molnar Logout

3.4.2.1 Gallery

La página de gallery es una página compuesta por dos secciones, una en la que se muestra la imagen seleccionada y se permite descargar la imagen o borrarla, todo mediante funciones javascript. Y otra en la que se muestran todas las imágenes almacenadas en el sistema para el usuario en cuestión, estas imágenes pueden ser filtradas por día para una mejor visibilidad de las mismas. Cada imagen tiene una función *onclick* que permite que, al pulsarla se muestre en la sección de “display”.

Para obtener las imágenes del servidor se hace uso del servlet *get_images* que, por medio de una sesión JSP, ofrece tres listas paralelas con rutas, etiquetas y marcas de tiempo, respectivamente. Es decir, las imágenes solo se actualizan cuando se accede a la galería, no en tiempo real.

Dashboard Gallery Streaming Contact us About us Settings

nicolae.molnar Log out

WebCam Image Gallery

Image properties

Label: Undetected

Timestamp: 18-11-2021 10:51:22

Download

Delete

Captured images: 83

Image filter

Filter

Captured images: 83

Image filter
dd/mm/aaaa



El resultado de pulsar una imagen es:

WebCam Image Gallery



Image properties

Label: face_

Timestamp: 2022-01-14 16

[Download](#)

[Delete](#)

es: 83

Image f
dd/mm

Los botones de descarga y borrado solo están operativos cuando hay una imagen seleccionada y cuando se elige un nuevo filtro, la imagen se deselecciona.

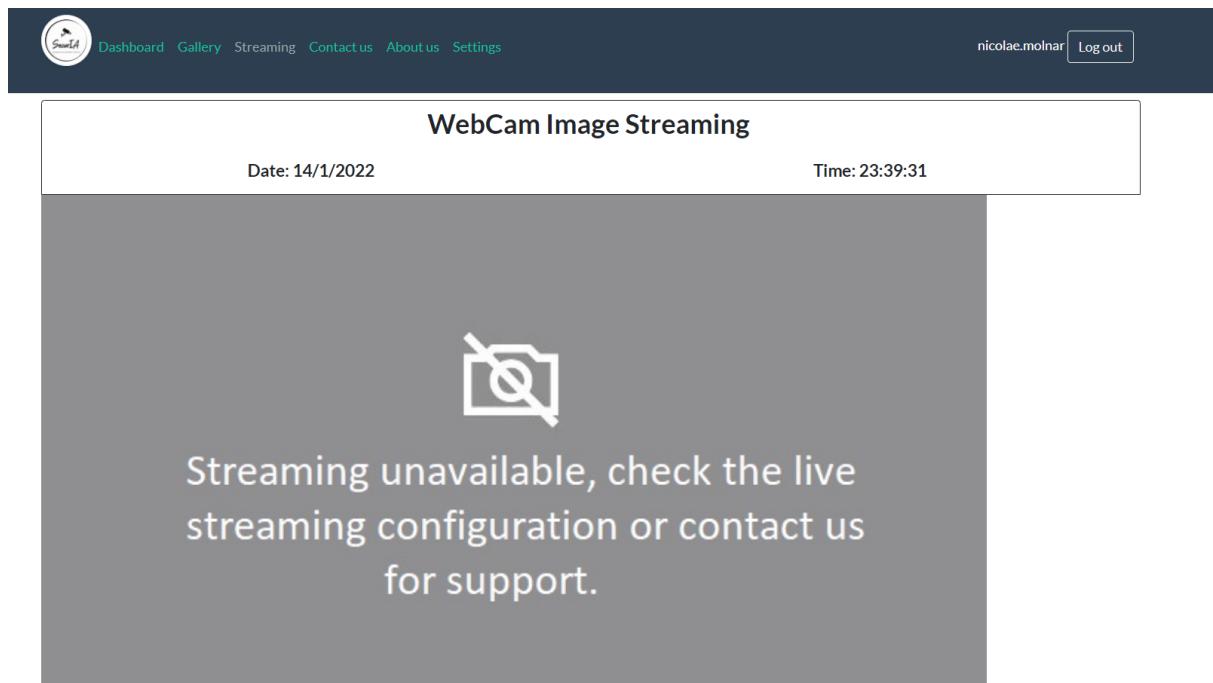
Ejemplo de filtro del día 28/12/2021 (0 imágenes):



3.4.2.1 Streaming

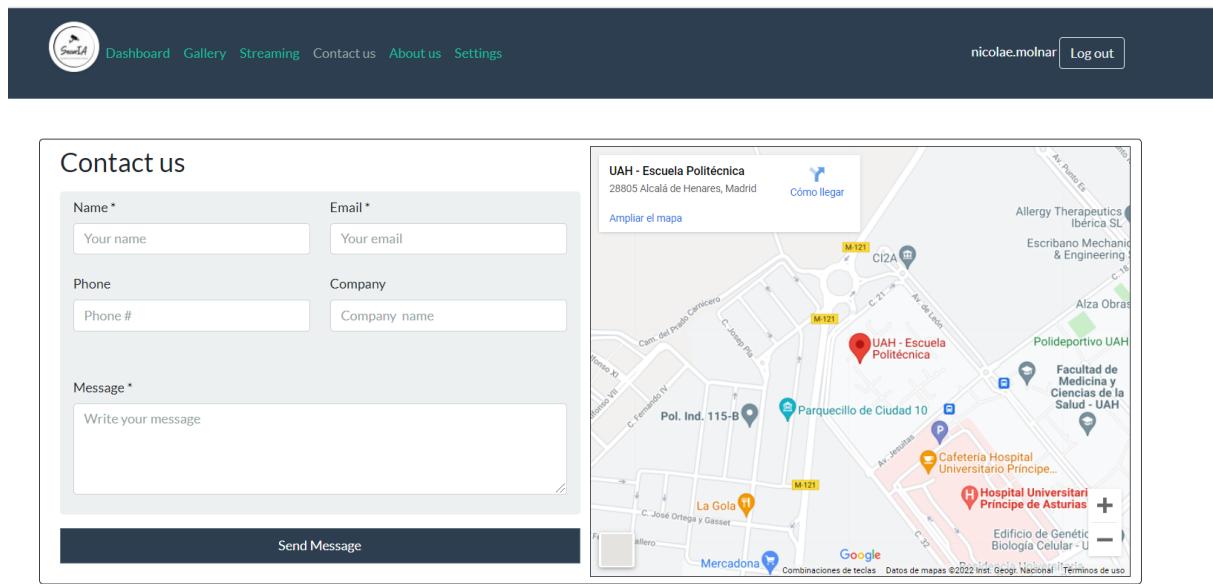
La página de streaming consiste en una función de intervalo de JavaScript que solicita, por medio del servlet `get_stream`, una imagen codificada en base64 y la muestra en un iframe llamado `stream_img.jsp`. De esta manera, la función interval, cuando recibe una respuesta adecuada, actualiza la imagen a lo recibido, en caso de recibir una respuesta errónea, se muestra una imagen de error con indicaciones para el usuario.

The screenshot shows a web application titled "WebCam Image Streaming". At the top, there is a navigation bar with links for "Dashboard", "Gallery", "Streaming", "Contact us", "About us", and "Settings". On the far right, it shows the user "nicolae.molnar" and a "Log out" button. Below the navigation bar, the title "WebCam Image Streaming" is centered. At the top of the main content area, it displays "Date: 14/1/2022" and "Time: 23:37:39". The main content area contains a large video frame showing a person wearing headphones. The background of the video frame is dark, and the person's face is clearly visible.



3.4.2.1 Contact

La página de contacto es una página que funciona con JSP y que utiliza el servlet `request_contact` para enviar una petición de contacto o soporte al servidor. Además contiene un iframe que apunta a google maps, mostrando la sede actual de la empresa, la Universidad de Alcalá de Henares.



3.4.2.1 About

La página about us contiene información sobre los integrantes del grupo, el objetivo del proyecto y las políticas seguidas.

 Dashboard Gallery Streaming Contact us About us Settings

nicolae.molnar [Log out](#)

About us

SecurIA is a project lead by four students of the University of Alcalá de Henares to achieve the objective of a safe environment for living.

An unprotected home is obviously vulnerable to attacks, but even the ones that have an alarm system are often robbed without major consequences. At SecurIA we believe in prevention rather than correction, this means, we offer a system that will record any suspicious activity around your home and report it to you through our notification system.

Thanks to the IoT, you can be aware of the situation of your home even if you are away on vacation, with the only need to be connected to the internet and react to the problem before it even begins.



3.4.2.1 Settings

La página settings es una página que funciona completamente con JSP y cuando se accede a ella, llama al servlet `get_settings`, que le ofrece la información del usuario que auto completará en los campos respectivos. De esta forma, el usuario solo debe editar su información en lugar de introducirla de nuevo entera.

El botón “Save changes” enviará los datos del formulario al servlet `set_settings`, que comprobará si los datos son válidos y, si lo son, los actualizará en la base de datos.

El botón “Delete {username}” enviará el email del usuario que a iniciado sesión al servlet `delete_account` que se encargará de eliminar no sólo el usuario de la tabla users, sino también los sistemas asociados a él y las imágenes y alertas asociadas a esos sistemas

User Settings

Email: nicolae.molnar@edu.uah.es

Password: Confirm password:

First Name: Alexandru Surname: Molnar

Phone number: 666333999 Birth date: 08/03/2001

System Settings

Capture photos Send notifications Live Streaming

Save Changes

3.4.2.1 Admin

Si se introducen en el login las credenciales del administrador, este nos redirige a esta página. Esta página funciona con JSP para comprobar que se ha accedido con el email del admin y javascript para proporcionar el soporte de una consola SQL desde el sevidor Web. Los botones de la izquierda utilizan JavaScript para autocompletar el textarea SQL y crear “Prepared Statements” automáticos.

Welcome: admin@securia.com

Admin Page

Prepared SQL

Select all alerts
Select all contacts
Delete contact
Delete images

SQL Console

SQL:

Result:

Execute

4. Conclusiones

Como conclusión general del trabajo nos ha resultado muy didáctico sacando ideas más claras a la hora de crear, y diseñar aplicaciones Android y páginas Web. También hemos aprendido a interconectar una gran variedad de software de diferentes usos, que hemos utilizado en otras

asignaturas, y hasta la llegada de esta no hemos podido probar la cohesión real de los mismos, ya sea desde el hardware de la placa con los sensores hasta las propias aplicaciones en las que vemos todos esos datos que hemos guardado, modificado con el uso del Servidor y las diferentes funciones que hemos implementado en el mismo y el envío de esos datos a lo largo de la arquitectura del sistema mediante el uso del Wi-fi y métodos de publicar esta información gracias a MQTT. En resumen hemos intentado usar todas las partes vistas en las clases de la asignatura (o casi todas) logrando un sistema que consideramos: inteligente, interconectado, intuitivo, discreto en una versión real y útil en nuestro día a día.

5. Bibliografía

Anexo I – Manual de instalación

Para la instalación del proyecto en otra máquina es necesario seguir los pasos indicados en el documento con un título “**Instalación**”. Cabe destacar que el contenido del servidor web, que debe ir a “C:\xampp\tomcat\webapps” se encuentra almacenado en la carpeta “SecurIA.com” dentro del directorio entregado. Este contenido debe ser copiado a la ruta mencionada junto con su archivo .war.

En caso de que necesite cargar el programa en un ESP32, deberá acceder al archivo “/Arduino/sensor_layer/sensor_layer.ino” y cargarlo. Cuando quiera simular la cámara web en el prototipo, deberá acceder al script “/python_scripts/detection_IA.py” y ejecutarlo.

Anexo II – Manual de usuario de la aplicación Móvil

1. Instalación

Una vez descargada la aplicación en el teléfono, la instalación de la misma se realiza de manera automática. Para comprobar que la aplicación se ha instalado correctamente es preciso ubicar el ícono ejecutable de la aplicación descargada (como se muestra en la figura 1). Para hacer uso de la aplicación SecurIA, sólo es necesario tocar el ícono de la aplicación para abrirla e iniciar sesión en ella.

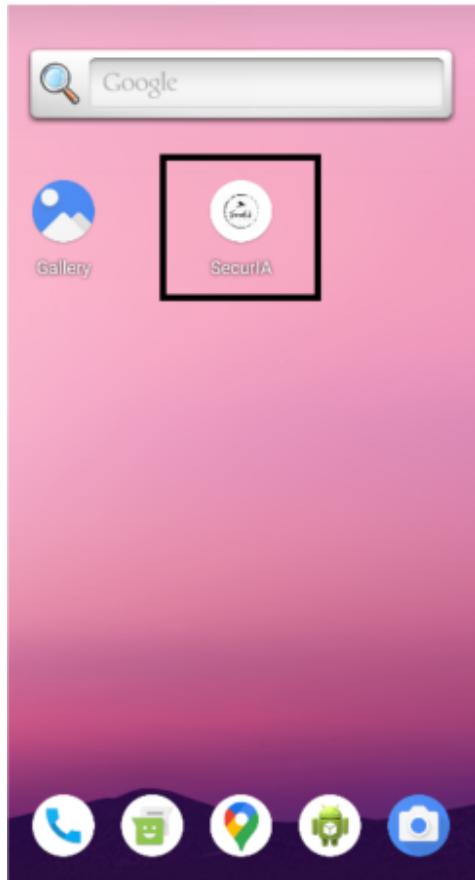


Fig. 1. Pantalla en el teléfono que muestra el ícono de la aplicación SecurIA.

2. Como registrarse

Una vez entremos en la aplicación se nos mostrará una pantalla de inicio de sesión (como se ve en la figura 2). Sin embargo, si somos nuevos usuarios y no disponemos de usuario y contraseña debemos registrarnos.

Para ello debemos pulsar en la etiqueta que pone *Register here!*, la cual nos redireccionará a una página de registro perteneciente a nuestra página web donde debemos poner nuestros datos y validarlos (como podemos ver en la figura 3).



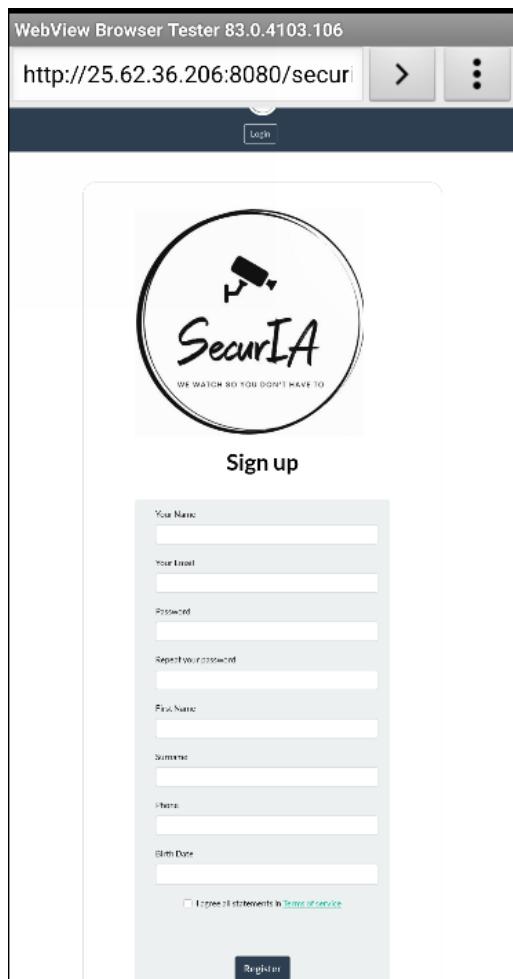
Email

Password

You are not registered yet? [Register here](#)

Fig. 2. Interfaz de Inicio de sesión donde se remarca la etiqueta de Register here.

Una vez registrados ya podremos volver a la ventana de antes, es decir, el Login, para poder entrar con nuestro correo y contraseña con el que nos hemos registrado.



Vista del registro desde el teléfono.

3. Inicio de sesión

Ya registrados y con nuestro correo y contraseña podremos acceder a la aplicación a través de un Login o inicio de sesión que se nos mostrará nada más abrir la aplicación.

Para iniciar sesión debemos pulsar el botón *LOG IN* después de llenar los campos solicitados.

Es estrictamente necesario completar correctamente ambos campos, en caso contrario no podremos acceder a nuestra cuenta y se nos mostrará un mensaje de credenciales erróneas (como se muestra en la figura 4).

NOTA: el usuario administrador solo estará disponible para página web.

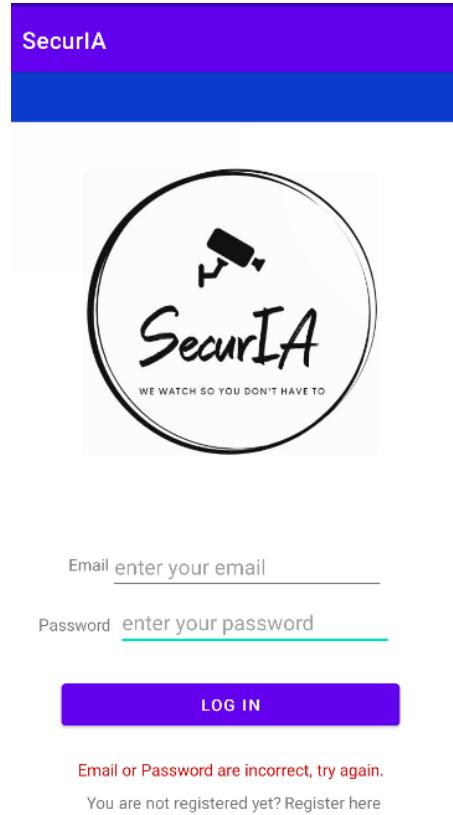


Fig. 4. Interfaz de Inicio de sesión donde se muestra el mensaje de error por credenciales erróneas

4. Menú

Una vez entremos a nuestra cuenta, lo primero que veremos será un menú (como se muestra en la figura 5). En este menú podremos realizar diferentes acciones.

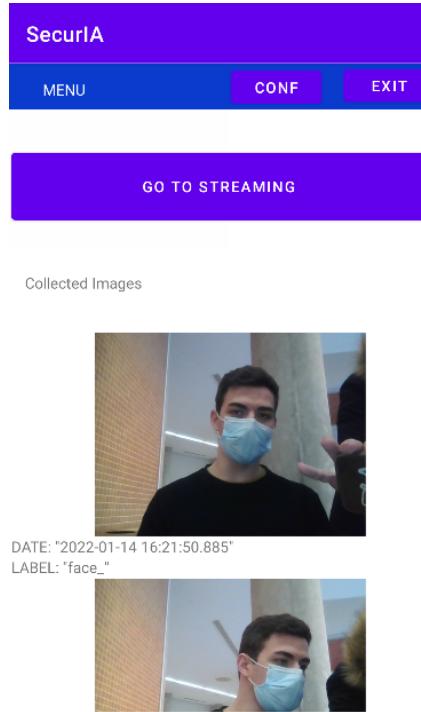


Fig. 5 Interfaz del menú donde se muestran las imágenes y sus etiquetas correspondientes.

En primer lugar, en la parte inferior de la pantalla veremos una galería de imágenes donde tendremos que ir bajando en la pantalla para que nos vayan apareciendo, ordenadas de más actual a más antigua, todas las imágenes. Estas imágenes se corresponden a las alertas que nuestra mirilla captará, de forma que si la mirilla detecta una cara, mano o ambas, capturará la imagen y nos la enviará a esta galería. En caso de que dicha detección sea constante se enviarán capturas cada 5 segundos.

Cada una de las imágenes tendrá asociado una marca de tiempo, que indica el momento en el que fue tomada la fotografía, y una etiqueta que especificará que se está mostrando el la imagen, es decir, si ha capturado una cara, una mano o ambas.

Por otra parte, en la parte superior de la ventana podemos ver que hay un botón más grande que los demás llamado *GO TO STEAMING*. Si pulsamos dicho botón nos llevaría a la ventana de streaming, para poder ver en directo lo que está ocurriendo en nuestra puerta, es decir, lo que nuestra mirilla está capturando en ese momento.

Por último, en la parte superior derecha podemos encontrar dos botones, el botón *CONF*, que nos enviará a una configuración para modificar nuestros datos y opciones de la aplicación, y el botón *EXIT* el cual pulsaremos en caso que queramos cerrar nuestra sesión y volver a la ventana de inicio de sesión.

4.1 Notificaciones

Mientras tengamos la aplicación activa, ya sea en primer o segundo plano, nos llegan notificaciones cada vez nos llega una nueva alerta de detección, es decir, una nueva imagen.

Estas notificaciones se basan en mensajes informativos que varían según el tipo de detección que haya ocurrido. Existen tres tipos de notificaciones:

- Si se detecta un intruso (reconociendo una cara, mano o ambas) se enviará una notificación indicando que se ha detectado una persona en tu puerta.

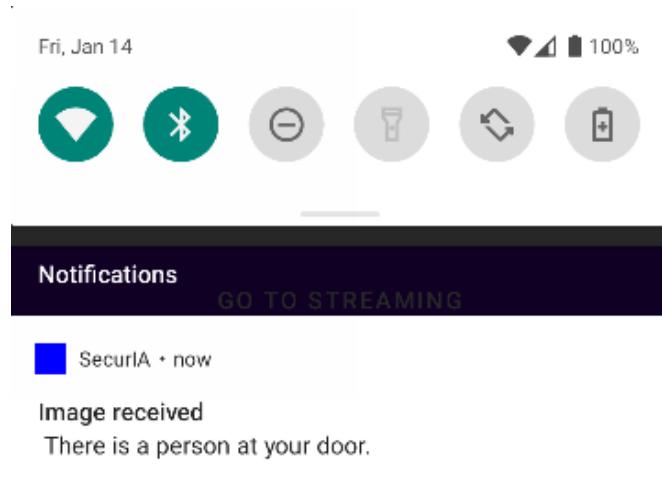


Fig. 6 Notificación de intruso reconocido por la mirilla

- Si se detectase movimiento en algún sensor, distinto al complementario a la mirilla, también se enviará una notificación alertando de esa actividad sospechosa e indicando en qué sensor se a producido la alerta.

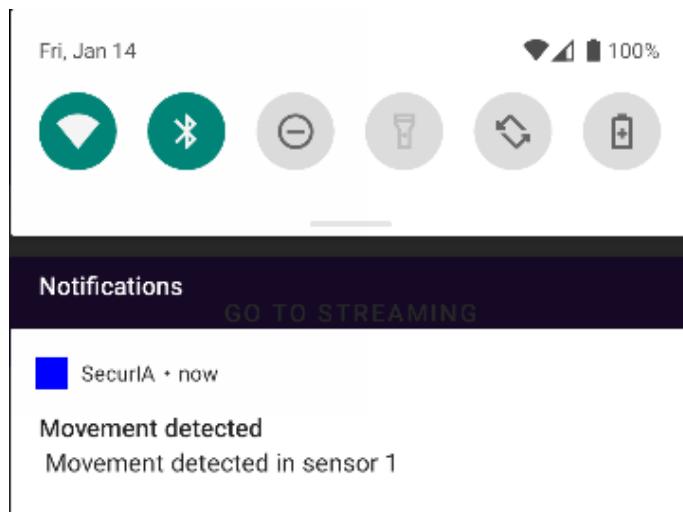


Fig. 7 Notificación de actividad detectada por el sensor 1.

- Si se detecta que la cámara está tapada o no se reconoce nada pero percibimos movimiento, a través de los sensores, en tu puerta, enviaremos una notificación de posible ataque detectado.

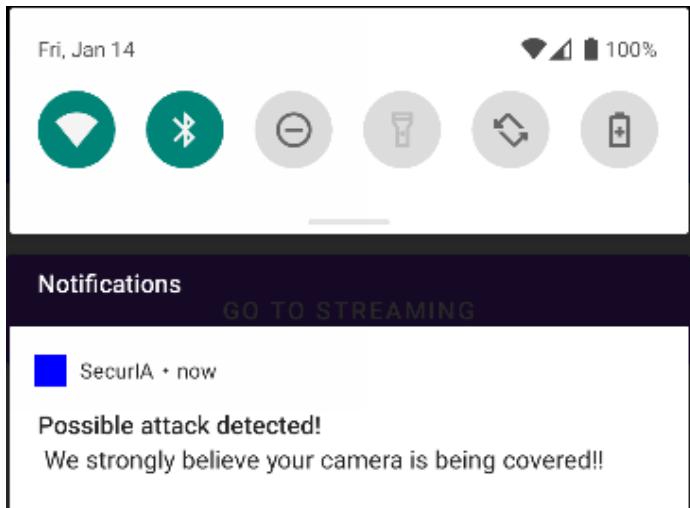


Fig. 8 Notificación producida por cámara tapada pero se detecta movimiento en la puerta.

5. Configuración

Si pulsamos en el botón *CONF* desde el menú, como mencionamos anteriormente, entramos en una ventana de configuración. En esta configuración podemos distinguir dos apartados, que son modificaciones de nuestra cuenta o datos personales, y por otra parte modificación de las opciones de la aplicación.

Para confirmar y guardar los cambios es necesario que ningún campo esté vacío o en blanco. Si alguno lo está no se guardaran ninguno de los cambios realizados y se mostrará el error: *Some field is empty*.

En la parte superior derecha tenemos dos botones *BACK* y *EXIT*, el primero volverá al menú principal del que venimos y el segundo nos servirá para cerrar sesión y volver al inicio de sesión.

5.1 Cuenta

En el apartado de configuración personal podremos cambiar todos nuestros datos excepto el email, que será único para cada cuenta.

Para confirmar cualquier cambio debemos modificar los campos o marcar opciones que necesitemos y finalmente pulsar el botón *SAVE & UPDATE*. Es imprescindible pulsar dicho botón para guardar los cambios.

5.1.1. Cambio de contraseña

Para el cambio de contraseña tendremos que modificar dos campos, *Password*, donde introduciremos la nueva contraseña, y *Confirm password* donde repetiremos la contraseña por seguridad.

Es obligatorio que ambos campos sean iguales, en caso contrario, se mostrará un error: *The password are different*.

5.1.2. Cambio de datos

Si queremos cambiar nuestros datos, tan solo debemos modificar los campos que queramos cambiar y pulsar el botón *SAVE & UPDATE*. En el caso del campo fecha de nacimiento, la fecha debe ser anterior al día actual, si esto no se cumple se le avisará con un mensaje de error: *Date is invalid*.

5.2 App

En la parte inferior de la configuración encontraremos las opciones para manejar los recursos de la aplicación, las cuales serán tres switches con dos posiciones, activado y desactivado:

- *Send notifications*: trata las notificaciones que nos llegan en caso de alguna detección. Si se desactiva no llegarán notificaciones.
- *Capture photos*: trata la captura de las fotos cuando detectamos una intrusión (detección de cara, mano o ambas por la mirilla). Si se desactiva no guardaremos las capturas y por tanto no nos aparecerán en la galería.
- *Life Stream Capture*: trata el streaming. Si se desactiva no se retransmitirá el streaming.

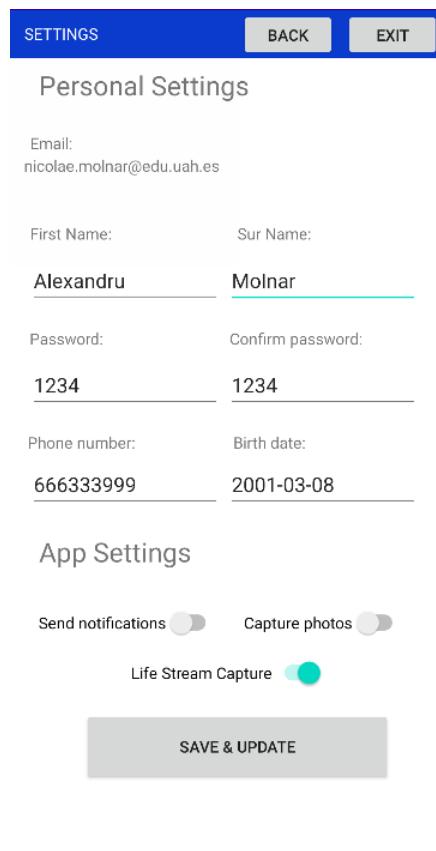


Fig. 9 Vista de la configuración de la aplicación.

6. Streaming

Esta es la clase a la que llegaremos si pulsamos el botón *GO TO STREAMING*. En esta ventana podremos ver una pantalla en grande donde se no retransmitirá el streaming, en el caso de que lo tengamos activo en nuestra configuración, o una imagen un un mensaje de información, si lo tenemos desactivado o hubiese algún problema en nuestra cámara asociada.

El mensaje en caso de no estar el streaming es el siguiente: *Streaming unavailable, check the live streaming configuration or contact us for support.*

Además, encima del streaming nos aparecerá un tiempo que indica la hora actual, la hora en la cual estamos viendo y recibiendo ese streaming.

Por último, en caso de querer abandonar el streaming, tan solo nos hará falta pulsar el botón situado en la parte superior derecha *BACK TO MENU* el cual nos devolverá al menú.

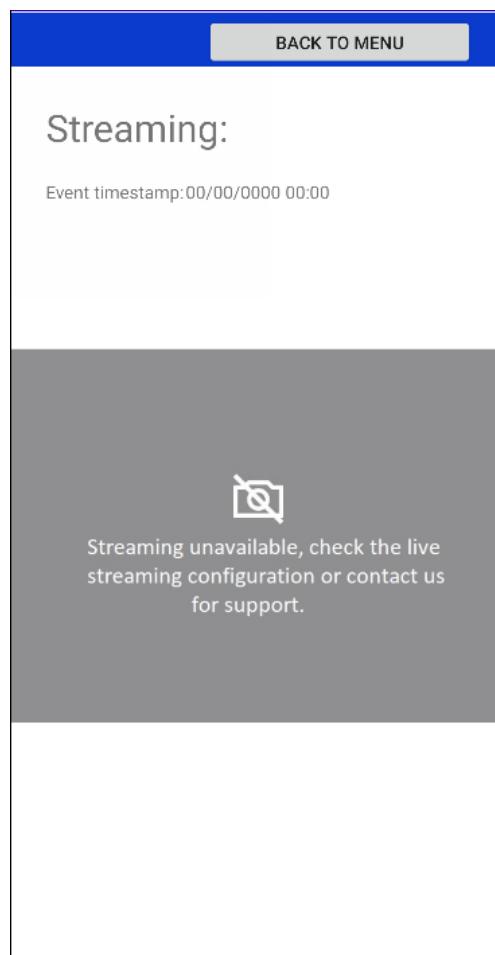


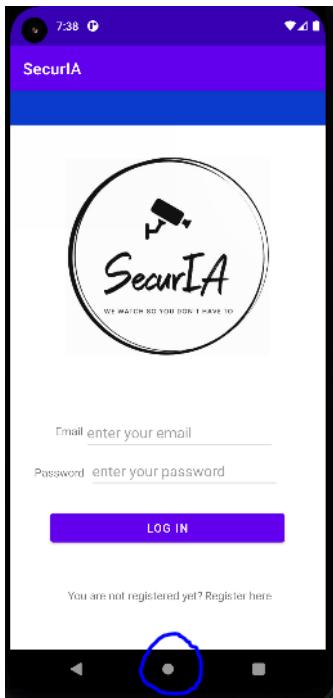
Fig. 10 Interfaz del Streaming donde vemos que la opción de streaming está desactivada.

7. Como salir de la app

Una vez hecho uso de nuestra aplicación, si queremos abandonarla tenemos varias opciones para hacerlo:

- Si queremos simplemente cerrar sesión podemos salir desde el botón *EXIT* del menú o desde el botón *EXIT* de la ventana de configuración. Otra opción es pulsar el botón de “atrás” en el móvil.
- Si queremos cerrar la aplicación por completo tendremos que pulsar el botón del centro de nuestro teléfono o el botón asignado a volver al menú y desde ahí podremos cerrar la aplicación con el botón asignado ver las aplicaciones abiertas de nuestro teléfono y arrastrandola para así cerrarla.

Pasos:



1 Pulsamos el botón para ir al menú



2 Pulsamos el botón actividades abiertas y arrastramos

Anexo III – Manual de Usuario de la aplicación Web

1. Inicio de sesión

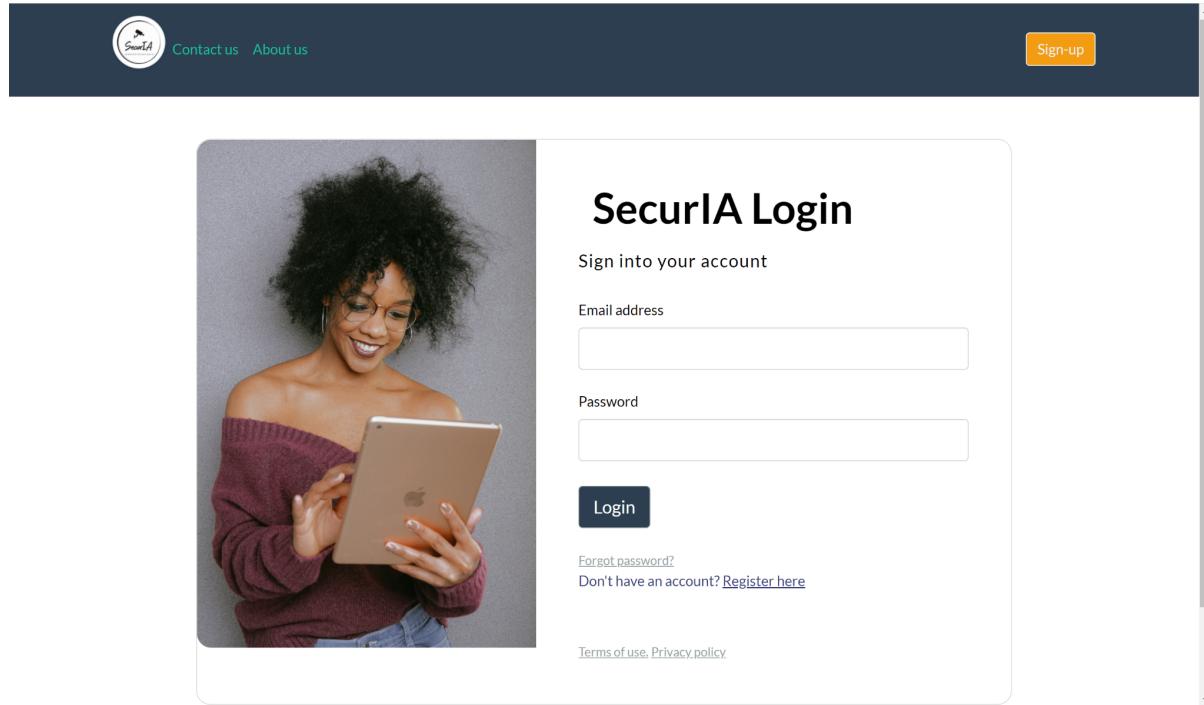


Fig. 1. Pantalla para inicio de sesión.

Al acceder a la página, lo primero que mostrará es una pantalla para iniciar sesión (Figura 1) en la cual, podremos acceder a la página para registrarnos (pulsando el botón Sign up o en Register here), acceder a la página de Contact us (para contactar con los encargados de la página en caso de problema pulsando en Contact us), acceder a la página About us (donde encontrará información sobre SecurIA y sus integrantes, pulsando en About us) y podremos iniciar sesión si ya nos hemos registrado previamente.

Si ya estamos registrados, debemos introducir en el campo Email address la dirección de correo electrónico con la que nos hemos registrado en la página y en el campo Password la contraseña y pulsar el botón de Login.

Si el correo y la contraseña son correctos se le redirigirá al Dashboard de su usuario. Si el correo o la contraseña son incorrectos se quedará en la misma página mostrando un mensaje de error.

2. Como registrarse

The screenshot shows the 'Sign up' form for the SecurIA website. The form includes fields for personal information like name, email, and birth date, along with a checkbox for accepting terms of service and a 'Register' button. To the right of the form is the SecurIA logo, which features a camera icon and the company name.

Fig.2. Página para registrar usuario

Para registrarnos debemos acceder a la página de registro de usuario, como está indicado en el primer punto de este anexo.

Una vez en la página para registrar usuario deberemos llenar todos los campos poniendo su nombre de usuario en el campo de Your Name, su email en el campo de Email, su contraseña en los campos Password y Repeat your password (ambas han de ser la misma); su primer nombre en el campo de First Name, su apellido en el campo de Surname, su número de teléfono en el campo de Phone y su fecha de nacimiento en el campo de Birth Date. Una vez introducidos todos los datos solicitados pulsaremos sobre el botón de “I agree all statements in Terms of service”, si aceptamos los Términos de servicio y para terminar el registro pulsaremos el botón de Register.

Si el usuario se registra exitosamente se le redirigirá a la página de inicio para que inicie sesión en su cuenta como se indica en el primer apartado de este anexo.

Si ha habido algún error al registrar su usuario, se quedará en la misma página con un mensaje de error indicando el problema.

3. Dashboard

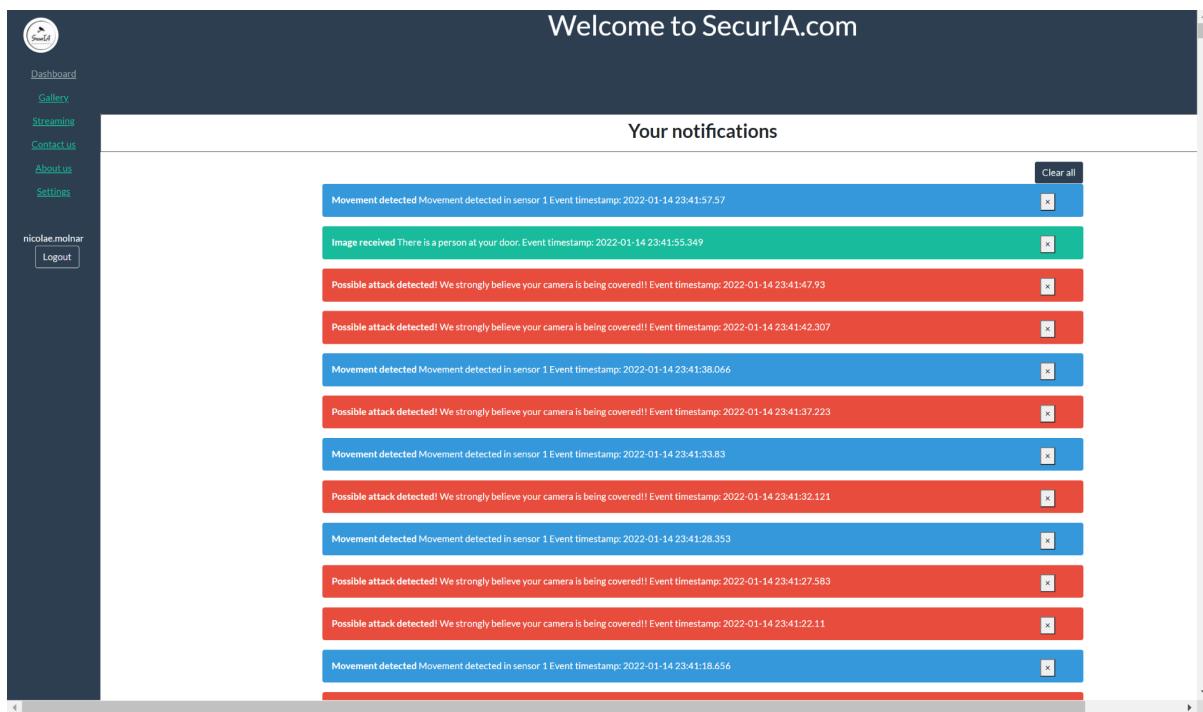


Fig.3. Pantalla de Dashboard

Una vez haya iniciado sesión correctamente, el sistema le mostrará el Dashboard donde podrá ver las notificaciones de su usuario, borrar dichas notificaciones de una en una o borrarlas todas.

Para borrar una de las notificaciones, deberá pulsar sobre el botón blanco con la x situado en la parte derecha de la notificación.

Para borrarlas todas, deberá pulsar sobre el botón Clear all situado encima de las notificaciones.

Las notificaciones informaran sobre la causa de la alerta y cuando se produjo. Hay tres tipos de notificaciones:

- Alerta por detección de movimiento: se notifica cuando alguno de los sensores detectan movimiento delante de su puerta. Es de color azul.
- Alerta por imagen recibida: se notifica cuando hay una persona delante de la puerta y se captura una imagen. Es de color verde.
- Alerta por posible ataque detectado: se notifica cuando alguien tapa la mirilla de la puerta. Es de color rojo.