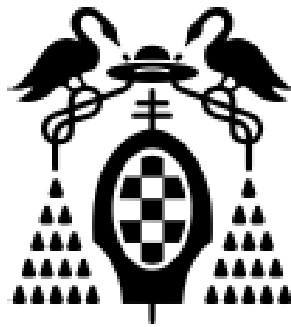


Computación Ubicua

PECL3 / PECL4

Paquete Inteligente

Grado en Ingeniería Informática
Universidad de Alcalá



Aya Atefich
George Dinu
Marcos Fuster
Paula Payo

Enero 2024-2025

Índice

Índice.....	2
1. Introducción.....	4
2. Objetivo del proyecto.....	4
3. Arquitectura del proyecto.....	4
3.1. Capa de procesado.....	5
3.1.1. Broker.....	5
3.1.2. Apache Tomcat.....	7
1. Objetivo de la aplicación.....	11
2. Elementos internos de la aplicación.....	11
9.1.1. Base de datos.....	17
9.2. Capa de aplicación.....	19
9.2.1. Aplicación móvil.....	19
10. Conclusiones.....	25
11. Bibliografía.....	26
Anexo I – Manual de instalación.....	27
1. Requisitos previos.....	27
2. Configuración en Arduino.....	27
3. Configuración de la Base de Datos.....	28
4. Configuración de la aplicación Android.....	29
Anexo II – Manual de usuario de la aplicación Móvil.....	31
Anexo IV – Simulación de datos.....	34

Resumen

Este documento describe el desarrollo de la capa de percepción y aplicación de un sistema de envío para el transporte de fármacos experimentales. Se volverán a mencionar los objetivos tenidos en cuenta desde los inicios del desarrollo del proyecto, seguido de una explicación detallada de la arquitectura e implementación de las capas de percepción y transporte, finalizando con las conclusiones obtenidas tras el desarrollo del proyecto.

Palabras clave: Transporte seguro de paquetes, Arduino, Servidor, Sensores, Arquitectura 4 capas, ESP32, MQTT, MySQL/MariaDB,

1. Introducción

El trabajo se centra en el desarrollo de un sistema integral para el transporte de fármacos experimentales. Este sistema busca garantizar la conservación de las condiciones de los paquetes mediante sensores y mecanismos controlados remotamente. El documento detalla la arquitectura del sistema, incluyendo la capa de procesado, la aplicación móvil y web, y la interacción con un servidor centralizado. También se incluyen anexos que explican la instalación, configuración y simulación de datos para replicar el entorno de operación.

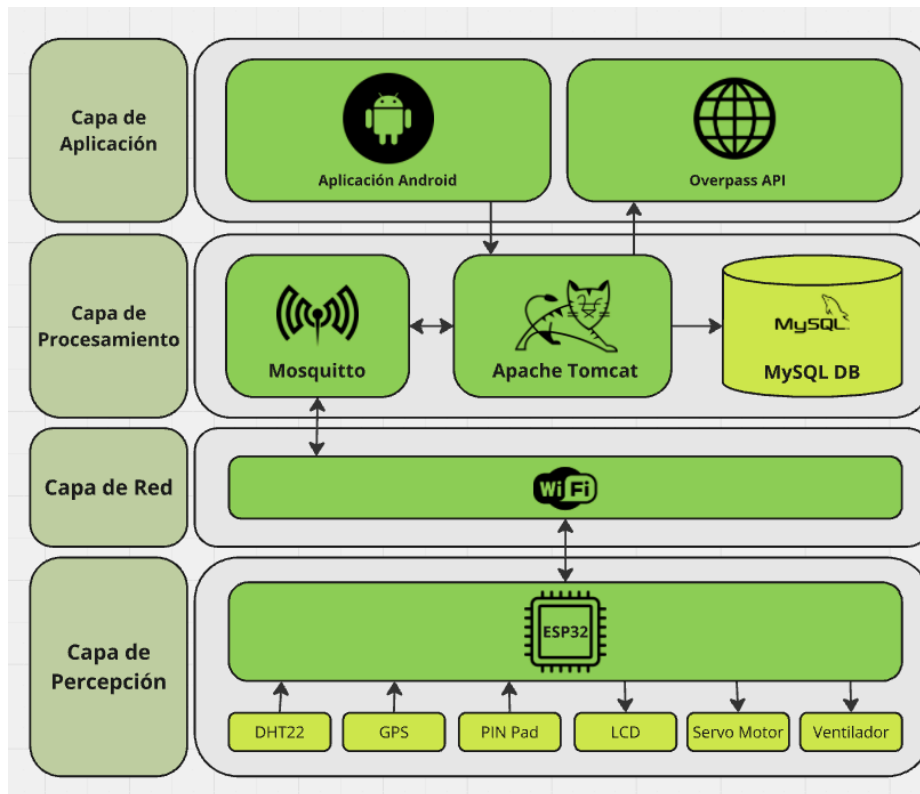
2. Objetivo del proyecto

El objetivo principal del proyecto es desarrollar un sistema integral para el transporte seguro y controlado de fármacos experimentales, asegurando la conservación de condiciones críticas como temperatura y humedad durante el envío. Para ello, se implementa una arquitectura modular de cuatro capas que integra percepción, procesado, aplicación y transporte, garantizando escalabilidad y funcionalidad. Además, se ha buscado diseñar una aplicación para móviles intuitiva que permita a usuarios y empresas gestionar envíos, monitorear condiciones en tiempo real y acceder a datos históricos. El proyecto también prioriza la trazabilidad de los paquetes mediante un sistema de comunicación basado en MQTT, sensores avanzados y bases de datos relacionales, facilitando su implementación mediante manuales detallados que aseguran una instalación y operación accesibles para distintos perfiles de usuarios. Además se ha buscado añadir una mejora basada en un sistema de puntuaciones que valora a los transportistas en función del tiempo que tardan en repartir los paquetes y de su responsabilidad en mantener las condiciones óptimas de temperatura durante el transporte.

3. Arquitectura del proyecto

La arquitectura se mantiene muy similar a la presentada en la PL1, sin embargo hemos eliminado la aplicación web, puesto que con la android ha sido suficiente. Por otra parte aunque no esté completamente implementado por fallos de petición. Se ha intentado usar un API para obtener la velocidad de la vía en la que circula el transportista. Sin embargo, no ha sido posible.

Por otra parte, la aplicación Android no ha necesitado conectarse al broker de MQTT. Puesto que no hemos implementado ninguna notificación a la aplicación.



3.1. Capa de procesado

En este apartado se analizarán los elementos que componen la capa de procesado, indicando primero el objetivo de esta capa y después descomponiendo el resto en subapartados que traten sobre cada uno de los elementos de la capa. En este caso los elementos en los que se descompondrá serían el bróker, apache Tomcat y la base de datos.

3.1.1. Broker

El broker utilizado, como se explicó en la entrega pasada, es Mosquitto, un bróker MQTT que ya viene instalado en la máquina virtual proporcionada para el desarrollo del proyecto. Ya viene configurada de antemano, pero si no funcionara o se quisiera modificar los parámetros de configuración, no tendríamos que ir a `/etc/mosquitto` y ejecutar el comando `nano mosquitto.conf`. Por defecto el puerto de conexión es el 1883 y se utiliza como protocolo de transporte TCP. En el servidor de Java utilizamos 3 clases, las cuáles están dentro del paquete Mqtt, siendo estas:

- **MQTTBroker:** Inicializa los datos de conexión con el broker (Qos, dirección IP de la máquina virtual, id del cliente, usuario y contraseña).

- **MQTTPublisher**: es la clase que se encarga de gestionar la publicación de tópicos mediante mensajes. Inicializa la conexión con el broker mediante el usuario, la contraseña y las opciones de conexión. Solo tiene un método, que se encarga de iniciar la conexión, publicar el mensaje en el topic correspondiente (ambos se pasan como parámetros de entrada) y de cerrar la conexión
- **MQTTSuscriber**: es la clase que se encarga de las suscripciones a los tópicos. Tiene un método de suscripción, el cuál inicia la conexión de manera similar a MQTTPublisher, para después generar un hilo que gestiona los mensajes que lleguen a los tópicos suscritos. La gestión de los mensajes está en el método messageArrived, donde separamos los tópicos por los slash ('/') y transformando a los tipos de datos que necesitamos (id's numéricos a integer o double, nombres o cadenas de texto a String...). Posteriormente comprobamos el tópico del mensaje correspondiente y dependiendo de este, haremos un registro en la base de datos en la tabla correspondiente.

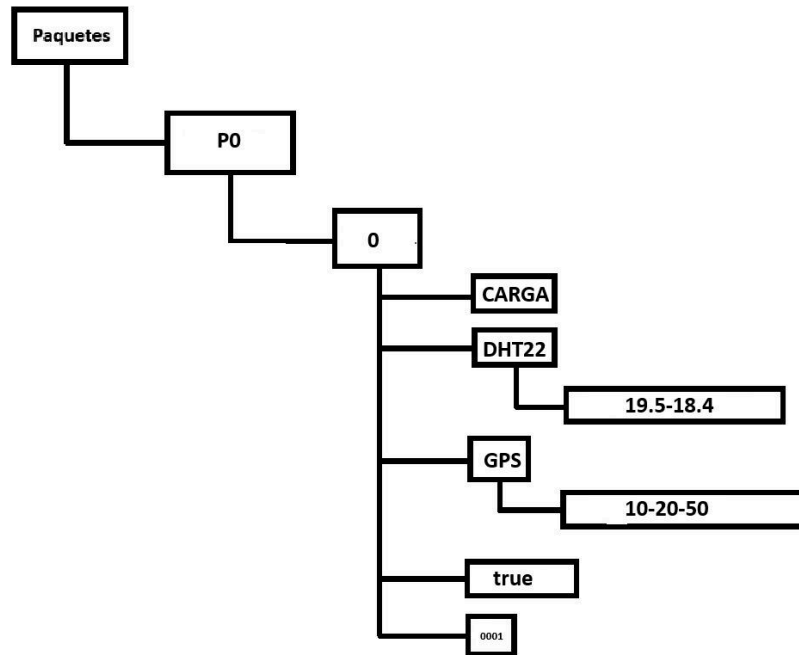
El funcionamiento es el siguiente: cuando se crea un paquete con sus respectivos tópicos, se llama al MQTTPublisher para que mande los tópicos iniciales del paquete, donde después llamará al MQTTSuscriber para que se suscriba a los tópicos de dicho paquete. Se ha intentado la implementación de varias maneras (poner Publishers y Suscribers en un método de crear paquete, intentar crear sub hilos para gestionar procesos de suscripción, etc), pero no ha habido manera de hacer que el suscriber se suscriba al tópico del paquete.

La estructura de etiquetas que se ha elegido es la siguiente



Esta estructura de tópicos permite hacer un seguimiento de los estados y de las mediciones de los sensores de los paquetes en cada envío que tengan.

Por ejemplo:



Si queremos acceder a la temperatura y la humedad del paquete p0, tendríamos que suscribirnos al siguiente topic: Paquetes/P0/0/DHT22/temperatura-humedad. Si quisiéramos suscribirnos a todos los estados de los envíos del paquete p0 nos suscribimos al siguiente topic: Paquetes/P0/+/estado. Si la temperatura interna detectada supera el umbral establecido para el paquete p0, se publicaría en el topic Paquetes/P0/0/ventilador el valor true, para que se active el ventilador y refrigere el interior del paquete.

3.1.2. Apache Tomcat

Para comenzar explicaremos la instalación y configuración del servidor de Apache Tomcat, así como el proyecto desarrollado en él.

Instalación de Tomcat

Para este proyecto, se utilizó la máquina virtual proporcionada a través del aula virtual, que ya tenía instalado Apache Tomcat. El directorio principal de Tomcat está ubicado en **/opt/tomcat**.

Dentro de este directorio, destacamos tres subdirectorios esenciales para la configuración y el despliegue de nuestra aplicación Java:

1. conf

Este directorio contiene los archivos de configuración esenciales para el servidor Tomcat. En particular, hemos trabajado con el archivo context.xml, que se utiliza para configurar diversos recursos que se cargan al inicio de la aplicación.

En nuestro caso, hemos definido una configuración específica para la conexión con la base de datos. El fragmento de configuración utilizado en context.xml es el siguiente:

```
<Resource name="jdbc/ubica" auth="Container" type="javax.sql.DataSource"
    maxTotal="50" maxIdle="5" maxWaitMillis="15000"
    username="root" password="ubicua" driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/mydb"
    validationQuery="SELECT COUNT(*) FROM Cliente"/>
```

En este archivo se configuran los parámetros principales de conexión a la base de datos. El campo url define la dirección de la base de datos, en este caso jdbc:mysql://localhost:3306/mydb, que incluye el protocolo JDBC, la ubicación del servidor, y el nombre de la base de datos (mydb). El campo validationQuery especifica una consulta SQL (SELECT COUNT(*) FROM Cliente) para validar la disponibilidad de las conexiones en el pool.

2. webapps

Este directorio es el lugar donde se despliegan las aplicaciones web que se ejecutan en el servidor Tomcat. En nuestro caso, hemos trabajado con un archivo .war que contiene nuestra aplicación Java.

Para desplegar la aplicación, utilizamos la interfaz de administración de Tomcat, accesible desde un navegador web mediante la URL:

http://IP:8080/manager

Desde esta interfaz, cargamos el archivo .war y ejecutamos nuestra aplicación. El servidor Tomcat automáticamente descomprime el archivo y organiza los recursos necesarios para que la aplicación esté disponible. En concreto el archivo se llama: /ServerExampleUbicomp-1.0-SNAPSHOT.war



Gestor de Aplicaciones Web de Tomcat

Mensaje:	OK									
Gestor										
Listar Aplicaciones		Ayuda HTML de Gestor			Ayuda de Gestor			Estado de Servidor		
Aplicaciones										
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos					
/	Ninguno especificado	Welcome to Tomcat	true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
ServerExample/Welcome	Ninguno especificado		true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
ServerExample/Welcome-1.0-SNAPSHOT	Ninguno especificado		true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
docs	Ninguno especificado	Tomcat Documentation	true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
examples	Ninguno especificado	Servlet and JSP Examples	true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
host-manager	Ninguno especificado	Tomcat Host Manager Application	true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
manager	Ninguno especificado	Tomcat Manager Application	true	1	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
ubicaaa	Ninguno especificado	UbiCompWeatherExample	true	0	Arrancar	Parar	Recargar	Replegar		
					Expirar sesiones	sin trabajar a: 30 minutos				
Desplegar										

3. logs

Este directorio contiene los registros generados por Tomcat y las aplicaciones desplegadas en él. En este caso, configuramos los registros utilizando un archivo llamado log4js.xml.

Dentro de este archivo, se han definido dos tipos principales de registros:

1. log.log: Es el archivo de registro general, donde se almacenan los eventos y mensajes más relevantes generados por la aplicación. Se almacenan en "\${sys:catalina.home}/logs/log.log", es decir "/opt/tomcat/logs/log.log"
2. logmqtt.log: Este archivo se utiliza específicamente para registrar eventos relacionados con MQTT. Se almacenan en "\${sys:catalina.home}/logs/log.log", es decir "/opt/tomcat/logs/log.log"

Configuración adicional:

- En el archivo log4js.xml, configuramos que los archivos de registro no superen los 10 MB. Si un archivo de registro alcanza este tamaño, se comprime automáticamente para optimizar el uso del almacenamiento y facilitar la gestión de los logs.

```

</xml>
<Configuration status="info">
  <Appenders>
    <!-- LOG -->
    <RollingFile name="LogFile" fileName="${sys:catalina.home}/logs/log.log"
      filePattern="${sys:catalina.home}/logs/Ubicomp-%d{yyyy}-%i.log.gz">
      <PatternLayout>
        pattern="%n%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %-30l %logger{36} ### %msg"
        header="--- UBICOMP SERVER EXAMPLE LOG ---" />
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="10 MB" />
      </Policies>
      <DefaultRolloverStrategy max="20"/>
    </RollingFile>
    <!-- LOG -->
    <RollingFile name="LogFileMQTT" fileName="/opt/tomcat/webapps/ServerExampleUbicomp/logs/mqtt.log"
      filePattern="/opt/tomcat/webapps/ServerExampleUbicomp/logs/Ubicompmqtt-%d{yyyy}-%i.log.gz">
      <PatternLayout>
        pattern="%n%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %-30l %logger{36} ### %msg"
        header="--- UBICOMP SERVER EXAMPLE MQTT LOG ---" />
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="10 MB" />
      </Policies>
      <DefaultRolloverStrategy max="20"/>
    </RollingFile>
  </Appenders>

```

Por otra parte también destacamos el pom.xml con las dependencias, hemos tomado las del proyecto de referencia y añadido, org.json para la lectura de ficheros json y com.mysql para el manejo del driver de la base de datos, el cual nos había dado problemas.

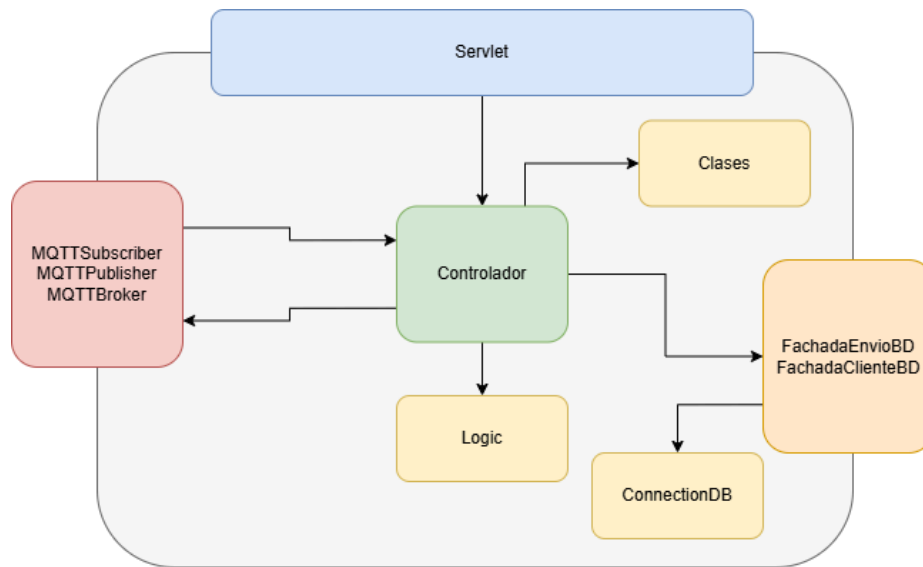
```

<dependencies>
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.1.0</version>
  </dependency>
</dependencies>

```

Aplicación del servidor

A continuación, se realiza un análisis detallado de la aplicación del servidor que se alojará en el .war anteriormente descrito y se relaciona con los elementos del diagrama de clases conceptual proporcionado, explicando cada componente y su funcionalidad. La explicación se dividirá en puntos organizados para cada elemento, siguiendo una estructura lógica:



1. Objetivo de la aplicación

La aplicación tiene como principal objetivo servir como punto intermedio entre las entidades externas (como bases de datos, usuarios, u otras capas) y la lógica del sistema. En ella se definen las clases que controlan la interacción y el flujo de datos entre los diferentes componentes.

Se distingue entre dos tipos de componentes:

- Componentes de interfaz: Interactúan directamente con las capas adyacentes.
- Clases internas: Manejan la lógica central del sistema.

2. Elementos internos de la aplicación

Controlador

- Representa la clase principal. Actúa como intermediario entre los diferentes componentes, asegurando que los datos fluyan correctamente y que los métodos se llamen según lo esperado.
- Funciones Principales:
 - **Actualizar:** Permite modificar datos en el sistema, como un cambio de estado en un envío. Esto se relaciona con el flujo del controlador hacia las Fachadas y la Base de Datos.
 - **Obtener:** Recupera datos almacenados en la base de datos con la ayuda de las fachadas (como FachadaEnvioBD o FachadaClienteBD).

- Registrar: Almacena valores en la base de datos. Esto incluye la funcionalidad de "poblar" la base de datos con datos sintéticos, útil para pruebas y simulaciones.
- **Relaciones:**
 - El **Controlador** centraliza la comunicación con otros elementos como:
 - Servlet: Para manejar solicitudes externas.
 - MQTT: Es capaz de recibir y mandar mensajes al broker
 - Clases: Almacena entidades como Clientes, Envios, o Transportistas.
 - Fachadas: Para interactuar con la base de datos de manera estructurada.

Clases

- Estas son clases que representan las principales entidades del sistema. Son usadas por el controlador para manejar los datos de manera estructurada como objetos.
 - Ejemplos incluyen Clientes, Datos, Envios, y Transportistas.

Lógica

- Encapsula operaciones más complejas que no son responsabilidad directa del controlador. Estas operaciones incluyen:
 - Cálculo de promedios: Por ejemplo, calcular el tiempo promedio que tarda un transportista en completar un envío.
 - Monitoreo de la cadena de frío: Determina el tiempo durante el cual la temperatura supera un umbral crítico para los paquetes.
 - Lógica no implementada: Existe un método para consultar la velocidad de una vía en función de la ubicación. Sin embargo, esto no se ha integrado por problemas con la API.
- La clase **Lógica** es llamada directamente por el **Controlador** para realizar estas operaciones complejas. No tiene contacto directo con otras capas.

3. Elementos de la interfaz de la capa de aplicación

FachadaBD:

La FachadaBD sirve como una capa intermedia que simplifica el acceso a la base de datos. Encapsula la lógica necesaria para interactuar con las tablas específicas y abstrae los detalles de las consultas SQL, permitiendo que el controlador trabaje con una interfaz más sencilla y estructurada.

1. ConnectionDB

- Esta clase contiene los métodos necesarios para:
 - Establecer una conexión satisfactoria con la base de datos.
 - Ejecutar las consultas SQL que las fachadas requieran.
- Funcionamiento:
 - Establecimiento de Conexión: ConnectionDB se asegura de que exista una conexión válida con la base de datos antes de ejecutar cualquier operación.
 - Gestión de Consultas: Las consultas SQL son generadas y ejecutadas a través de esta clase. Además, maneja posibles errores de conexión o ejecución.
 - En el diagrama, ConnectionDB se conecta directamente con las Fachadas (FachadaEnvioBD y FachadaClienteBD), ya que es la base para cualquier interacción con la base de datos.

2. FachadaEnvioBD

- Maneja las consultas SQL relacionadas con las tablas de la base de datos que gestionan información sobre envíos.
- Su funcionamiento consiste en:
 - Recibe parámetros de la consulta enviados desde el Controlador.
 - Flujo de Trabajo:
 1. Establece conexión con la base de datos utilizando ConnectionDB.
 2. Crea la consulta SQL correspondiente para la operación requerida (por ejemplo, insertar un nuevo envío, actualizar su estado o consultar envíos pendientes).
 3. Introduce los parámetros en la consulta.
 4. Ejecuta la consulta SQL.
 5. Gestiona los errores que puedan surgir durante el proceso (como errores de sintaxis SQL o fallos en la conexión).

3. FachadaClienteBD

- Realiza operaciones relacionadas con las tablas de clientes en la base de datos.
- Funciones:
 - Similar a la FachadaEnvioBD, pero enfocada en las tablas y consultas de clientes, como:
 - Agregar nuevos clientes.
 - Consultar información de clientes existentes.
 - Actualizar datos de clientes.
 - Flujo de Trabajo: Idéntico al de la FachadaEnvioBD, pero adaptado a las tablas relacionadas con clientes.

Aunque ambas fachadas tienen un funcionamiento similar, se dividen en dos para: mejorar la organización del código y facilitar la búsqueda y comprensión de las consultas SQL. Especializándose FachadaEnvioBD en la gestión de datos

relacionados con los envíos y FachadaClienteBD para operaciones relacionadas con clientes.

MQTT:

Esta interfaz se encarga de gestionar las peticiones de suscripción y publicación del servidor hacia el arduino, mediado por el broker mqtt. Respecto al Controlador es bidireccional puesto que los publish serán llamados por el controlador y los subscribe se llamarán funciones del Controlador. Técnicamente se ha explicado en el 3.1.1.

Servlets:

Para comunicar la aplicación con el servidor se han usado los siguientes Servlet:

Validar Cliente

Objetivo: Autenticar a un cliente en el sistema.

Datos de Entrada: Solicitud HTTP POST con los siguientes parámetros:

- nombre: Nombre de usuario del cliente.
- pw: Contraseña del cliente.

Datos de Salida:

- idCliente: ID del cliente autenticado.

Obtener Receptores

Propósito: Devuelve la lista de receptores registrados en el sistema.

Datos de Entrada: No requiere ningún parámetro en la solicitud.

Datos de Salida:

- Lista de listas con los datos:
 - ◆ id: ID del receptor.
 - ◆ nombre: Nombre del receptor.
 - ◆ ubicacion: Objeto que incluye:
 - latitud: Latitud de la ubicación.
 - longitud: Longitud de la ubicación.

Registrar Cliente

Objetivo: Registra un nuevo cliente en el sistema.

Datos de Entrada:

- nombre: Nombre del cliente.
- pw: Contraseña del cliente.
- longitud: Longitud de su ubicación.

- latitud: Latitud de su ubicación.
- tipo: Tipo de cliente (receptor/remitente).

Datos de Salida:

- idcliente: ID del cliente registrado.

Obtener Transportistas

Objetivo: Devuelve la lista de transportistas registrados en el sistema.

Datos de Entrada: No requiere parámetros.

Datos de Salida:

- Lista de listas con los datos:
 - ◆ id: ID del transportista.
 - ◆ nombre: Nombre del transportista.
 - ◆ tiempoEnvio: Tiempo promedio de envío.
 - ◆ tiempoPerdida: Tiempo promedio de pérdida.

Generar PIN de Envío

Objetivo: Genera un PIN único para la apertura de un envío específico.

Datos de Entrada:

- idEnvio: Identificador del envío.

Datos de Salida:

- pin: PIN generado para el envío.

Obtener Envíos por Cliente

Objetivo: Devuelve la lista de envíos asociados a un cliente.

Datos de Entrada:

- idCliente: Identificador del cliente.

Datos de Salida:

- Lista de listas con los datos:
 - ◆ idEnvio: ID del envío.
 - ◆ transportistaId: ID del transportista asignado.
 - ◆ paqueteId: ID del paquete.
 - ◆ receptorId: ID del receptor.
 - ◆ remitenteId: ID del remitente.
 - ◆ finalizado: Estado del envío.

Obtener Temperaturas y Humedades

Objetivo: Devuelve el historial de temperaturas y humedades registradas durante un

envío.

Datos de Entrada:

- idEnvio: Identificador del envío.

Datos de Salida:

- Lista de datos con los atributos:
 - ◆ idDato: ID del paquete registrado.
 - ◆ temperatura: Valor de la temperatura.
 - ◆ humedad: Valor de la humedad.
 - ◆ fecha: Fecha y hora de la medición.

Obtener Ubicaciones

Objetivo: Devuelve el historial de ubicaciones registradas durante un envío.

Datos de Entrada:

- idEnvio: Identificador del envío.

Datos de Salida:

- Lista de datos con los atributos:
 - ◆ longitud: Longitud de la ubicación.
 - ◆ latitud: Latitud de la ubicación.
 - ◆ velocidad: Velocidad del paquete en movimiento.

Cancelar Envío

Objetivo: Cancela un envío específico.

Datos de Entrada:

- idEnvio: Identificador del envío a cancelar.

Datos de Salida:

- Devuelve true.

Crear Envío

Objetivo: Registra un nuevo envío en el sistema.

Datos de Entrada:

4. idTransportista: ID del transportista.
5. idPaquete: ID del paquete.
6. idReceptor: ID del receptor.
7. idRemitente: ID del remitente.
8. temperatura_max: Temperatura máxima del paquete.
9. temperatura_min: Temperatura mínima del paquete.

9.1.1. Base de datos

Instalación y Configuración de la Base de Datos

La instalación y configuración de la base de datos se realizaron utilizando la máquina virtual proporcionada, que ya incluía una instalación de MariaDB. A continuación, se describen detalladamente los pasos realizados:

1. Creación de un Usuario en MariaDB

- Accedemos al servicio de *MariaDB* preinstalado en la máquina virtual utilizando el comando:

```
mysql -u root -p
```

- Creamos un usuario llamado ubicua con la contraseña ubicua para facilitar las conexiones remotas:

```
CREATE USER 'ubicua'@'%' IDENTIFIED BY 'ubicua';
```

- Otorgamos permisos al usuario ubicua para que pueda realizar operaciones en la base de datos:

```
GRANT ALL PRIVILEGES ON *.* TO 'ubicua'@'%' WITH GRANT OPTION;
```

- Actualizamos los privilegios para que se apliquen los cambios:

```
FLUSH PRIVILEGES;
```

2. Conexión Remota mediante MySQL Workbench

- Configuramos el acceso remoto para MariaDB asegurándonos de que el archivo de configuración (/etc/mysql/my.cnf) permita conexiones desde cualquier dirección IP (modificando la línea correspondiente):

```
bind-address = 0.0.0.0
```

- Reiniciamos el servicio de MariaDB para aplicar los cambios:

```
sudo systemctl restart mariadb
```

- Desde MySQL Workbench, nos conectamos a la base de datos utilizando:

- Usuario: ubicua

- Contraseña: ubicua

- Dirección IP del servidor

- Puerto: 8080

- Una vez conectados, creamos un esquema para organizar las tablas y datos:

```
CREATE SCHEMA mydb;
```

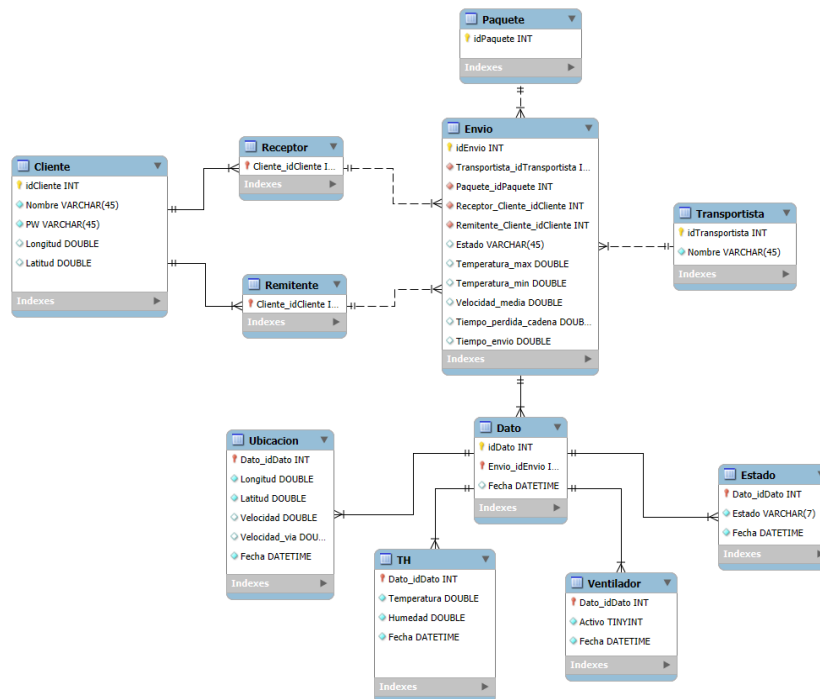
```
USE mydb;
```

Finalmente para la creación del diagrama de tablas hemos usado el creador de modelos de MySQL Workbench y una vez creado lo hemos exportado como script sql y ejecutado en la consola de scripts de la aplicación.

Explicación ER

El modelo representa la información de los clientes (los cuáles se dividen en receptores y remitentes) y los envíos (los cuáles agrupan la información de los paquetes, el transportista y los datos)

- La entidad Cliente almacena un identificador único para cada cliente, junto a su nombre y su longitud y latitud, las cuáles sirven para la localización del cliente. Las entidades Receptor y Remitente tienen una relación de uno a muchos con esta clase a través de la clave primaria de Cliente (idCliente), por lo que, gracias a ellas, podemos distinguir entre clientes.
- La entidad Transportista registra el id y el nombre de los transportistas que realizan los envíos.
- La entidad Dato agrupa las mediciones que se capturan durante el envío, las cuales derivan en las entidades Ubicación, TH (temperatura y humedad), Ventilador y Estado. Todas estas Entidades están relacionadas con Dato mediante una relación de uno a muchos (N:M).
- La entidad Envio almacena información sobre los clientes (receptor y remitente), los transportistas, el paquete y los datos del envío. Es la entidad más relacionada de toda la base de datos.
- Los atributos que destacan son:
 - Tiempo_perdida_cadena: almacena el tiempo en el cuál la temperatura ha estado por encima del umbral (temperatura máxima permitida)
 - Tiempo_envio: tiempo que tarda en enviarse el paquete
 - Velocidad_media: velocidad promedio del transporte



9.2. Capa de aplicación

9.2.1. Aplicación móvil

El proceso de desarrollo de la aplicación comienza con la creación de su diseño visual, estableciendo las interfaces basadas en la estética definida en el mockup de la primera entrega. Este enfoque nos permite garantizar que el producto final mantenga coherencia con las expectativas visuales planteadas desde el inicio del proyecto.

La primera etapa consistió en diseñar la interfaz principal, que corresponde al archivo *activity_main.xml*. Este archivo representa la estructura visual y funcional de la pantalla principal de la aplicación. En esta etapa, se encuentran dos principales botones:

1. **Botón "Registrarse"**: Este botón dirige al usuario a la interfaz de registro, definida en el archivo *activity_register.xml*. En esta pantalla, se muestran los campos necesarios para que el usuario complete su registro en la aplicación, como:
 - Nombre de usuario.
 - Contraseña.
 - Tipo de usuario.
 - Dirección

2. Botón "Inicio de Sesión"

- Este botón dirige al usuario a la interfaz de inicio de sesión, definida en el archivo *activity_inises.xml*.
- En esta pantalla, se presentan los campos necesarios para que el usuario acceda a su cuenta, tales como:
 - Nombre de usuario.
 - Contraseña.
- Una vez que el usuario ha iniciado sesión correctamente, es redirigido a la interfaz de historial, definida en el archivo *activity_historial.xml*.

La pantalla de historial, representada por el archivo *activity_historial.xml*, muestra un registro de los paquetes, clasificados en tres categorías principales:

1. Paquetes en Envío:

- Muestra los paquetes que aún están en proceso de envío. Si un paquete en envío es seleccionado podrán aparecer dos interfaces distintas según el tipo de usuario:
 - i. **Empresa o Remitente:** Mostrará información del envío como la ubicación y la temperatura del paquete. Además se muestra la fecha estimada y un botón para cancelar el envío. La pantalla está representada por el archivo *activity_ver_paquete_e.xml*
 - ii. **Usuario o Receptor:** Mostrará información del envío como la ubicación, la temperatura del paquete o la fecha estimada de entrega. También tiene un botón para generar pin, el cual tras pedir una confirmación de que el paquete ha sido recibido genera un pin para abrir el paquete. La pantalla está representada por el archivo *activity_ver_paquete_c.xml*.

2. Paquetes Enviados:

- Esta sección muestra los paquetes que ya han sido entregados.

3. Paquetes Cancelados:

- En esta categoría se muestran los paquetes que han sido cancelados.

Estos paquetes se muestran basándose en la interfaz *list_element.xml*

Botón para Empresas

Si el usuario es una empresa, debajo de la lista de paquetes se muestra un botón específico para "**Registrar Envíos**". Al presionar este botón, la aplicación redirige a la interfaz que permite a la empresa registrar nuevos envíos, proporcionando los campos necesarios para completar el envío de paquetes. La pantalla está representada por el archivo *activity_envio.xml*.

Registrar envíos

Una vez en la pantalla para registrar el envío existen dos tablas que albergan los transportistas y los receptores. También contiene campos para introducir la temperatura máxima y mínima a la que se quiere mantener el paquete y el ID del paquete. Por último, existe el botón de “CONFIRMAR PEDIDO”, el cual al ser pulsado muestra un mensaje tanto en caso de que de error debido a que algún campo esté sin rellenar o un mensaje de que el envío se ha realizado con éxito. La pantalla está representada por el archivo

Menú Lateral

En la esquina superior izquierda de la pantalla de historial, se encuentra un **menú lateral** (también conocido como "drawer") que ofrece opciones adicionales:

- **Acceder al Historial:**
 - Permite al usuario volver a la pantalla de historial si está en otra sección de la aplicación.
- **Cerrar Sesión:**
 - Esta opción permite al usuario cerrar su sesión actual y regresar a la pantalla principal.

Este menú lateral proporciona una navegación rápida y accesible para el usuario, asegurando que pueda moverse fácilmente entre las secciones clave de la aplicación. Además, la interfaz de historial está diseñada para ser clara, fácil de entender y visualmente atractiva, con un enfoque en la experiencia del usuario.

Una vez desarrollada la interfaz visual de la aplicación, se procedió a la programación de las funcionalidades correspondientes a las diferentes estructuras y botones con los que el usuario puede interactuar. Este proceso implica conectar los elementos visuales con la lógica de la aplicación, asegurando que cada acción tenga el comportamiento esperado. Para ello se han implementado las siguientes clases con su respectivas funcionalidades:

“MainActivity”

- Se muestra una pantalla de inicio con dos botones.
- Al pulsar el botón "Registrarse" en la pantalla principal, la aplicación redirige al usuario a la interfaz de registro (*activity_register.xml*).
- Al pulsar el botón "Iniciar Sesión" en la pantalla principal, la aplicación redirige al usuario a la interfaz de inicio de sesión (*activity_inises.xml*).

"RegistrarseActivity"

- En la interfaz de registro, se validan los campos introducidos como por ejemplo su dirección, se comprueba que está exista. Si algún campo está vacío, se muestra un mensaje de error al usuario.

- Una vez los datos sean válidos, el usuario puede completar su registro y será redirigido a la pantalla de inicio de sesión.
- El usuario podrá volver a la pestaña anterior clickeando a la flecha o al inicio de sesión clickeando al texto “¿Ya tienes cuenta? Iniciar Sesión”

“IniciarSesiónActivity”

- En la interfaz de inicio de sesión, se validan los campos introducidos. Para esto, se realiza una interacción con el servidor a través de un servlet llamado *ValidarCliente*, que verifica si las credenciales introducidas coinciden con los registros almacenados en la base de datos. Si algún campo está vacío, se muestra un mensaje de error al usuario.
- Una vez los datos sean válidos, el usuario puede acceder a ver su historial de paquetes.

“HistorialActivity”

- Se muestran 3 RecyclerView para envíos en curso, envíos enviados y envíos cancelados. Para ello primero obtiene el ID del cliente desde el Intent y llama a dos funciones: *obtenerEnviosCliente(id)* que obtiene los envíos del cliente desde el servidor y *obtenerTipo(id)* que obtiene el tipo de cliente (remite nte o destinatario). La respuesta de este segundo método la usamos para *actualizarInterfaz()*, ya que si eres una empresa te sale un botón para generar envíos. El ListAdapter es un adaptador que se utiliza para mostrar y manejar datos en un RecyclerView en Android. En este caso, el ListAdapter se encarga de adaptar los datos de la lista (elementsE, elementsEs, elementsC) y mostrarlos en las vistas correspondientes dentro de la actividad HistorialActivity.
- Se usan otras funciones como:
 - **mostrarMenu()**: Este método configura un menú emergente (PopupMenu) en la pantalla. El menú tiene dos opciones:
 - Historial: Recarga la actividad actual.
 - Cerrar sesión: Redirige al usuario a la pantalla principal (MainActivity).
 - **generarColor()**: Genera un color aleatorio a partir de una lista predefinida de colores. Se usa para asignar colores a los elementos en la lista de envíos.
 - **obtenerEnviosCliente()**: Este método realiza una solicitud HTTP para obtener los envíos del cliente desde el servidor. Crea una conexión HTTP utilizando el ID del cliente y realiza una solicitud GET al servidor. Si la respuesta es exitosa, la respuesta se procesa en formato JSON y se filtra según el estado del envío (en curso, enviado, cancelado). Los envíos se añaden a las listas correspondientes (elementsE, elementsEs, elementsC).
 - **procesarEnvios()**: Este método procesa la respuesta del servidor (en formato JSON) para filtrar los envíos según su estado y luego

añadirlos a las listas correspondientes. Posteriormente, actualiza la interfaz de usuario llamando a *actualizarRecyclerViews()*.

- Además, la función *onResume* se llama cuando la actividad vuelve a ser visible. Limpia las listas de envíos y vuelve a cargar los datos del cliente llamando a *obtenerEnviosCliente(idCliente)*.

“VerPaqueteActivity”

- Se muestra la información del paquete en envío como la temperatura y humedad, la ubicación del paquete y la fecha estimada.
- Sus funciones son:
 - **onCreate():**
 - Es el punto de entrada de la actividad. Determina si el usuario es un Remitente o Receptor, y carga el layout correspondiente.
 - Diferencia entre tipos de usuarios para mostrar una interfaz distinta. En caso de ser una empresa aparecerá el botón de cancelar el envío y de ser un usuario aparecerá el botón de generar el pin.
 - Configura un mapa utilizando *SupportMapFragment* y obtiene datos relacionados con el paquete como ubicaciones y temperaturas.
 - **mostrarMenu():**
 - Muestra un menú emergente cuando el usuario hace clic en un ícono. Las opciones del menú permiten navegar a otras pantallas de la aplicación.
 - **onMapReady():**
 - Callback para cuando el mapa está listo. Llama a *obtenerUbicaciones()* para cargar las ubicaciones de un paquete en el mapa.
 - **obtenerUbicaciones():**
 - Realiza una solicitud HTTP para obtener las ubicaciones de un paquete desde el servidor. Las ubicaciones son mostradas en el mapa.
 - **parseUbicaciones():**
 - Convierte la respuesta del servidor en una lista de objetos *LatLng* (coordenadas geográficas) para su visualización en el mapa.
 - **mostrarUbicacionesEnMapa():**
 - Muestra las ubicaciones obtenidas en el mapa, añadiendo marcadores y centrando la cámara en la primera ubicación.
 - **obtenerTemperaturasYHumedades():**
 - Solicita las temperaturas y humedades a través de una solicitud HTTP y luego muestra esos datos en un gráfico.
 - **parseTemperaturasYHumedades():**
 - Convierte la respuesta JSON de las temperaturas y humedades del servidor en una lista de pares de valores.:

- **mostrarDatosEnGrafico():**
 - Muestra los datos de temperatura y humedad en un gráfico combinado (líneas) usando LineChart.
- **generarCodigoEnvio():**
 - Solicita al servidor que genere un código PIN para el envío.
- **mostrarCodigoPin():**
 - Muestra el código PIN generado en un AlertDialog al usuario.
- **cancelarEnvio():**
 - Solicita al servidor que cancele el envío de un paquete y maneja posibles errores de conexión.

“EnvioActivity”

- En la interfaz se muestran dos tablas TableLayout, una con los **receptores** y otra con los **transportistas** disponibles. Para seleccionar un transportista, primero es necesario elegir un receptor, ya que al hacerlo se muestra un mensaje con información personalizada para ese receptor, que incluye el **tiempo medio de envío** y el **tiempo de conservación de la temperatura** del paquete. Esta información es esencial para elegir el transportista adecuado según las necesidades de cada receptor y las condiciones del envío.
- Sus funciones son:
 - **onCreate():** Es la función principal que se ejecuta cuando la actividad es creada. Aquí se configura la vista, se inicializan las variables, se realizan operaciones de red en el hilo principal (para pruebas), se cargan las tablas de receptores y transportistas, y se configura la selección de filas. Inicializa la actividad, carga las tablas y gestiona la UI, como mostrar el menú de opciones y configurar los botones.
 - **cargarTransportistas():** Esta función carga los transportistas asociados con el receptor seleccionado. Se vacía la tabla de transportistas y luego se solicita la información desde un servidor web.
 - **setupTableRowSelection():** Configura el evento de clic para las filas de una tabla. Dependiendo de si la tabla es de "receptores" o "transportistas", se selecciona la fila correspondiente. Si es un receptor, se actualiza el ID del receptor; si es un transportista, se valida que haya un receptor seleccionado antes de actualizar el ID del transportista. Maneja la interacción del usuario con las tablas, permitiendo seleccionar un receptor o transportista.
 - **mostrarMenu():** Muestra un menú emergente cuando se hace clic en un ícono (usualmente un botón de menú). Dentro del menú, el usuario puede seleccionar diferentes opciones, como ver el historial o volver a la pantalla principal.
 - **cargarTablaR():** Carga los datos de los "receptores" desde un servidor web y los muestra en una tabla. Hace una solicitud GET al

servidor, parsea la respuesta JSON y agrega las filas correspondientes a la tabla. Obtiene y muestra los datos de los receptores en la tabla correspondiente.

- **cargarTablaT():** Similar a cargarTablaR, pero para los transportistas. Carga los datos de los transportistas desde un servidor web, los parsea y los muestra en la tabla de transportistas. Obtiene y muestra los datos de los transportistas en la tabla correspondiente.
- **mostrarMensajeDeTiempo():** Muestra un mensaje con el tiempo de envío y el tiempo de pérdida de un transportista seleccionado. Si el tiempo de pérdida es demasiado alto, clasifica el estado (bien, aceptable o mal). Muestra un mensaje en pantalla con los tiempos de envío y pérdida de un transportista seleccionado.
- **redondearTiempo():** Redondea el tiempo recibido a dos decimales. Ayuda a dar formato a los tiempos de envío y pérdida para presentarlos de manera más legible.
- **ClasificarTiempo():** Clasifica el tiempo de pérdida en tres categorías: "Bien conservado" (si es menor o igual a 3 minutos), "Aceptable" (si es entre 3 y 5 minutos), y "Mal" (si es mayor a 5 minutos). Proporciona una evaluación del tiempo de pérdida para ayudar a clasificar el estado de un transportista.
- **sendPostRequest():** Envía una solicitud POST a una URL con los parámetros proporcionados. Los parámetros incluyen el ID del receptor, transportista, paquete, temperaturas y otros datos relevantes. Se maneja de forma asíncrona, pero el uso de StrictMode permite realizarlo en el hilo principal (aunque no se recomienda en producción). Envía los datos del envío al servidor y muestra un mensaje indicando si la solicitud fue exitosa o si hubo un error.

Por último, hemos implementado una clase para almacenar las variables globales, en nuestro caso, la IP que se utiliza en las URL de los Servlets. De este modo, se facilita y agiliza el proceso de modificar la IP en todos los métodos que hagan uso de los Servlets.

10. Conclusiones

A pesar de no haber logrado que algunas partes no se logaran conectar, se ha podido desarrollar un sistema encaminado a los objetivos planteados, superando parte de los desafíos técnicos durante su implementación. La arquitectura de cuatro capas ha permitido manejar e integrar herramientas como MQTT, sensores, bases de datos relacionales y aplicaciones móviles y web, garantizando un monitoreo continuo y una trazabilidad eficiente de los envíos, a pesar de que el MQTT no logre realizar correctamente el paso de mensajes al broker..

Se ha logrado una gran familiarización con todas las herramientas usadas en cada capa, consiguiendo así una unificación de todos los conocimientos adquiridos en un solo proyecto completo (en su mayoría).

11. Bibliografía

- **Postman:** Herramienta para pruebas de API que permite diseñar, simular y realizar solicitudes a APIs de manera eficiente. Más información en: <https://www.postman.com/>
- **Figma:** Plataforma de diseño colaborativo para crear interfaces de usuario, prototipos y gráficos. Más información en: <https://www.figma.com/>

Anexo I – Manual de instalación

A continuación se detallan los pasos necesarios para instalar y configurar el proyecto a partir del material proporcionado en la entrega. Este manual incluye la configuración de Arduino, la creación de la base de datos y la instalación de los programas necesarios.

Todos los recursos vienen en el siguiente repositorio de github:

https://github.com/MarcosFP812/Ubicua_Paquete_Envio

1. Requisitos previos

Antes de comenzar con la instalación, asegúrese de contar con los siguientes elementos:

- **Hardware:**
 - Placa ESP32
 - Componentes del proyecto (sensores, cables, etc.).
 - **Software:**
 - **Arduino IDE:** Instalado en su ordenador (versión recomendada: 1.8.x o superior).
 - **Base de datos:** Software de gestión de bases de datos como MySQL, nosotros hemos usado MySQL Workbench
 - **Máquina Virtual:** VMWare y la máquina dada en el aula virtual.
 - **Librerías adicionales** de arduino
-

2. Configuración en Arduino

Paso 1: Conectar la placa ESP-32

1. Conecte su placa Arduino al ordenador utilizando un cable USB.
2. Abra el **Arduino IDE** y seleccione el tipo de placa y el puerto correspondiente desde el menú **Herramientas > Placa y Herramientas > Puerto**.

Paso 2: Cargar el código en Arduino

1. Abra el archivo de código proporcionado (PL2.ino).
2. Haga clic en el botón de **Cargar** para cargar el código en la placa Arduino.
3. Una vez cargado, la placa debería comenzar a ejecutar el programa.

Paso 3: Configurar las conexiones de red (si aplica)

- Incluye la conexión de Arduino a una red, deberá configurar la IP del servidor de red dentro del código de Arduino.

Paso 4: Importar las librerías

- Dentro de la carpeta arduino, se observa una que contiene librerías. Con esta debemos de dar a la opción Sketch > Incluir biblioteca y añadiremos todos los zips de la carpeta.

Paso 5: Compilar con la placa seleccionada.

- Apretaremos el botón de compilar y esperaremos.

3. Configuración de la Base de Datos

Paso 1: Creación de un Usuario en MariaDB

1. Accedemos al servicio de *MariaDB* preinstalado en la máquina virtual utilizando el comando `mysql -u root -p`
2. Creamos un usuario llamado ubicua con la contraseña ubicua para facilitar las conexiones remotas: `CREATE USER 'ubicua'@'%' IDENTIFIED BY 'ubicua';`
3. Otorgamos permisos al usuario ubicua para que pueda realizar operaciones en la base de datos: `GRANT ALL PRIVILEGES ON *.* TO 'ubicua'@'%' WITH GRANT OPTION;`
4. Actualizamos los privilegios para que se apliquen los cambios: `FLUSH PRIVILEGES;`

Paso 2: Conexión Remota mediante MySQL Workbench

1. Configuramos el acceso remoto para MariaDB asegurándonos de que el archivo de configuración (/etc/mysql/my.cnf) permita conexiones desde cualquier dirección IP (modificando la línea correspondiente):
`bind-address = 0.0.0.0`
2. Reiniciamos el servicio de MariaDB para aplicar los cambios: `sudo systemctl restart mariadb`
3. Desde MySQL Workbench, nos conectamos a la base de datos utilizando:
 - Usuario: ubicua
 - Contraseña: ubicua
 - Dirección IP del servidor
 - Puerto: 8080
 - Una vez conectados, creamos un esquema para organizar las tablas

y datos: `CREATE SCHEMA mydb;`
`USE mydb;`

4. Configuración de la aplicación Android

Paso 1: Instalar Android Studio

- Descargue **Android Studio** desde su sitio oficial: <https://developer.android.com/studio>.

Paso 2: Importar el proyecto Android

1. Abra Android Studio.
2. En la pantalla principal, haga clic en **Open an existing project**.
3. Seleccione la carpeta del proyecto Android proporcionada en el material entregado.
4. Espere a que Android Studio sincronice el proyecto. Esto incluye descargar las dependencias del proyecto definidas en el archivo build.gradle.

Paso 3: Configurar las variables globales en la aplicación

1. Abra el archivo `Globales.kt` y modifique la url con la ip del servidor-
2. Abra el archivo `network_properties.xml` y modifique la ip del servidor.
3. Abra el archivo `local.properties` y modifique la ruta del SDK a la suya.

Paso 4: Ejecutar la aplicación en un emulador o dispositivo físico

1. Conecte un dispositivo físico al ordenador mediante un cable USB o configure un emulador de Android en Android Studio:
Para un dispositivo físico:
 - Asegúrese de que las opciones de desarrollador y la depuración USB estén habilitadas en el dispositivo.
 - Seleccione el dispositivo en el menú de ejecución de Android Studio.
2. Para un emulador:
 - Cree un nuevo dispositivo virtual en **Tools > Device Manager > Create Virtual Device**.
 - Configure el dispositivo virtual con la imagen del sistema recomendada y ejecútalo.
3. Haga clic en el botón *Run* (icono de un triángulo verde) en Android Studio para compilar e instalar la aplicación en el dispositivo/emulador seleccionado.

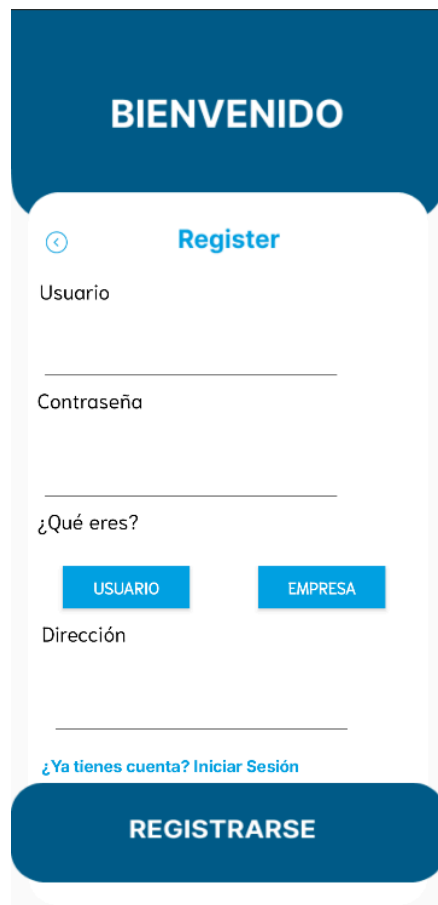
Paso 5: Probar la conexión con el servidor

1. Abra la aplicación en el dispositivo/emulador.
2. Verifique que la aplicación se conecta correctamente al servidor configurado (por ejemplo, enviando datos al servidor o recibiendo información desde la base de datos).
3. Si hay problemas de conexión, revise:
 - La dirección IP configurada en la aplicación.
 - La conectividad de red entre el dispositivo y el servidor.

Anexo II – Manual de usuario de la aplicación Móvil

En primer lugar, para poder hacer uso de la aplicación hay que estar previamente registrado, si ya está registrado se puede acceder al inicio de sesión directamente.

En el registro se debe introducir el usuario y la contraseña (las cuales serán necesarias para el futuro inicio de sesión). En el registro también se tiene que indicar si el usuario será una empresa (botón empresa), encargada de enviar paquetes, o un cliente (botón usuario) encargado de recibir los paquetes. También habrá que introducir la dirección.



En caso de haberle dado al botón Registro de manera accidental, se puede ir a la pestaña de Iniciar Sesión presionando en “¿Ya tienes cuenta? Iniciar Sesión”.

Después del registro se redirecciona automáticamente a la pantalla de inicio de sesión, en la cual, introduciendo los datos, se podrá acceder a la aplicación en sí.

A partir del inicio de sesión, se diferenciará entre usuarios y empresas, aunque para ambos existe un solo menú que cierra la sesión o que devuelve a la pantalla del historial la cual se verá más adelante.



Para acceder al menú habrá que pinchar en las tres líneas que aparecen en la parte superior de la pantalla (lo cual aparece rodeado en la figura que aparece a la izquierda)

Manual Usuario (Cliente)

Una vez iniciada la sesión como usuario aparece un historial con los paquetes en proceso de envío (Envío), los paquetes que se han recibido (Enviados) y los paquetes cuyo envío ha sido cancelado (Cancelados).

Al pinchar en un paquete en proceso de envío se muestra información sobre su localización, el progreso de temperatura y humedad y la fecha de entrega estimada.



Cómo abrir un paquete recibido

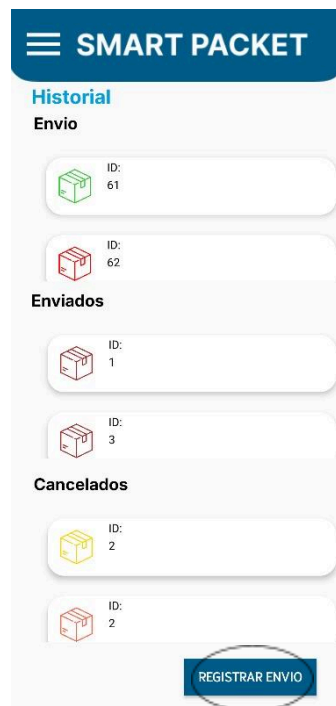
En el historial de paquetes se pincha en el paquete que está en envío. Una vez hecho esto aparecerá la fecha en la que ha sido recibido y un botón para generar un pin (sale rodeado en la imagen inferior). Al darle a este botón se generará un pin el cual se deberá introducir en el paquete cuando sea solicitado. Aparecerá una advertencia para confirmar que el paquete ha sido recibido, **no darle a aceptar si no ha sido recibido**.



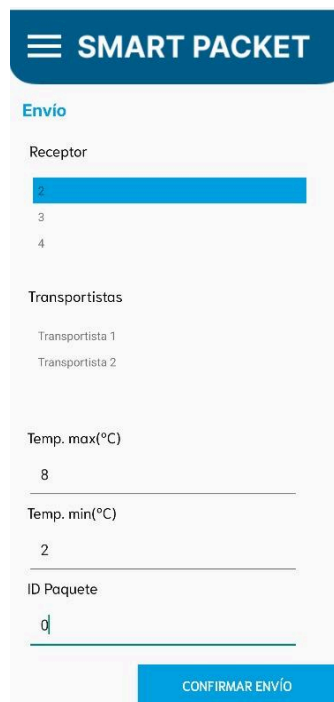
Manual Empresa

Una vez iniciada la sesión como empresa aparece un historial con los paquetes en proceso de envío (Envío), los paquetes que se han enviado (Enviados) y los paquetes cuyo envío ha sido cancelado (Cancelados). También aparece un botón para registrar un nuevo envío.

Cómo registrar un envío



Historial usuario empresa



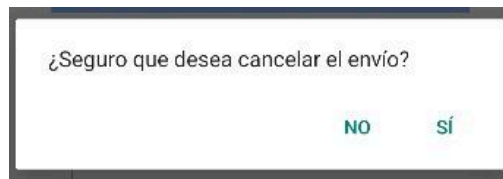
Pantalla registro envío

Primero se deberá pulsar el botón “Registrar envío”.

Posteriormente aparecerá una pantalla con la lista de receptores y la lista de transportistas, y se tendrá que seleccionar uno de cada. También habrá que rellenar los campos de temperatura máxima, temperatura mínima y el ID del paquete. **Sí no se rellenan todos los campos no se podrá realizar el envío.** Finalmente se pulsa en el botón “CONFIRMAR ENVÍO”, y si los datos son correctos el envío se realizará correctamente.

Cómo cancelar un envío

Al pinchar en uno de los paquetes en proceso de envío aparecerá información al respecto, como la localización del paquete, la gráfica de temperatura y humedad, la fecha de entrega estimada y por último el botón “Cancelar envío”. Al pulsar este botón aparecerá un mensaje de confirmación, **no darle a aceptar si no se quiere cancelar el envío.**



Anexo IV – Simulación de datos

Para la generación de rutas hemos utilizado la herramienta de MyMaps de Google y creado 2 rutas, una desde la EPS de la UAH hasta el hospital universitario de Torrejón y otra hasta el hospital de Guadalajara. Esto lo hemos exportado a .kml, donde se pueden leer las coordenadas

Para simular los datos hemos utilizado 2 scripts en python que se alojan en DB/src.

population.py

Este script genera datos sintéticos de envíos simulados basados en coordenadas geográficas extraídas de un archivo KML. Su propósito principal es crear datos de prueba para sistemas de logística, transporte o seguimiento de envíos, modelando movimientos entre puntos geográficos con variaciones en tiempo y velocidad.

1. **Extrae coordenadas geográficas:** Lee un archivo KML para obtener puntos de interés con longitud y latitud.
2. **Simula envíos:**
 - Genera datos para múltiples envíos (movimientos entre puntos).
 - Calcula velocidades basadas en límites de velocidad obtenidos de un servicio de mapas (Overpass API) o establece valores predeterminados.
 - Ajusta tiempos y distancias entre puntos simulando trayectos realistas.

3. **Asegura días laborables:** Garantiza que los envíos ocurran solo en días hábiles.
4. **Genera y guarda datos:** Crea un archivo JSON con toda la información generada para su uso posterior. con esta forma:

```
{
  "0": [
    {
      "Longitud": -3.347906403743479,
      "Latitud": 40.51335110432337,
      "Velocidad": 46.92520930438417,
      "Fecha": "2024-11-05 09:00:00"
    },
    {
      "Longitud": -3.347588140240544,
      "Latitud": 40.51302827857987,
      "Velocidad": 45.017086568460165,
      "Fecha": "2024-11-05 09:00:03"
    },
    {
      "Longitud": -3.347510789758639,
      "Latitud": 40.51302579468961,
      "Velocidad": 44.911993856522685,
      "Fecha": "2024-11-05 09:00:04"
    },
    {
      "Longitud": -3.3472591773067073,
      "Latitud": 40.51268282134628,
      "Velocidad": 43.46200561188414,
      "Fecha": "2024-11-05 09:00:07"
    },
    {
      "Longitud": -3.3469303240144024,
      "Latitud": 40.51249591669912,
      "Velocidad": 41.9718342399298,
      "Fecha": "2024-11-05 09:00:11"
    },
    {
      "Longitud": -3.3466242432824966,
      "Latitud": 40.51232042523267,
      "Velocidad": 39.64847742675504,
      "Fecha": "2024-11-05 09:00:15"
    }
  ]
}
```

temp.py

Este script genera tres conjuntos de datos en formato JSON basados en ubicaciones y tiempos extraídos de un archivo ubi.json. Sirve para simular mediciones de temperatura, estados de ventilación, y estados operativos en un proceso logístico o industrial.

1. **Carga ubicaciones y tiempos:**
 - Lee un archivo JSON (ubi.json) que contiene datos de ubicación con marcas de tiempo.
2. **Genera datos:**

- **Temperatura y humedad (temp.json):** Crea registros simulados de temperatura y humedad a intervalos de 10 segundos entre el tiempo inicial y final de cada ubicación.
- **Estado del ventilador (ventilador.json):** Indica si el ventilador está activo o inactivo según un umbral de temperatura (7.0°C).
- **Estados operativos (estados.json):** Define transiciones de estados como "CARGA", "CERRAR", "ENVIO" y "APERTURA" basadas en tiempos clave.

3. Guarda los datos generados en archivos JSON separados.

Envio.json

```
{
  "Transportista_idTransportista": 1,
  "Paquete_idPaquete": 1,
  "Receptor_Cliente_idCliente": 2,
  "Remitente_Cliente_idCliente": 1,
  "Estado": "Enviado",
  "Temperatura_max": 8,
  "Temperatura_min": 2
}
```

Estado.json

```
{
  "0": [
    {
      "Fecha": "2024-11-05 08:59:00",
      "Estado": "CARGA"
    },
    {
      "Fecha": "2024-11-05 08:59:50",
      "Estado": "CERRAR"
    },
    {
      "Fecha": "2024-11-05 09:00:00",
      "Estado": "ENVIO"
    },
    {
      "Fecha": "2024-11-05 09:22:42",
      "Estado": "APERTURA"
    }
  ],
  "1": [
    {
      "Fecha": "2024-11-06 09:49:28",
      "Estado": "CARGA"
    },
    {
      "Fecha": "2024-11-06 09:50:18",
      "Estado": "CERRAR"
    },
    {
      "Fecha": "2024-11-06 09:50:28",
      "Estado": "ENVIO"
    },
    {
      "Fecha": "2024-11-06 10:13:24",
      "Estado": "APERTURA"
    }
  ],
  "2": [

```

Ventilador.json

```
{
  "0": [
    {
      "Fecha": "2024-11-05 09:00:00",
      "Activo": false
    },
    {
      "Fecha": "2024-11-05 09:00:10",
      "Activo": false
    },
    {
      "Fecha": "2024-11-05 09:00:20",
      "Activo": false
    },
    {
      "Fecha": "2024-11-05 09:00:30",
      "Activo": false
    },
    {
      "Fecha": "2024-11-05 09:00:40",
      "Activo": true
    },
    {
      "Fecha": "2024-11-05 09:00:50",
      "Activo": true
    },
    {
      "Fecha": "2024-11-05 09:01:00",
      "Activo": true
    },
    {
      "Fecha": "2024-11-05 09:01:10",
      "Activo": false
    }
  ],

```

temp.json

```
{
  "0": [
    {
      "Fecha": "2024-11-05 09:00:00",
      "Temperatura": 5.11,
      "Humedad": 32.58
    },
    {
      "Fecha": "2024-11-05 09:00:10",
      "Temperatura": 5.12,
      "Humedad": 38.31
    },
    {
      "Fecha": "2024-11-05 09:00:20",
      "Temperatura": 1.6,
      "Humedad": 64.17
    },
    {
      "Fecha": "2024-11-05 09:00:30",
      "Temperatura": 6.81,
      "Humedad": 76.45
    },
    {
      "Fecha": "2024-11-05 09:00:40",
      "Temperatura": 8.19,
      "Humedad": 33.68
    },
    {
      "Fecha": "2024-11-05 09:00:50",
      "Temperatura": 7.38,
      "Humedad": 72.63
    }
  ],

```

Todos estos datos se han aplicado en la base de datos con el método poblarEnvios de la clase Controlador.

